Beyond Solo End-User Programming: A Scientific Basis for Supporting Reuse

Christopher Scaffidi, School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA

ABSTRACT

Many end-user programming environments are “reuse environments” that store and organize code so users can extend, adapt, and combine existing code. Yet to date, no well-validated, theory-based body of design principles exists for guiding the development of these reuse environments. The contribution of this survey paper is to identify relevant theoretical perspectives and candidate design principles by which these theories could be extended, adapted, and/or applied to the problem of understanding how and when end-user programmers reuse code. In particular, based on this survey, candidate principles are identified that (1) could be used to guide the design of environments for end-user reuse of code, (2) are grounded in theory, and (3) have preliminary empirical support. This contribution is beneficial because successfully applying these principles could increase the benefit of end-user programming by helping software engineers to deliver effective programming environments to end users.

Keywords: Design, End-User Programming, Programming Environments, Reuse, Theory

1. OVERVIEW

Today, online sharing of code has become possible among the millions of end-user programmers who create mashups, scripts, spreadsheets, and other programs to speed up computations, to capture and share knowledge, and to analyze or explore data (Burnett et al., 2004; Good 2011; Ko et al., 2011). The potential benefits of code reuse are clear: users can execute existing code to automate tasks, examine it to learn programming skills, and adapt it to create new programs that serve their own needs. Existing code can also inspire users to create and share even more code, driving a virtuous cycle of enhanced user creativity and productivity. With reuse, code moves from being disposable to being a form of valuable intellectual capital, as users become a participatory community and help to make software systems more relevant, engaging, and useful (Fischer, 2008; Giaccardi & Fischer, 2008). In short, code-reuse should help end-user
programmers meet their own individual needs by drawing upon the resources of their peers.

Unfortunately, these potential benefits are largely untapped. Studies show that end-user programmers still make remarkably little reuse of one another’s code (Bogart et al., 2008; Dahotre et al., 2010; Rosson et al., 2004; Segal, 2004; Stolee et al., 2011; Wiedenbeck, 2005).

These empirical studies leave significant questions unanswered, chief among them, How can we predict whether, when, and how end-user programmers will successfully share and reuse code? In other words, we lack fundamental knowledge about how end users try to reuse code and about how to design effective environments to support code reuse.

Such knowledge about the world and knowledge about how to design for the world are the two key forms of scientific knowledge required for successful engineering (Shaw, 1990; Simon, 1996). Having obtained these forms of knowledge, less experienced engineers “can then apply this knowledge to solve problems far faster than they otherwise could” (Shaw, 1990). Because we lack this knowledge about end-user programmers, software engineers must rely on intuition or empirical studies (e.g., Stolee et al., 2011) when designing “reuse environments” to support end-user code reuse. Refining intuition and conducting studies takes years of iteration and struggle, and along the way, reuse environments can starve for lack of code contributors (Gulley, 2006) or become choked with useless code (Bogart et al., 2008; Stolee et al., 2011).

The contribution of this survey paper is to identify relevant theoretical perspectives and candidate design principles by which these theories could be extended, adapted, and/or applied to the problem of understanding how and when end-user programmers reuse code. This objective is achieved by presenting potential design principles grounded in theory that future research could extend, adapt and/or apply to the creation of end-user code repositories and related tools. Achieving this objective is a valuable contribution because future research could then apply, extend, and empirically validate these candidate design principles.

As a concrete illustration, consider how one design principle might be grounded in theory, applied, and evaluated. Innovation Diffusion Theory predicts that innovations are more likely to be adopted if they have low complexity and high benefits (Rogers, 1995). A design principle based on this theory might stipulate that “reuse environments should provide features to help users find and create code that has low complexity and high functional size.” Such a principle would be evaluated for usefulness by applying it—for example, by creating repository search features that find code based on measures of complexity-of-use and functionality—and then measuring how well the prescribed features help people to find and reuse code.

Section 2 introduces theories that might account for reuse of end-user code. Based on these theories, Section 3 outlines candidate design principles. Section 4 concludes by briefly outlining some future research based on adapting, extending and/or applying these principles the design of end-user code repositories and related systems.

2. THEORETICAL FOUNDATIONS

Engineering can be defined as “the disciplined application of scientific knowledge to resolve conflicting constraints and requirements for problems of immediate, practical significance” (Shaw, 1990). Design principles enable “engineers of ordinary talent” to quickly produce reliable solutions (Shaw, 1990).

Yet the scientific foundation of software engineering is rather uneven (Shaw, 1990). One of the most important, yet least theoretically well-founded areas is the design of environments to support code reuse. Research has only begun to provide theories, expanding on Distributed Cognition Theory (Hollan et al., 2000), to describe how professional programmers interact and, often, reuse code (Herbsleb & Mockus, 2003). System developers are just starting to use these theories to guide the design of programming environments (e.g., Halverson et al., 2006).
An Extension of the Technology Acceptance Model to Determine the Intention to Use Biometric Devices
www.igi-global.com/chapter/extension-technology-acceptance-model-determine/18153?camid=4v1a

A Visual Code Inspection Approach to Reduce Spreadsheet Linking Errors
www.igi-global.com/article/visual-code-inspection-approach-reduce/3756?camid=4v1a