End-User Software Engineering and Why it Matters
Margaret Burnett, Oregon State University, USA

ABSTRACT

End-user programming has become ubiquitous; so much so that there are more end-user programmers today than there are professional programmers. End-user programming empowers—but to do what? Make bad decisions based on bad programs? Enter software engineering’s focus on quality. Considering software quality is necessary, because there is ample evidence that the programs end users create are filled with expensive errors. In this paper, we consider what happens when we add considerations of software quality to end-user programming environments, going beyond the “create a program” aspect of end-user programming. We describe a philosophy of software engineering for end users, and then survey several projects in this area. A basic premise is that end-user software engineering can only succeed to the extent that it respects that the user probably has little expertise or even interest in software engineering.

Keywords: End-User Development, End-User Programming, End-User Software Engineering, Interactive Problem Solving

INTRODUCTION

It all started with end-user programming.

End-user programming enables end users to create their own programs. Researchers and developers have been working on empowering end users to do this for a number of years, and as a result, today end users create numerous programs.

The “programming environments” used by end users include spreadsheet systems, Web authoring tools, and graphical languages for creating educational simulations (e.g., Burnett, Chekka, & Pandey, 2001; Lieberman, 2001; Little, Lau, Cypher, Lin, Haber, & Kandogan, 2007; Pane, Myers, & Miller, 2002; Repenning & Ioannidou, 2006). Using these systems, end users create programs in forms such as spreadsheets, dynamic Web applications, and educational simulations. Some ways in
which end users create these programs include writing and editing formulas, dragging and dropping objects onto a logical workspace, connecting objects in a diagram, or demonstrating intended logic to the system.

In fact, research based on U.S. Bureau of Census and Bureau of Labor data shows that there are about 3 million professional programmers in the United States—but more than 12 million more people who say they do programming at work, and more than 50 million who use spreadsheets and databases (Scaffidi, Shaw, & Myers, 2005). Figure 1 shows the breakouts. Thus, the number of end-user programmers in the United States falls somewhere between 12 million and 50 million people—several times the number of professional programmers.

Clearly then, end-user programming empowers—it has already empowered millions of end users to create their own software.

Unfortunately, there is a down side: the software they are creating with this new power is riddled with errors. In fact, evidence abounds of the pervasiveness of errors in software end users create (e.g., EUSPRIG, 2009). These errors can have significant impact. For example, one school faced a £30,000 shortfall because values in a budget spreadsheet had not been added up correctly (EUSPRIG, 2009, no. 67). TransAlta Corporation took a $24 million charge to earnings after a bidding error caused it to buy more U.S. power transmission hedging contracts than it bargained for, at higher prices than it wanted to pay, due to a spreadsheet error (French, 2003). Even when the errors in end-user-created software are non-catastrophic, however, their effects can matter. Web applications created by small-business owners to promote their businesses do just the opposite if they contain broken links or pages that display incorrectly, resulting in client frustration, loss of revenue, and/or loss of credibility. Software resources linked by end users to monitor non-safety-critical medical conditions can cause unnecessary pain or discomfort for users who rely on them. Such problems are ubiquitous in two particularly rapidly growing types of software end users develop: software built using open repositories of shared computational resources (such as public databases, program examples, program templates, entire programs, and portions of programs), and dynamic Web applications.

Thus, the problem with end-user programming is that end users’ programs are all too often turning out to be of too low quality for the purposes for which they were created.

A NEW AREA: END-USER SOFTWARE ENGINEERING

A new research area is emerging to address this problem. The area is known as end-user software engineering (Burnett, Cook, & Rothermel, 2004), and it aims to address the problem of end users’ software quality by looking beyond the “create” part of software development to