# Chapter 3
# The participation problem: roles, participation, and the conflation of contexts in groupware development

In the previous chapter, I have discussed the issues and challenges surrounding groupware development. I have explained that Brooks's (1987) "No Silver Bullet" argument has particular relevance for groupware development as a special branch of software development. Based on Brooks's argument, the most efficacious groupware should be developed by the group of people[42] who will eventually use it, where the groupware *grows* until its final disposal. This implies that efficacious groupware development requires incremental development and continuous *fine-tuning* similar to that in crafting (cf. Brooks, 1987) in order to provide adequate support to the evolving contexts[43]. This *crafting* process may be continued in the use context. In this sense, neither the activity of coding nor the activity of use can be used to characterize the *stage* of the evolving groupware. In other words, the context of development and the context of use are conflated.

The conflation of the development context and the use context challenges the traditional concept of participation in groupware development through pre-conceived roles. In this conception, the roles of the participants are usually assumed unchanged throughout the

[42] Although this usually refers to end-users, the group may consist of other stakeholders such as managers and system developers.

[43] The evolving contexts take account of environment, human activity, the understanding of the developers, users and other stakeholders.

development and participation usually refers to end-user participation in the developer's space. In a conflated context, however, the concepts of role and participation are blurred. The roles of participants are usually dynamic throughout the development and participation may be conceived as multi-directional. It is the case that the end-user participates in the developer's space, but it is also the case that the developer participates in the end-user's space. In fact, the conflation of contexts intertwines with the phenomenon of dynamic roles. On the one hand, the conflated context seems to promote dynamic roles by allowing participants to interact with the evolving groupware in various ways, such as to construct, to (re-)configure, to observe, or to use[44]. On the other hand, dynamic roles will also cause conflation of the contexts. This can happen when the participants are developing and using the evolving groupware in the same shared context.

In this chapter, I discuss the issues surrounding the conflation of the context of development and the context of use, the concept of role and participation, role-shifting phenomenon in systems development, and the impacts of these issues and conceptual variations on groupware development. In section 3.1, I examine the way in which the developer's understanding and the artifact co-evolve. This co-evolution phenomenon has been noted by Naur (1985a) and has been discussed among activity theorists in interaction design, e.g. Kaptelinin and Nardi (2006). It also exemplifies reality construction that Floyd (1992) has observed in software development. The co-evolution phenomenon suggests that the development of the artifact is intertwined with the developer's understanding of the problem situation. Therefore, granting equal opportunity to all participants to *craft* the groupware may enhance its fitness for the evolving context and enhance the understanding of the participants.

Developer and user are two crucial roles in the evolution of groupware. However, the concepts of developer and user remain controversial. In the traditional conception, the role of an individual in groupware development is more often determined by their organisational

[44] There are many different ways to interact with the evolving groupware, e.g. observing the changes being made by other participants at real time, learning about the groupware by exploring without an explicit goal, etc. From a software engineering perspective, the "construction" task can further be broken down into coding, testing, debugging, evaluating, etc.

roles rather than by their capabilities. This leads to the situation in which individuals usually play a single, static and separate role throughout the groupware development in the context of a high division of labour. In practice, it is possible for a *user* to do the programming task if she is equipped with the programming knowledge. In section 3.2, I look critically at the concepts of developer and user. I argue that the tension between developers and users is caused by the assumption that an individual can only play a single and static role. I then turn to discussing the possibilities and the issues of developer-as-user and user-as-developer.

Theoretically, an individual rarely plays multiple roles in groupware development because this requires quite different sets of skills and knowledge. In practice, however, it is not uncommon for individuals to have to shift their roles in different situations and at different times. In section 3.3, I report two examples of systems development where this role-shifting behaviour was observed, one by Friis (1998) and the other by Danielsson (2004). I argue that this role-shifting phenomenon should not be viewed either as unstable, temporary, or exceptional behaviour of individual participants. In these examples, a number of participants shifted their roles from passive users to active users, and from active users to pseudo-developers.

In section 3.4, I revisit the notion of participation (see section 2.5.2). I argue that the implicit assumption behind the concept of participation – as the term in used in PD – that the *participants* are non-developers – is problematic when individuals can shift their roles throughout the development process. In fact, the development process should be viewed as a co-construction process where individuals' understanding may co-evolve with the groupware throughout its development and evolution.

In section 3.5, I summarise this chapter, highlighting the implications for developing an *environment* for groupware development, drawing on the discussions from section 3.1 to section 3.4. I argue that the traditional concept of role and participation may be an obstacle to realising efficacious groupware development. To remedy this situation, we may want to consider an alternative conception of role and participant in a conflated context of development and use. This chapter concludes with a discussion on the implications of these changes for the existing conception of groupware development.

## 3.1 The co-evolution between the developer's understanding and the artifact[45]

It has been suggested that there is a close connection between the developer's understanding that is associated with the software development process and the construction of the actual artifact, i.e. the software under construction. Peter Naur (1985a) was one of the first researchers to acknowledge the importance of recognising such a relationship:

> *"a vital issue is the proper view of the development process considered as an interplay of intuitive knowledge had by the programming person and the real world accessible to that person, which includes the texts used or produced by the person." (Naur 1985a, p.73)*

As figure 3.1 depicts, the developer's intuitive knowledge will gradually evolve in relation to the development of artifacts throughout the development. In a software development project, the developer begins with her initial knowledge (K1) of the real world (W). As the development progresses, the developer's knowledge of the problem domain and the artifact will gradually be improved (K2) as the artifact is being developed (A1). At the same time, the artifact (i.e. the software and any documentation describing the software including any specifications and manuals) is actually developed within the real world. Further development will bring the artifact into a more complete state (A2) and the developer will achieve a better understanding of the artifact (K3). Although the development of the artifact may be restricted by specifications[46], there is no limit to how far the developer's understanding may be developed. Indeed, this co-evolution occurs no matter which methodology for software development is adopted (Naur, 1985a).

[45] Part of the argument in this section has been presented in a paper published in The 20th Annual Psychology of Programming Interest Group Conference 2008 (Beynon et al. 2008).

[46] The scope of the artifact might be limited in the context of traditional software development, but is subject to no limits if the development adopts an evolutionary perspective, as we discuss in section 2.4.

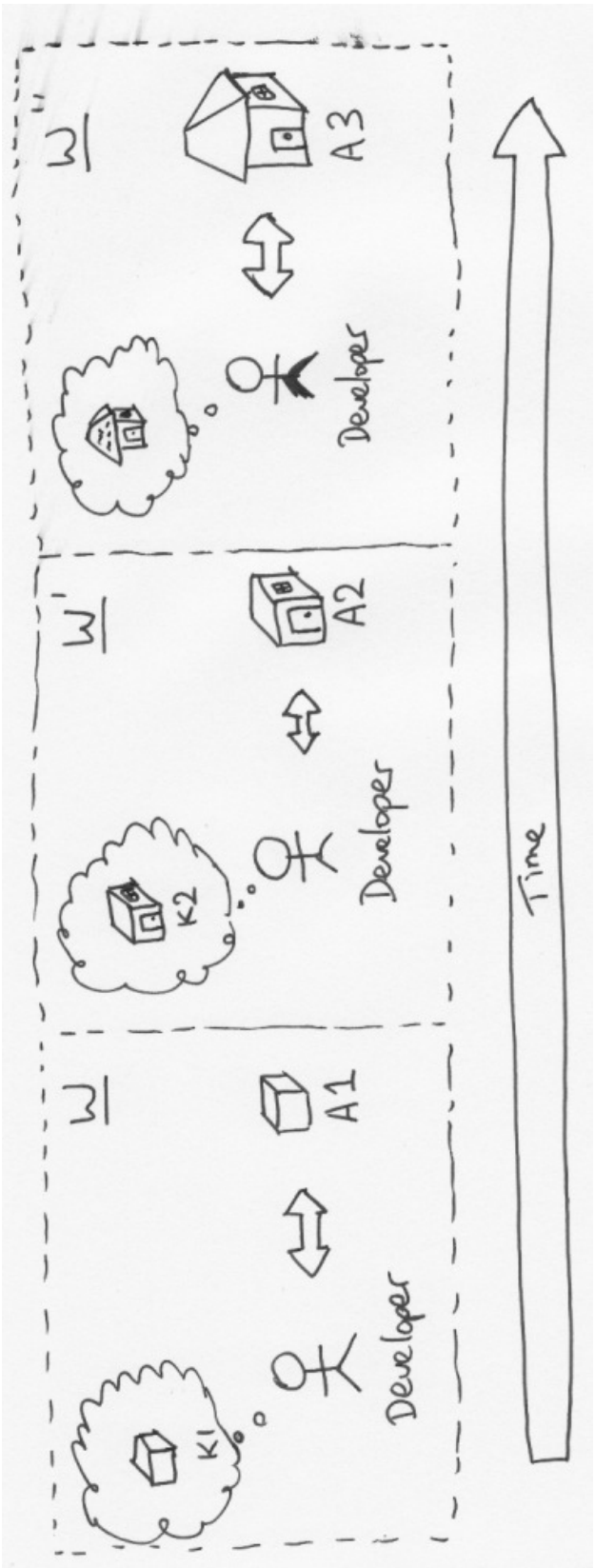



*Figure 3.1 – The co-evolution between the developer's understanding and the development of an artifact*

For Naur (1985a), the interplay between the developer's understanding and the artifact produced by the developer is driven by human intuitive knowledge. Following the above discussion; in order to grasp K2, the developer has to relate her knowledge of the artifact A1 to her knowledge of the world, i.e. part of K1. This interplay only makes sense if it is "understood as elaborate, purposeful actions undertaken by the programmer, depending on intuitive insight at every turn" (Naur, 1985a).

However, there is a subtle difference between what can and cannot be regarded as intuition. According to the Oxford English Dictionary (2008), *intuition* refers to "The immediate apprehension of an object by the mind without the intervention of any reasoning process"; "Immediate apprehension by the intellect alone"; "Immediate apprehension by sense"; or an instance of "Direct or immediate insight". What Naur (1985a) refers to as intuition here is more closely matched to immediate apprehension without the intervention of any reasoning process, by the intellect alone, or direct or immediate insight, rather than merely an "immediate apprehension by sense". Indeed, what we can apprehend immediately is highly affected by experience and training, i.e. what we already know (Beynon et al., 2008). Therefore, the extreme forms of guesswork with minimal understanding that novice developers may resort to might not involve intuition in this sense – such activity might not involve significant "apprehension of the program by the mind" (Beynon et al., 2008). In contrast, a purposeful trial-and-error activity of an experienced developer might be guided by "apprehension by the intellect alone".

Naur (1985a) considers the potential hazards of human intuition. On the one hand, Naur is concerned with ways of avoiding the flaws of human intuition through the use of methodologies and techniques. On the other hand, he argues that formal methods do not guarantee the absences of flaws and acknowledges the importance of human intuition:

> *"The claim is often made that certain forms, or formalizations, will guarantee the absence of flaws of arguments. What seems to lie behind such claim is the fact that by the use of certain kinds of formalizations it is possible to formulate the connections between statements corresponding to a proof in terms of rules for manipulating the statements. While this property is of great*

> *interest as a matter of principle, and also is the necessary basis for mechanical proof construction and verification, and occasionally is used in the reasoning carried out by people, it provides no guarantee for the absence of flaws in the arguments used in software development making use of formalizations … Avoiding flaws in that modelling undoubtedly depends to some extent on the form or language used for the description. However, what is the most suitable form or language for any situation will depend not only on the degree of formalization of the description, but equally on the character of the aspect of the world to be modelled and the background experience of the programmer who has to establish the description. " (Naur 1985a, p.77)*

From Naur's perspective, the involvement of human intuition is indispensable in software development. Naur (1985a; 1985b) observes that the developer's intuitive knowledge, i.e. the developer's understanding, is so pervasively involved in software development that it has only attracted researchers' attention when it is exposed in exceptional circumstances, e.g. failure as a result of acting on intuitions. Naur (1985a) also observes a difficulty in discriminating developer's intuitive knowledge while acknowledging that the developer's knowledge "at any time is an indivisible whole". This view is closely connected to the outlook of William James (1912; cf. Beynon et al., 2008). Furthermore, such a kind of knowledge can hardly be formulated into rules, as such rules further require other sets of rules to explain how to apply them and so forth, and this becomes "an infinite regress" (Naur 1985a; 1985b).

In another work, Naur (1985b) proposes a "programming as theory building" perspective on software development[47]. Unlike Turski and Maibaum (1987), who adopted a formal conceptualisation for their theory building perspective, Naur adopts a 'softer' notion of theory, as developed by Gilbert Ryle (1949). Naur (1985b) explains briefly that, in Ryle's sense, "a person who has or possesses a theory … knows how to do certain things and in

[47] Naur (1985b) views programming embracing "the whole activity of design and implementation of programmed solutions", i.e. the entire lifecycle of software development.

addition can support the actual doing with explanations, justifications, and answers to queries, about the activity of concern." This means that the developer who builds the software possesses a *knowledge that enables her to perform activities* that transcend what is captured in other document-based artifacts (Naur, 1985b). According to Naur (1985b), these activities may include:

i) relating the software to the affairs of the world which will be supported by the software;

ii) explaining and justifying each part of the software; and

iii) responding to any demand to change the software *constructively*.

As stressed by Naur (1985b), this notion of theory is not the same as what is often called *tacit knowledge*, nor is it the same as intuitive knowledge such as I discussed earlier in this section. While the former is closely related to the implicit knowledge of its design rationale and the structure of software, the latter involves a more general insight into the program.

The most interesting thing about Naur's theory building perspective is that it emphasizes the development of the theory, which has "primacy" over other document-based artifacts such as specifications and user manuals (Naur, 1985b). This also contributes to the growth of the developer's intuitive knowledge of the software. As Naur argues, "an essential part of any [software], the theory of it, is something that could not conceivably be expressed, but is inextricably bound to human beings", therefore, "the proper, primary aim or programming is, not to produce programs, but to have the programmers build theories of the manner in which the problems at hand are solved by program execution." (Naur, 1985b). This puts Naur's theory building perspective in a position that is highly contrasted with the product development perspective, where the primary focus is on documentation and the software itself, and where the development is thought of as guided by a prescriptive specification. To some extent, Naur's theory building perspective conveys the need to attend to the social issues and the situated nature of software development – which are not popular concerns in software engineering, but are topical in research areas such as PD, CSCW, and HCI. These concerns are echoed in Brooks's (1987) *growing* software metaphor and Floyd's (1992) notion of *reality construction*. Floyd (1992, p. 95; cited in Rönkkö, 2007, p.689) argues that:

- *We do not analyze requirements; we construct them from our own perspective. This perspective is affected by our personal priorities and values, by the methods we use as orientation aids, and by our interaction with others constructing requirements from their perspective. Requirements are governed by perspective. In most cases they reflect divergences in perspective and are subject to temporal changes.*
- *We do not apply predefined methods, but construct them to suit the situation at hand. There are no such things as methods per se – what we are invariably concerned with are processes of situative method development and application. We select methods and adopt them. What we are ultimately doing in the course of design is developing our own methods.*
- *We do not refer to fixed means of implementation that only take effect later on when working out the details of implementation decisions. Instead, we construct the meaningful use of means of implementation by testing, selecting, or complementing what is already available*

Brooks's (1987) *growing* software metaphor, to a certain degree, highlighted the need to understand how support for the developer's intuition and communication can be embodied in the artifact itself (cf. Beynon et al., 2008). The idea that understanding can be implicit in an artifact and interpretations of interactions with it is vividly expressed in Gooding's notion of *construal* (Beynon et al., 2008). The idea that the developer's understanding of the artifact will evolve throughout the construction of the artifact is a central tenet of constructivist computing (cf. Harfield, 2008), and the idea that interaction with the artifact may evolve from time to time due to its changing environment is one of the main concerns in Activity Theory (cf. Kaptelinin and Nardi, 2006).

## 3.2 Reconceptualising the developer and the user

In this section, I argue that the traditional narrow conceptions of developer and user maintain a divide between two parties (cf. §3.2.1). The divide in the conception has raised two issues:

i) The disparity between the influence that the developer and the user can

exercise over the construction of the artifact (cf. §3.1)

ii) Difficulty in conceptualise the role-shifting phenomena when user involvement and workspace democracy is increased. (cf. §3.3)

To understand the root causes of these issues, I examine the broadening conception of developer and user in section 3.2.2 and section 3.2.3 respectively. Despite a trend towards broadening the conceptions of developer and user, the confusions that arise within the narrow conceptions may not be completely resolved. This is because (cf. Kuutti, 2001; Bannon, 1992b):

i) The narrow conceptions are not completely eradicated.

ii) Discussion is still set within the traditional conception of systems development, i.e. the divide still exists and when the user can perform supposedly developer's activities it breaks out of the traditional conception.

Consequently, I suggested a social role perspective for the conception of developer and user (cf. §3.2.4). That is, they should be thought as roles that can be taken by persons instead of entities of systems development in the traditional narrow conception. In addition, each person may play the role of developer and user at the same time or different times, as well as other social roles.

The discussion in this section focuses on the relationship between developer and user. The discussion of other issues, viz. the role-shifting phenomena in systems development and the need to revise the notion of participation due to the conflation of contexts will be postponed to later sections (cf. §3.3 and §3.4) of this chapter.

## 3.2.1 The tension between the developers and the users

In systems development, different stakeholders usually possess different or even conflicting perspectives about what artifact is to be built and how it should be built. This potentially creates a tension among all stakeholders, particularly between the developers and the users due to their different interests. In the worst case, the systems development process becomes an arena for negotiating their private interests (Howcroft and Wilson, 2003). From the organisational perspective, the existence of the tension is unrelated to the development

approaches practised, and it is hard to dismiss due to its social nature – it exists anyway irrespective of the role of the actors. Since the difference of interests is mainly between the developers and the users, I argue that the tension is caused by the preconceived roles of developer and user (i.e. what they can do, are supposed to do, and are allowed to do) and the implicit assumption that a person may play only one role in most of the conceptual frameworks for systems development. Furthermore, the co-evolution between the construction of the artifact and developer's understanding (cf. §3.1) hints at an imbalance between the developers' and the users' access to the knowledge acquisition through the construction of the artifact – as far as the traditional conception is concerned, the developers have direct access, while the users can only passively influence the construction.

In the traditional conception of systems development, there is a clear separation between the developers and the users. An individual who has a *developer* role in systems development is expected and is allowed to build the system, while an individual who has a *user* role is expected and is allowed to exploit the system for her work. This separation is due to several implicit assumptions in the traditional conception:

i) The developers and the users possess different sets of skills, knowledge, and abilities, and they are working in different workspaces, on different sites, at different time, etc.

ii) The developers and the users are working in separated contexts: the developer (role) is working within the development context, and the user (role) is working within the use context [48].

Fischer (1999) even goes further to suggest that there is a *symmetry of ignorance* between the developers and the users, i.e. the developer is ignorant of the user's space, and the user is ignorant of the developer's space, and these spaces are disconnected. From the role

[48] The separation of contexts becomes apparent in product development, when the systems development is considered as a software factory. In the author's own working experience in the e-commerce industry, the development environment (i.e. the software and hardware configuration for developer to construct, test, and evaluate the system) is usually separated from, and often different from, the production environment due to resource limitations. Even in some in-house development projects, customers are usually unwilling to invest in equipment for the developers and unwilling to share their brand new production hardware for the development. These differences make it hard for the developers to discuss and relate innovative ideas to the users.

theory perspective, the traditional conception of the developer and the user are reciprocal (Biddle and Thomas, 1966). That is, there is a producer-consumer relationship between the developer role and the user role. This may promote a conflict of interest between the developer and the user.

The divide between the conceptions of developer and user can also be found in the fundamental concepts of Information Technology (IT). Alter (2000) observed that the developer-oriented perspective (with reference to the narrow conceptions of developer and user; in Alter's terms, the IT perspective) tends to emphasize the interests of developers and hence view the users as more distant from the developer compared to the business perspective[49]. In this sense, the conception seems to suggest a necessary divide between the developer and the user in order to achieve the integrity of the traditional conception of systems development – i.e. the relationship between the developer and the user has to be reciprocal and is necessarily facilitated by document-based communication such as requirements specification. Another reason for this divide may be due to the fact that the roles of individuals in relation to the development process are often restricted by their organisational roles and the culture of participation in the organisation (cf. §2.5.2).

As I discussed in section 3.1, there is an intimate co-evolutionary relationship between the developer's understanding and the software (Naur, 1985a), and to that extent the developer who constructs the software may be considered as holding the *theory* of the software (Naur, 1985b). Since the users cannot interact with the artifact as the developers do (lacking the intimate co-evolutionary relationship), the developer can be viewed as having a privileged influence over the artifact in the systems development process, and the user can only passively influence the development of the artifact even though the *ownership of the artifact* is later transferred to the user. In other words, the intimate interplay between developer's understanding and the artifact results in a disparity of influence between the developers and the users. This disparity, together with the divide between the developers and the users in

[49] According to Alter (2000), the business perspective is the perspective that is taken by non-IT professionals such as staff in sales, marketing, engineering, general management, etc. These people are interested in how the system achieves their goals effectively and how it might provide a better work life (Alter 2000). They may like, dislike, or tolerate doing work in relation to systems development (Alter 2000).

the traditional conception, contributes to a tension between the developers and the users in systems development.

### 3.2.2 Broadening the conception of user

Common sense suggests that the term *user* refers to the individual who will ultimately use the artifact as constructed by the developer. However, careful examination in the literature suggests that the concept of user covers a wide range of possibilities. Drawing on von Hippel (1986), Bannon (1992b), Wallace (1999), Alter (2000), Kuutti (2001), Lamb and Kling (2003), Iivari (2006), the user can be thought as:

- a component in an organisational system (influenced by organisational studies)
- a source of errors (influenced by cognitive psychology and human factors)
- a social actor / a human actor (influenced by anthropology and micro-sociology)
- a consumer (influenced by marketing and design)
- a learner (influenced by learning theory and pedagogy)

The conceptions of the user has also been characterised in narrower and broader terms in different contexts. In the narrow conception, such as was topical during the first wave HCI, the users are conceived as "… at worst, idiots who must be shielded from the machine, or at best, simply sets of elementary processes of "factors" that can be studied in isolation in the laboratory" (Bannon, 1992b). The adoption of this limited scope user conception can be found in many disciplines such as Software Engineering (SE), Information Systems (IS), and Human Computer Interaction (HCI). For instance, the user is treated as the source of information within the traditional waterfall systems development model. In interactive systems and products design within the research community of HCI, this notion of user is exemplified by the popularity of user-centred design (UCD), which is "a philosophy based on the needs and interests of the user" (Norman, 2002, p.188). Although UCD claims that it will create a more usable and understandable system or product for the user (Norman and Draper, 1986), it is based on some implicit assumptions:

i. The user can be studied in laboratory settings and can be modelled accurately

with conceptual tools such as user and task modelling[50] (Bannon, 1992b).

ii. The interaction between the user and the machine can be isolated from the social context in which the user is living (Lamb and Kling, 2003).

These implicit assumptions were broadly made in HCI but were not questioned until the "second-wave" of HCI (Kaptelinin et al., 2003; cf. Bannon, 1992b). The main issue with this the narrow-scoped user conception is that it does not take the complex *interface* that the user is facing into account (Bannon, 1992b). Grudin (1990) argues that the user not only interacts with the computer, but also other people, artifacts and processes in the context in which the user is situated, and all of these are significant in shaping the user's interaction with the computer. That is, the computer is only one of the many entities in the *user's interface*. Indeed, it is through these seemingly non-essential everyday social interactions that we articulate our work, and many groupware applications have failed due to their lack of concern for the social aspects of the user (as I discussed in §2.2.2).

Consequently, broader conceptions of user have emerged, e.g. *user as human actor* (Bannon, 1992b), *user as social actor* (Lamb and Kling, 2003). In these conceptions, the social aspects of the user, such as the organisational structure and the social environment that circumscribe the user, are taken into account. Moreover, user are conceived as active agents who are competent in their work practices and "wish to accomplish tasks, to understand what is going on, and are willing to jump ahead and explore the computer system on their own if, for example, the tutorial material is unclear or too pedantic" (Bannon, 1992b). Bannon (1992b) argues "By using the term "human actors" emphasis is placed on the person as an autonomous agent that has the capacity to regulate and coordinate his or her behaviour, rather than simply being a passive element in a human-machine system." Furthermore, the users are thought of as "partners in an ongoing and continuous social interaction with various communities … their thinking is shaped by their personal histories and by their membership in those various cultural communities" (Kuutti, 2001). Besides

[50] User modelling and task modelling are the conceptual tools that a usability engineer may use to analyse the user's behaviour and the user's task during interactive systems design. These conceptual tools are based on the first generation cognitive psychology theories. From my observation, these conceptual tools are still being used, at the moment, in practice.

these social factors, the broadening conception of user has recently been influenced by the consumer perspective, which originated from consumer product design. In this consumer perspective, the user is conceived to have "emotions and needs for pleasure and self-expression" apart from rationality and reason (Kuutti, 2001; also cf. McCarthy and Wright, 2004).

### User-as-learner

Apart from the social and consumer experience aspects, the broader conception should include the learning aspect of the user. Quite often, the user has to going through a learning process before she can fully adapt to the system and properly configure the system for her work. However, this aspect is usually overlooked. As Kuutti (2001) pointed out, "When designers think about users, they rarely think of learning and dynamics, neither in the small – users as learning to master the technology device itself – nor in the large – users as learning something through the technology." In fact, the learning of the user not only occurs in the use context, but also occurs in the development context. For instance, if the user is actively involved during the development process, she may also need to learn about the future configuration of the computer system and changes to her work and workspace in order to give appropriate comments regarding the design of the system. All these suggest that the learning of a user is pervasively intertwined with the entire development/use process.

### User-as-developer

Apart from the above aspects, there is still (at least one) aspect missing from the scene: the *user as developer*. If users are considered to be social actors who are able to perform operations informed by their knowledge (i.e. capabilities, skills, trainings, experiences, etc) and relate the artifact to their work *in situ* (cf. Suchman, 2007), there is no reason why the users should be treated as "technically-incompetent" (i.e. as only able to carry out activities other than development activities, even though they may be technically-competent). Furthermore, users may take the "first steps" to programming if they believe the profit of the program is greater than the investment of their attention (Blackwell 2002; 2004). For instance, if the user believes learning to use the macro feature in a word processor and

writing a macro may save time in the future for a repetitive task, she may choose to learn and write the program rather than perform it manually. Indeed, the (main) difference between professional developers and end-user developers is whether they regard programming as their primary job:

> *"A "professional" programmer might be defined as someone whose primary job function is to write or maintain software. A "novice" programmer might be defined as someone who is learning how to program ... "end-user programmers" (EUP) are people who write programs, but not as their primary job function ... but some EUPs, such as chemists or other scientists, may need to learn to use "regular" programming languages such as C or Java to achieve their programming goals" (Myers, 2006)*

This argument is supported by recent research, which suggests that, in the US, a large number of end-users may do programming in their job (Scaffidi, 2005; Myers et al., 2007) In fact, members of the younger generation are often more experienced in using computer technologies than their teachers. It is not uncommon for high school and college students to be required to perform information search from the web, to write reports using word processors, and to do a few programming exercises (Xiao et al., 2005). More recently, Berlinger et al. (2008) argue that the next challenge for the HCI community is to find ways to enable non-professional software developers (i.e. initially conceived to be users) to create, modify, and extend software artifacts. All of these arguments suggest that the users may play a developer role in the development process.

### 3.2.3 Broadening the conception of developer

To some extent, developer is a collective term which is used to refer to the people who carry out technical activities that are directly related to the software construction process, e.g. requirements analysis, designing the artifact, programming, testing, debugging, and deployment. However, these activities are often carried out by different people in large software teams due to high division of labour. In these teams, the role of a developer may be further broken down into several roles. For instance, system analysts capture requirements from the users and design the system, programmers implement and debug the

system, and testers evaluate the system. In contrast, a developer in a small development team may be responsible for all these activities at the different stages, or even concurrently, during the systems development process. This suggests that the conception of developer can vary from several specialised roles to a general role. This can be partly due to the chosen paradigm for development, and partly due to the division of labour in the organisation structure.

The term “developer” can be viewed as an umbrella term useful to people outside the software development community. It can refer to system analysts, programmers, testers, etc. However, the diverse activities in the systems development process raise questions such “What characterises a developer?” and “What is the role of the developer in the development process?”

In the traditional conception, the developer is conceived as a constructor whose main responsibility is to carry out technical activities that are at the core of the software construction process, e.g. requirements analysis, designing the artifact, programming, testing, debugging, and deployment. Such a conception is narrow – at worst, the developer is thought as a person who merely converts the given specification of the system into the real system – and it presumes minimal contact, communication and interaction between the developers and the users. Furthermore, this conception emphasizes a reciprocal role relationship between the developer and the user, which may provoke the tension that I discussed earlier (cf. §3.2.1).

To some extent, the narrow conception of developer is derived from the traditional conception of software development. The main issue in this conception is that it does not take other aspects (faces/facets) of the developer into account other than the formal systems development aspect. In practice, developers are not merely responsible for software construction; they may also need to carry out non-technical activities that are related to the construction, which may include learning the user’s domain/workspace (as a learner), playing the role of the user to envisage user’s experience in the future configuration (as a user), teaching and training the users to use and to configure the system (as a teacher and/or consultant). Indeed, these non-technical activities broadened the conception of

developer. As with the broader conception of user, the broader conception of "developer" views the developer as a *participant* who brings in all of his knowledge and his life into the systems development process, rather than merely as a technical guru. To some extent, this broader conception is motivated by the growing trend towards increasing user involvement in systems development and it is highly influenced by the approach that the development team has adopted.

### Developer-as-learner

It is very difficult, if not impossible, to design an artifact with prior knowledge. This is because the design problem is a *wicked problem* (Rittel and Webber, 1973). It is not always possible to be a *reflective practitioner* (Schön, 1983). Developers often have to face new domains and unfamiliar situations in this era of rapid technological change. Indeed, systems development is always situated – there are no two identical problems and no two identical groups of stakeholders. This makes reusing or relating to previous experience difficult.

Quite often, developers will acquire knowledge (or learn) about the users' domain and their workspace throughout the design and development of the artifact when practicing an incremental development approach to systems development. When practising the traditional systems development process, the developers may have to gain substantial knowledge about the user's domain before an appropriate design is developed. No matter which development approach is practised, the systems development process can be viewed as a mutual learning process between the developers and the users which has the learning activity at the centre of its focus (Béguin 2003). In fact, learning occurs among all stakeholders. In the light of the symmetry *of ignorance* perspective (Fischer 1999), this is because stakeholders (developers, users, managers, etc) have different perspectives and expertise such that they learn from each other through the development process (a common space which can be thought of a *boundary object* (Star and Griesemer, 1989).

Besides learning about the users' domain and their workspace, the developer also learns through the construction of the artifacts (cf. §3.1). Furthermore, it is not uncommon that

novice developers have to learn how to use the development tools[51] on-the-fly. This suggests that the learning process (both individual and collective) is intertwined with the development process, and the developer sometimes has to *act* as a learner in the systems development process.

**Developer-as-user**

During the development process, developers may sometimes, arguably, play the role of a user of the developing artifact. This often happens in exploratory systems development (cf. e.g. Yung, 1993), in which the developers may play the role of users and *use* the artifact to envisage the users' experience in order to gain insight for further development. However, unless the developers are developing software for themselves (e.g. in the developers of the IBM Jazz project[52]), it is questionable whether such development is effective, as the role-playing users lack the skills that *real* users possess. Consequently, the *use* scenarios that the developers engage in are closely connected with the objective of learning how the users interact with the system. Such interactions would otherwise be regarded as no more than testing or debugging activities.

### 3.2.4 Reconsidering the conception of developer and the user: a role or a person?

The purpose of the following discussion is to establish a coherent view and interpretation of human-centred design, a topic to be discussed throughout the rest of this thesis.

A few researchers have already argued that the separation of roles between developers and users in systems development should not be binary (Nardi, 1993; Fischer et al., 2005). The spectrum between users and developers should range continuously, from passive consumer to active consumer, to end-user, to normal user, to power user, to domain designer, to meta-

[51] This includes seeking references for API libraries and components off the shelf (COTS).

[52] This is an example of a context in which the developers can play the role of users actively. (cf. the presentation at http://www.eclipsecon.org/2007/index.php?page=sub/&id=4248 on "Jazz in Action - Building Jazz with Jazz" in EclipseCON 2007)

designer[53] (Nardi, 1993; Fischer et al., 2005). The discussion above (i.e. §3.2.2 and §3.2.3) suggests a trend towards expanding the scope of design (cf. Kaptelinin and Nardi, 2006), and a trend towards broadening the conceptions of developer and user. The broadening scope of developer and user has motivated us to reconsider the conception of developers and users in systems development. On the one hand, the terms "developer" and "user" can be conceived as labels for entities in the development process, so that a developer is a person whose main responsibility is to construct the artifact, but (depending on her knowledge, ability, skills and training) is also capable of performing any other activities as a human being. Similarly, a user can be a person whose main responsibility is to use the artifact in her work context, but who is capable of performing any activities as a human being. On the other hand, these concepts can be conceived as roles, so that a person can be a *user* as well as a *developer*, at different times or at the same time. This can be explained by *social role* theory, which suggests that a person can have multiple roles (cf. e.g. Biddle and Thomas, 1966).

Taking the above discussion into consideration, the narrow conceptions that I discussed in section 3.2.2 and 3.2.3 seem to suggest *a single static role perspective*, while the broader conceptions seem to suggest *a person perspective with multiple roles and continuously being broadened with newly discovered roles*. The person perspective is more aligned with social role theory, which suggests that each person may have multiple roles at different times. However, the broader conceptions seem to maintain the divide between developers and users that can be found in the traditional narrow conceptions. The broader conceptions still pay little or no attention to the fact that developers and users are capable of performing other activities, e.g. user-as-developer or developer-as-user, which may be thought of as irrational or irrelevant within the thinking derived from the traditional narrow conceptions (cf. §3.2.1).

[53] Fischer (1999) uses the term 'designer', while I use the term 'developer' to refer to the similar conceptions but to place particular emphasis on the fact that systems development is not merely *design* – it includes both design and implementation, and these two activities are intertwined.

Although many researchers have criticised the narrow conception of user (e.g. Bannon, 1992b; Kuutti, 2001), they have also stated that there is nothing wrong with taking such narrow conceptions. This reflects the fact that research communities still have reservations about adopting a broader conception that is independent from the traditional conception, or an alternative conception that is better suited to human-centred design. This may be because there are implicit interdependencies between the conceptions of developer and user and the development paradigms. To some extent, the conceptions of developer and user are bound up with the development paradigms – development paradigms set the scope of the design process. The conception of the development paradigm influences how the developers and users are involved (this is a collaboration issue) and how the development process is carried out (this is structural issue). If adopting broader conceptions of developer and user entails dispensing with the traditional (i.e. narrower) conceptions entirely, this may violate the underlying assumptions and integrity of the development paradigm. Since the broader conceptions are derived from the narrower conceptions within the traditional conception of systems development, it is hard to conceptualise the dynamic role-shifting phenomena in systems development (cf. §3.3). While the narrow conceptions are restricted to a single static role, the broader conceptions do not account for how and why the roles may be shifted. Indeed, the imbalanced relationship between "developer" and "user" becomes apparent when human-centred aspects are taken into consideration.

Because of all of the issues discussed above, I argue that, at the very least, a new methodology should not be based on the narrow conceptions of developer and user, as this limits our understanding of what users are *really* capable of and blinds us to the possibility that the users may progress towards developer roles as they gain sufficient technical knowledge regarding the artifact in construction. However, the diverse conceptions associated with the same terms may lead to confusion. Kuutti (2001) criticises the fact that the term "user" is adopted for different perspectives at different times, and different focuses in different research areas. This can lead to confusion, as the same terms are being used for different meanings and these meanings are often implicitly associated with different perspectives taken by practitioners and researchers (Alter, 2000). The narrow conceptions of user have not and will not completely disappear, and we do not seem to have a better

term other than *user* even though the term has been confusing (Kuutti, 2001). For this reason, adopting the role perspective (as mentioned above), in which developer and user are roles and a person can hold both roles at different times, seems less problematic.

## 3.3 The role-shifting phenomena[54]

If the developer and the user are roles for a person, it is necessary to conceptualise the possible role-shifting in systems development. In this section, I argue that the role-shifting phenomenon is not uncommon in general group work and in systems development, where human actions (including group work) are highly situated (Suchman, 1987). In these contexts, the shift in roles is enabled by the configuration of the situation (e.g. members of the group having overlapping skill-sets) and is motivated by local needs (e.g. to cope with changing context of work).

From a social role theory perspective, a *role* is "any set of behaviors that has a socially agreed upon function and an accepted code of norms" (Newman and Newman, 2007; also cf. Biddle and Thomas, 1966), and it is filled with expectations of self and others (Newman and Newman, 2007). Put simply, a role can be used to predict the behaviour of an individual who performs the role, and at the same time the behaviour of an individual may reflect the role that she is playing. For instance, an individual who has a *developer* role in systems development is expected and is allowed to build the system. Similarly, an individual who has a *user* role is expected and is allowed to exploit the system for her work. However, it is inappropriate to assume that a person can only play a single static role. Since a person may have multiple roles[55] (Merton, 1968), the notion of role-shifting as I discuss below can be considered as the shift of an individual's primary role.

---

[54] Part of the argument in this section has been presented in the Warwick Postgraduate Colloquium in Computer Science (WPCCS '06), University of Warwick (Chan 2006), and later in a paper (Beynon and Chan 2006, available at http://extra.shu.ac.uk/paperchaste/dpd/participants.html, last accessed on 17 February 2009) for the Distributed Participatory Design workshop that was held in conjunction with NordiCHI 2006 in Norway.

[55] In role theory, there is a subtle difference between what is known as *multiple roles* and *role-set*. According to Merton (1968), *multiple roles* refers to the set of roles that are associated with multiple social statuses that a person is holding, while a *role-set* refers to the "complement of role relationships which persons have by virtue of occupying a particular social status" (p.423). The notion of multiple roles I use here is closer to Merton's notion of role-set.

### 3.3.1 Role-shifting in group work

In order to achieve a common goal effectively and efficiently, it is not unusual that a group activity is broken down into a number of tasks and carried out by different people, i.e. division of labour. It is also not unusual that members of a group shift their roles due to local needs throughout the course of a group activity. For example, consider the event of organising a house party of a small group of friends, e.g. a dozen of people. There is usually a significant amount of work between preparing to hold the party and cleaning up after the party. Although the party may be "managed" by party organisers (i.e. a few individuals of the group), the work is often broken down into many tasks and shared by most, if not all, of the participants. Initially, individuals may have "well-specified" duties, e.g. preparing food, setting up a tent, serving drinks, etc. However, the duties and the roles of individuals may be shifted either temporarily or permanently during the party. For instance, Anthony was socialising and serving drinks to other party participants in another corner of the house initially, but then went to help Mark in the kitchen to make the barbecue skewers because there is shortage of labour in the food preparation "team". Anthony initially knows nothing about how to make skewers and is not aware of hygiene issues in preparing food. However, through observing how Mark is making the skewers and learning-by-doing, Anthony becomes proficient in making the skewers.

In this example, Anthony's role has shifted, from drink server to a food preparer, due to the need of extra labour in food preparation. This role-shifting was enabled by the possibility that he can learn the new skills for food preparation and takes up the new role. Upon reflection, Anthony may be reluctant to shift his role if he does not see the need (i.e. shortage of labour), or if he does not want to move into the new role because he is a bad learner or the new role requires professional skills of which he is not capable. Consequently, one may argue that such role-shifting behaviour (i.e. flexible roles) only occurs in informal activities such as voluntary work, which can be characterised by its softer and decentralised power structure. Furthermore, it is suggested that people conform to the expectation of a given role if the punishment for shifting away from the given role is substantial enough (Milgram 1974). However, one counter-argument is that decentralised power structures (i.e. decision by

consensus) can also be implemented in work. For instance, Constantine (1991; 1993) introduced the concept of structured open team, which can be viewed as the combination of the *closed paradigm* (i.e. traditional hierarchical organisation paradigm) and *open paradigm* (i.e. adaptive collaboration process). The merit of this 'hybrid' model is that it promotes flexible communication and efficient collaboration for group work yet it is within a formal and static hierarchical organisational structure (Constantine, 1993). It is of particular interest that it promotes "rotation of roles to promote flexibility and skill acquisition" (Constantine, 1993). In this sense, it is well-aligned with the kind of role-shifting that I am concerned with in the example of organising a house party. Indeed, the concept of structured open team has been successfully applied in software development projects (e.g. Rettig, 1990; Thomsett, 1990).

### 3.3.2 Role-shifting in systems development

In the context of systems development, it is not uncommon for individuals to have to shift their roles (in relation to the development activities) in different situations and at different times, especially in PD projects when participants are actively involved in the systems development process.

In Friis (1998) paper[56], Friis reports a field study of a systems development project for an organisation across two sites. In Friis's study, all participants were considered as actors – this included researchers, future users of the computer system, and developers (i.e. system analysts and programmers) – and about half of the future users had participated in the development. The development process followed an exploratory prototyping approach, in which the prototype served as complementary to the other requirement specification documentation (cf. Friis, 1988). Further, the development was highly influenced by its future users (to some extent, it was user-driven) as the users sat together with the researchers and built a "logical data base model" that was based on their structured descriptions of the problem, and later implemented by the in-house developers (Friis, 1988).

The virtue of Friis's study lies in the harmonious collaboration between various actors, and

[56] Friis (1988) was published as "Action research on systems development: case study of changing actor roles" in "Computer and Society".

the shifts in their roles, in relation to the systems development activities, throughout the development process. Friis (1998) observed a number of participants who had shifted their roles from time to time. The role-shifting behaviour was evidently reflected from the participants' interaction styles with the system and their behaviour towards the other personnel. To increase the validity of the observation, Friis discussed the role-shifting findings with the participants (Friis, 1988). Interestingly, most of them (except two programmers who showed no interest in this discussion) agreed with Friis's findings regarding their changing roles in the development process (Friis, 1988). Throughout the development, Friis (1998) had detected five user roles: the "traditional user", the "interested user", the "analyzing user", the "designing user", and the "evaluating user". As the development progressed, the users became bolder in their interaction with the computer system (Friis, 1988). Some users became curious about the feasibility of their proposed solutions ("interested users") and some even wanted to participate and to influence the development on the technical side ("analyzing user") (Friis, 1988). Some users wanted to make decisions about what would and would not be computerised ("designing user") (Friis, 1988). Friis (1998) observed that some "designing users" were interested in what is behind the user interface, and argued that these users might be able to build components of the system if appropriate tools were given. Friis (1998) further observed that some users felt that they were the "system owner", and therefore were confident in evaluating, testing, and making modifications to the system in a later stage of development.

Friis noticed that this role-shifting behaviour was not limited to the users, but also occurred among the developers. On the one hand, three system analyst roles were detected: the "traditional analyst", the "collaborating expert", and the "teaching-consultative expert". On the other hand, two programmer roles were detected: the "traditional programmer", and the "teaching programmer". Throughout the development, and through increased interaction with the users, the developers gradually shifted towards the role of teacher and/or consultant (Friis, 1988). Initially, the system analysts had no contact with the users or treated the users as source of information ("traditional analyst"). In the role of "collaborating expert", a system analyst invites users to actively participate in the analysis work (Friis, 1988). As the development progressed, the system analysts had shifted into the role of "teaching-

consultative expert", providing guidance and help to the users during the analysis and, in some cases, teaching the users to use high-level compilers to build the prototype (Friis, 1988). Although the programmers are the ones who actually implement the system with lower-level tools (and are not supposed to interact with the users in the first instance), a similar teacher-student relationship was also observed between the programmers and the users (Friis, 1988).

Friis (1998) considered the role-shifting behaviour as "preliminary" findings because they were unexpected results: the original aim of the study was "to develop a mutual code for better communication between the DP experts and the users for the work with requirement specifications" (ibis, p.30). However, despite its "preliminary" nature, it is clear that participants had transcended their preconceived roles in Friis's study. To some extent, the users in Friis's study had gradually shifted into a role close to what is known as end-user development or end-user programming (Nardi, 1993).

Apart from Friis's study, similar role-shifting behaviour was also detected in other PD projects. For instance, Danielsson (2004) observed a role-shifting behaviour among the participants in the development of a mobile learning environment for students in higher education over a two-year period. According to Danielsson (2004), the project initially followed a user-centred design (UCD) approach (Norman and Draper, 1986). In UCD, the focus is to develop an understanding of the possible interaction between the user and the artifact in the context of use. Hence, the participating students (who were the future users of the system) were initially treated as the source of information. However, researchers observed that the students were unable to envisage the future configurations (Danielsson, 2004). In order to enhance the students' ability to imagine the future environment, the development process was driven into a learner-centred design (LCD) approach (Danielsson, 2004; Soloway et al., 1994), where the development focus shifted to supporting diversity, motivation and growth (Quintana et al., 2001). As the development progressed, the students "could *interact with the designers* and directly see future design possibilities" (Danielsson, 2004, p.51). In a later stage, the project team found themselves using PD techniques, e.g. future workshops.

In Danielsson's study, the shift from user-centred, to learner-centred, to participatory design was strongly in evidence: the development focus was shifted, the scope of concerns was broadened, the conceptual roles of the students were shifted, and the techniques that were used were diversified throughout the project. The shift in the roles of the students were not merely a coincidence, but inevitable (Danielsson, 2004). The role-shifting was in part due to the fact that the students were not familiar with the future configuration – there was no counterpart in their experience to which the students could relate. In order to promote user participation, the focus of the project was shifted and, at the same time, the role of the participants was shifted from a user role to a learner role. In addition, role-shifting may also have been promoted by the situated nature of the systems development – it is unlikely that two systems and their development process would be identical. This makes it hard for the participants to maintain preconceived roles. As Danielsson (2004) noted, the approach that was used in their project cannot be properly reconciled with the description of the perceived approaches, i.e. UCD, LCD, or PD.

The role-shifting phenomenon among the participants is apparent in both Friis's and Danielsson's studies: a number of participants shifted their roles from passive users to active users, from active users to pseudo-developers, and from pseudo-developers to co-developers. From a traditional perspective on roles in relation to systems development, it is tempting to regard such role-shifting behaviour is exceptional, if not abnormal and irrational. However, from a social perspective, an individual requires some form of motivation (i.e. a need for changing role) in order to change her role, in addition to possessing appropriate skills and/or appropriate training (i.e. having the potential for changing role). Where motivation is concerned, it is clear that the users were motivated by their personal interest to the system and the needs for appropriate support to their existing and future work (in Friis's case), and learning (in Danielsson's case). Where potential for role-shifting is concerned, the users were able to obtain help and guidance to learn about the technical possibilities (and even the programming skill in Friis's case).

Another argument might be used to dismiss the role-shifting behaviour. It could be attributed to *powerful intervention* by the researchers as they entered the systems development

project so that the situation might revert to normal when the researchers had left the scene (Friis, 1988). In Friis's study, the situation did revert to normal when the researchers had left the project. As some of the users complained, "You come here and promise a lot, and you even show us the promised land, and how to get there – then you leave and all is back to 'normal'!" (Friis, 1988). Although researchers using participative observation might have made their best effort to avoid interference, it is as hard to guarantee the absence of such interference as to guarantee the absence of interference from the management of the organisation (cf. the political and organisational issues in user participation I discussed in §2.5.2). Hence, it is hard to see how such arguments may be dismissed entirely.

Despite these issues, there is a growing interest in recent HCI research (cf. e.g. Beringer et al., 2008; Fischer and Ostwald, 2002; ODS, 2009) in democratizing the design space for the users to create, modify, or to extend software systems. The rationale behind this trend is that the users are the owners of the problems and computer systems "may be better and more efficiently used if we … let the users solve their own problems, we should not do that for them" (Friis, 1984). In fact, this matter is closely linked to the espousal of user-oriented development, in which the development is driven by the users. Indeed, the high degree of workspace democracy (such as is advocated in PD) might be viewed as a catalyst for the role-shifting behaviour in systems development – as it encourages the users to contribute their diverse perspectives; but the users cannot do so without being deeply involved and engaged in learning about the developer's initial vision and the technical possibilities. At the same time, as the users seek help and guidance from the developers, the developers move to a teacher and consultant position. Eventually, the users shift into a role of pseudo-developer or co-developer. However, there is concern that users may not be able to produce such high quality systems as professional developers as they are not trained for rigorous quality assurance techniques nor able to identify an accurate and complete set of requirements (Friis, 1988; also cf. Davis, 1984). Even worse, the user may not be aware of the consequences of that their design. This could result in what Friis (1998) called *automatization in absurdum* – the worker redesign away his own job through a high degree of automation of the present work process that the worker is supposed to handle.

The role-shifting phenomenon becomes more apparent in the context of free and open-source systems development (FOSS). Nakakoji et al. (2002), based on their case studies in four FOSS projects, observed that some initially passive users may become active developers after a period of engagement and active contribution to the project They argue that the "evolution" of the users is facilitated by two factors:

i) the existence of motivated members who aspire to play roles with larger influence,

ii) the social mechanism of the community that encourages and enables such individual role changes.

Nakakoji et al's observation is consistent with Lave and Wenger's (1991) theory of Legitimate Peripheral Participation (LPP), which suggests that new members of a community of practice will learn the skills from the more senior members of the group (cf. Nakakoji et al., 2002).

To some extent, the interplay between the users' understanding of the developing artifact and the developers' role-shifting is similar to the co-evolution between the developer's understanding and the artifact that I discussed in section 3.1. Despite the concerns raised above, the role-shifting phenomenon is seemingly genuine. Neither should it be viewed as unstable, temporary, or exceptional behaviour of individual participants. Consequently, the question to ask is not "Will role-shifting occur or not", but "How far may we allow role-shifting to go?" and "How do we support it?"

## 3.4 Participatory development

In the previous sections, I argued that the disparity between the influence of the developer and the user over the construction of the artifact in systems development is a consequence of the traditional conceptions of developer and users (cf. §3.1 and §3.2). This disparity may be mitigated by promoting a higher degree of user involvement and workspace democracy in systems development. On the one hand, a higher degree of user involvement enables the users to envisage and experience the systems as early as possible and, therefore, to communicate and negotiate their perspectives with the developers if the system does not

fulfil their needs. On the other hand, a greater degree of workspace democracy allows users to actively influence the design decisions and, therefore, increases the level of satisfaction and the chance of acceptance.

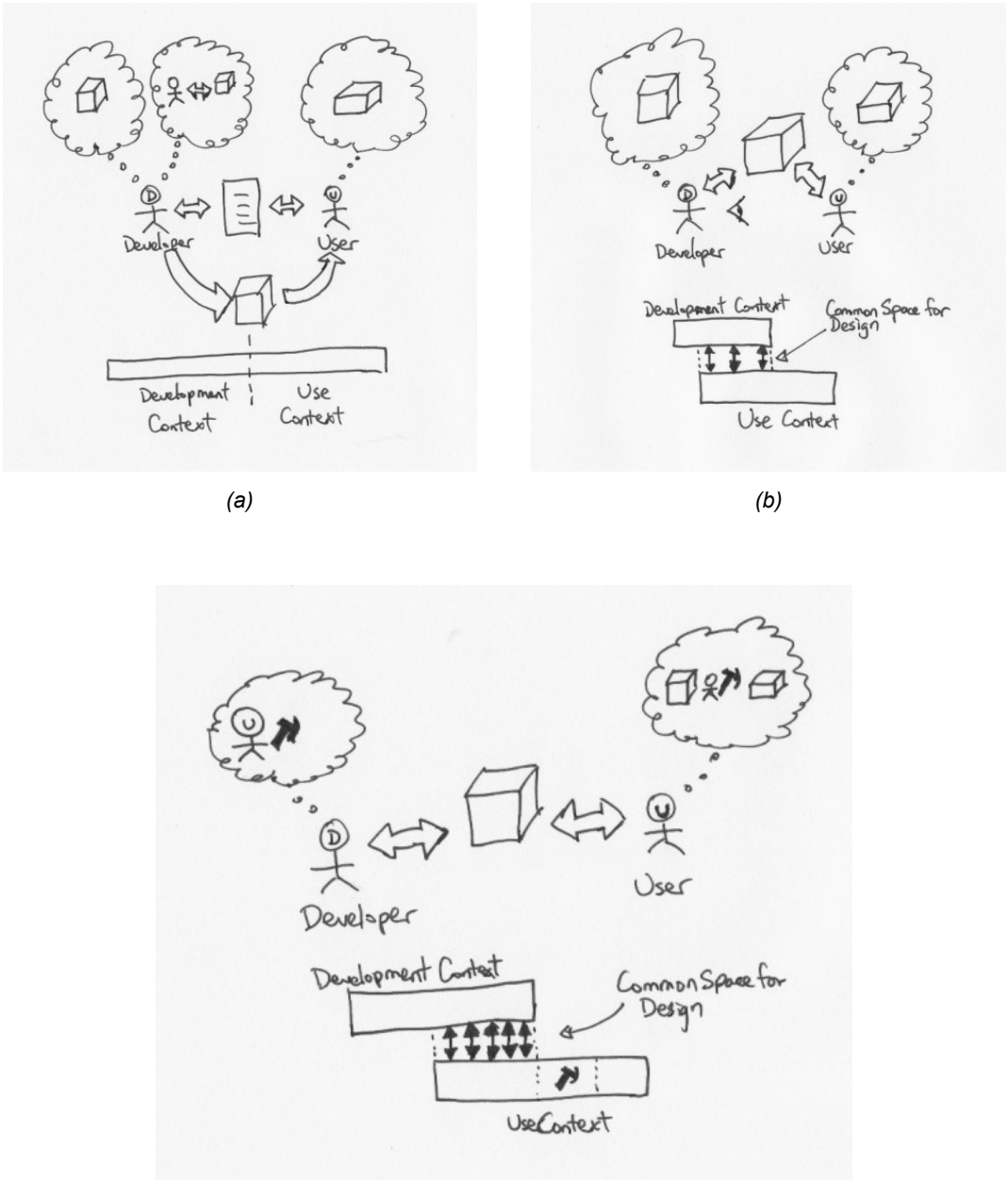
Apart from the practical issues discussed in section 2.5.2, the promotion of a higher degree of user involvement and workspace democracy probes issues in the traditional conception of systems development – the boundaries between contexts of development and use, and the boundaries between the roles of developer and user become blurred. Moreover, as the user involvement and workspace democracy progress towards *genuine participation* (cf. §2.5.2), shifting in roles between developer and user may result. This suggests that the existing conception of user participation may not reflect this conflation well. In fact, the concepts of developer, user, and participation are all interdependent and co-related; when revising the conceptions of developer and user, the conception of participation may have to be reconsidered as well.

In this section, I examine the conception of participation, its relationship with the conflating contexts and roles, and propose a conceptual framework which views the systems development as a co-construction and co-evolution process that incorporates the role-shifting phenomena I discussed in section 3.3.

### 3.4.1 Who is the participant?

Central to the conception of participation is the question "Who is the participant?" The answer to this question is dependent on which perspective is taken. From a *developer-oriented perspective*, it is the case that the user participates in the development process. In this perspective, the developer has the overall control of the process and, therefore, the user may be conceived as entering the developer's space (i.e. user participation). From a *user-oriented perspective*, the user has the overall control of the process and, therefore, it is the developer who participates in the user's world (i.e. developer participation). From a *holistic perspective* (e.g. project management), all stakeholders may be thought as participants and, therefore, all of them are participating in a shared workspace. From a practical perspective, it seems that there is nothing wrong in adopting any of these perspectives. Indeed, a number of techniques and paradigms have been developed for different situations, ranging

from a user-oriented to a developer-oriented perspective and from early to late stages in systems development (cf. Muller, 2001). However, from a conceptual perspective, I argue that both developer-oriented and user-oriented perspectives are biased and they may increase the tension between the stakeholders. Besides, the holistic perspective (i.e. common workspace) is well-aligned with the *symmetry of ignorance* perspective:




*(a)* *(b)*

*(c*

*Figure 3.2 – The spectrum of user participation in systems development*

> *"The 'symmetry of ignorance' requires creating spaces and places that serve as boundary objects where different cultures can meet. Boundary objects serve as externalizations that capture distinct domains of human knowledge, and they have the potential to lead to an increase in socially shared cognition and practice." (Fischer, 1999)*

Indeed, the holistic perspective not only offers better status for all participants, it also offers a more human-centred perspective:

i) It changes the relationship between the developer and the user, i.e. neither the developer nor the user dominates the development process.

It puts the emphasis on the collaboration instead of the separation between participants such as the developer and the user.

### 3.4.2 Participation and the conflation of contexts

The promotion of a higher degree of user involvement and workspace democracy blurs the boundaries between contexts of development and use. Figure 3.2 illustrates the spectrum of user participation in systems development that can be roughly divided into three different regions:

a) The level of user involvement in the development process is minimal. The developers and the users, therefore, mainly communicate through documents. The users may not have the opportunity to experience the artifact until late in the development process. The traditional "waterfall" approach to systems development falls into this category.

b) There is an increased level of user involvement but limited interaction between the developers and the users in the development process. At this level, users are invited to design meetings and are encouraged to express their diverse perspectives. Meanwhile the developers actively observe how the users interact with the artifact. Systems development approaches such as evolutionary prototyping (with or without ethnographical studies) fall into this category.

c) The developers and the users co-design and interact through the artifact, to the extent that face-to-face (or mediated) interaction becomes indispensable. In some cases, the users may construct or modify the artifact, with or without the guidance of the developers (e.g. tailoring and end-user development). This is an ideal scenario for systems development, yet none of the existing approaches fall into this category. However, Participatory Design comes closest – due to its advocacy of equal opportunity for participation and multiple communication channels (cf. Mambrey and Pipek, 1999).

As the degree of user participation increases, the context of development and the context of use are being conflated and create a common space for 'interaction' and 'participation'. In fact, the nature of interaction and participation are quite different in these cases, and they also establish different kinds of tensions between the developers and the users. In scenario (a), interaction and participation is often used to 'define' the boundary of development and the scope of use; thus the tension is stemming from negotiating the boundary. In scenario (b), interaction and participation is for verifying the observation and improving the fitness of the artifact from the developer's perspective; thus, the tension is stemming from interpreting the situation. In scenario (c), interaction and participation is an indispensable part of the whole process, in which the developers and users learn about each other's concerns, develop joint-ownership of the development process, and co-construct the artifact; thus the tension is stemming from the gap between what they can do and what they are allowed to do. It is worth noting that the above partitioning of the user participation in systems development should not be considered as representative of all possibilities. For instance, it is possible to collaborate as in scenario (b) but for the developers to also develop tools or mechanisms for the users to customerise the artifact as in scenario (c). However, it is in scenario (c) that the conflation of the development and the use contexts becomes apparent. On the one hand, the development process (i.e. both design and implementation activities) may continue in the user's workspace, i.e. in the use context in the traditional conception. On the other hand, the artifact may be trialled in the user's workspace, i.e. *using* the artifact in its state-of-development. In this sense, the context of development and the context of user are overlapped and the boundary between them is blurred.

### 3.4.3 Participation and the conflation of roles

As mentioned earlier in section 3.2, within the traditional narrow conception of systems development, developers are conceived as constructors and users are conceived as non-IT professionals who use the artifact constructed by the developers. As the paradigm of participation shifts towards the scenario depicted in figure 3.2c, it is not only the contexts of development and use that become blurred – the roles of developer and user also begin to conflate. This raises a question regarding the conceptions of developer and user – both conceptions break out of the scope of traditional conception, and have violated the implicit assumptions behind the traditional conception.

One way to restore the integrity of these conceptions is to take a social role perspective (cf. §3.2.4). This allows the participants to play multiple roles at different times or at the same time, and to shift their roles throughout the systems development[57]. That is, a participant can have a developer role and a user role. In fact, a person can be a learner, an observer, a consultant, etc. In this sense, the "roles" of a participant at different moments in time become conflated (i.e. *role conflation*). Moreover, as the "role" of the participant is changing from time to time, it becomes unclear which "role" the person is holding at any given moment (i.e. within a short time of observation). However, this conflation has its merits: it does not merely allow participants to envisage the developing artifact from different perspectives, but also allows them to contribute from different perspectives and with different skill sets. Furthermore, this role-shifting phenomenon is not uncommon in systems development (cf. §3.3), and it becomes apparent when the user involvement and workspace democracy progresses towards *genuine participation* (cf. §2.5.2). All these considerations suggest that the conception of participation, developer, and user are all inter-related. By reconsidering the conceptions of developer and user, the conception of participation (as well as the framework for participatory development) may have to be reconsidered.

### 3.4.4 Participation, co-construction, and co-evolution

There is a close relationship between participation, co-construction and co-evolution in

[57] The assumption is that the participant has or will possess the appropriate skill sets.

systems development. One the one hand, a high degree of user involvement and workspace democracy results in a co-construction relationship between participants instead of a reciprocal developer-user relationship. In this sense, participants are co-designing and co-constructing the artifact through interaction with other participants within and outside the artifact itself. This means the users can also construct or modify the artifact, with or without the guidance of the developers. On the other hand, *genuine participation* gives a better experience of learning about the artifact through the co-evolution process during artifact construction (cf. §3.1). In the case of *genuine participation*, this process is no longer proprietary to the developers; it is accessible by all participants, which enables them to gain insight about the artifact and to contribute freely with different perspectives and with different knowledge. This may lead to a more human-centred systems development.

Figure 3.4 illustrates a conceptual model for participatory development which is based on the co-construction scenario (cf. figure 3.2c), and the co-evolution relationship that I discussed in section 3.1. This model takes a social role perspective, which considers "developer" and "user" as roles rather than entities in systems development. It potentially removes the conceptual boundary between "developer" and "user" by incorporating the idea

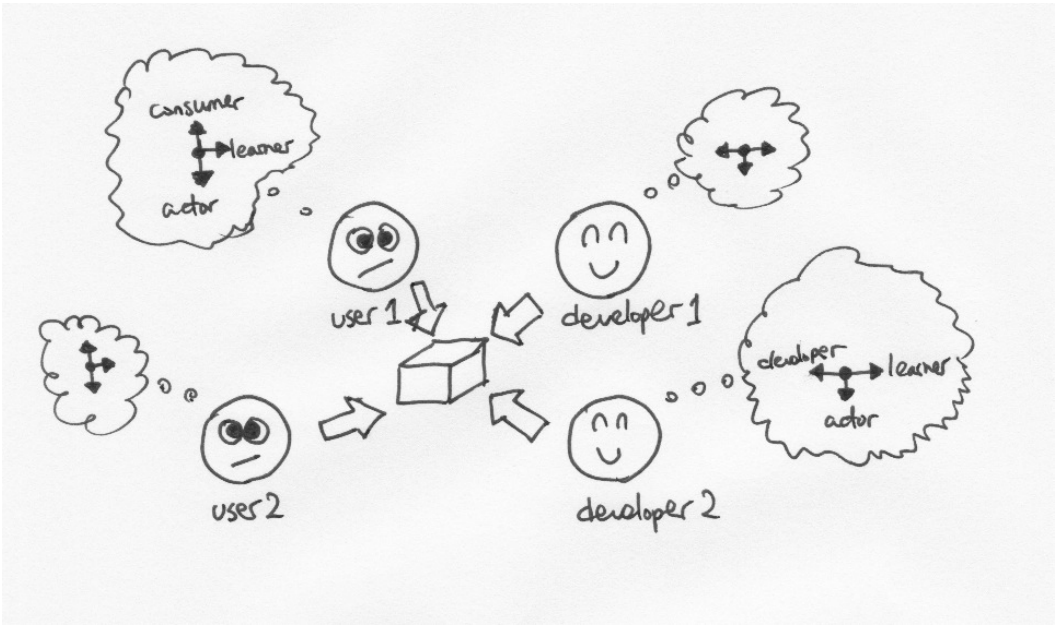



*Figure 3.3 – A model of systems development with broader conceptions of developer and user*

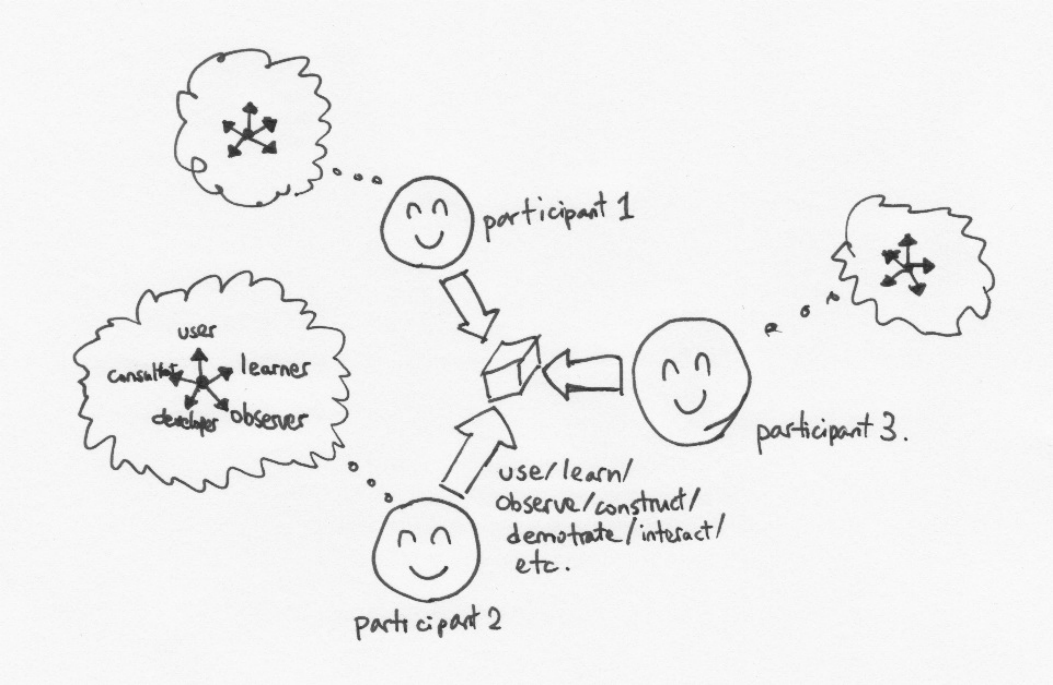



*Figure 3.4 – A conceptual model for participatory development*

of conflated roles that I discussed in the section 3.4.3 – so that a participant may have multiple roles, e.g. user, learner, developer, observer, consultant, etc, at any period of time. The conceptual differences between the participatory development model (cf. figure 3.4) and the systems development model with the broadened traditional conceptions of developers and users (cf. figure 3.3) are:

i) The participatory development model does not maintain a divide between the developer and the user as in the traditional conception. This divide limits our perception of what actions a "developer" and a "user" can perform.

ii) The systems development model with the broadened traditional conceptions is based on a single static role perspective, while the participatory development model is based on a social role perspective (that allows a person to play multiple roles). In the participatory development model, the "role" of a participant depends on the activity and the situation in which she is engaged, and her role may shift from time to time.

iii) There is virtually no limitation on the number of "roles" that a participant can

take in the participatory development model, as they are treated as human beings who can perform any action they are capable of. This is, perhaps, the most significant conflict with the traditional view of systems development.

## 3.5 Implications for groupware development

*"Groupware developers need to be conscious of the potential effects of technology on people, their work and interactions." (Ellis et al., 1991)*

In this chapter, I have examined the conception of developer and user in relation to the degree of user participation and workspace democracy. I argue that the boundary between contexts of development and use and the boundary between the "roles" (in the sense of traditional conceptions) of developer and user is blurred and becomes conflated when the degree of user participation and workspace democracy moves towards *genuine participation*. I have also discussed the co-evolution between the developer's understanding and the artifact, and the role-shifting phenomenon in the context of systems development in detail. Throughout this chapter, I gradually developed the argument that the conceptions of developer, user, and participation in systems development are intertwined. That is, reconceptualising one of these concepts will affect the relationship between them and our conceptual understanding of systems development. While these concepts are topical in the systems development research, our particular interest is in how these concepts affect research into groupware development paradigms. Drawing on the conceptual analysis and the discussion in the earlier sections, I conclude this chapter by identifying seven implications for groupware development:

i) Allow users as active and direct an influence over the artifact construction as the developer has

ii) Support seamless shifting of roles

iii) Support the acquisition and refinement of development related skills

iv) Support diverse perspectives and integration

v) Support genuine participation

vi) Do not limit the user engagement in development to a sample of users

vii) Consider an alternative conceptual framework for participatory development

These implications, as discussed below, can be grouped under four headings, namely, co-evolution (i), role-shifting (ii, iii), participation (iv, v, vi), and conception (vii). I argue that a conceptual framework and *environment* that takes these implications into account is potentially better suited to addressing the human-centred aspects (as discussed in §2.5.1) and leads to efficacious groupware development (as discussed in § 2.5.3).

## 3.5.1 Co-evolution

The way in which the developer's understanding co-evolves with the artifact suggests that there is an essential ingredient of knowledge which is embedded in the artifact itself. The developer's understanding is growing through continuous interaction with and construction of the artifact. The intimate relation between the developer's understanding and the artifact also suggests that only the developer who produces the artifact may have access to this knowledge. This knowledge can be viewed as 'a theory' in the sense of Ryle (1949). I shall refer to this knowledge, which is derived through interaction with the evolving artifact, as *co-evolved understanding*.

The extent to which the developer is privileged over the user through access to co-evolved understanding becomes apparent when the development process affords less user participation and workspace democracy. This is the exact opposite of the high degree of user participation and workspace democracy (compare figure 3.2a with figure 3.2c). This makes the developer the main and the most influential contributor to the development of the artifact, which is thus more developer-oriented.

### Implication I: Allow users as active and direct an influence over the artifact construction as the developer has

In the context of groupware development, this may raise the issues I discussed in section 2.2.2 because the users are not involved in the process. It may resulting systems that are based heavily on what the developers (including the ethnographers) can observe and interpret, which may not *fit* into group requirements (e.g. it may violate the work practice,

disturb the social process, etc). It may also make the users reluctant to adopt and adapt to the system. Therefore, as I argued in section 3.1, it is important for the users to have as active and direct influence over the artifact construction as the developer. Moreover, this allows the users to understand the system and context of the development process better.

### 3.5.2 Role-shifting

In the context of groupware development, role-shifting is actually a two-fold issue. On the one hand, the participants may shift their roles in relation to the development (e.g. developer, user). On the other hand, practitioners (i.e. future users) who are involved in the development process may also shift their roles in relation to their work (i.e. "user" roles) [58]. In section 3.3, I argue that the role-shifting phenomenon is not uncommon in systems development. In fact, the degree of role-shifting is influenced by the extent to which user participation and workspace democracy is implemented. In other words, a higher degree of user participation and workspace democracy invites shifting in development-related roles.

#### Implication II: Support seamless shifting of roles

As many have argued (e.g. Grudin, 1991b; Ellis et al., 1991; Karasti, 2001), a high degree of user participation and workspace democracy increases the likelihood of successful groupware adoption and adaption, and may result in more human-centred groupware. For these reasons, a high degree of user participation and workspace democracy is seemingly indispensable for efficacious groupware development. When the users are highly engaged in the development process (as in Friis's (Friis, 1988) case study), curious users may wish to shift into a more active development role as they become more engaged. Indeed, in this context, participants of systems development shifted their roles seamlessly – they were only aware of the shift in their roles when they were interviewed by the researchers (cf. §3.3.2). Therefore, I argue that groupware development should provide support for participants to shift development-related roles seamlessly whenever a high degree of user participation and

[58] The former issue has been discussed in detail in section 3.3.2, while the latter has been briefly discussed in section 3.3.1. Since the focus of this thesis is on the development process of the groupware, the latter issue is beyond the scope of this thesis.

workspace democracy is preferable.

#### Implication III: Support the acquisition and refinement of development related skills

In order to enable the user to *shift* their roles towards the developer seamlessly, it is necessary for groupware development frameworks to provide adequate support for the user to acquire and refine their development related skills. Furthermore, social role theory suggests that a person may have to learn about the behaviour of the new roles when the person takes the new role and before the person become proficient in the new role (cf. e.g. Biddle and Thomas, 1966; Merton, 1968). In this sense, section 3.3 confirmed that participants of group work in general and of systems development specifically might have to learn the new skills through on-the-job training, as in the party example (cf. §3.3.1) and both Friis's (Friis, 1988) and Danielsson's (Danielsson, 2004) case studies (cf. §3.3.2) respectively.

### 3.5.3 Participation

As a high degree of user participation and workspace democracy is crucial in groupware development, it is more likely the participants will generate more ideas on how the system should be built. This, in turn, may require support for more diversified perspectives and a development process more oriented towards *genuine participation*.

#### Implication IV: Support diverse perspectives and integration

When the imbalance between the degree of co-evolved understanding of the participants is removed (cf. §3.1) and the users are allowed to shift their roles towards "developer", the users will potentially be able to make contributions from different perspectives and different roles. In this sense, all participants may directly influence, construct, or modify the system (as in the case of end-user development). Consequently, the development process will gradually become user-oriented. From a conceptual perspective, this may result in conflated contexts and conflated roles (as I discussed in section 3.4). From a pragmatic perspective, this potentially brings in more diverse and potentially conflicting perspectives. Since the user-developers are now having active and direct influence over the artifact construction, the

degree of diversity in perspectives (as well as in interaction and communication between participants) is potentially much wider than the situation in developer-oriented development. Therefore, proper support for integrating diverse perspectives of participants becomes important.

### Implication V: Support genuine participation

As argued in section 3.2 and 3.4, increased user participation and workspace democracy may lead to *genuine participation*. From a holistic view, the development process is leaning more towards the notion of *genuine participation* when all participants act like developers who can directly influence, to construct, or to modify the system. However, *genuine participation* is not a "silver bullet" for groupware development – not only may *genuine participation* be hard to realise due to subtle social issues and organisational constraints, it can be problematic when the development is *too much* user-oriented, again, due to the very same factors. While these issues may beyond the scope of this thesis, they are unfortunately critical factors for success or failure in groupware development (cf. §2.2.1). In this sense, allowing and limiting the potential for *genuine participation* becomes a dilemma. Therefore, it is necessary to strike a balance between a high degree of user participation and workspace democracy and *genuine participation*. As far as groupware development frameworks are concerned, the important consideration is to allow room for such potential to be realised. This is because it is difficult to judge the degree of user participation and workspace democracy for the development of the groupware for a particular group – it depends on the social, cultural, and organisational characteristics of the group of participants (i.e. users, managers, developers, etc) rather than solely upon technical issues.

### Implication VI: Do not limit the user engagement in development to a sample of users

Despite the fact that user participation has gradually been incorporated into systems development methodologies in response to criticisms, real *user-centred* design is hardly achieved in reality because of the difficulties in getting active user involvement (cf. Bannon, 1992b). Consequently, a reduced degree of user involvement, e.g. representative users, is often employed in practice. This is particularly true when the systems development is

leading to mass selling products (Bannon, 1992b). The problem here is that a "sampling technique" of this nature cannot be used in groupware development. The individuals act differently when no account is being taken of the group dynamics, so such sampling methods are liable to fail if the group is not involved and studied as a single unit. For this reason, what is needed in groupware development is *group participation*, i.e. the participation of the entire future user-group, rather than *user participation*.

### 3.5.4 Conception

Although the traditional narrow conception adopted by user-centred design is, to some extent, based on individualistic cognitive science theories, there is nothing wrong in taking the traditional narrow conception of users and developers (cf. Lamb and Kling, 2003, Kuutti, 2001). However, it becomes problematic when it is applied to a context in which a high degree of user participation and workspace democracy is indispensable, such as groupware development.

#### Implication VII: Consider an alternative conceptual framework for participatory development

As mentioned earlier, the conceptions of developer, user, and participation in systems development are intertwined. Reconceptualising one of these concepts will affect the relationship between them and our conceptual understanding of systems development. Earlier in this chapter, I argue that there is a conceptual divide between the developer and the user in the traditional conceptions. This divide not only affects how we may conceive participation in systems development, but is also an obstacle to conceiving participatory development projects, where roles and contexts can be highly conflated when user engagement and workspace democracy are in place.

In contrast to other branches of systems development, groupware development can be characterised by its particular emphasis on tackling the socio-technical aspects that arise during its development process. Central to this concern is exploring the best possible ways to integrate the human activity and the technological system harmoniously. To transcend the traditional conception and move towards efficacious groupware development (in the sense

that I discussed in section 2.5), user engagement and workspace democracy seem indispensable. Therefore, the concept of developer and user, their contexts, and the "roles" of participants in the development process must of their *nature* be conflated. Moreover, if the participants' roles are not static, the conceptual framework has to support the notion of role-shifting. To this end, it may be worth considering an alternative conception that takes all these considerations into account, such as the participatory development model I described in figure 3.4 (cf. §3.4.4). This is not only to help the practitioners and the researchers to conceptualise the kind of participation and collaboration that is required in the context of groupware development, but also to serve as a guide to the development of socio-technical approaches for efficacious groupware development.

In the next chapter, I will describe the merits of Empirical Modelling (EM) as an alternative conceptual framework for systems development. In chapter 5 and 6, I will argue that EM has the potential for collaborative modelling. In chapter 7, I consider groupware development as an instance of collaborative modelling, and argue that EM may be a potential framework for efficacious groupware development.