The Place of User Enhanceability in User-Oriented Software Development

Nikolay Mehandjiev
University of Hull, UK

Leonardo Bottaci
University of Hull, UK

This paper is concerned with a particular form of end user development known as user enhanceability, in which software applications are produced initially by software professionals, and are later modified and enhanced by end users. This paper concentrates on a software development process and associated modelling languages, appropriate for user enhanceable systems. It argues that it is both feasible and advantageous for user enhanceability languages to be similar to graphical analysis languages, although they cannot be identical. The paper also discusses the role of designers in the context of user enhanceability, and the limitations of the user enhanceability approach.

User Enhanceability and End User Development

Traditional end user development proposals see end users as developers who take full responsibility for creating their application systems. These proposals argue that since end users have expertise in the application domain, the resultant systems will be successful. Thus the communication gap between developers and customers is avoided, and the shortage of IT skills is addressed by asking users to perform some of the IT work.

Developing complex software, however, requires professional discipline (Juliff, 1995) and end user development has been often criticised for:

- producing error-prone software (see the paper by Panko in this issue);
- producing software that is poorly integrated with the rest of the software applications (Cottermann and Kumar, 1989);
- slow development (Rivard and Huff, 1988);
- misusing domain expert’s time on tasks unrelated to their main skills (Avison and Fitzgerald, 1988).

For a comprehensive review of the end user development drawbacks see (Cottermann and Kumar, 1989; Panko, 1996). These drawbacks become more serious as the complexity of the applications increases, and end user development success has been largely limited so far to small, stand-alone applications.

User enhanceability approaches attempt to alleviate these drawbacks and thus allow end users to control bigger applications by restricting end user programming to the maintenance part of the lifecycle. They are based on the premise that it is easier to modify a well constructed existing system than to program one from scratch. Furthermore, at the maintenance stage developers’ work is more mundane, and they often act just as straightforward translators of user requirements into code.

Apart from avoiding some of the scaling up problems of end user development, user enhanceability aims to keep the application “in sync” with the changing organisation. Small modifications can be made rapidly by avoiding the need for slow and error-prone communication between users and maintainers, and by reducing the need for software professionals in short supply.
We have developed a user enhanceability approach, which addresses some of the problems in building user enhanceable systems (Mehandjiev 1997; Mehandjiev and Bottaci, 1996). This paper focuses only on the place of this approach within the software development process, and on the languages used during analysis and user enhancements stages.

User Enhanceable Applications

Clearly, many application domains are suitable for user enhanceable applications, including organisational information systems, management problem solving and manufacturing control. Some business and organisational issues, however, may limit the type of domains where user enhanceability is applicable. For example governmental regulations may forbid changing accounting packages.

Our user enhanceability approach was developed within the domain of organisational information systems, such as workflow systems. Many aspects of organisational work are inherently difficult to capture and formalise because of their social and behavioural nature, and user enhanceability enables the “fine tuning” of the software to be postponed until experience of the practical use of the application system is accumulated.

Some software applications are too simple to require frequent enhancements, for example a simple e-mail tool which provides only basic e-mail functionality will not be affected by organisational changes. Others are too complex for users to enhance. Our proposal targets those applications in the middle that provide comparatively sophisticated functionality, and are thus more vulnerable to changes in the working practices in the organisation. As an example of this vulnerability, consider an e-mail system that provides sophisticated coordination and mail filtering capabilities. This system will have to embody organisational knowledge such as to whom should all the urgent mail be forwarded if a particular recipient is absent. When employees change functions, mail filters will require changing. Organisational knowledge in the e-mail system will have to be updated in line with organisational changes, and this will require changes in the system.

Process Guidelines for User-Enhanceable Systems

The proposed user enhanceability process requires that a first-cut application system is produced by software professionals, and is then enhanced by application managers (Malone, Lai, and Fry, 1992). A pictorial outline of the process is shown in Figure 1. It consists of three phases, analysis, design and maintenance.

Analysis. The development of a user-enhanceable system starts with analysts and customers jointly developing a requirements document in a participatory development manner (Muller, Wildman, and White, 1993). It is important that the requirements identify the areas where enhancements are likely to be required. Note that the group of customers would include end users who will be application managers. They would be actively involved in the development process, making decisions about the application and hence about their future work environment.

Design and implementation. During the design stage software professionals transform the requirements document into a ‘detailed definition of how the system should be implemented’ (Sommerville, 1992). During the implementation