Prototyping Praxis: Constructing Computer Systems and Building Belief

Douglas Tudhope, Paul Beynon-Davies
School of Computing
University of Glamorgan

Hugh Mackay
Faculty of Social Sciences
The Open University

ABSTRACT

This paper explores the consequences of the uncertainty introduced into the system development lifecycle by a prototyping approach and the practical strategies employed by developers in prototyping projects. Drawing on various strands of the sociology of technology, the paper discusses findings from a multi-disciplinary research project, which investigated the use of prototyping in commercial information systems development in the UK during the period 1995-1998. Qualitative semi-structured interviews with commercial practitioners were followed by a series of mini case studies. We draw on interview and participant observation material and the practitioner literature on Rapid Application Development (RAD). In the course of the project, we encountered a variety of practical strategies which attempted to extend the sphere of developers' influence beyond the technical realm to affect (but not determine) how the user/customer participates in the development process. Various techniques attempt to create a climate of joint ownership and shared approaches to change management. For example, the role of an ambassador user encompasses shaping the environment in which the system will operate via information, training and advocacy. Rather than a cause and effect model from user requirements to specification to implementation, developer strategies can usefully be considered in terms of sociological work on reflexive elaboration of networks. From this perspective, prototyping is more akin to trying to stabilise a network of evolving prototypes, user expectations, requirements and working practices than meeting a fixed specification.

Douglas Tudhope is a computer scientist with interests in hypermedia, information science, and the application of interactionist social science to Participatory Design; he is a Senior Lecturer in the School of Computing, University of Glamorgan. **Paul Beynon-Davies** is a computer scientist with an interest in information systems development and information systems management; he is Reader in Information Systems at the School of Computing, University of Glamorgan. **Hugh Mackay** is a sociologist with an interest in technology and culture and is researching computer system design and new media technologies; he is Staff Tutor and Senior Lecturer in Sociology at the Open University.

CONTENTS

1. INTRODUCTION

Prototyping is a rich area of study for HCI. It collapses the usual separation in space and time between developer and user. While prototyping and user involvement are now mainstays of user-centred development approaches (Gould & Lewis, 1985), the implications leave many questions unanswered. Managing the practical problems of user involvement and prototyping is a very topical concern for commercial developers.

1.1 Overview of Paper

This paper discusses findings from a multi-disciplinary research project, which investigated the use of prototyping in commercial information systems development in the UK during the period 1995-1998. Qualitative semi-structured interviews with commercial practitioners were followed by a series of mini case studies. Previous work on prototyping (Alavi, 1984; Boehm, 1986; Gordon & Bieman, 1995; Miller-Jacobs, 1991; Thomsen 1993) has highlighted project management and user involvement as problem areas for developers and this was confirmed by initial findings from our interviews (Beynon-Davies, Tudhope & Mackay, 1999). Notwithstanding potential benefits, iterative development poses greater uncertainty for the management of a project, compared with more prescriptive waterfall methodologies, for example in defining the final closure of the project and in the likelihood that user requirements will change during the development. Indeed, in attempting to set up interviews we came across developers who did not use prototyping for these reasons.

In this paper, we take these problems as our point of departure. Our aim is to provide some understanding of the strategies and practical methods evolved by developers to deal with prototyping and user involvement, drawing on various strands of the sociology of technology that emphasise the reflexive elaboration of social/technical networks. In the course of the project, we observed a set of practices which attempted to extend the sphere of developers' influence beyond the technical realm to affect (but not determine) how the user/customer participates in the development process. Section 2 illustrates some of the practical problems for developers as we encountered them in the interviews. Prototyping was frequently associated by our respondents with Rapid Application Development (RAD), sometimes taken loosely to denote some kind of rapid prototyping approach. However, there is a more precise definition and the practitioner literature on RAD is discussed in Section 3. Section 4 outlines key sociological perspectives informing the discussion of practical developer strategies in Sections 5 and 6. Section 7 draws conclusions and discusses future research.

1.2 Related Work

Previous work on prototyping supports the relevance of the issues discussed here. Evolution of requirements is cited as a potential advantage of prototyping as early as Carey and Mason's (1983) survey and has continued to be stressed by some studies (e.g., Harker, 1991), and by advocates of a process perspective on system design (Floyd, 1987; Grønbæk, Grudin, Bødker & Bannon, 1993). Alavi's (1984) comparison of prototyping with more traditional development found that prototyping improved user-developer communication, but that developers experienced more difficulty in managing and controlling the process, due to a lack of know-how in project management. Frequent changes in user requirements could frustrate designers unless they learned to view them positively. Hartson and Smith (1991), in making a case for rapid prototyping tools, list evolution of requirements as a potential benefit and differences from established management procedures in prototyping projects as a possible pitfall. Hardgrave's (1995) mail survey identified various factors employed by managers to decide whether a prototyping approach was appropriate for a given project. Requirements determination was a key factor – a major advantage of prototyping was in situations where requirements were ambiguous or likely to change. Gordon and Bieman's survey (1995) of published case studies found evolutionary prototyping strongly favoured over throwaway, although they noted that developers must be careful to address performance issues early on. Their findings on whether prototyping led to proliferation of requirements (five cases) were mixed; in three cases prototyping was observed to lead to a decrease of features, since critical features were emphasised at the expense of non-critical ones. In two cases the need for effective procedures for managing prototyping effort was raised. Axtell, Waterson & Clegg (1997) stressed the need for greater understanding of organizational issues and that more careful selection of users was an important issue for developers.

Current work in the sociology of technology attempts to show how technical systems do not evolve under their own imperative but have an inextricable social dimension. Recently various authors have sought to represent technical and social elements as part of a seamless web (for overviews: Bijker, Hughes & Pinch, 1987; Bijker & Law, 1992; for an application to design: Suchman & Trigg, 1993). This perspective informs our study and speaks to a growing concern in Participatory Design (PD) with the organizational context of prototyping activity, although there are important differences between the practical developer strategies in our study and PD's principled position on democratic user participation. While work in

PD has successfully demonstrated various techniques for facilitating user involvement in prototyping projects, several authors have stressed the need for further exploration of the pragmatic issues faced in extending PD to wider commercial practice (eg Bødker, 1996; Clement & Van den Besselaar, 1993; Grudin, 1993). Kensing and Blomberg (1998) note that PD researchers have begun to look for new ways of engaging with user organizations and suggest that a degree of co-operation with management may be necessary for the success of many PD projects. A recent collection of papers on PD (Trigg & Anderson, 1996) highlighted the importance of the social and organizational context of system design. Blomberg, Suchman and Trigg (1996) discuss the use of case-based prototypes to uncover how a new system might be used in rich detail and to explore a new technology's relationship with underlying organizational politics and work practice. In the same collection, Gartner and Wagner's (1996) case studies of the introduction of computer technologies in large corporations drew on actor-network theory (ANT) as a tool to analyse the political and organizational context of design work. They explored the competing interests of the various actors (shop stewards, management, etc) in the network with a view to understanding the broader political framework within which the design activity was embedded. We also apply an ANT perspective but focus at a more micro level on the specific practices employed by developers when engaged in prototyping activity with users. Another case study of the effect of organizational context focused on the working practices of in-house system developers, including their 'political' strategies (Symon, 1998). Symon's concluding suggestion is that user consultation might be seen as much a political process as an exchange of information – and that further research is needed on how user participation, beyond assisting with requirements, may serve the function of conferring credibility on the developed system. In this paper, we develop the theme that political and communicative functions are inextricably inter-related. We explore the constitutive nature of developers' practical strategies for creating stable (for a time) networks of technical and non-technical elements, within which a system can be seen as 'working'.

1.3 Approach

During the first half of 1995, we conducted 40 telephone and face-to-face interviews with commercial software developers who claimed to employ prototyping as part of their software development practice. A wide range of organizations and geographical areas were covered, including banks and financial institutions, the software industry, utilities, manufacturing, local and national government agencies, media, transport, and the computer departments of large commercial organizations. Half of the organizations were in the South Wales/ Bristol/ Swindon area while the other half were in London or other parts of England. The bulk of the sample was obtained via the School of Computing's student industrial placement database, augmented by contacts known to the researchers and by some snowball sampling (this use of interviewees to suggest other potential respondents may have resulted in some slight bias towards RAD practitioners). Although a wide range of organizations was covered, the intention was to conduct an exploratory study that allowed interviewees to discuss their opinions and prototyping practice. We were looking to generate themes for future study rather than confirm existing hypotheses (Golovchinsky, Chignell & Charoenkitkarn, 1997). A checklist reminded the interviewer of topics for the semi-structured interview, which covered the subjects' definition of prototyping, perceived advantages and disadvantages, how prototyping was employed in the organization, and the

nature of user involvement. Interviews were audiotaped[1]. Content summaries and field notes were written while reviewing the recording, with partial transcription. Following the interviews, observations were conducted of commercial prototyping practice, seminars and RAD training events over a two year period. Further interviews were conducted with RAD facilitators and consultants as part of these case studies. This paper draws on the interview material and practitioner literature, supported by field notes and in some cases audio recordings from observations of development activity and RAD training courses. Unless otherwise identified, quotes are from the interviews.

The majority of applications in our interview sample were forms of database or information systems (IS) developments. Beynon-Davies, Tudhope & Mackay (1999) gives a quantitative analysis of the results from the interviews and surveys prototyping in the IS literature. Prototyping occurred both as part of requirements capture and also development method. It was employed contingent on situation and a wide variety of tools were used. Following Grudin's (1991a) breakdown of types of development context, roughly three quarters of our sample were in-house developers (often in large distributed organizations), with one quarter contract development. In fact, the latter fell more under bespoke development than contract bidding (somewhat of a hybrid category). In part this distribution is due to the initial sampling criteria that respondents be engaged in prototyping. More than one third of the interviewees characterised their practice as employing some aspects of RAD, while more than two thirds had heard of RAD. Time emerged as an important concern. While some interviewees acknowledged the theoretical benefits of throw-away prototyping, the vast majority employed an evolutionary technique, due to the time pressures of commercial development contexts. There was general agreement on the potential benefits of prototyping: a better fit with user needs and relationship with users, greater communication, more user ownership of the final system, early identification of misunderstandings, a faster development on the whole if well managed. The ability to deal with changing requirements was seen as both an advantage and a disadvantage.

2. PROBLEMS OF PROTOTYPING

This section attempts to give a flavour of some of the practical problems posed by prototyping projects, as we encountered them in our interviews with developers and project managers. Within the broad areas of user involvement and project management, perceived problems included proliferation of user requirements, difficulties with managing and achieving closure of the project, unrealistic expectations by users and obtaining satisfactory user involvement.

Proliferation of requirements can arise from more concrete visualisation of the technology involved and what might be achievable. Users may be encouraged to ask for more features if initial suggestions are incorporated. While this may ultimately result in a better fit with requirements, it poses problems for project management. For example, a commercial services developer had a very practical concern: "If you don't control it properly you go into continual development" and thus fail to achieve satisfactory closure of the project. In another interview, a manager of the development group at a major London bank, where requirements capture was conducted with focus groups in diverse locations, noted that he employed strict *timeboxing* (see discussion of RAD in Section 3 and also Section 5.1) of these discussions to avoid the proliferation of demands for features as knowledge of possibilities became more

[1] Interviewees were guaranteed anonymity but in a few cases requested not to be taped.

detailed. For him, setting strict time limits was important to stop projects "taking on a life of their own" – the situation was compared to shutting the departure gates to allow a train or an aeroplane to depart on time: “you're locked out ... they know that." Another reason for proliferation is that new requirements arise in time as business needs change. For example, the manager of the IT bureau at a regional hospital faced challenges following reorganization of the National Health Service (NHS). In this situation, prototyping was useful when requirements changed frequently. Project management strategies for prototyping are discussed in Section 5.1 - these also include the managing of user expectations.

Unrealistic user expectations are often due to a mistaken view of the capabilities of the system being developed or the underlying technology. The manager of an in-house system development department in a regional house construction company was enthusiastic about prototyping's potential to meet user needs more closely (systems are not just "thrown over the wall"). However, he went on to elaborate that users may not realise they are only seeing “the outside of a car” without an engine. For him, “The largest single potential problem in prototyping is that it raises expectation levels early” since users have difficulty understanding why they *still* cannot have the system (“it’s there – I’ve seen it!”). Thus “Keeping those [expectation levels] up and keeping the enthusiasm for the system is a problem as the system goes further down the line [and does not seem to change] … getting the expectation right and keeping it right.” In fact, some developers go so far as to introduce code for the purpose of producing delays in the speed of operation of early prototypes to avoid unrealistic user judgements of final system response.

Problems achieving satisfactory user involvement was sometimes expressed as the need to involve the 'right kind' of user. We encountered a rich vein of trouble stories involving lack of promised users, the futility of having the wrong types of users or accurately meeting the wrong requirements. For example, although the users involved might know the business case for the changes, they might be unskilled at work tasks. Conversely, end-users may prove unable to imagine changes to existing work practices. The hospital IT manager above stressed that one of the main problems in prototyping is getting the correct users to specify the correct parts of the product - one would 'get the right user' by being sceptical regarding those who are put forward as users; end-users also needed to be brought into the development process. Even when suitable user representatives were present, they might ultimately be repudiated by their organization, or developers would end up with user representatives unable/unwilling to sign off. A frequent problem was that promised user participation simply failed to materialise. Section 5.2 explores methods evolved for attempting to facilitate user involvement. Developers would often make a link between user involvement during prototyping and user ownership of the resulting system. There are various aspects to this issue, which we explore in Sections 5.2 and 5.3.

3. RAPID APPLICATION DEVELOPMENT

Frequently (but not always) developers in our study associated prototyping with Rapid Application Development (RAD). The background of RAD is a perception that a competitive economic climate requires businesses to deliver systems within a faster timescale than traditional methodologies provide (Bates, 1995; DSDM, 1995; Martin, 1991). Reasons given include the growth of international markets, deregulation, and privatisation of public utilities. Traditional requirements gathering is seen to take too long in large, multi-site organizations. For developers to arrange separate meetings with spatially distributed stakeholders often takes significantly longer than expected, and, furthermore, possibly conflicting requirements

from different user groups are not easily reconciled by developers alone. The technical literature from one software tool (among many) associated with RAD makes a case against traditional approaches:

> Such traditional approaches to development are often referred to as waterfall methods … However, this approach often results in applications that take a long time to deliver and aren't fit for purpose when they do finally arrive. One cause of this is that users don't really know what they need until they have seen the system, or at least a prototype, in action. … Often a project takes so long to complete, that business conditions and therefore the application requirements have changed in the meantime. What is even more frustrating for the intended business user is that because the traditional approach follows such a strict sequence, this disparity remains hidden until the system is finally delivered ready for use. (Antares, 1995, p8)

RAD is essentially a project management philosophy, normally including Joint Requirements Planning (JRP) and Joint Application Design (JAD) workshops but with a broader focus. Like JAD (Carmel, Whitaker & George, 1993; Wood & Silver, 1989), RAD has evolved in industry with little influence from the academic world. RAD might be regarded as a compilation of lessons from commercial prototyping experience. A much-quoted axiom states that 80% of a system can be produced in 20% of the time required to build the complete system. This underlies the concept of 'timeboxing', a change in the relative priority of time and requirements compared to waterfall models. If a deadline is in danger of being missed, lower priority requirements are moved back to a later timebox, or the next phase of an incremental delivery. "Requirements can slip, timing never does" (DSDM, 1995, p70). Martin's (1991) book, *Rapid Application Development*, is credited as playing a key role in promoting RAD, along with other rapid prototyping influences including JAD and Boehm's (1986) spiral model of iterative development. Martin argues that business needs change substantially during long development times and that meeting current business needs when the system comes into operation is a better aim than meeting a long frozen specification. RAD appears to have grown out of the experience of large US organizations, employing code-generating tools in evolutionary prototyping. In particular, Martin highlights a 1989 shift at Dupont to a highly automated manufacturing environment, and a subsequent need for 90 day application software development times (later extended to 120 days to allow the re-organization of user activities)[2]. The term timeboxing appears to have been coined at Dupont at this time. Timeboxing is combined with an iterative approach to product development involving incremental releases.

Subsequently, the DSDM (Dynamic Systems Development Method) consortium of vendor and user organizations was formed in 1994 to produce a public domain, tool-independent RAD method, first in the UK and then internationally. Stapleton (1997) provides an overview of DSDM and case studies, while Beynon-Davies, Carne, Mackay & Tudhope (1999) survey documented RAD projects. The DSDM manual (DSDM, 1995) includes many of Martin's points and further elaborates various practical techniques. For example, the 'clean room' emphasises the importance of commitment to user participation: users are removed from other duties and their usual work environment to a dedicated area with developers. Projects should commence with a feasibility study, where a 'filter' as to the

[2] One of the UK RAD consultancy teams we interviewed had made an information gathering trip to Dupont.

suitability of RAD is applied – typically database-oriented applications not computationally complex, where the interface is important. Subsequent stages include a business study, a functional model iteration (with prototypes), a system build iteration, and 'implementation' (includes installation and training). Teams should be relatively small, with project length from 2-6 months. The importance of higher management involvement to champion the project is emphasised. In many ways the manual represents a collection of practical methods for project management and enlisting user commitment and participation, rather than any formal methodology. Key features are a prioritised set of requirements, timeboxing, evolutionary prototyping with active user involvement, change control procedures, rapid development tools and (possibly) incremental delivery. In our experience, the majority of developers who were aware of RAD tended to pragmatically select elements rather than follow it strictly, while others appeared to employ broadly similar techniques without identifying them as RAD.

4. MESSY NETWORKS[3]

Various strands within sociology and the human sciences locate objects of analysis within emergent social and technical networks. Much of this literature has been influenced to some degree by an ethnomethodological perspective, which is (in part) concerned with how everyday social reality can be seen as a situated accomplishment, achieved by participants' taken-for-granted practical activities (Garfinkel, 1967; for an overview: Sharrock & Anderson, 1986; with reference to system design: Dourish & Button, 1998; Suchman, 1987; Suchman & Trigg, 1993). We draw on key themes from this literature and apply them to the consideration of developers' practical methods for managing prototyping activity in the following sections.

4.1 Indexicality

In iterative design, users and developers communicate via various forms of prototype, which can include computer artefacts, sketches, mockups, etc. A prototype is a boundary object (Star & Griesemer, 1989), around which users and designers attempt to negotiate and share an understanding. As a glossing practice or 'mock-up' (Garfinkel & Sacks, 1970), it is also indexical in so far as it is intended to resemble the intended system in some important aspects, but inevitably is not completely specified. Users will be less familiar than developers with technical features being glossed (one of the reasons for the unrealistic expectations discussed earlier) and developers may be unaware of the significance of omissions from current work practice.

4.2 Reflexivity[4]

As we have seen, the notion that requirements, or 'needs', necessarily stand prior to and apart from experience with a prototype is challenged in iterative development. Requirements, prototypes and working practices are mutually constitutive -

---

[3] The term 'messy networks' is taken from Bijker & Law (1992, p12).

[4] The term 'reflexivity' tends to be over-loaded. Sometimes it is taken to denote a critical reflection by the author on the warrant for an argument or alternative points of view. The use here corresponds to Garfinkel and to Woolgar's (1988, p21) *constitutive reflexivity*.

> … a description [read prototype], for example, in the ways it may be a constituent part of the circumstances it describes, in endless ways and unavoidably, 'elaborates' these circumstances and is 'elaborated' by them. (Garfinkel & Sacks, 1970, p337, in relation to natural language).

In general, this reflexive elaboration of descriptor and the circumstances being described (of category and instance) underlies Garfinkel's (1967) discussion of the documentary method of interpretation. Through local sense making practices an underlying concept is elaborated within local contexts. That elaborated concept in turn provides the basis for interpreting future contexts, and so on. Thus in our case, paradoxically, the indexical and reflexive nature of the prototype is both a source of troubles and an advantage – it affords a mutual alignment of the elements of the network. A prototype is indexical of the system and of the user requirements and future work practices that are part of the system's context. The prototype's incompleteness and fluidity is a resource for the iterative and reflexive shaping of both prototype and the context of use. Prototypes are modified in light of evaluation but requirements (and user motivation) also evolve along with the system; inevitably, what users think is possible comes to influence their articulation of requirements. In part, the very process of interacting with prototypes shapes user requirements by giving a concrete focus to the imagination. This can simply lead to a request for changes or additional features in the prototype, but can have more complex outcomes, such as triggering a reflection by the user on (new) work practices hitherto invisible to the developers (Trigg, Bødker & Grønbæk, 1991). Indeed, the developer can act as provocateur, using the prototype to challenge the user to consider new possibilities for work practice (Mogensen, 1992). The involvement of user representatives in prototype development can also feed into design of training activities for new systems and affect the climate of reception.

4.3 Actor Network Theory

This ethnomethodological perspective is applied in Actor Network Theory (ANT). ANT and related work sees technological productions as messy networks of interacting elements, technical, social, and economic (Callon 1987; Law & Hassard, 1999). Our concern is with developers, various kinds of users, perceived business needs, prototypes and work patterns. Engineers are seen as lay sociologists or heterogeneous engineers (Law, 1992) in that they attempt to mould society as part of the work of designing objects[5]. However they are not privileged; all actors (human and non-human) are continually acted on by others as they attempt to shape a network, of which they themselves are part. Within the computing field, CSCW research into the organizational context of collaborative technologies is concerned with the 'co-evolution' of artefacts and work patterns over time (O'Day, Bobrow & Shirley, 1994). This reflexive or feedback network is also central to Winograd and Flores' (1986) critique of cause and effect rationalistic models for human action and to their alternative programme for design. Drawing on various hermeneutic traditions, including Maturana's notion of structural coupling in biological systems, they advocate designs which evolve "in alignment" (p53) with their domain of application. Our focus in this paper is on the development itself rather than adaptation of the delivered system but they can be seen as different stages of the same process.

---

[5] Hughes (1987) discusses the *social* engineering work of technological entrepreneurs such as Edison in attempting to influence the audiences and cultural contexts for new technologies.

A related concern focuses on the construction of identity and its role in attempts by actors to achieve their aims (Callon, 1991; Michael, 1996) and in stabilising a network. *Interressement* refers to the general process by which one actor attempts to define an identity for another actor, including a set of needs and interests, which will result in their enrolment in a supportive place in the network the actor is trying to stabilise. To take an example, Callon (1987) analyses a case study involving attempts by scientists to develop technologies to prevent the depletion of scallop stocks. Actors include scientists, fisherman, scallops etc. and identities and interests are imputed to fishermen (say) to encourage their playing a desired role. Such attempts need not be successful (as in Callon's example). Callon notes the difficulty of maintaining a stable network; actors may 'betray' their roles and refuse to play their assigned part, may be subject to other influences, and will in fact be part of a multitude of overlapping socio-technical networks. This problem is shared by developers attempting to enroll users, who are also part of many other networks. ANT's focus on building networks and its operationalization of the ethnomethodological concept of reflexivity can be seen in the practical strategies used by developers and project managers.

5. PROTOTYPING STRATEGIES

This section discusses a number of practical strategies employed by developers for dealing with the complexities of prototyping projects and the problems outlined in Section 2. It is important to emphasise that these strategies are contingent, adapted to the situation, influence and preferences of developers. In the RAD context, this was recognised in one training course, where the method was held out as a source of techniques to adapt according to context[6].

5.1 Project Management

Developers were concerned to facilitate the right relationship with users, where input was encouraged but not an endless sequence of requests for changes leading to the 'continual development' mentioned in Section 2. For that commercial services developer, prototyping was seen to require structure and clearly identified review points:

> And I know we struggled for quite a while to actually find a methodology that supported rapid development, because it typically flies in the face of all the formal methods that you've known in other formal development approaches. I think that what you have to do is get the balance right – you need to have the formal structure round it and you need to have formal review points in it and then you use the power of the tools to get you to those review points in as quick a way as possible.

This was reinforced in another interview. A backlog in customer application development had led the solutions/consultancy arm of an international computer corporation to create their own rapid development tool. Subsequently, however, underlying problems between developer and user communities were uncovered: "There was no relationship. They hated each others' guts. Didn't want to talk to each other. Both blamed each other for what

[6] On occasion the term 'RADish' was humorously employed by developers to denote projects incorporating just a few RAD elements.

was going wrong." A reaction against "very structured methodologies" (such as PRINCE, SSADM), unhelpful for faster development, initially led to a naïve ("hacking") approach:

> Essentially, we were saying that four years ago the way to do RAD is: You get a user in and sit them at your desk with a screen and then start building screens and the system with them. We started some projects that way and they were pretty well an unmitigated disaster. In the sense that there was no change control, there were no proper management, there was no sign off, there was no end point. You could just go on and on forever.

These experiences led to a research effort culminating in a more structured methodology with three major stages of prototype. "You actually need stronger project management techniques and standards in doing a RAD approach, a prototyping approach, than you would maybe even in a standard project". In fact, sometimes these prototype stages were sold separately to customers, as one way round the difficulties of developing cost estimating metrics for large projects. The methodology was now marketed as part of their development product:

> The thing that really seems to strike home and get us brownie points is that our running those Joint Requirements Planning sessions, because often you find that it's the first time that you've ever managed to get the deliverers from the company and the business users in the same room in an affable frame of mind. … A team transforms from there as opposed to a request there and those swine over there [developers] who never do anything we ask them to do, sort of thing. It's a bit of a trick I suppose, because in a sense you're making them part of a team and therefore responsible for the success or failure. But it's not. It's so true. It's kind of a trick in a political way I suppose, but effectively you are tying together. You're making them as responsible as the rest of the team for any changes they agree on. Whereas with the sort of vendor/supplier situation, they feel as though, I suppose, when they make a request it doesn't cost them anything. You have an IT department. There's a fixed cost. They should just go and do what I damned well ask them. … There's no real cost.

User involvement can act as a form of change management. We return to this issue in Section 6.

Given that requirements will evolve, a recurring theme was a concern with managing how users participated, and how changes were made to requirements, in order to avoid proliferation and 'unreasonable' change requests ("dropping out of the sky" as a developer put it in one seminar).

> Management of traditional projects is about control: trying to prevent drift from the signed off specification, controlling resources, etc. Managing a DSDM project is about enabling constant change while continually correcting the course of the project to in order to maintain its aim at the target - a fixed delivery date for a usable system. To be successful with DSDM, the organisation must change organisational, social and technical elements at the same time. These all have impact on project management. (DSDM, 1995, p62).

Various rules of thumb are suggested for user and developer project managers to decide whether change requests are inside or outside the original scope of the business requirements and it is suggested conventional contractual arrangements may need to be modified. The pragmatics of arranging contracts for prototyping projects is a very real concern for project managers. Thomsen (1993) describes Danish commercial experience with a spiral model

which lays out in advance an initial calculation and a template for negotiating changes to system price and delivery time based on estimates of complexity of system components. The Danish model allows the delivery schedule to be adjusted in the negotiations, unlike RAD which prioritises time.

In RAD, timeboxing acts to package development units into manageable chunks, with potential for flexibility if not all of the deliverables are "must haves"[7]. A DSDM consultant recounted persuading a company with a very detailed set of requirements that they didn't know what things would be like in five years time and instead to go for shorter timeboxed contracts where they could "envisage the horizon". The system is less likely to evolve out of step with its intended context. A key ingredient is the emphasis on prioritising requirements, a discipline which reminds the user of the impact of changes and militates against the attitude graphically depicted by the developer above. For example:

> We document the requirements from the first JAD workshop. Then we ask the owners to prioritise the requirements - this is very important - as "must have", "ought to have", or "nice to have". What we find is that half are "must have" and half are "nice to have". We never get any "ought to haves" on the first pass. Then we go through the whole process again, asking "which of these must haves are really ought to haves", so we can deliver some key business benefits early. We ask "why is it a must have? If we didn't deliver this, would the system not work?" and questions like that. We shift some of the "must haves" to "ought to have". (DSDM, 1995, p48)

Note the emphasis on prioritisation. In effect, if this can be achieved then, taken in conjunction with timeboxing and an agreed limit to the number of iterations, the prioritisation sets up a future trajectory for the development where the real time management of temporal constraints vis a vis implemented functionality is rendered more routine and less problematic. A decision in principle has been taken that there are priorities within the requirements and a framework set up for future discussions on particular lower priority requirements in particular timeboxes.

5.2 Getting the 'Right Users'

Various authors have highlighted the complexities underlying the term *user* (Agre, 1995; Bødker & Grønbæk, 1991b; Grudin, 1993; Low, Malcolm & Woolgar, 1993), an abstraction with many varying uses in practice. We encountered varied characterisations by developers of the type of users that should be involved. Usually the formulation was in terms of a mix of qualities, including higher management or budget holders, end-users for detailed specification and knowledge of work practices, skilled or competent users, knowledge of overall business needs and the rationale for the system, people with vision of different ways of doing things, possible champions of the system, ability to make decisions and sign off, and simply availability and commitment to the project. An in-house developer for a large commercial service organization elaborated:

> It needs to be someone who's involved directly in the resultant system or the operation of it. So it's got to be someone who will have the authority to

[7] In the intranet case study below, 'MoSCoW' rules were applied: 'Must have, Should have, Could have, Want to have.' Another project applied a second level of prioritisation within the main set of categories.

> implement it at the end if you like, someone who understands the environment in which it's going to be used. So there's no point in having, if you like, a manager in charge of a department being your user if he's not
> a) got the authority to implement it amongst them above him
> b) he's not going to be the sharp-end user who has to use it in the final resting place. I think you need to get a balance, and it's probably not just one person. It's probably a team. Again that depends on the size of the system.

There was widespread agreement that getting the right users was problematic. Higher management support was important but it was also recognised that involvement posed practical problems for user representatives. This was nicely illustrated in the intranet case study discussed below, where developers and users were isolated for three weeks in a setting geographically separate from users' normal workplaces. The user team members spent a significant amount of time fielding questions on the phone from their usual workplace. One user, having spent the day with the developers on the RAD project, complained about now having to work at the "day job" that evening in the hotel.

User Roles

The notion of roles for users came up in the interviews and is also treated at some length in the DSDM manual. There is a colourful typification of ideal-type users and their responsibilities. The Executive Sponsor (RAD champion) is a high level executive role, essential in securing commitment to user participation, who owns the system and can resolve organizational issues. The Visionary's role is to retain the initial vision for the project to guard against excessive requirement drift. Advisors are end-users with detailed working knowledge. The Ambassador User is a key role and would normally be part of the development team. This should be a user with some working knowledge, who represents different user groups, works closely with developers in the project team and, crucially, is empowered to make decisions. Note the prescriptive element in the definition of the roles. The manual suggests setting the Ambassador User involvement at the start, fixing regular times in staff weekly diaries when the project has first priority. Agreements with users and their managers on time commitment is also suggested, as well as training for user organizations on how to participate in RAD. While we did not come across any faithful duplication of these roles in our field work, we did encounter subsets of the roles, pragmatic adaptations or attempts by developers to suggest such roles[8].

The user taxonomy, in effect, enlists a user organization's involvement in a development project, by laying out a template of roles (with responsibilities) for the organization to take part in the project. User organizations are encouraged to take part in RAD awareness courses and RAD literature places some emphasis on user training. In general, such efforts, part of the promotion of RAD, can be seen as an interressement of customer organizations by identifying them as wanting quicker and better systems, and asserting the benefits to be gained from departing from the traditional hands-off role during development. Thus the concept, user, is translated to a number of well defined roles, involving an active and orderly participation in the development process from the developer's point of view.

---

[8] From our experience it does not seem easy for developers to gain access equivalent to the full set of DSDM user resources. However, case studies of RAD projects exist - see Stapleton (1997).

Users as Ambassadors

It is worth considering the notion of Ambassador User in more detail, who both represents user needs to the development team and represents the (evolving) system to the wider world.

> The holder of the Ambassador User role must have the desire, authority, responsibility and knowledge to be able to ensure that the right system is built for the business. This does not necessarily imply a senior position within the organisation, but a level of empowerment during the project to fulfil the role and an allocation of time to fully participate in the project as required. (DSDM, 1995, p82) … The Ambassador Users write their test scripts. They also build their training course. They know how to train people on their system. They'll also write the user manuals. Finally they're also capturing information for the workshop to kick off the next timebox. This is how the change in mindset takes place. It's not an IT system; it's *their* system (DSDM, 1995, p92).

More traditional systems development may also be concerned with building networks. Woolgar's (1991) ethnographic study of a printer manufacturer discusses how users are "configured" in a product development (mass-market) situation by a variety of means, including documentation and helplines for appropriate use of the product. It was important that the product be seen in the right context. Woolgar gives an example of a non-preferred reading, in which a key reviewer did not attend a publicity briefing where the desired context for the machine was presented and subsequently reviewed the machine using criteria not seen as appropriate by the developers. In our study, developers were not only concerned with configuring the participation of users during the development but ultimately in the context for receiving the final system. One mechanism by which this can occur is part of the role scripted for the Ambassador User, which involves taking the system back to the organization and organising training. We asked a DSDM consultant about user roles:

> Question: Do you think users have a role in the implementation process?
> Absolutely. DSDM defines the ambassador role very clearly and makes sure they don't drift off. They have to be present throughout the project. They have to be present throughout the project (sic). They certainly are there early on defining the high-level requirements, but then they should be present when you build prototypes. Then refining their requirements, detailing them. At that point they should be producing detailed business scenarios, they should be putting together help texts to be read by other users. Going back to my warehousemen, they were thrilled that they were going to be writing their own help texts because they would be able to understand it. Even when it gets technical they should be there, monitoring, testing, but at that point they should be thinking about what sort of training to do, how is change going to happen.

This was illustrated in a three day DSDM in-house training course run by an international corporation. In the introduction to RAD on the first day, the instructor motivated the course by citing the long time taken by traditional methods, competitive pressures from smaller startup companies, a failure to meet (internal) customer expectations and significant time taken up in maintenance activities. A RAD approach was contrasted with a personal bad experience as a user:

> Two days after [System X] had been mentioned to me somebody walked into the room with a disk in their hand and shoved it into my PC and twiddled away on the keys for 15 minutes and said: There you are – you've got X. I said: Great, now what can I do? They said: well there's a training session being held in three weeks time [which he couldn't attend] and here's the training book.

The system was not well liked. This was contrasted with a hypothetical development where:

> Maybe people from their own work group that are part of the team that's developing it there and they do take on ownership because these people come back and they say that you know well that the old system never used to allow us to do this. But I just insisted that they do and it does it now. And they go away and they're really good ambassadors. … And of course if you have users involved during the development, they will be trained. They will actually get to know what the different key strokes are, what the mouse does and things. And even if they don't know we'll have some users in there who'll be able to write proper training packages and train the end users. The people that actually know what they are doing will be training you and hopefully people that speak the same language that you do.

When new work practices are involved, appropriate training and user receptivity can be critical to a system's 'success'.

5.3 Building Belief

The importance of users' confidence or belief in a system was a concern that surfaced several times. The development manager of the housing construction company was asked what made a successful system:

> It's got to satisfy needs, satisfy a requirement … I think there's a more crucial thing than that. What makes a successful system? It's got to be used, it's got to be usable. People must want to use it. If they don't want to use it or they don't like using it … if it's not pleasant to use … then the system will fall into disrepute, disuse and all the failings under the sun, from any loss that the company makes to the coffee machine running out of sugar, will be blamed on that system. … There is a problem we've found with the systems. If you wind up in a situation like that where you get a system that is not in favour for whatever reason that may be, it's actually very very difficult to get of that loop because the next system is the same. Your iteration, your review of it, is the same, is tarred by the same brush – 'this is another one of those. This will be really hard to use' - Unless there is major break of the mould, a major change, look, feel, operation, it will just go the same way. Again it doesn't matter how good it is.

The identity of a system does not rest only in its technical components – motivational factors are key elements of the network. Evaluative judgements, as we see above, are often based on identities retrospectively constructed. It is also possible to set up prospective trajectories for future motivation and action. Latour (1987) has been concerned to follow the actors in the process of stabilising (black-boxing) technological achievements, in order to see apparently solid scientific or technical facts "in the making". A recent essay (Latour, 1997) deconstructs the dichotomy between facts and beliefs and critiques the tendency to pass over the role of

human agency in the fabrication of each. In place of subjective beliefs opposed to 'bald' facts, Latour holds out the (social) construction of 'dishevelled' facts, surrounded by the context that gives meaning. The reflexive network in which the evolving prototype is embedded includes user confidence in the system. In one interview, a developer reflected back on a career where prototyping had played an important part before the introduction of modern tools or RAD and emphasised the prospective nature of user ownership and belief:

> If you've been out there in the hard world, I'm sure you'll know there have been some *beautifully* written systems which have been a total failure, because essentially the customer didn't believe in them, or the users didn't believe in them. And there have been some *absolute* monstrosities, especially mickey mouse systems written on PCs by the users themselves, that fell down at every instance, that had no integrity and all of the dreadful sins, but they loved it because it was theirs. And a key difference in prototyping is that you sit down with the user with a PC, or a screen painter or whatever it is, but you sit there and you go through it with him. And by the time that you finished this exercise whether it's just taken a day, a week, a month. I mean usually it was, it was for 2 or 3 hours every day or two over a period of weeks, and together you built the system and by the end of it he felt it was so much his system or her system that they'd *make* the bugger work. It didn't matter how good, bad, or indifferent it was, they would make it work. In fact it was usually pretty good as well. But the winning of their hearts and minds was a major breakthrough in my opinion because they invested the time and they took ownership. (emphasis in original)

The developer emphasises that crucial factors in whether the system will eventually 'work' are the *future* activities and motivation of the user and that these are strongly influenced by previous prototyping experience, either direct involvement or, as we have seen, via 'ambassadors'.

A concern with reflexively building user belief in a system resonates with studies of innovation in other fields, both technical and cultural. Hoogma and Schot's (1996) case study of attempts to involve users in the development of electric vehicles critiques the notion that all that is required is for innovators to learn about users' needs and requirements. "Double-loop learning"[9] between users and producers is needed and intermediary structures that allow the "mutual adaptation of a technology and its context/selection environment". The importance of lead users, in developing new technologies is stressed. Lead users are characterised by competence in their understanding of (and ability to express) user experience and tasks, by resourcefulness in their access to resources and know-how, and by their incentive for innovation[10]. Creating a *market of expectations* is an important part of the development process - a speculative market of early promises, solutions to problems and possible scenarios. One factor encouraging user take up of stocks when floating a new option in this market is an early demonstration of success in an initial trial. This builds user confidence, encourages buy in and sets up a trajectory to the future, particularly if sympathetic hearings of early performances (prototypes) can be encouraged. This also holds

---

[9] The need for mutual learning has been well established in PD (Ehn 1988; Greenbaum & Kyng 1991).

[10] Possible disadvantages include a potential for conservatism due to lack of vision or experience with alternative ways of working, and a risk that lead users selected for their incentive to benefit may be unrepresentative of the larger user population and may come to exert undue influence.

for new computer systems. For example, an in-house developer found prototyping facilitated partnership with users: "they get so possessive … that I shudder to think of any manager who would criticise it".

A similar emphasis on building belief can be found in studies of aesthetic innovation. For example, in the cultural domain, DeNora's (1995) study of Beethoven's early career traces how he and his supporters attempted to build a context for hearing his music, including reform of the piano manufacturing technology of the day, musical-critical discourse and the criteria for reviewing new performances. A controversy of the time concerned user confidence in the new system - whether the new work could be clarified as a continuation of an established musical style ("Mozart's mantle"). Although aesthetic constructions have important differences from computer systems, some of the problems facing the innovator are common. Hennion's study (1989) of popular music producers also highlights the role of iterative prototyping and intermediaries in cultural production.

6. THE CONTINGENCY OF STRATEGIES

In this section, we draw on elements of case studies conducted as part of the research project together with interview material, in order to provide examples of network building strategies and also to illustrate the complexities of real situations. Our focus on methods employed by developers should not be read as a claim that they will be 'successful', meaning that networks can be established or will prove stable, or that there is some simple formula. As mentioned earlier, it was generally recognised that such techniques were contingent, to be called on as contexts and organizational constraints permitted. In many situations developers are not in a position to exert much influence. One site for observation was a large financial institution with a specialist 'new technology' development team. The team were concerned to present themselves as employing RAD, to strengthen possibilities for future RAD projects in the organization. The development was iterative and involved prototypes, but in practice developers were unable to achieve much direct user participation in this project (a key RAD requirement)[11]. Another (not explicitly RAD) case study concerned a prototyping development in a utility company moving from mainframe databases to PC-based applications. The project manager was a technically aware business manager, referred to as a 'SuperUser' by the developers, who played a key role in explaining and promoting the prototypes to the general user community and in fact had developed a first, less elaborate mainframe version of the proposed system. In demonstrations of the system to immediate end-users, the SuperUser translated between development features and user work practices and the lead developer frequently directed his gaze to the SuperUser when explaining the system. The interressement (to use Callon's term) was successfully extended in another meeting to a neighbouring section of the organization which also decided to adopt the proposed system for their purposes. However yet another meeting with a more distant group did not succeed in enlisting them into the project. The SuperUser was less familiar with the culture of the other group, differences in work practices and perceived status of the end-users emerged, and an alternative design was produced to counter the proposal.

6.1 Intensive RAD Case Study

[11] See also Button and Sharrock (1993) on accounting strategies in development activity.

Another inhouse case study[12] occurred in the same international corporation whose RAD training courses we discussed in Section 5.2. The project was seen by the organization as the purest form of RAD they had employed, due to its 'intensive' nature and the high degree of user involvement. Developers and user representatives were sequestered for three weeks (going home at weekends) in a setting remote from users' normal work places. The objective was to build an intranet website to provide resources for corporate PR activities and avoid duplication of effort by different centres throughout the country. An extensive set of web pages with associated Perl scripts was developed, building on and significantly extending a previous iteration along similar lines. Acceptance criteria for the project included approval of the 'look and feel', by user representatives on the RAD team, testing by user team members and by user representatives from outside the team – a widening circle of user exposure. The initial project plan included a formal review with three higher management representatives midway though the three weeks. In fact, this occurred towards the end of the second week and resulted in some changes to the combined project team's design to take account of the managers' input. Of course, this resulted in some extra work for the team and departs from standard RAD ideas on how development should proceed. (The notion of an 'empowered' joint design team is central to RAD guidelines but in practice this takes place within the constraints of organizational structure.) However, explicitly incorporating management approval into the prototyping process acts to raise their level of ownership and belief in the system. In the third week, a live demonstration of the current prototype at a national company meeting was marked as an important event by the team. The project manager only brought back minor changes and reported that the demonstration had gone well. This would have raised the value of the system's stock in the local 'market of expectations', discussed in Section 5.3. The project manager was from a user section of the organization (in fact the distinction between developer and user is not always clear) and was indeed able to exercise some control in the selection of team members (rejecting some candidates as unsuitable for the project). Not all users could be retained for the full three weeks of the project – a problem in intensive projects for continuity and dependence on any specialised knowledge. However, providing the user experience is positive, the more users with experience of the development, the more potential ambassadors to promote the system to other users[13], as in the training course discussion (Section 5.2).

A strong team spirit evolved – 'us' and 'them' referred to the boundary between the user-developer team and the rest of the organization. User representatives were actively involved in both design and development activity. They participated in 'brown papering' design sessions aided by the use of low technology mockups and also authored HTML documents while the developers worked on Perl scripting. The use of a 'clean room' dedicated project area allowed both design discussions and online sessions. Demonstrations of prototypes by the developers were informal and often involved live modifications (Bødker & Grønbæk, 1991a), a capability much appreciated by the users. The intensive development did put some strain on users since it did not prove possible for them to leave behind all other responsibilities. Typically, user representatives spent an hour on the phone each morning

[12] A detailed discussion is given elsewhere (Tudhope et al., 2001).

[13] Another potential facet to the role of Ambassador User, as a relayer of news and events from 'abroad', was suggested when an ex-user representative, replaced half-way through the project due to other duties and now at their usual workplace, faxed a warning of a last minute change in arrangements for the demonstration back to the project team. In fact, Stapleton (1997, p42) makes the comparison with diplomatic ambassadors.

dealing with issues from their usual workplace. Training activities were another responsibility of user representatives. In fact, a planning document for this development stressed that a lack of formal training with the previous iteration's product had led to ignorance of its full capabilities and some customer resistance. Accordingly training material was included as an output, to be used later to help develop regional 'supertrainers' who would in turn train local users on a one-to-one basis. Preparation of training material appeared on two occasions in the work plan constructed in the closing review session each day, with two user representatives involved in this activity on two days out of the fifteen allotted to the project. Stabilising a network involves a reflexive interaction with the context in which the system will operate, from input on current work practices to training and confidence building measures.

Requirements had been prioritised into 3 levels at a meeting before the development with management representatives present. A time-boxing exercise during the project descoped the lower priority requirements and the project met the level 1 priorities, except those identified at the requirements meeting as impractical in the time scale. The development we observed was the second in a series of three iterations. (RAD facilitators had created a video about the first iteration, in order to promote the use and acceptance of RAD within the organization.) The system was put into operational use and a year or so later a third iteration, run on similar lines, developed an extranet extension for communication with international PR agencies. Incremental delivery of system components makes it easier for work practice and system functionality to co-evolve and to keep in step with changing business conditions and requirements.

6.2 Discussion

Mediators, including surrogate users and third-party organizations, who act as indirect channels of communication between users and developers, have been identified by Grudin (1991) as an important channel of information for both camps. They have also been found useful in CSCW (e.g., Okamura & Fujimoto, 1994), installing or tailoring an implementation to a local context and shaping user interaction. Developers were identified in Symon's (1998) case study as potential organizational (business) change agents beyond the purely technical realm. As seen here, this potential exists with users also. It is clearly a key element in the identity of the DSDM Ambassador User[14] and some interviews identified users as an important mechanism for effecting change. One developer pointed out that a system had to be sold to its users. A "change agent" learned how the system worked and then returned to "sell" the system and also assisted with any further iterations. Another talked of increasing ownership of a system by inviting "power users" to play with a prototype and then report back on progress. The construction company development manager, quoted in Section 2, also emphasised ownership (belief):

> The prime value [of user involvement] is ownership, ownership of the system. From day one, it's their system … It's our [the users] system because we devised it. It also means that if there's six people around the table, you know there'll be six people at the end who'll want to have a go at it and will be keen to push it. Whereas any system that's owned almost purely by IT and IS, you then have to install it and you have to gain this acceptance of systems. All our

[14] In fact, the part played by the user representative, Andy, in Symon's study is a good example of the Ambassador User role.

past bad systems have been done really by ourselves with virtually no user involvement.

When talking with developers, unfavourable references would sometimes be made to non-prototyping approaches, where various means might be employed to manipulate users into 'signing off' on systems, with little understanding. Of course, it has to be understood that these utterances play a performative role in the contrast structure they set up. From one viewpoint, the strategies described here could be viewed simply as another manipulation of user expectations by developers. There is an agenda implicit in these methods of enlisting user participation in a process which is anticipated will result in a higher probability of acceptance than a 'lobbing it over the wall' approach. However to view this as simply manipulation by developers would be to miss the point. The developer (in Section 5.1), who reflected whether joint Requirements Planning sessions might be a kind of 'trick' in that users are made to feel part of a team, went on to suggest a broader perspective for viewing this activity ("It's not. It's so true."). Users make input and have agency. They take on some responsibility for the future system. For their part, developers do not stand outside the network and in fact surrender some control of project management for a sharing of constraints and more active participation by users. This requires a change of attitude on the part of both developers and users. For example, one of our interviewees referred to a need for customers to throw away some of their traditional "comfort blankets" (fixed specifications) in prototyping projects. This is not to say that identity of interests can always be achieved. Relative power and status between developers and users and between user groups is distributed differentially according to the particular situation. Although developers in our study generally took pride in giving examples where convergence had been achieved with users (and between different sections of customer organizations), there was awareness that conflicting interests could arise in connection with changes to work patterns (see Blomberg, Suchman & Trigg, 1996; Gartner & Wagner, 1996; Symon, 1998).

7. CONCLUSIONS

This study supports previous work on potential advantages and problems of prototyping with user involvement and sheds new light on the practical methods employed by developers as a response. The practices explored here attempt to influence both the choice of which users are involved and the manner in which they are involved via user roles. This template of roles not only encourages participation but also attempts to configure the manner of participation. User requirements and expectations change as business constraints change and experience with prototypes clarifies understanding of the evolving system and future working practices. Managing user expectations and requirements is in part an attempt to share with user representatives an appreciation of the constraints under which the development takes place. Various techniques, including joint design teams and an early emphasis on prioritisation, attempt to create a climate of joint ownership and shared approaches to project management. RAD's time-constrained development techniques, based around timeboxing and phased delivery, may have wider interest.

Rather than a cause and effect model from user requirements to specification to implementation, the developer strategies we have discussed can usefully be considered in terms of sociological work on reflexive elaboration of networks. From this perspective, prototyping is more akin to trying to stabilise a network of evolving prototypes, user expectations, requirements and working practices than meeting a fixed specification. The

emphasis on time constraints and incremental delivery recognises that the stability of any configuration is subject to external influences. The developer cannot hope to command the context surrounding the system but may attempt to influence it and build confidence. Conversely, the important role that lead users have to play, beyond requirements specification and evaluation of interfaces, should be valued and realistically resourced. Although our study focused on developer strategies, the sometimes offstage activities of users came through as key elements of stable networks. There are different attributes of 'user ownership'. Production of training material can be performed by user representatives. The role of a lead or ambassador user can extend to shaping the environment in which the system will operate by providing information, training, tailoring and advocacy or motivation. There are implications for organizational culture and professional and computer science education. While the reflexive nature of the prototyping process can lead to a virtuous circle between artefacts and work practice, with positive outcomes in terms of perceived fit to user needs, it should also be noted that this channelling of expectations can sometimes encourage a blindness to alternative solutions (Bødker & Grønbæk, 1991b).

Looking to future research, longitudinal, ethnographic studies incorporating viewpoints of different actors, from initial requirements gathering through development to observations of a system in operational use, are difficult to set up but would yield valuable insights. On the practical side, it might be possible to apply existing PD techniques (e.g., Greenbaum & Kyng, 1991) to the commercial practices of our study - the developers we talked to had no awareness of work in PD. For example, although end-user involvement was seen as a key source of information, there seemed limited awareness of the potential for encouraging users to transcend current practice (Mogensen, 1992). Furthermore, while we observed informal sessions involving future work practice scenarios, a more systematic approach to the iteration of scenarios along with prototypes (Kyng, 1995) could be incorporated.

As noted in Section 1.3, our experience has been with in-house and contract developers of bespoke systems. It is probably easier to connect with users in these situations than in mass market product development organizations (Grudin, 1991b). The fact that our study was centred on developers lends itself to an account where they figure as the technological entrepreneurs, both working for customers and translating their needs. However, in our daily activities we all inevitably belong to multiple, overlapping networks, assuming varying identities and social roles and responsibilities. Distinguishing a particular network and the identity of particular actors is, in part, a necessary product of the analyst's need to produce an account. In this paper, we have concentrated on networks forged during system development. Studies which focus on user communities, their appropriation of systems and their network building activities would be a fruitful area for research.

NOTES

Background. Early ideas leading to this paper were presented in 1996 at a Participatory Design session at the Joint Conference of the Society for Social Studies of Science (4S) and European Association for the study of Science and Technology (EASST), Bielefeld.

Acknowledgements. We would like to thank all those who talked to us about their work, and all participants in the case studies. We would like to acknowledge the work of the project's two research fellows, Chris Carne and Roger Slack. Geoff Cooper, Daniel Cunliffe, Tia DeNora, Lucy Suchman and the three anonymous reviewers made helpful comments on previous drafts.

Support. The project was funded by the UK Economic and Social Research Council (Grant No. R000 23 5505).
Authors' Addresses. Douglas Tudhope, Paul Beynon-Davies, School of Computing, University of Glamorgan, Pontypridd, CF37 1DL, Wales, UK. E-mail: dstudhope@glamorgan.ac.uk, pbeynon@glamorgan.ac.uk. Hugh Mackay, Faculty of Social Sciences, The Open University in Wales, 24 Cathedral Road, Cardiff, Wales, CF1 9SA, UK. Email: A.H.Mackay@open.ac.uk

REFERENCES

Agre P. (1995). Conceptions of the user in computer system design. In P. Thomas (Ed.), *Social and interactional dimensions of human-computer interfaces* (pp. 67-106). Cambridge: Cambridge University Press.

Alavi M. (1984). An assessment of the prototyping approach to information systems development. *Communications of the ACM,* 27(6), 556-563.

Antares. (1995). *Huron ObjectStar Release 3.0 Technical Overview: a developer's perspective.* Antares Alliance Group, Dallas, Texas. Document No. OSTECD-1095.3.

Axtell C., Waterson P. & Clegg C. (1997). Problems integrating user participation into software development. *International Journal of Human-Computer Studies,* 47, 323-345.

Bates P. (1995). 'Rapid Application Development - Concept, Methodology, or What?'. In M. Stephens (Ed.), *Proceedings of the Seminar Series on New Directions in Software Development: RAD - How Rapid is Rapid?* (pp. 1-21). University of Wolverhampton.

Beynon-Davies P., Carne C., Mackay H. & Tudhope D. (1999). Rapid Application Development (RAD): an empirical review. *European Journal of Information Systems,* 8, 211-223.

Beynon-Davies P., Tudhope D. & Mackay H. (1999). Information Systems Prototyping in practice. *Journal of Information Technology,* 14, 107-120.

Bijker W., Hughes T. & Pinch T. (Eds.). (1987). *The social construction of technological systems.* Cambridge, Mass: MIT Press.

Bijker W. & Law J. (Eds.). (1992). *Shaping technology/ building society.* Cambridge, Mass: MIT Press.

Blomberg J. & Kensing F. (1998). Participatory design: issues and concerns. *Computer Supported Cooperative Work,* 7(3-4), 167-185.

Blomberg J., Suchman L. & Trigg R. (1996). Reflections on a Work-Oriented Design Project. *Human-Computer Interaction,* 11, 237-265.

Bødker S. (1996). Creating conditions for participation: conflicts and resources in systems development. *Human-Computer Interaction,* 11, 215-236.

Bødker S. & Grønbæk K. (1991a). Cooperative prototyping: users and developers in mutual activity. *International Journal of Man-Machine Studies,* 34, 453-478.

Bødker S. & Grønbæk K. (1991b). Design in action: From prototyping by demonstration to cooperative prototyping. In J. Greenbaum & M. Kyng (Eds), *Design at work: cooperative design of computer systems* (pp. 197-218). Hillsdale, NJ: Lawrence Erlbaum Associates.

Boehm B. (1986). A spiral model for software development and enhancement. *ACM SIGSOFT Software Engineering Notes,* 11(4), 14-24.

Button G. & Sharrock W. (1993). Practices in the work of ordering software development. In A. Firth (Ed.), *The discourse of negotiation* (pp. 159-180). Oxford: Pergammon.

Callon M. (1987). Some elements of a sociology of translation: domestication of the scallops and the fishermen of St. Brieuc Bay. In J. Law (Ed.), *Power, action and belief* (pp. 196-233). London: Routledge and Kegan Paul.

Callon M. (1991). Techno-economic networks and irreversibility. In J. Law (Ed.), *A Sociology of Monsters* (pp. 133-161). London: Routledge.

Carey T. & Mason R. (1983). Information systems prototyping: techniques, tools, and methodologies. *INFOR - Canadian Journal of Operational Research and Information Processing*, 21(3), 177-191.

Carmel E., Whitaker R. & George J. (1993). PD and Joint Application Design: a transatlantic comparison. *Communications of the ACM*, 36(4), 40-48.

Clement A. & Van den Besselaar P. (1993). A Retrospective look at PD Projects. *Communications of the ACM*, 36(4), 29-37.

DeNora T. (1995). *Beethoven and the construction of genius: musical politics in Vienna, 1792-1803.* Berkeley: University of California Press.

Dourish P. & Button G. (1998). On "Technomethodology": foundational relationships between ethnomethodology and system design. *Human-Computer Interaction*, 13, 395-432.

DSDM Consortium. (1995). *Dynamic Systems Development Method* (v2). Farnham, UK: Tesseract Publishing.

Ehn P. (1988). *Work-oriented design of computer artifacts.* Arbetslivscentrum: Stockholm.

Floyd C. (1987). Outline of a paradigm shift in software engineering. In G. Bjerknes, P. Ehn & M. Kyng (Eds.), *Computers and democracy – a Scandinavian challenge* (pp. 191-212). Aldershot, UK: Avebury.

Garfinkel H. (1967). *Studies in ethnomethodology*. Cambridge: Polity.

Garfinkel H. & Sacks H. (1970). Formal structures of practical actions. In J. McKinney & E. Tiryakian (Eds.), *Theoretical Sociology: perspectives and developments* (pp. 337-366). New York: Appleton Century Crofts.

Gartner J. & Wagner I. (1996). Mapping actors and agendas: political frameworks of systems of design and participation. *Human-Computer Interaction*, 11, 187-214.

Golovchinsky G., Chignell M. & Charoenkitkarn N. (1997). Formal experiments in casual attire: case studies in information exploration. *New Review of Hypermedia and Multimedia*, 3, 123-157.

Gordon V. & Bieman J. (1995). Rapid prototyping: lessons learned. *IEEE Software*, 30(1), 85-95.

Gould J. & Lewis C. (1985). Designing for usability: key principles and what designers think. *Communications of the ACM*, 28, 300-311.

Greenbaum J. & Kyng M. (Eds.). (1991). *Design at work: cooperative design of computer systems.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Grønbæk K., Grudin J., Bødker S. & Bannon L. (1993). Achieving cooperative system design: shifting from a product to a process focus. In D. Schuller & A. Namioka (Eds.), *Participatory Design: Principles and Practices* (pp. 79-97). Hillsdale, NJ: Lawrence Erlbaum Associates.

Grudin J. (1991a). Interactive systems: bridging the gaps between developers and users. *IEEE Computer*, 24(4), 59-69.

Grudin J. (1991b). Obstacles to user involvement in software product development, with implications for CSCW. *International Journal of Man Machine Studies*, 34, 435-452.

Grudin J. (1993). Obstacles to participatory design in large product development organizations. In D. Schuller & A. Namioka (Eds.), *Participatory Design: Principles and Practices* (pp. 99-122). Hillsdale, NJ: Lawrence Erlbaum Associates.

Hardgrave B. (1995). When to prototype: decision variables used in industry. *Information and Software technology*, 37(2), 113-118.
Harker S. (1991). Requirements specification and the role of prototyping in current practice. In J. Karat (Ed.), *Taking software design seriously: practical techniques for human-computer interface design* (pp. 339-354). San Diego: Academic Press.
Hartson H. & Smith E. (1991). Rapid prototyping in human-computer interface development. *Interacting with Computers*, 3(1), 51-91.
Hennion A. (1989). (Transl. G. Bowker). An intermediary between production and consumption: the producer of popular music. *Science, Technology and Human Values*, 14(4), 400-424.
Hoogma R. & Schot J. (1996). How innovative are users: A critique of learning-by-doing and -using. Paper presented at 4S/EASST Conference, Bielefeld.
Hughes T. (1987). The evolution of large technical systems. In W. Bijker, T. Hughes & T. Pinch (Eds.), *The social construction of technological systems* (pp. 51-82). Cambridge, Mass: MIT Press.
Kyng M. (1995). Creating Contexts for Design, In: J. Carroll (Ed.), *Scenario-based Design* (pp 85-107). New York: Wiley.
Latour B. (1987). *Science in Action.* Cambridge, Mass: Harvard University Press.
Latour B. (1997). A few steps towards an anthropology of the iconoclastic gesture. *Science in Context*, 10 (1), 63-83.
Law J. (1987). Technology and heterogeneous engineering: the case of Portuguese expansion. In W. Bijker, T. Hughes & T. Pinch (Eds.), *The social construction of technological systems* (pp. 111-134). Cambridge, Mass: MIT Press.
Law J. & Hassard J. (Eds.). (1999). *Actor network theory and after*. Oxford: Blackwell/ Sociological Review.
Low J., Malcolm B. & Woolgar S. (1993). 'Do users get what they want?' - Introduction and workshop report. *SIGOIS Bulletin*, 14(2), 3-7. New York: ACM.
Martin J. (1991). *Rapid Application Development.* New York: Macmillan.
Michael M. (1996). *Constructing identities: the social, the nonhuman and change.* London: Sage.
Miller-Jacobs H. 1991. Rapid prototyping: an effective technique for system development. In J. Karat (Ed.), *Taking software design seriously: practical techniques for human-computer interface design* (pp. 273-286). San Diego: Academic Press.
Mogensen P. (1992). Towards a provotyping approach in systems development. *Scandinavian Journal of Information Systems*, 4, 31-53.
O'Day V., Bobrow D. & Shirley M. (1994). The social-technical design circle. *Proceedings of the CSCW'94 Conference on Computer Supported Cooperative Work*, 160-169. New York: ACM.
Okamura K. & Fujimoto M. (1994). Helping CSCW applications succeed: the role of mediators in the context of use. *Proceedings of the CSCW'94 Conference on Computer Supported Cooperative Work*, 55-65. New York: ACM.
Sharrock. W. & Anderson B. (1986). *The Ethnomethodologists.* Chichester, UK: Ellis Horwood.
Stapleton J. (1997). *Dynamic Systems Development Method: the method in practice.* Harlow, UK: Addison-Wesley.
Star S. & Griesemer J. (1989). Institutional ecology, 'translations' and boundary objects: amateurs and professionals in Berkeley's museum of vertebrate zoology, *Social Studies of Science*, 1907-1939, 19, 387-430.

Suchman L. (1987). *Plans and situated actions: the problem of human machine communication.* Cambridge: Cambridge University Press.

Suchman L. & Trigg R. (1993). Artificial Intelligence as Craftwork. In: J. Lave & S. Chaiklen (Eds.), *Understanding practice* (pp. 144-178). Cambridge: Cambridge University Press.

Symon G. (1998). The work of IT System developers in context: An organizational case study. *Human-Computer Interaction*, 13, 37-71.

Thomsen K. (1993). The Mentor Project Model: A model for experimental development of contract software. *Scandinavian Journal of Information Systems*, 5, 113-131.

Trigg R. & Anderson S. (Eds.). (1996). Special Issue on current perspectives on participatory design. *Human-Computer Interaction*, 11.

Trigg R., Bødker S. & Grønbæk K. (1991). Open-ended interaction in cooperative prototyping: a video-based analysis. *Scandinavian Journal of Information Systems*, 3, 63-86.

Tudhope D., Beynon-Davies P., Mackay H., Slack R. (Forthcoming 2001). Time and representational devices in Rapid Application Development. *Interacting with Computers*, 13(3).

Winograd T. & Flores F. (1986). *Understanding computers and cognition: a new foundation for design.* Reading, MA: Addison-Wesley.

Wood J. & Silver D. (1989). *Joint Application Design: how to design quality systems in 40% less time.* New York: John Wiley.

Woolgar S. (1988). *Knowledge and reflexivity: new frontiers in the sociology of knowledge.* London: Sage.

Woolgar S. (1991). Configuring the User: the case of usability trials. In J. Law (Ed.), *A Sociology of Monsters* (pp. 58-100). London: Routledge. An updated version appears in: Grint K. & Woolgar S. (1997). *The machine at work: technology, work and organization* (pp. 65-94). Cambridge: Polity.