Chapter II

Design for Evolution: Fostering Continuity and Responding to Change

Darren Dalcher, Middlesex University, UK

ABSTRACT

This chapter highlights the pervasiveness of change in most developed systems, and the resulting implications on development practice. A key problem with developed software systems is the focus on discrete one-off development. Emphasising continuity and evolution requires a way of distinguishing between systems that can be assumed to remain stable and those that continuously “live” and evolve in their environment. Developing systems that are expected to evolve calls for a dynamic activity of design that is responsive to changes in the environment. The chapter concludes by discussing the role of design in such environments and characterising design as a complex, responsive and negotiated search for an acceptable, yet changeable trade-off. This is taken forward by highlighting the implications for the adoption of adaptive design in dynamic environments, thereby justifying the move from discrete and project-based software engineering to a more strategic, continuous and evolving discipline of adaptive design.

INTRODUCTION: THE PROBLEM OF CHANGE

Software systems are often designed using the assumptions of “fixed” reality and a process that freezes out change and sees little need for allowing alterations. In fact, such a process typically assumes that once the requirements are “objectively” understood and fixed, the rest of the process can be rigorously engineered, or even described through...
a set of (formal) transformational implementations. Proponents of this approach often cite bridges and buildings as proof that formal engineering can be utilised independently from the need to consider change. This enables optimisation with respect to a specific equilibrium point in an effort to maximise potential gains. Maintenance and adaptation occur post-design and construction and thus fail to be considered during these activities. However, change appears to be an inevitable facet of reality and thus a major issue in the context of development for future use.

The chapter focuses on some of the implications of change (and the need to address it) in the context of software development and evolution. It begins by highlighting some of the problems associated with the view of a fixed and change-free reality. Software cannot be designed independently from its environment and will require a capacity to evolve and adapt in order to deal with change in its environment. The rapid rate of change requires a continuous focus capable of dealing with open-ended change and responding to and evolving as a result of new knowledge. The discussion is then re-focused to the context of different software development environments, thereby emphasising the continuous nature of development in most environments. This provides the foundation for developing a framework of software development domains that also highlights the different modes of problem resolution required to populate each of the domains. The interest in the continuous nature of development evolves into a discussion of design as a resolution approach, and more particularly, to the identification of adaptive, dynamic design as the approach that needs to be adopted in evolving and change-intensive environments. The penultimate section highlights the implication of dynamic design in change-intensive and evolving environments by outlining the special characteristics of adaptive design, thereby leading the way to defining the broader context of software development in Domain III. The final section rounds off the discussion by developing the characterisations into a set of implementation issues that enable designers to start taking account of change and foster continuity in an effort to design living and evolutionary information systems that can continue to grow and adapt in response to change.

Traditional software development has attempted to freeze out change. However, empirical evidence presented by Lehman and others (see for example, Lehman, 1985, 1994, 1998, 2000; Rajlich, 2000; Pereira, 2001) demonstrates in general that a program that is used, regardless of the original assumptions, must be continually adapted or otherwise it will become progressively less satisfactory. The implicit assumption that an agreed specification will remain valid and offer a solid basis for all subsequent work is unrealistic, as observed by Lehman (1985, 1998), Wand (1995), and Pereira (2001), as complete and final requirements are seldom available at the start of a project (Cave, 1978; Distaso, 1980; Brooks, 1986; Parnas, 1986; Yourdon, 1993; Schneider, 1998; Pressman, 2000), and are likely to change, with their impacts rippling through all parts of the system (Gibson, 1989; Littlewood, 1992). In reality, software is often forced to adapt to various changes (Land, 1982; Longworth, 1985; Cleland, 1998; Rajlich, 2000), including to the evolving requirements and tastes of users, to new regulations, to the developing capabilities of machines, and to the rising level of expectations. This often calls for constant refinement and gradual modification and reshaping of function in response to context. The need to maintain the fit rather than the initial manifestation of structure forces developers to mature from being artists of shape and structure to artists of time and adaptation (see for example Brand, 1997; Rechtin, 2000).
Enterprise Resource Planning Acceptance Model (ERPAM): Extended TAM for ERP Systems in Operational Phase of ERP Lifecycle
www.igi-global.com/chapter/enterprise-resource-planning-acceptance-model/63453?camid=4v1a