IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITJ2530

Silver Pellets for Improving Software Quality

Evan W. Duggan, University of Alabama, USA

ABSTRACT

In his timeless article, Fred Brooks asserted that the essential difficulties of developing software would continue to ensure the futility of any search for a "silver bullet" to reproduce (for software engineering) the catalytic effects that electronics, transistors, and large-scale integration had on computer hardware development. Since his article, software development has become even more difficult and organizations have magnified the struggle to overcome what has been called "the software crisis." There is unlikely to be a silver bullet, *but this article discusses a variety of user-centered and process-oriented systems delivery methods, philosophies, and techniques available to the software engineering community, that may be used in innovative permutations to tranquilize the dragon of poor software quality. The context for the applicability of these approaches and some advantages and weaknesses where indicated in the research literature or gleaned from practitioner accounts are also discussed.*

Keywords: software engineering; software development; software quality

INTRODUCTION

Despite several years of information technology (IT) innovations, the persistent theme is that "software development is in trouble."[1] Information systems providers have failed to exploit IT capability to establish systems that provide appreciable impact on their sponsoring organizations' value chain (Brynjolfssen, 1993; Gibbs, 1994). Consequently, organizations (and eventually the national economy) seldom derive commensurate benefits from IT investments. This, presumably, has contributed to the perception of a productivity paradox, which was first insinuated by economist Robert Solow.

Brooks (1987) has metaphorically linked the search for solutions to the software crisis to the search for a silver bullet to slay the legendary werewolf, and suggested two major categories of software development problems—essential and accidental difficulties. The former, which are inherent in the process of conceptualizing the several constructs that comprise this largely intangible product, have no known solution. The latter are controllable; they are merely incidental properties of the programmatic representation of these constructs. He concluded that no silver bullet existed to slay this werewolf of essential software difficulties, and none was foreseeable on the distant horizon. In an equally

conclusive article, Cox (1995) offered the apparent minority view that a silver bullet indeed existed for those who will summon the tenacity to shift from the software building paradigms that we have embraced toward engineering-based construction models.

In this article I describe several techniques that have been employed to address the quality concerns of information systems delivery. I recommend an attack on the problem from several sources simultaneously, with relevant combinations of philosophies, methods, and techniques—**silver pellets**—that may assist developers and users chip away at the vital areas of the monster's anatomy. No single silver bullet is presumed; only the disciplined, faithful application of relevant combinations of these approaches will eventually tranquilize the werewolf beyond its capacity to generate terror.

ANALYZING THE PROBLEM FURTHER

None of the four reasons—conformity, changeability, invisibility, and complexity—to which Brooks (1987) attributed the essential difficulties of software engineering, has disappeared. Software continues to "conform" to organizational politics and policy, human institutions, and other systems. Computer systems are still subject to many corrections, adaptations to business process changes, and extensions beyond their original scope. We are yet unable to create visual models to make intangible design products appreciable during construction; software quality is still only experienced through utilization.

In fact, the complexity of software systems development has increased (Al-Mushayt et al., 2001), primarily because of the increased reliance of businesses on IT to support corporate missions and priorities, either for strategic enablement or competitive necessity. This accentuates the challenges that face IS developers and user domain experts to interact more effectively (Vessey & Conger, 1994). Another reason for the increased complexity stems from organizational drives to apply more sophisticated technologies and establish flexible communications networks to accommodate a variety of data and processing distribution strategies, multimedia operations, and integrated systems. Yet another reason is the increased security concern that now attends the greater movement of data. The need for high quality software is glaring.

There is general agreement that the crucial parameters that set the tenor of expectations to determine what value software systems can eventually deliver are scope, quality, timing, and cost (Friedlander, 1992). These interdependent factors, which must be monitored and controlled during systems development, may be traded off against each other, but one cannot be adjusted independently without repercussive effects on the others. The tendency has been to use quality as a surrogate, aggregating construct to capture the value-adding attributes of these parameters.

From the combined definitions of several scholars (Erikkson & McFadden, 1993; Grady, 1993; Hanna, 1995; Hough, 1993), a high-quality information system (IS) is one that reliably produces required features (with a high probability of correct response) that are relatively easy to access and use. It should provide consistently good response times, and is delivered on time so that it retains business relevance beyond deployment. Problems that occur during field use are discoverable with normal effort, and can be easily isolated and corrected. It is scalable both with regard to functionality and usage, and the overall