Supporting Appropriate User-Developed Applications: Guidelines for Managers

Laura Lally
Hofstra University

This paper examines User Developed Applications (UDAs) with the goal of providing a framework that managers can use for identifying and supporting appropriate UDAs. Tradeoffs between UDAs and traditionally developed systems throughout the systems development life cycle are identified. Steps that managers can take to: 1) ensure that only appropriate UDAs are developed and 2) improve the logical correctness and usability of the systems in an organizational context are indicated. Finally, suggestions are made for further research aimed at increasing the likelihood of UDA success. Three mini cases are provided as an appendix to illustrate the ideas developed in this analysis.

The increasing power of personal computers and the growing functionality of productivity tools have provided end-users with an increased potential for developing their own software applications. This trend is further supported by continued large organizational investments in end user computing (EUC) (Amoroso & Cheney, 1991), and a blurring of the distinction between EUC and traditional information systems (McLean & Kapplemen, 1993).

The resulting User Developed Applications (UDAs) differ from systems developed by traditional methods in a number of important ways:

1) They are generally developed by an individual solving a job-related problem, rather than by a professional analyst and programmer solving someone else’s problem.
2) They are more likely to be developed using available productivity tools, rather than third generation programming languages.
3) They are more likely to be developed on PCs, rather than on mainframes.

Organizations encourage the development of UDAs by providing PCs, productivity tools, and end-user support. These organizations are motivated by a desire for:

1) a decreased reliance on, or even elimination of, mainframe computers with their high overhead costs and large support staffs, (Korzeniowski, 1991), (Radding, 1989),
2) an elimination of the large backlog of applications in the IS department (McLean, 1979), and
3) An elimination of the communication problems that can arise between developers and users, resulting in systems that are more likely to be appropriate for the user’s tasks (Amoroso & Cheney, 1991), (Grudin, 1991), (Weinberg, 1982).

Although a great deal of money has been spent supporting EUC, research has focused primarily on user satisfaction as an outcome measure of success (Doll & Torkzadeh, 1987), (Amoroso & Cheney, 1991), (Rivard & Huff, 1989) and not on the fate of the systems these users develop. Cotterman and Kumar (1991) emphasize the need to distinguish between EUC support that leads to user satisfaction, and supporting the development of UDAs that are logically correct, and usable.
A few striking success stories about UDAs exist (Meyer & Boone, 1988), (Santeusus, 1992). Many UDAs, however, are never developed to completion (Klepper & Sumner, 1988), are error-prone (Edberg, 1993), and never used for the purpose for which they were developed.

Even UDAs that are successfully adopted by their developers may be duplications of other systems already available in the organization, and therefore, represent a waste of resources. Resources may also have been wasted in using expensive user man-hours (user-developers are often well-paid managers and professionals) to develop a system that could have been developed in fewer hours, and at a lower hourly rate, by in-house analysts and programmers.

This paper, therefore, defines a successful UDA as one which:

1) is appropriate for user development,
2) is logically correct, and
3) is usable for the purpose for which it was designed.

The goal of this paper is to help managers of end-user computing environments and user-developers support successful UDAs by providing insight into the process of UDA system development. The analysis will indicate steps managers can take to minimize the money and time lost as a result of abandoned, inappropriate, or error-plagued systems.

Three mini cases are provided to illustrate the issues raised in this analysis. The cases were developed based upon the experience of three end-users who developed job-related UDAs as part of a graduate level microcomputer projects course. All three UDAs were logically correct when submitted as projects. Follow up interviews were conducted six months later with the developers to determine the fate of these systems and the factors that lead to their abandonment, diffusion, or evolution.

The Emergence of User Developed Applications

In 1971, Weinberg contrasted applications developed by novices for their own use and applications developed by professional analysts and programmers for others. Weinberg assumed that user-developed systems would be relatively simple and stated that developing a system for oneself “conditions the work in a number of ways” (Weinberg, 1971, p. 122):

1) Novices can do their thinking before or after they begin programming. No pre-planning or specifications which must be approved by the user are required.
2) Novices know what is expected of the system.
3) Novices do not need to have changes authorized.
4) Novices are interested in gaining new insight into their problem, not in becoming a better programmer.
5) Novices do not need to create documentation.

What emerged from Weinberg’s analysis was the key issue to be considered in the development of UDAs -- the professional’s superior understanding of the process versus the user’s superior understanding of the problem.

Weinberg also prophetically warned (Weinberg, p. 124) that as software becomes more user-friendly, the gap in methodological skill between novices and professionals will grow. User-friendly software, he argued, is likely to mask the details of the programming task, making novices over-confident in their abilities and increasing the potential for errors.

Since 1971, many productivity tools have been developed which further remove developers from the technical details of programming. Productivity tools may lead as well to greater user over-confidence and the potential for errors.

As the range of productivity tools increased, the systems users can develop for themselves grew larger and more complex. As early as 1983, Rockart and Flannery began to notice the emergence of these larger systems (Rockart & Flannery, 1983). Panko (Panko, 1987), developed a taxonomy of application development environments that distinguish applications by developer and size. Panko distinguishes between the simple applications developed by early user developers such as the ones Weinberg envisioned (Panko’s Second Environment), and systems involving more complex modelling and a much wider range of functionality in the underlying tools (Panko’s Third Environment).

Panko identified a number of key differences between Second and Third Environment systems. First, there was no clear cut indication in terms of size and complexity as to whether or not the development of Third Environment systems should be done by their users or delegated to professionals. Secondly, user developers were more likely to need more help in the early, problem formulation stage of development as the problem grew more complex. Third, user developers were more likely to encounter technical problems developing the software. Finally, Panko suggested a need for greater control over the development of these more expensive, time-consuming systems.

This last concern was also expressed by the EUC managers interviewed by Munro, Huff and Moore (1987). The authors suggested that organizations typically went through two stages in End User Computing: 1) expansion — where users were freely encouraged to develop applications and 2) control — where limitations were placed on tool availability, access to corporate data, and the types of applications users were able to develop.

Alavi and Weiss (1986) also mentioned serious problems with the quality of more complex systems. The authors also expressed concerns about the ownership of UDAs should developers leave the firm.