2000

Design Breakdowns, Scenarios and Rapid Application Development

Paul Beynon-Davies
University of Galmorgan

Design Breakdowns, Scenarios and Rapid Application Development

Paul Beynon-Davies
University of Glamorgan, UK

Abstract - **In this paper we consider the way in which two representational forms, scenarios and design breakdowns, which have emerged in the tradition of participatory design are relevant within the recent commercial emphasis on rapid application development (RAD). RAD is a contingent approach to interactive software development that is characterised by large amounts of user involvement, incremental prototyping and product-based project management. Scenarios have become popular as an intermediate representation within the human-computer interaction and computer supported co-operative work communities. Design breakdowns have been suggested as a useful organising device and design technique within the co-operative prototyping literature. Both these representational forms are not currently utilised within the RAD tradition. In order to detail the relevance of these concepts to commercial development, we describe the 'natural history' of one particular RAD project and show how scenarios, breakdowns and the resolution of such breakdowns contributed to the successful implementation of an information system within a small commercial organisation**

I INTRODUCTION

For the last three years a research team, of which one of the authors is a part has observed on a number of rapid application development (RAD) projects in industry. RAD is an iterative and contingent approach to interactive software development that is characterised by large amounts of user involvement, the use of incremental prototyping and product-based project management. To help us understand and interpret the material we have been collecting we began to compare and contrast aspects of RAD practice with an academic area which bears a family resemblance to the RAD approach: the area of Participatory Design (PD) of software systems. A second strand of our research, described in this paper, has been the testing of a number of techniques from the PD area within the context of our own development work utilising the RAD approach.

In this paper we consider two representational forms with different ancestry, which have informed our own work in commercial information systems development: scenarios and design breakdowns. We particularly focus on the latter concept, but also illustrate how scenarios and design breakdowns have a complementary status in commercial design work. We wish to emphasise the use of scenarios and breakdowns as representations for communication and joint understanding between developers and users. In this sense, our interest is clearly towards what might be called user-centric rather than developer-centric notions of representation. We believe that whereas the area of developer-centric representations is well served at least in terms of the number of representational formalisms currently available, user-centric representations are comparatively scarce within commercial systems development.

The structure of the paper is as follows. First, we briefly examine the philosophical and methodological background of the concept of a design breakdown and that of a scenario. Second, we describe some of the core components of RAD. Third, we describe the background to our development project. Fourth, we illustrate the importance of scenarios and design breakdowns to RAD work with examples taken from a series of design meetings between developers and users throughout this project. We conclude with some of the lessons for both PD and RAD and some ideas as to our future work in this area.

II THE CONCEPT OF BREAKDOWNS

One of the earliest references to the concept of breakdown in relation to computer systems design appears to be due to Winograd and Flores [1]. They adapt this concept from Heidegger's insistence that objects and properties are not inherent in the world but arise only in an event of breaking down, a process in which human actors undergo an experiential shift in which objects change from being *ready-at-hand* to being *present-at-hand*. They use the classic example of the hammer and the nail to explain this experiential shift. To a person hammering in a nail the hammer as such ceases to be foregrounded in perceptual terms. It is *seen-but-unnoticed*; part of the background readiness-to-hand that is taken for granted. The hammer presents itself as a hammer only when there is some kind of breaking down, such as when it breaks, slips from the hammerer's grasp or bends the nail. In a similar manner a computer system, and more particularly its properties or design, are normally ready-to-hand. Only when there is some breakdown, such as when a software bug causes the machine to crash or when the system behaves in an unexpected way, do we experience it as being present-at-hand.

This leads Winograd and Flores [1] to propose that breakdowns are an essential characteristic of design and also that as a consequence design is an inherently cyclical process. They cogently sum up their position as one in which: '...the development of any computer-based system will have to proceed in a cycle from design to experience and back again. It is impossible to anticipate all the relevant breakdowns and their domains. They emerge gradually in practice.'

The concept of breakdowns in design have been particularly taken up and used by members of the Aarhuus school of PD [3]. Bødker and Grønbœk [4] use the concept of breakdown within a co-operative prototyping approach in which users and designers participate actively, creatively and mutually in the design process. In this approach, prototyping is used either for idea generation and exploration or for work-like evaluation of prototypes. Particularly in relation to the use of prototypes for work-like evaluation, breakdowns in design are seen as leading to further modifications to a prototype.

In their work Bødker and Grønbœk distinguish four types of situations where breakdowns are relevant. Two feature dimensions seem to be evident in this categorisation: the area under consideration; the time-related nature of the focus. In terms of area we can distinguish between a focus on the computer system, on work activity, or on the design process. In terms of time, we can distinguish between a focus on the current situation as compared to a focus on future situations.

This leads us to extend Bødker and Grønbœk's categorisation to include six possible types of breakdown situation, as illustrated in figure 1. Types 1 through 4 on figure 1 correspond to those originally proposed by Bødker and Grønbœk. Type 5 is useful in the context of situations where an existing computer system has to be re-designed or extended. Type 6 is particularly useful for developers in that breakdowns may occur in suggestions as to changes in an envisioned design process.

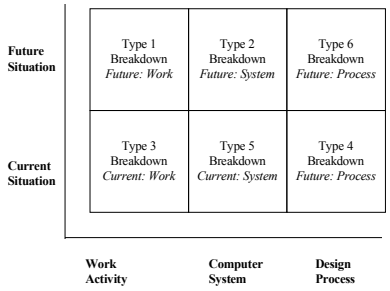

Fig. 1 Types of Design Breakdown

In situations of types 1 (Future: Work) and 3 (Current: Work) potentialities exist for making the work become present-at-hand. In situations of types 2 (Future: System) and 5 (Current: System) the computer system/prototype becomes present-at-hand and may be reflected upon. In situations of types 4 (Current: Process) and 6 (Future: Process), the design process itself may become foregrounded.

III SCENARIOS

Scenarios are described by Carroll [4] as, '... a narrative description of what people do and experience as they try to make use of computer systems and applications'. Although discussed in a number of different ways in the literature, the concept of a scenario seems to have a number of characteristics in common:

1. Key Episodes. Scenarios normally constitute key situations or episodes in the work activity of people working with computers.
2. Concreteness. The emphasis in a scenario is on concrete representation of use rather than abstraction of use. The focus is on specific instances of use located within a work context.
3. Groundedness. Scenarios should be grounded in the existing or potential work activities of users of computer systems.
4. Informality. Scenarios tend to be open-ended, fragmentary, informal, rough. They are particularly directed at enhancing communication and envisionment rather than displaying the representational characteristics of consistency, rigour and completeness.
5. Middle-level abstractions. Carroll [4] sees scenarios as a much-needed middle-level abstraction between the formality of a systems specification and the informality of everyday discussions between developers and users. Consequently a scenario tends to be more of a user-centric rather than a developer-centric design representation.
6. Multiple media. Although usually expressed in narrative form, scenarios may also comprise storyboards of annotated cartoon panels, video mock-ups, scripted prototypes, or physical situations contrived to support certain activities [5].
7. Applicability. Carroll [4] describes a number of distinct uses of scenarios in terms of the life cycle of software development. He distinguishes between scenarios used for: requirements analysis; user-designer communication; design rationale; envisionment; software design; implementation; documentation and training; evaluation; abstraction; team building.

Scenarios as a representational formalism have a considerable history within the domain of participatory design. However, it is interesting that scenarios do not appear to be discussed in relation to RAD currently. Because of the family resemblance between RAD and PD, our interest has therefore been in the applicability of scenarios within the rapid applications development process.

IV SCENARIO-BASED DESIGN AND BREAKDOWNS

We maintain that there are clear links between scenarios and design breakdowns [6]. Here we formulate this linkage as two propositions:

1. Proposition 1: In their use either as a means of representing current or envisioned work activity scenarios may be used as a vehicle for stimulating breakdowns of types 1 (Future: Work) and 3 (Current: Work).
2. Proposition 2: Since scenarios have proven useful as intermediate representations of user interactions with computer systems they may have potential in stimulating design breakdowns of types 2 (Future: System) and 5 (Current: System).

Breakdowns of types 4 (Current: Process) and 6 (Future: Process) are types not covered by the practical experience described in this paper. However, they form elements of the study of information systems failure and remain types worthy of further investigation.

We find it useful to conceive of the linkage between scenarios and design breakdowns within the context of a reflective, cyclical model of design activity. In this mould, Schön [7] discusses design as a reflective conversation with the situation. The designer 'shapes the situation, in accordance with his initial appreciation of it, the situation 'talks back' and he responds to the situations back-talk. In a good process of design, this conversation with the situation is reflective. In answer to the situation's back-talk, the designer reflects-in-action on the construction of the problem, the strategies of action, or the model of the phenomena, which have been implicit in his moves.'

This metaphor of design as a reflective conversation with the situation is certainly useful in understanding the way in which a project team within RAD work continually debates the utilisation of technology. Design in a RAD project is

however different from that portrayed in Schön's work because of its inherent collaborative nature. In collaborative design the 'situation' is not an individual cognitive construct but the result of group appreciation and negotiation. Clearly to achieve such a collective appreciation of the situation certain representations are needed to communicate effectively amongst the group the current appreciation of the design situation. It is here that we feel that scenarios have a part to play as organising devices for user reviews of prototype systems, which in turn stimulate breakdowns, which then feedback into another cycle of design work. This reflective cycle is illustrated in figure 2.

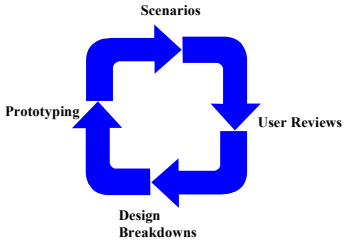

Fig. 2 The Reflective Cycle of RAD Design

V RAPID APPLICATIONS DEVELOPMENT

RAD first appears to have become topical with the publication of a text by James Martin [8] with the same title. Martin defines the key objectives of RAD as: high quality systems, fast development and delivery, and low costs. The following appear to be the common components of RAD approaches discussed in the literature:

1. RAD Project. RAD projects seem to be typically of relatively small-scale and of short duration. Two to six months is a normal project length. Project work may be placed away from the business and developer environments in 'clean' rooms - that is, places free from everyday work interruptions. Regular user-developer interaction is characteristic of most RAD projects – sometimes formalised in Joint Application Design and Joint Requirements Planning workshops. Interaction may be phased at regular intervals throughout the project or intensive in the sense of developers and users communicating on an hour-by-hour basis.
2. RAD Team. RAD is characterised by small development teams - typically four to eight persons - made up of both developers and users who are empowered to make design decisions. Users frequently act as managers on projects.
3. RAD Product. Usually, the products of RAD projects appear to be information systems. Such systems also seem to be characterised by high levels of interactivity and low-levels of background complexity.
4. RAD Process. Project control in RAD is seen to involve scoping the project by prioritising development and using negotiated delivery deadlines or 'timeboxes'. RAD is frequently discussed in terms of incremental prototyping and phased deliverables.

VI OUR STUDY

A Organisational Context

Our project was conducted in a small, commercial organisation employing approximately forty people. The main business of the company is to sell and deliver short training courses to commerce and industry. One of the authors was initially invited into the organisation by it's managing director to consider developing an information system to support the work of his sales and marketing team. A few months later a developer was contracted to implement the software system. A system for sales as well as other functionality was eventually delivered at the end of a six-month period. A further six months was spent delivering a cognate module in the area of course administration, and a third phase was spent in the construction of a marketing function as well as making some further adjustments to the course administration module. We concentrate in this paper on describing elements from the first and second phases of this project.

At the start of the project the development team took the view that an incremental development approach was the most suitable for the type of systems being proposed by the organisation. There were a number of reasons for choosing this approach. The company had undergone a rapid increase in business over the last two years. One of the overt reasons for the planned introduction of information technology was to attempt to cope with this increased volume without an increase in staffing levels. However, the organisation had only limited prior exposure to computers and no previous bespoke information systems had been built for the organisation. All of this meant that the requirements were necessarily uncertain since the organisation members had no prior experience of the potentialities of information technology. Also, one of the authors was familiar with aspects of two bodies of literature, which influenced this decision: RAD, and PD. On reflection, aspects of RAD were heavily fore-grounded during the duration of the project because of the commercial/consultancy nature of the work. The emphasis was on producing a working system to a given time-scale, within a finite budget, and with a clear objective of productivity improvement.

In the course of conducting this project both the authors maintained detailed field diaries. Audio and videotapes were also made of all user-developer meetings, amounting to some 100 plus hours of audiotape for the first two phases of the project. Also, all project documentation was collected in a project database, and all versions of the software system were archived. Second-tier reflections on project activities were maintained by one of the authors in an attempt to continuously reflect on the development activity using the strategy proposed by Schön [7].

B Project Planning

Because of manpower constraints this particular project proved to be a phased type of RAD. An initial JAD workshop was held with the 10 members of staff from the organisation likely to be effected by the development. This took a complete day. Following this workshop it was decided to organise the project in terms of three phases: Sales, Marketing and Courses. Within each phase at least three developed

prototypes were produced. User meetings were held both on delivery of each module as well as regularly if aspects of clarification were needed. An informal evaluation of each deliverable occurred some one to two months after each phase. In general terms, the project was an instance of both incremental development (within phases) and incremental delivery (between phases).

C *Development Process*

The structure of the development process within each phase of the development work is illustrated in figure 3. Each phase generally consisted of three timeboxes and led to the development of a version of the prototype for the domain being considered in the phase. User representation was determined before the start of each phase, usually by the user community suggesting representatives themselves. The start of each phase was marked by an initiation meeting between developers and user representatives. The main purpose of this meeting was to scope the phase, which on a practical level meant negotiating a series of *ability-tos* for the system module under consideration. This idea is an interpretation of a technique much-used in RAD work at the UK company Norwich Union [9]. Norwich Union developers tend to phrase their requirements in terms of *ability-tos* such as: *the system must be able to register claims made against policies or should be able to flag to the customer service manager the number of outstanding policy claims per month*. In our initiation meeting a global list of ability-tos was generated first. Then the ability-tos were prioritised in terms of something known as the MoSCoW framework.

Within RAD there is a continual emphasis on the listing and prioritisation of requirements within the constraints of a timebox. Timeboxing means setting a fixed deadline by which a set of business objectives must be met, rather than specifying when a task must be completed. The rationale is: first, that timeboxes enable the customer to see concrete examples of progress being made on a regular basis with the delivery of products; second, frequent small deliveries are preferable to one large delivery. Timeboxes are usually between 60 and 90 days duration. If, as the deadline or the timebox approaches it proves impossible to deliver against the planned business objectives, the timebox is never extended. Instead, business objectives (in our case, ability-tos) are removed from the plan.

Time-boxing is primarily used as a means of avoiding the problem of creeping functionality or scope which has been experienced in relation to incremental development projects and is usually expressed in terms of the generation of a list of requirements or business objectives. There are conventionally four categories of requirement on this list:

1. Must Haves. Products that an organisation must have. These products satisfy the critical success factors for the project
2. Should Haves. Products that an organisation should have. These products will directly benefit the business in a cost-effective way
3. Could Haves. Products that an organisation could have. These are products which will not directly benefit the organisation but would be nice to have
4. Won't Haves. Products that organisations would like to have, but probably will not have. These are products where it is not known if they will be of any use to the organisation

These four types of product are frequently phrased in terms of the mnemonic MoSCoW - Must Have, Should Have, Could Have, Won't Have. As part of the process of defining a timebox the products need to be ordered in terms of criticality or desirability. As the timebox progresses, other requirements may be identified or some of the products may take longer to develop. Thus the *could haves* and *won't haves* need to get displaced by the more critical *must haves* and *should haves*.

Hence in the initiation meeting the global list of ability-tos was first prioritised in terms of the MoSCoW categories. Then those ability-tos categorised as *must haves* and *should haves* were assigned to each of the three timeboxes in relation to estimates made by the developers as to the length of time needed to produce each ability-to.

Each timebox consisted of the development of a version of the prototype. Towards the end of each timebox an agenda-setting meeting was held in which a series of scenarios were prepared for structuring the demonstration of the prototype. At the end of the user review session the global list of ability-tos was reviewed and those requirements to form the next timebox were negotiated. After the completion of the user review a de-briefing meeting was held between the developers to document the agreed activities for the next timebox. A brief description of the work for the next timebox was prepared and distributed to user representatives. This development process is illustrated in figure 3.

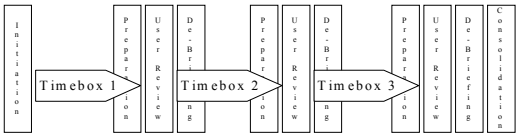

Fig. 3 The Development Process

VII SCENARIOS AND DESIGN BREAKDOWNS IN OUR PROJECT

A *Scenarios*

We usually set up scenarios as a means of organising the demonstrations of the prototype. All user review sessions were therefore initially scripted in these terms. Such scenarios were developed from our initial understanding of the current work activity of users as well as our envisionment of future work with the system. In this respect they corresponded to a mixture of Bødker and Grønbœk's type 1 (Future: Work) and type 2 (Future: System) situations. We always used a simple, terse textual description of the scenario to facilitate ease of understanding by the user.

For instance, in relation to the demonstration of a refined prototype of the Sales module, the following represent examples of scenarios used to demonstrate the system:

1. Scenario 1. Sent a mailer to an existing contact at the Royal Mint. 30,000 records in demo system. Selected company. Go to contacts, pick off contact with details. Memos replace actions - metaphor with memo-pad. Log in-coming call. Details of fax to be sent off logged as new memo-item. Looked in to-do list; notify of sending fax.

2. Scenario 2. Advert placed in Western Mail. Person phones up and not in database. Want to record his details while on the phone. Enter company details including interests as tags. Company details copied across to contact form. Enter contact details. Tag with interest in PC courses. Place memo against contact - posting a letter. Back to to-do list. Removing items from to-do list.

Our scenarios represented key, concrete episodes in the activity of people working with our information system. However, we acknowledge that our use of scenarios did not involve much user personalisation - for instance, we did not say that Gordon (a particular sales-person) would do such and such. Also, our scenarios were notably short and succinct – informal rather than rigorous.

B Breakdowns

In the following section we discuss a number of examples of breakdown situation organised in terms of the typology presented in section 1.1.

Type 1 Breakdowns (Future: Work)

By this type of breakdown we are considering situations in which issues relating to the design of future work in association with an information system and the utility of the information system within this context becomes the focus of awareness.

In our review sessions, users would use the prototype as a means of proposing re-design to their own future interaction with the system. Here the prototype was used as a basis for idea exploration. On reflection, the users only began treating the prototype in this way some way into the project when they appeared to be reasonably confident and comfortable with the approach that was taken.

One particularly good example of this arose in discussion of the maintenance of a 'to-do list', a facility which allowed sales people to record actions needed to be carried out in relation to contacts, e.g., send them further information about a course, re-telephoning them at some future date etc. The developers demonstrated this facility in the prototype and made the explicit point that the design of the to-do list assumed that it was accessible by all users. Some debate arose around the issue with a provisional conclusion that a 'to-do list' should be specific to each user, and only accessible by that user. But then formulations of a number of breakdown situations emerged. One user maintained the need to be able to distinguish between who takes a telephone call against who's to do something in relation to the call. He formulated this as: *occasionally I might take a call, but I might wish to delegate an appropriate action to somebody else*. After some debate, a 'solution' was eventually proposed by the user group in terms of a design that involved a to-do item defaulting to the current user identifier but with the ability to override this in terms of future actions.

Then the question was proposed: *What happens if a user is off sick?. I may need access to their list to take over their work*. An initial suggestion was that somebody logs on as the sick user and takes up their to-do list. Concerns were then expressed in terms of the need for privacy and security of personal to-do items. Another solution was then proposed, to have an administrator's to-do list. One of the developers then re-formulated this in terms of an administrator having access to all to-do lists with the ability to switch user identifiers against actions.

Type 2 Breakdowns (Future: System)

By this type of breakdown we are considering situations in which issues relating to the usability of a prototypical system becomes the focus of awareness.

For instance, in the same meeting in which the scenarios in section A were used, the prototype being demonstrated assumed that users would primarily access contact information via a company key. In response to this, one user asked if it was possible to access contact information by keying on contact name? The developer explained that this was not possible in the current prototype. Both users then emphasised the need for keying by contact as well as company name because frequently they didn't remember the company name of a contact, but did have a rough idea of the contact name.

In another user review session the development team became depressed at what appeared to be a substantial number of perceived problems with the prototype being demonstrated. In retrospect, many of the issues raised did not actually lead to a significant re-framing of functionality. This was because it was decided jointly between users and developers that a major part of the problem was one of interface terminology rather than system functionality. For instance, it was decided to rename business classification and contact classification fields as *company tags* and *contact tags* to accord with the use of these terms in a packaged computer system previously used by the sales team.

Type 3 Breakdowns (Current: Work)

By this type of breakdown we are considering situations in which issues relating to the way in which current work is organised perhaps in association with an information system, and the utility of the information system within this context becomes the focus of awareness.

As has been mentioned, all of our generated scenarios were primarily designed as type 1 situations (Future: Work), i.e., situations which attempted to simulate future work activity with the system. Frequently however, in the process of demonstration, the focus tended to shift to type 3 situations (Current: Work) where the users would propose additional (frequently exceptional) scenarios of current work practice in an attempt to see if these would 'break' the system.

For example, in a meeting in which an iteration of the Courses module was demonstrated, over half of the meeting was taken up with users proposing potential breakdown situations. In one instance, a user proposed the question: *now what happens in the situation that a contact wants to make four provisional bookings, but doesn't want to assign names to delegates yet?* In the demonstrated prototype only one delegate had been allowed against each booking, and each course name-delegate name pair had to be unique. Hence, one person could not book many places on a course on other people's behalf without the system first knowing the names of the proposed persons. It became clear that this was untenable in organisational terms as the scenario proposed by the user was a commonplace one. This led to the re-design of the

system such that it allowed an initial contact name to be duplicated across several bookings, with the ability to enter delegate details later.

Type 5 Breakdowns (Current: System)

By this type of breakdown we are considering situations in which issues relating to the usability of an existing, delivered information system becomes the focus of awareness. In classic usability testing, this seems to be the type of breakdown that is most often covered.

In relation to our own project, these types of breakdown only became relevant in the second phase of the development of a software module. That is, after delivery of a system module and a suitably elapsed period of use in context had taken place. This is probably a consequence of our use of what Bødker and Grønbœk would call demonstration prototyping rather than co-operative prototyping.

One of the key instances of breakdowns of this type emerged after we had delivered the first version of a module for enquiry management. This module was intended to be used to log enquiries made directly into the system as and when sales-people were on the telephone. After a couple of months of use of this software it became apparent that very few enquiries were being logged against the system. On questioning user representatives as to the reason for this within a user review session, it became apparent that the use of both the contacts and enquiry management modules in the way they were originally intended to be used had proven impracticable for two major reasons: firstly, users found it difficult to concentrate on the content of a conversation with a person on the end of a phone while needing to utilise the system in parallel with this activity; secondly, users found the performance of the system inadequate to the task of rapidly searching for and adding contact details. As one user put it, *I don't want to be asking the person on the end of the line to wait while I pull up his details*.

VIII Lessons from Our Study

Our main aim has been to investigate in what way some of the lessons and techniques from PD can be adapted to RAD. The concept of a design breakdown does not have any pedigree within the contemporary RAD literature. However, we think it has much potential for acting as a more pragmatic focus for design workshops and review sessions, particularly in relation to a technique such as scenarios. It is interesting that scenarios also do not appear to be discussed in relation to RAD currently. We believe that scenarios can act as useful design representations in a number of different ways within RAD work. In this section we include some reflections on firstly our use of design breakdowns and secondly our use of scenarios.

A Design Breakdowns

Our experience leads us to disagree with Bødker and Grønbœk that incremental prototyping sessions lack the ability to involve users actively in design. In lieu of full co-operative prototyping, the elaboration of scenarios in correspondence with a usable prototype seems sufficient to foreground many of the issues of breakdowns and their resolution. Therefore, we believe that breakdowns are equally relevant to an incremental prototyping session, particularly if managed through a scenario. However, it is important that users feel comfortable with developing and exploring their own scenarios in the context of the system. It is also equally important for developers not to invest their egos in products - 'egoless' development is even more foregrounded with an incremental prototyping approach.

Having said this, we did frequently follow up our review meetings with user trials of the system if the prototype remained a relatively stable output from the design session. In such sessions further breakdowns did emerge. One user was puzzled by this. He made the comment that he had seen versions of the system many times and hence he did not know why he had not spotted these problems before. This, and other instances like this, lends support to the PD emphasis on simulated hands-on work-situations as a necessary factor in stimulating effective analysis of breakdown situations.

However, there does seem to be something of a contradiction in the co-operative design literature. On the one hand breakdowns are considered as a useful emergent phenomena in participatory design. On the other hand breakdowns are something either to be avoided in the design of an interface or they are to be handled by the interface. This means that there does seem to be an assumption that a system is perfectible in that breakdowns can be designed out. We feel that this assumption is somewhat simplistic. Whilst acknowledging that the aim of good computer design is to confront breakdowns head on and adjust systems accordingly it is unlikely that a delivered system will remain untroubled by further breakdowns in use. This is likely because, for example, the delivery and use of a new computer system impacts upon and causes adjustments to be made in work which were unintended in the original design [10]. We therefore place some credence in the RAD aphorism:

The application is always complete, but never finished

B Scenarios

In relation to scenarios our interest has been in the fit between this concept and commercial development work. In this sense we concur with In terms of Carroll's typology of scenario usage described in section A, because of the intensive and iterative nature of RAD work, our use of scenarios tended to overlap with a number of scenario types.

We used scenarios primarily as a technique to facilitate user-designer communication. Explicitly, we developed scenarios of use primarily as a means of structuring demonstrations of a prototype at user review meetings. In this sense, scenarios were used as ways of encapsulating and exploring design rationale with users in review sessions. At such sessions, users evaluated increments of a prototype in terms of the specific tasks (ability-tos) the increment was designed to support.

Scenarios were also used as a way of framing aspects of the usability of the interfaces being demonstrated to users. However, scenarios as a usability engineering technique differ in two fundamental ways from the range of usability inspection techniques currently being proposed. First, the

focus of scenarios tends to be broader than merely that concerned with issues of interface usability. Second, inspection is a generic term for a range of usability engineering approaches that do not involve end-users. Instead, evaluation is based on the considered judgement of a set of expert evaluators [11].

In reviewing our tape material of user meetings we can clearly identify a host of occasions where users are utilising a scenario model in either describing their work processes to the development team or in elaborating on design issues which were seen as being important to them. In their descriptions of current work activity, users frequently started with a verbal description of what they regarded as the most typical case, such as the making of one booking on a course in the name of one delegate. Then they attempted to break the typicality of the case by discussing exceptional cases such as when a person makes a booking on a course on behalf of a group of other people.

It is interesting that we have also found users quite prepared to shift focus within describing their current work to discussing envisionment scenarios, frequently by identifying breakdowns in their current work practice that suggest the need for change. An example here is the way in which our user community readily acknowledged that in terms of their current paper-based activities it was very difficult for a person to take over the work of colleagues in their absence. The ability of the system to enable this change to working practice was proposed as a major strength of a future system.

On reflection, our sensitivity to many of these issues can be seen to be a by-product of our use of audio tape as a representation of requirements. Following user review meetings at least one, but sometimes both developers, would listen to the tape material to raise their awareness of design problems. It is in relation to such listening and reflection that we have found the mutual concepts of scenarios and breakdowns of most practical use.

Therefore, without wishing to deny the importance of gaining an in-depth appreciation of organisational work through detailed study wherever possible [6], we would concur with Nardi [12] that scenarios of this form may act as a useful, if admittedly limited substitute for full and rigorous work analysis.

REFERENCES

[1] Winograd, T. and F. Flores. *Understanding Computers and Cognition: a new foundation for design*. Norwood, N.J., Ablex Publishing, 1986.

[2] Bødker, S. *Through the interface: a human activity approach to user interface design*. New Jersey, Lawrence Erlbaum, 1991.

[3] Bødker, S. and K. Grønbœk. Co-operative Prototyping: users and designers in mutual activity. *International Journal of Man-Machine Studies*, Vol. 34, 453-478, 1991.

[4] Carroll, J. M., Ed. *Scenario-Based Design: envisioning work and technology in systems development*. New York, John Wiley, 1995.

[5] Kuuti, K. Work Processes: scenarios as a preliminary vocabulary. *Scenario-based design: envisioning work and technology in systems development*. J. M. Carroll. New York, John Wiley, 1995.

[6] Kyng, M. Creating contexts for design. *Scenario-Based Design: envisioning work and technology in systems development*. J. M. Carroll. New York, John Wiley, 1995.

[7] Schön, D. A. *The reflective practitioner: how professionals think in action*. Aldershot, Ashgate, 1983.

[8] Martin, J. *Rapid Application Development*. Englewood Cliffs, Prentice-Hall, 1992.

[9] Webster, S. Focus groups. *RAD, JAD and DSDM*. Heathrow, London, Unicom Seminars, 1996.

[10] Giddings, R. V. (1984). Accommodating Uncertainty in Software Design. *Comm. ACM*, Vol. 27. No. 5, 428-434, 1984.

[11] Nardi, B. Some reflections on scenarios. *Scenario-Based Design: envisioning work and technology in systems development*. J. M. Carroll. New York, John Wiley, 1995.

[12] Nielsen, J. and R. L. Mack. *Usability Inspection Methods*. New York, John Wiley, 1994.