4 RACING THE E-BOMB: HOW THE INTERNET IS REDEFINING INFORMATION SYSTEMS DEVELOPMENT METHODOLOGY

Richard Baskerville
Jan Pries-Heje
Georgia State University
Atlanta, Georgia 30302
U.S.A.

Abstract

A case study in three companies working on Internet time reveals that the present notion of methodology seems to be changing. In fact, the lack of methodology in its traditional form seems to be characteristic. Instead of methodology, time pressure and requirements ambiguity are found to be at the core of 10 properties of a new and redefined methodology for Internet time development. In this paper, each of the properties is briefly described together with causal links between the properties and using examples from the study of three Danish companies. Furthermore, it is discussed why methodology seems to be undergoing a redefinition when working on Internet time: the underlying philosophical foundation for the change seems to be pragmatism.

1. INTRODUCTION

The Internet is being adopted faster than nearly any other technology. It took 30 years (1920-1950) for the telephone to reach a 60% penetration in the USA. It took 15 years for computers to reach a 60% penetration. But it took only two years for the Internet to reach 60% penetration (*Atlanta Constitution* 2001). Thus the growth of the Internet seems to be similar to an exploding bomb. We have called this phenomenon the "e-bomb," and we have called the frantic speed that companies are developing applications for the Internet and for electronic commerce "racing" since the risk of becoming obsolete—swallowed by the explosion—seems to be imminent.

Working at "Internet time" is not a new phenomenon. The term was coined at Netscape and publicized by Cusumano and Yoffie (2000). However, we have found that a consequence of Internet time is that a metamorphosis is underway in the concept of methodology in certain systems development sectors. This metamorphosis may continue the migration of meaning for "methodology" and involve yet another reformulation of some fundamental assumptions about what constitutes systems development methodology. This reformulation is being driven by the dramatic increase in the influence of time on the systems development process, and the dominance of "time-to-market." Furthermore, this change is not just a new methodology, but a new form of methodology that operates from a largely separate philosophical foundation. The new form implies a change in what methodology means to systems developers.

The current concept of methodology in systems theory regards ordered systems development processes and products. Systems development unfolds in a series of phases or stages. Each phase operates with a defined notation and will often result in a prescribed artifact, such as a design document or program. This artifact is also typically in a prescribed form.

However, methodology is not a crisp concept. Methodology has rather a messy terminological history among scholars in information systems. The term has already undergone quite a migration in meaning. The "-ology" originally implied a study of methods:

> It would be better, as in the philosophy of science to speak of "methods" when referring to specific ways of approaching and solving problems, and to reserve "methodology" for comparative and critical study of methods in general; otherwise this vital field of study is nameless. (Stamper 1988)

Gradually, the distinction between methodology and its underlying term, "method," has become blurred by common usage. For example, compare two authoritative and current definitions of method and methodology:

> Methodology: A systematic approach to conducting at least one complete phase (e.g., design; testing) of software production, consisting of a set of guidelines, activities, techniques and tools, based on a particular philosophy of system development and the target system. (Wynekoop and Russo 1993)
>
> A *method* consists of prescriptions for performing a certain type of working process...is characterized by its *application area*—i.e., the type of working processes in which the method may be

> applied—and its *perspective* (i.e., some assumptions) on the nature of these working processes and their environment. The prescriptions of method are given in terms of techniques, tools, and principles of organization. (Mathiassen and Munk-Madsen 1986)

Methodology and method now seem largely conflated concepts in information systems parlance. Not surprisingly, when Stamper recognized this conflation on the horizon, he admitted that he felt obliged to use the term methodology as a substitute for method, although "under protest bowing only to customary usage" (Stamper 1988).

In the last decade, and perhaps under the weight of loose and unstable conceptualizations, the practicality of methodology has been questioned altogether. A growing number of studies have suggested that the relationship of methodologies to the practice of information systems development is altogether tenuous (Fitzgerald 1997, 1998, 2000; Wynekoop and Russo 1997). It seems that methodology has been so dominant in our thinking about systems development that it may have become somewhat imaginary: a self-confirming hypothesis. For example, one alternative viewpoint situates systems development as "amethodical": the management and orchestration of systems development without the predefined sequence, control, rationality, or claims to universality implied by much of methodological thinking (Truex et al. 2000).

The paper will be developed as follows. In section 2, we will describe the research approach that has given rise to the discovery of a reconceptualized methodology in empirical settings. Section 3 describes the new methodology. Section 4 describes how the new methodology differs from the previous conceptualizations, first in a practical sense and second in a philosophical sense. Finally, section 5 summarizes the discovery and draws some concluding implications from this study.

2. RESEARCH METHODOLOGY

The aim of the research was an exploration of the influence of working on Internet time (Cusumano and Yoffie 2000). At a very general level, one could say that we were testing the hypothesis that working on Internet time would have to cause some changes in the way software development work was organized. But beyond this, no hypotheses were pre-formulated and tested.

Data collection was carried out using open-ended interviewing following an interview guide. The topics in the interview guide are shown in Figure 1.

1. The firm and product and services
2. The interviewee
3. Projects in the organization—from start to end
4. Development model used?
5. Internet time—What does it mean to you?
6. The development process itself
7. Talent, learning, training and knowledge
8. Transfer of knowledge
9. Your biggest problem/Greatest challenge

Figure 1. Topics in Interview Guide

We interviewed in three Danish companies in 1999 and 2000. Two of the companies were new to the authors and the third was a company we had visited over a period of time for a longitudinal study. The main facts about the three companies are given in Table 1.

Table 1. Facts on the Companies Studied

COMPANY	NEWWAYS	PROFWEB	ALFAWEB
NO. OF EMPLOYEES	2000: 50	1999: 40 1998: 50 1996: 20	1999: 12
NO. OF PEOPLE INTERVIEWED	Four people: One project manager, a development manager and two developers/coders.	Two people interviewed in 1999: A development manager and a developer/ coder. Earlier (1996, 1997, 1998) we interviewed four people several times	CEO and development manager interviewed
PRODUCTS	Custom-tailored Internet products for major customers internationally.	Custom-tailored Internet and intranet products interfacing with large existing databases.	A general web-based product sold on the market as a standard product for e-commerce.

The interviews at Alfaweb and Profweb were tape and video recorded. The interviews at Newways were recorded at the site using a portable computer. All of the interviews were transcribed and the transcripts used for analysis.

To analyze the data we had collected, we used the grounded theory methodology (Strauss and Corbin 1990). This research methodology allows the development of a substantive theory of a problem under investigation without prior hypotheses. The chosen grounded theory approach is composed of an alternation between three different coding procedures to analyze the collected data: open, axial, and selective coding. The goal of open coding is to reveal the essential ideas found in the data. Open coding involves two essential tasks. The first task is labeling phenomena. This task involves decomposing an observation into discrete incidents or ideas. Each discrete incident or idea receives a name or label that represents the phenomenon. These names represent a concept inherent in the observation. An example of our coding is shown in Figure 2.

Example excerpt from interview at PROFWEB, May 1999	**Open Coding**
We talk about a generation 1 and a generation 2 Internet application....Generation 1 is just a home page in Danish. A kind of a business card on the net. No functionality. Whereas generation 2 is what we call a web application...from having just	*Two generations of Internet applications called generation 1 and generation 2*
simple functionality we now have a lot of complexity, and we have integration to other systems of different kinds and we have administration.... And often to go from generation 1 to generation 2	*Generation 2 has more complexity and integration*
is not a development it is a revolution. A rejection of what we saw in generation 1 because the foundation wasn't solid enough. We also had a number	*Generation1 was not solid*
of databases supporting some kind of stupid architecture that didn't keep up. Thus generation 2 is characterized by moving further. We needed to	*Architecture didn't keep up*
begin again and make it a formalized process. We were to begin software development. This was a realization that surfaced in an executive meeting.	*Generation 2 requires a formalized process*

Figure 2. Example of Open Coding

The second essential open-coding task is discovering categories. Categorizing is the process of finding related phenomena or common concepts and themes in the accumulated data and grouping them under joint headings, thus identifying categories and sub-categories of data. Developing a better and deeper understanding of how the identified categories are related is the purpose of axial coding. Axial coding involves two tasks further developing the categories and properties. The first task connects categories in terms of a sequence of relationships. For example, a causal condition or a consequence can connect two categories, or a category and a sub-category. The second task turns back to the data for validation of the relationships. This return gives rise to the discovery and specification of the differences and similarities among and within the categories. This discovery adds variation and depth of understanding.

The authors did the first part of the axial coding together. One author presented the results of the open coding to the other. Similarities and differences were noted and discussed. Categories and relationships were identified, discussed, corrected, and changed, until a common understanding of the categories, sub-categories, and their relationships was reached. Finally one of the authors returned to the data for validation. This led to further changes and additions.

Selective coding is the process of determining a core category, relating it to all other categories, validating these relationships, and elaborating the categories that need further refinement and development. The definition of only one core category is usually recommended to maintain clarity and precision and to achieve a tight integration of the categories. However, the authors found that the analysis of the data led to two core categories, one about time pressure, and another about ambiguity of requirements.

Strauss and Corbin clearly advocate grounded theory coding of the data until one core category stands out. However, we decided to have two core categories since we found no empirical evidence in the data that time pressure preceded ambiguity when on Internet time. Besides the two main categories, we identified nine sub-categories. In Figure 3, the relationships between the categories are shown and section 3 of this paper presents an account of each category and sub-category.

3. NEW METHODOLOGY: INTERNET TIME DEVELOPMENT

In the cases, we noted 10 properties of the new methodology. Each of these properties is briefly described below, along with examples of how these properties are manifested in the cases. We will also describe the chain of causal links that we discovered among these properties, which helps explain why this particular set of properties has come to characterize Internet time development. These properties and the causal chain are depicted graphically in Figure 3.

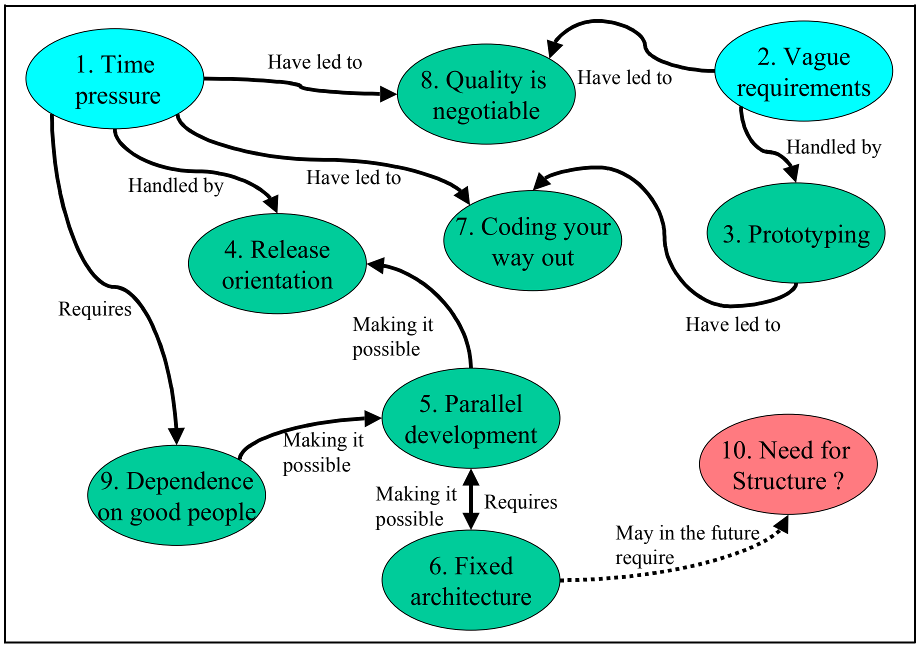

Figure 3. The Model Consisting of Concepts Identified in the Study and the Causal Chains Between Them

3.1 Time Pressure

First-to-market is the central, defining high-priority goal of Internet time development. Minimizing time-to-market from concept to customer use is an all-consuming activity and achievement of this goal drives almost all other elements of the methodology. This goal is not altogether new in business (Smith and Reinertsen 1995) nor in software development (Cusumano and Selby 1995; Iansiti and McCormack 1997); however, the degree to which it has inflamed widespread systems development methodology has not yet been recognized.

We found time pressure to be a condition permeating software development in all the three Internet web companies we studied. At AlfaWeb, they experience the time pressure as a pressure to keep up with all the needs and requirements from the market: "Time...is not what we have now....You just need to move to fulfill customer needs and requirements and then you don't have the time to think."

At NewWays, we studied a web site being developed for a major company in the travel industry. This project was under extreme time pressure right from the beginning. The launch of the web site to be developed was timed with a direct mail on campaign March 1, 2000, and the project didn't really start before January!

And at ProfWeb, they point to the relationship between time pressure and the fast changing technology. A few years ago (1998), they earned a profit on what they call generation 1 web sites, mainly giving companies a face on the Internet. But today, all of the technology that went into those web sites is obsolete. The lead programmer states:

> The operational knowledge we have...has a very short time-span. We can live from the hands-on technological knowledge we have for a year or maybe two, and then it will be a totally different set of tools [with which] we are working.

3.2 Vague Requirements

An inability to pre-define system requirements is the central, defining constraint of Internet time development. The requirements specification that defines the operational goals and strategies in the systems project has traditionally been the heart of systems methodology. However, Internet time methodology accepts a starting point in which the goals, and consequently the specific strategies, are permitted to persist in near or full ambiguity. A good example of this ambiguity was given us at NewWays. A project manager relates:

> Often a project starts without a requirements specification. The project manager says: "Companies come to us and say: We believe there is a treasure buried in the World Wide Web. But we can't find it! We don't just want to copy and change something others have made available on the Internet. We want something new."

Another good example of vague requirements and the ambiguity that it leads to also comes from NewWays. After having struggled to fix a very fluid set of features, it was decided to require written requirements specifications. However, since the customer did not have sufficient knowledge to do that, NewWays did it themselves and just asked the customer to comment. In that way, eight specifications were written over the course of two months with nearly 50% change from version 1 to version 8 of the specification.

At ProfWeb, we find another way of handling vague requirements. Again it involves writing a specification but this time the specification is only specifying future use situations (known as use cases). The interesting thing here is that it is done in such a speedy way, as the following quote by the development manager at ProfWeb illustrates:

> It is now our way of making a receipt for an assignment, saying; if we have understood you correctly then tomorrow you will have a description of what you want specified as use cases and a prototype. Then we send it to the customer, and if they say it's correct, that is what I want, then we move on.

3.3 Prototyping

The idea of using prototypes seems to be widespread and permeating both early and late work in development projects. ProfWeb describes their use of prototypes—as described above—as being part of their core competence. The R&D manager says:

> We live from being technologically in front of our competitors, and from *being able to visualize* more far-reaching and wide-ranging solutions to our customers than our competitors are able.

At NewWays, prototypes are not being used only in cooperation with the front end of a project. In fact, NewWays makes three or four prototypes within a project. The development manager states:

> We start out with a hand-drawn graphical mock-up. It doesn't derail the discussion with technical details, thus we can concentrate on getting the systems ***architecture*** in place. Then we write a basic requirement specification with use cases and visual sketches included. This ensures that two people reading the same specification get kind of the same picture inside their head. The next thing we do is to make a clickable HTML-version. It has no content but it gives the user an experience. Then we make a prototype and then the final system.

3.4 Release Orientation

The vague requirements are not just something we see in the beginning of a project. In fact, it continues throughout the project. One consequence is what we have named a "release orientation." Software systems are produced in a series of ever more refined and extensive versions of the product, and each release contains bug fixes and new features. These maturing product cycles characterize major Internet software development in which competition demands significant product and feature changes every few months (Cusumano and

Yoffie 2000, p. 299). Our cases share this release orientation notable in this well-known experience. This release orientation helps relieve some of the time pressure because there always seems to be a new release one or two months ahead. If a feature doesn't make it for the contemporary release, it is less of a crisis because it can simply be postponed to the following release, which is never very far behind.

3.5 Parallel Development

It appears that the release orientation demands a fast cycle time that is impossible to meet in a serial process. Parallel development processes seem to flourish along with release orientation, meaning that a number of activities take place at the same time. For example, typical NewWays projects have a duration of two to three months. A waterfall-like model is seen as much too slow. Therefore, NewWays will have several parallel development processes running at the same time.

This means that products and releases have to be designed and coordinated for parallel development, another aspect common to large-scale Internet software development (Cusumano and Yoffie 2000, p. 14). To make sure that the parallel development activities still result in a consistent and coherent product, a very important role in projects at NewWays is the coordinator. In every project, the coordinator ensures that the interfaces between groups are kept well defined.

3.6 Fixed Architecture

To make parallel development possible, it is also necessary to have some method to divide work. Interestingly, we have found in all three cases this necessity has led to a fixed three-tier architecture. At NewWays, the development manager describes it in the following way:

> Architecture is important to NewWays. Typically an application has three layers: At the bottom you have a database with content. In the middle you have the business logic. And, at the top, you have the HTML generating logic, typically written in Visual Basic Scripts.

Thus the architecture is used as an ***important coordination mechanism*** to divide the work in the project, as the following quotes clearly illustrate: "Typically the graphical person is drawing something in PhotoShop which the HTML person then can cut up and put into tags," says one developer and another continues, "Which means that we are released from worrying about presentation and can concentrate on the heavy things" (the business logic and the database).

At ProfWeb, a very similar architecture was found. The development manager reveals:

> We have also developed a ProfWeb three-layer architecture where you can see the databases we want to specialize in....In the middle is our logic layer, which we are making as objects. In the future we believe there will be many standard products here...and then we place the roles in our ProfWeb development model in relation to the architecture.

3.7 Coding Your Way Out

The short time frame allowed for developing applications also introduces a coding focus or even hacking as a project manager from NewWays expresses it: "You have to accept that ***hacks are being made***. That you don't have time to think systematically. And that you don't reuse because of the time pressure."

Another very interesting example of coding your way out was found at AlfaWeb. They simply developed their own programming language to be able to speed development to what was needed. The CEO states:

> PMA is our own programming language...we have created our own server-site programming language to Unix allowing us direct access to the databases....One good reason is that when we need something we just put it into the language...it allows us to do things fast, incredibly fast.

3.8 Quality Is Negotiable

Quality is a term often used—and misused. Everyone is for it, everyone feels they understand it, and everyone feels that others are causing the problems (Crosby 1980). Three different ways of looking and talking about quality have appeared over the last 20 years. One school of thought focuses on fulfilment of customer expectations, thus suggesting that quality is the degree of fulfilled expectations. Another way of thinking focuses on measurable product attributes: defining quality as conformance to requirements. The third way of looking at quality believes that a good development process will lead to quality. The three resulting kinds of quality can be named expectation-based, product-based, and process-based quality.

As a consequence of both time pressure and vague requirements, we have found that both product-based and process-based quality in our cases seems to be ignored. However, since quality also can be defined as expectation-based, thus fulfilling customer and user expectations, and since customers and users seems to expect low quality, we decided to call this phenomena negotiable quality.

NewWays is an example of negotiable quality. The product under development during our study was released in the beginning of March 2000. However, it wasn't finished. "In the week after we corrected defects," tells one of the developers, "and we also made some of lowest priority scenarios after the release." The project manager adds: "We are doing module testing, but typically we don't have enough time."

ProfWeb was also struggling with quality. They knew it wasn't good enough. They knew that there might be potential catastrophes waiting, and they had started thinking about what to do, as the development manager relates:

> We collect a test group for every project. At least that is the plan for the future, but right now we are running the pumps, not financially, but we are very busy....I have a capacity planning system and the UNIX department is booked four months ahead.

Thus time pressure definitely is a cause of the negotiable quality, as is also clearly seen from the following quote about maintaining an existing site originally developed by ProfWeb two years before:

> Documentation works fairly well....for smaller maintenance projects...but if you have larger sites to maintain like the XYZ site we did two years ago and are updating right now, then so many new technologies and new tools have arrived, that in fact you start from the beginning again.

3.9 Dependence on Good People

Time pressure is the primary reason why good people are in high demand. As one of founders of ProfWeb phrases it: "I believe the largest bottleneck we have is to get enough qualified employees."

However, not all kinds of IT people seems to be in high demand. Traditional analysts are not in as high demand as the technical folks who are close to the code, as the following quote from the development manager at ProfWeb illustrates: "I also realized that the job market is such that I could find 25 new consultants tomorrow but I wouldn't be able to find two new programmers."

The reliance on good people is even more important in the smaller company, AlfaWeb. The CEO says:

> The people I have sitting in there are bright. I thought I was good at coding back then but I can't follow these people. They are lightning intelligent, and maybe, at times, a little strange, but very bright and very intelligent....they also have a good salary, that is part of this business isn't it?

Thus the last part of the quote above leads us to another interesting observation, namely that many nontraditional perks are put into place for retaining employees. It might be offering computer games and room to play, or it could be cappuccino machines or pool tables, as we have seen in some places when visiting for interviews.

However, there is also a downside to the reliance on good people. Gold plating (adding things that the customer never asked for) seems to be common. "That may be the biggest problem we have in this business. Everyone wants to be proud of what he or she is doing so they put in some extra," acording to the project manager at NewWays.

3.10 Need for New Kinds of Structure

An issue that is closely related to methodology and to a number of issues we have addressed above is structure. We haven't been able to establish a solid causal relationship but we have indications that seem to reveal that the older and larger the organization and/or the customers, the larger the need for structure.

AlfaWeb, which was only half a year old at the time of the interview, was not feeling any need for structure. The CEO, answering the question of what it is that creates quality and innovation, said: "I believe it is the informality but also the lack of formal structures. If people have too close-knit a framework to work in, you may cut down on creativity."

NewWays, being close to two years old and having 50 employees, had started creating some structures. In fact, they had (in one single project mentioned earlier as the project having a requirements specification) placed a number of object-oriented techniques in use. The technique used for the requirements specification was, for example, use cases:

> We gathered people from content editing, HTML, project managers and me as developer. We then focused on the central themes and identified use cases....Based on that we then made state-activity-diagrams, and then we wrote the requirements specification.

NewWays is planning to use the experience from this project to see whether this structure can be used in other (or maybe in all) projects.

Finally, ProfWeb tried to enforce structure twice without succeeding. The first time was a development manager who, as part of his graduate studies at a university, developed a methodology. However, that methodology was too traditional and was never really used. The same happened in a second attempt, which originally was presented in the following way by the development manager:

> generation 2 is what we call a web application...from having just simple functionality we now have a lot of ***complexity***, and we have ***integration*** to other systems of different kinds and we have administration....Thus we needed to...create a formalized process....And based on this we decided to take up a classic development methodology with five phases: pre-analysis, analysis, design, implementation, and editing.

However, this second attempt to add structure also collapsed. In fact, the development manager quoted above left the company partly because of the lack of acceptance for the structure he wanted to add.

We conclude that the breadth of resources available for systems development drives the degree of structure in Internet time methodology. There seems to be a need for structure, but the traditional structures seem to fail. When resources grow in breadth without structure, quality seems to be driven down, perhaps through ineffective resource use, inefficiency, and poorly coordinated activity. Structure is added almost begrudgingly, and only as little as may be necessary to keep the development activities focused on the goals of fast delivery of desired features.

4. DISCUSSION

A first reaction to the above description of Internet time methodology might be: "Ok, these are the properties of the methodology, but where is the methodology itself?" This reaction betrays a set of assumptions about methodology rooted in the conflated concept we have been using for methodologies in the 1990s. Such a reaction overlooks the possibility that there is nothing more in this methodology, nor actually necessary for the methodology, than that stated by the 10 properties described in the preceding section.

Internet time methodology is not a methodology in the traditional sense (if indeed there is anything stable enough in the concept of methodology that could be characterized as traditional), but is rather a different form of methodology that can only be understood in practice if we rethink what it means to follow a methodology or to be a methodology. This rethinking is similar to the earlier rethinking that conflated methodology and method. It is not dissimilar to the conceptual leap necessary for procedural programmers to make the transition from procedural programming languages (e.g., Fortran or Pascal) to declarative programming languages (e.g., Lisp or Prolog). First, we must recognize that in the conflation of methodology and method, we have imbued the concept with certain procedural and reductionist properties. From the definitions given earlier in section 1 for method and methodology (Mathiassen and Munk-Madsen 1986; Wynekoop and Russo 1993) we discover a concept laden with certain assumptions. Among these assumptions we find that methodology involves distinct periods: phases, stages. We also find that activities in these phases are prescribed as processes, prescriptive guidelines, etc. We also find an assumption that these activities should revolve around a definite set of techniques and tools. There is a context: a use-setting or an application area. Finally we find an assumption set or philosophy to which the methodology responds.

Second, we need to realize that the concept of methodology to which Internet time methodology corresponds is dissimilar in certain ways. We still find a context and we still find a philosophy. However, the remaining assumptions have been shifted: either changed in essence, de-emphasized, or entirely missing. For example, the stages may no longer be fixed, and may routinely change depending on the nature of each release; or these stages may iterate continuously through the life of the product; or they may merge for one release and separate for others. The activities for each release may be innovational or redefined daily, depending on the features and the time available. The techniques and tools may be completely left up to the individual programmer/developer, or they may be opportunistically selected according to the latest technological advances.

In a "traditional" sense, this development environment may not be a methodology at all. Indeed, it could be characterized as entirely amethodical (Truex et al. 2000). However, it may represent the beginning of a transition of our understanding of information systems methodology away from its current position, one in which it has been conflated with method, toward one that more accurately responds to information systems development practice. This may be an entirely natural migration of the meaning of information systems development methodology, when we consider the historical basis of this meaning.

Information systems development methodology has traditionally aligned within systems methodology in general. According to Klir (1991), the systems movement emerged from three principle roots: mathematics, computer science, and "systems thinking." A key aspect of systems thinking is holism, the antithesis of reductionism, which originated in Gestalt theory. Consequently, systems methodology in general can be associated with the last century's protest against reductionism and positivist science. However, systems development methodologies are a design science, rather than more abstract pure mathematical science:

> Hence, the *primacy of problems* in systems methodology is in sharp contrast with the *primacy of methods* in applied mathematics. It is the most fundamental commitment of systems methodology to develop methods or solving genuine systems problems in their natural formulation (Klir 1991, p. 88).

There are two kinds of systems methodology: hard and soft systems methodologies (Flood and Carson 1988). Hard systems methodology is based on the assumption that problems can be structured, through a problem formulation stage, in terms of well-defined objectives and constraints. Hard systems methodology remains closely related to its mathematical roots. However, the primacy of problems in systems science, moves methods out of their mathematical driver's seat. The methodological tools for dealing with a problem are chosen in such a way as to best fit the problem rather than the other way round. Even in hard systems methodologies, the tools may not only be mathematical in nature, but also consist of a combination of mathematical, computational, heuristic, experimental, or any other desired methodological trait.

Soft systems methodology deals with ill-structured problems, in which objectives or purposes are themselves problematic. Its strongest proponent in the systems science community has been Peter Checkland, whose formulation of soft systems methodology has been persistently adapted in information systems development (cf. Avison and Wood-Harper 1990; Checkland 1981). However, Checkland's contribution also shifted the foundations of systems methodology. Checkland drew heavily on action research, both as the scientific method by which he developed and validated his methodology, and as the philosophical basis of his soft systems methodology. In this way, the philosophical roots of action research have crept into the foundations of the body of systems science.

Notably, action research had already been applied to information systems development. The action research underpinnings of the socio-technical movement were independently brought from the Tavistock Institute into the information systems arena by Mumford (cf. Mumford and Weir 1979). Thus the influence of action research on information systems methodologies arrived in both systems science and socio-technical forms.

Thus action research sprang forward in response to the primacy of information systems methodology, both from Checkland's ill-structured systems problems and Mumford's concern for the human problems of organizational information systems. Action research is itself an attempt to structure the solution-seeking behavior of social scientists in rather familiar terms to systems designers: stages, guidelines, and a set of techniques (cf. Lewin 1947a, 1947b; Susman and Evered 1978).

But Lewin was, to a certain degree, only instantiating Dewey's logic of controlled inquiry (Dewey 1938) as a generalized scientific methodology for clinical use organizational development. If we return to Dewey's original philosophy, however, Dewey was not merely prescribing a logic of scientific inquiry, but also describing the methodology of everyday human enquiry. The overarching practical need to address high-pressure Internet time goals has forced a return of systems development methodology to Dewey's original, basic formulation of human problem-solving behavior, one in which the exact stages, guidelines, and techniques were fluid. Only a logic of goals and constraints define the methodology, and beyond this the basic human problem solving philosophy of logic will take over. Such is the case with Internet time development methodology. The philosophy of logic is defined in the causal relationships between the central goal and constraint of the development setting: time is limited, and requirements are ambiguous. The rest is pragmatism:

> …the controlled or directed transformation of an indeterminate situation into one that is so determinate in its constituent distinctions and relations as to convert the elements of the original situation into a unified whole. (Dewey 1938, p. 104)

Internet time methodology does not attempt to define how this transformation is to be implemented for each cycle, stage, or phase. The human beings *in situ* define and redefine this transformation each day depending on the situation. On one day, they will need to prototype, on another day, they will just need to hack: code their way out of a problem. The procedures, stages, and tools cannot be predefined, only the logical relation between the properties of the setting. Beyond this, they need to apply the best human intellect toward pragmatically resolving each day's problems.

5. CONCLUSION

Internet time methodology makes no attempt to predefine certain methodical elements that are common in recent methodologies. However, it does focus on more fundamental and older methodological elements. Because the new methodology does not "look" like the usual methodology, it is necessary for methodologists to rethink what it means to be a methodology.

Internet time methodology is comprised of a central goal and constraint (time and ambiguity), and is defined by a logic of properties that proceed from the central goal and constraint. Internet time methodology assumes a philosophy of pragmatism that defers substantial decisions about process to the everyday logic of human inquiry. It is by all means a complete methodology within its assumption space. More importantly, it works for its purposes. Internet time methodology pushes an information product out the door in a satisfying way.

The human beings involved feel that they matter, and that their problems are getting resolved.

Further work is needed in this area. There are the usual limitations to grounded theory analysis. The results above were derived through the use of interpretive techniques using data from exploratory interviews. There was no attempt to consider statistical-style sampling. The interpretive techniques limit imposition of positivist criteria for evaluation of the findings, such as reliability or validity. However, we believe exploratory and theoretical research such as this is necessary in the development of knowledge, since a redefinition of the assumption space is indicated, and this redefinition must proceed before more objective approaches can continue on to confirm the findings (Galliers 1991). However, we agree that additional confirmatory research is called for.

6. ACKNOWLEDGMENTS

A warm thanks to our interviewees who dedicated their valuable time to help us advance our knowledge in this area. We are also grateful for the thoughtful and constructive comments of two anonymous reviewers.

7. REFERENCES

Atlanta Constitution. "Internet Growing by Leaps and Bytes: Study Says Americans' Trek to Cyberspace Is Now a Stampede," *The Atlanta Constitution*, February 19, 2001, pp. A1, A9.

Avison, D., and Wood-Harper, A. *Multiview: An Exploration in Information Systems Development*, London: McGraw Hill, 1990.

Checkland, P. *Systems Thinking, Systems Practice*, Chichester, England: J. Wiley, 1981.

Crosby, P. B. *Quality Is Free: The Art of Making Quality Certain*, New York: New American Library, 1980.

Cusumano, M., and Yoffie, D. *Competing on Internet Time: Lessons from Netscape and Its Battle with Microsoft* (1st Touchstone Edition), New York: Touchstone, 2000.

Cusumano, M. A., and Selby, R. W. *Microsoft Secrets: How the World's Most Powerful Company Creates Technology, Shapes Markets and Manages People,* New York: Free Press, 1995.

Dewey, J. *Logic: The Theory of Inquiry*, New York: Henry Holt and Co., 1938

Fitzgerald, B. "The Use of System Development Methodologies in Practice: A Field Study," *Information Systems Journal* (7:3), 1997, pp. 201-212.

Fitzgerald, B. "An Empirical Investigation Into the Adoption of Systems Development Methodologies," *Information and Management* (34), 1998, pp. 317-328.

Fitzgerald, B. "Systems Development Methodologies: The Problem of Tenses," *Information Technology and People* (13:2), 2000, pp. 13-22.

Flood, R. L., and Carson, E. R. *Dealing with Complexity: An Introduction to the Theory and Application of Systems Science,* New York: Plenum Press, 1988.

Galliers, R. "Choosing Appropriate Information Systems Research Approaches: A Revised Taxonomy," in *Information Systems Research: Contemporary Approaches and Emergent*

Traditions, H.-E. Nissen, H. K. Klein, and R. Hirschheim (eds.), Amsterdam: North-Holland, 1991, pp. 327-346 .

Iansiti, M., and McCormack, A. "Developing Products on Internet Time," *Harvard Business Review* (75:5), 1997, pp. 108-117.

Klir, G. *Facets of Systems Science*, New York: Plenum Press, 1991.

Lewin, K. "Frontiers in Group Dynamics," *Human Relations* (1:1), 1947a, pp. 5-41.

Lewin, K. "Frontiers in Group Dynamics II," *Human Relations* (1:2), 1947b, pp. 143-153.

Mathiassen, L., and Munk-Madsen, A. "Formalizations in Systems Development," *Behaviour and Information Technology* (5:2),1986, pp. 145-155.

Mumford, E., and Weir, M. *Computer Systems Work Design: The ETHICS Method*, London: Associated Business Press, 1979.

Smith, P. G., and Reinertsen, D. G. *Developing Products in Half the Time* (2nd Edition), New York: Van Nostrand Reinhold, 1995.

Stamper, R. "Analyzing the Cultural Impact of a System," *International Journal of Information Management* (8), 1988.

Strauss, A., and Corbin, J. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*, Newbury Park, CA: Sage Publications, 1990.

Susman, G., and Evered, R. "An Assessment of the Scientific Merits of Action Research," *Administrative Science Quarterly* (23:4), 1978, pp. 582-603.

Truex, D., Baskerville, R., and Travis, J. "Amethodical Systems Development: The Deferred Meaning of Systems Development Methods," *Accounting, Management and Information Technology* (10), 2000, pp. 53-79.

Wynekoop, J. L., and Russo, N. L. "Studying System Development Methodologies: An Examination of Research Methods," *Information Systems Journal* (7:1), 1997, pp. 47-65.

Wynekoop, J., and Russo, N. "System Development Methodologies: Unanswered Questions and the Research-Practice Gap," in *Proceedings of the Fourteenth International Conference Information Systems*, J. I. DeGross, R. P. Bostrom, and D. R. Robey (eds.), Orlando, FL, 1993, pp. 181-190.

About the Authors

Jan Pries-Heje holds M.Sc. and Ph.D. degrees from Copenhagen Business School, Denmark, and is currently a visiting professor in the Department of Computer Information Systems of Georgia State University. His research interests include information systems development, diffusion and adoption of IT, and software process improvement. He is a certified ISO 9000 auditor and BOOTSTRAP assessor and has been a project manager for a number of multimedia and organizational change projects. He is the Danish national representative to IFIP Technical Committee 8 on Information Systems. He was conference and organizing chair for the European Conference on Information Systems (ECIS) in Copenhagen, June 1999. Jan can be reached by e-mail at jpries@cis.gsu.edu.

Richard Baskerville holds M.Sc. and Ph.D. degrees from the London School of Economics and is department chair and professor of information systems in the Department of Computer Information Systems of Georgia State University. His research specializes in security of information systems, methods of information systems design and development, and the interaction of information systems and organizations. His interests in methods extends to qualitative research methods. He is an associate editor of The *Information Systems Journal* and *MIS Quarterly.* Richard's practical and consulting experience includes advanced information system designs for the U.S. Defense and Energy Departments. He is a Chartered Engineer under the British Engineering Council. Richard can be reached by e-mail at baskerville@acm.org.