Software Engineering and HCI

Shawren Singh
University of South Africa, South Africa

Alan Dix
Lancaster University, UK

INTRODUCTION

Technology Affecting CBISs

As computer technology continues to leapfrog forward, CBISs are changing rapidly. These changes are having an enormous impact on the capabilities of organizational systems (Turban, Rainer, & Potter, 2001). The major ICT developments affecting CBISs can be categorized in three groupings: hardware-related, software-related, and hybrid cooperative environments.

Hardware-Related

Hardware consists of everything in the "physical layer" of the CBISs. For example, hardware can include servers, workstations, networks, telecommunication equipment, fiber-optic cables, handheld computers, scanners, digital capture devices, and other technology-based infrastructure (Shelly, Cashman, & Rosenblatt, 2003). Hardware-related developments relate to the ongoing advances in the hardware aspects of CBISs.

Software-Related

Software refers to the programs that control the hardware and produce the desired information or results (Shelly et al., 2003). Software-related developments in CBIS are related to the ongoing advances in the software aspects of computing technology.

Hybrid Cooperative Environments

Hybrid cooperative environments developments are related to the ongoing advance in the hardware and software aspects of computing technology. These technologies create new opportunities on the Web (e.g., multimedia and virtual reality) while others fulfill specific needs on the Web (e.g., electronic commerce (EC) and integrated home computing).

These ICT developments are important components to be considered in the development of CBIS's. As new types of technology are developed, new standards are set for future development. The advent of hand-held computer devices is one such example.

BACKGROUND

A Software Engineering View

In an effort to increase the success rate of information systems implementation, the field of software engineering (SE) has developed many techniques. Despite many software success stories, a considerable amount of software is still being delivered late, over budget, and with residual faults (Schach, 2002).

The field of SE is concerned with the development of software systems using sound engineering principles for both technical and non-technical aspects. Over and above the use of specification, and design and implementation techniques, human factors and software management should also be addressed. Well-engineered software provides the service required by its users. Such software should be produced in a cost-effective way and should be appropriately functional, maintainable, reliable, efficient, and provide a relevant user interface (Pressman, 2000a; Shneiderman, 1992; Whitten, Bentley, & Dittman, 2001).

There are two major development methodologies that are used to develop IS applications: the *traditional systems* development methodology and the *object-oriented* (OO) development approach.

The *traditional systems* approaches have the following phases:

- **Planning:** this involves identifying business value, analysing feasibility, developing a work plan, staffing the project, and controlling and directing the project.
- **Analysis:** this involves information gathering (*requirements gathering*), process modeling and data modeling.
- **Design:** this step is comprised of physical design, architecture design, interface design, database and file design, and program design.
- **Implementation:** this step requires both construction and installation.

There are various *OO* methodologies. Although diverse in approach, most *OO* development methodologies follow a defined system development life cycle. The various phases are intrinsically equivalent for all of the approaches, typically proceeding as follows:

- **OO Analysis Phase** (determining what the product is going to do) and extracting the objects (**requirements gathering**), **OO design phase**, **OO programming phase** (implemented in appropriate OO programming language), **integration phase**, **maintenance phase and retirement** (Schach, 2002).

One phase of the SE life cycle that is common to both the traditional development approach and the OO approach is *requirements gathering*. Requirements' gathering is the process of eliciting the overall requirements of the product from the customer (user). These requirements encompass information and control need, product function and behavior, overall product performance, design and interface constraints, and other special needs. The requirements-gathering phase has the following process: requirements elicitation; requirements analysis and negotiation; requirements specification; system modeling; requirements validation; and requirements management (Pressman, 2000a).

Despite the concerted efforts to develop a successful process for developing software, Schach (2002) identifies the following pitfalls:

- Traditional engineering techniques cannot be successfully applied to software development, causing the software depression (software crisis). Mullet (1999) summarizes the software crisis by noting that software development is seen as a craft rather than an engineering discipline. The approach to education taken by most higher education institutions encourages that "craft" mentality; lack of professionalism within the SE world (e.g., the failure of treating an operating system's crash as seriously as a civil engineer would treat the collapse of a bridge); the high acceptance of fault tolerance by software engineers (e.g., if the operating system crashes; reboot hopefully with minimal damage); the mismatch between hardware and software developments. Hardware and software developments are both taking place at a rapid pace but independently of each other. Both hardware and software developments have a maturation time to be compatible with each other, but by that time everything has changed. The final problem for software engineers is the constant shifting of the goalposts. Customers initially think they want one thing but frequently change their requirements.

Notwithstanding these pitfalls, Pressman (2000b) argues that SE principles *always* work. It is never inappropriate to stress the principles of solid problem solving, good design, thorough testing, control of change, and emphasis on quality.

The Web is an intricate and complex combination of technologies (both hardware and software) that are at different levels of maturity. Engineering Web-based EC software, therefore, has its own unique challenges. In essence, the network becomes a massive computer that provides an almost unlimited software resource that can be accessed by anyone with a modem (Pressman, 2000a). We illustrate these intricacies in Figure 1, which is a representation of a home computer that is attached to the Internet. It depicts the underlying operating system (the base platform), the method of connection to the Internet (dial up, the technology that supports Web activities), browser, an example of a Web communication language (HTML), and additional technology that may be required to be Web active.