Chapter I
Introduction to This Book

AGILITY AND THE PROBLEM OF CHANGE

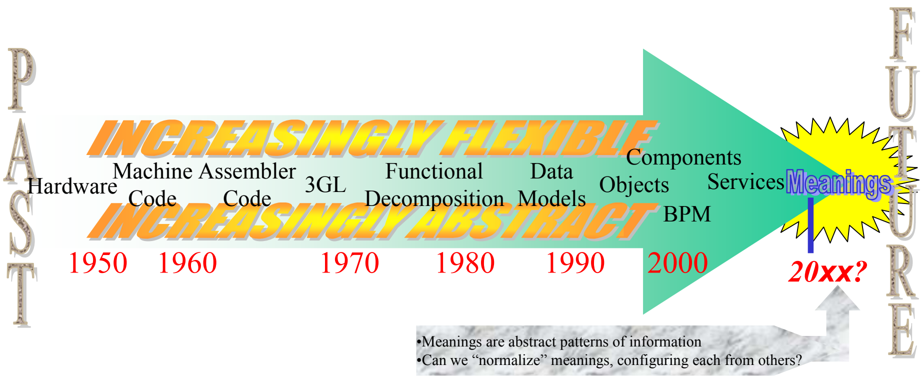
Change is difficult, complex, and risky because it has unintended side effects. Effects of change ricochet through systems via interactions between its parts. The larger the number of components, the more convoluted the system and the greater the chance of unintended side effects of change: more interactions imply a greater risk of multiple, complex impacts of change. Each impact has many consequences, which in turn will have many more until there is a cascading avalanche of changes, interactions, and impacts, which are difficult to manage, foresee, and qualify. This is the problem of change.

The problem of change has persisted through 50 years of automation. Its solution has resisted every technology devised by man. In the beginning, our systems were small, simple, and of limited scope. However, automation opened up new opportunities to improve and integrate processes by coordinating ever larger numbers of elements in previously unanticipated ways. This meant coordination of information across continually broadening horizons, which led to processes that were more dependent on automated systems; further, these processes and systems were more complex, had even more interactions, and were therefore even riskier to change. Paradoxically, it also led to the information economy, which thrives on change and innovation. We have created new technologies and methodologies at a prodigious rate to solve the burgeoning problem of change as our systems have evolved, matured, and integrated over the last 50 years. However, a solution to the problem of change has eluded us because every new approach has been the catalyst for the next level of complexity, which has then required a better, more sophisticated approach to managing change and innovation (Figure 1.1).

Change impacts diverse business processes and cascades through multiple layers of the legacy information systems in a rapidly growing avalanche such that the initiator of the change is faced with the Hobson's choice—either to make the change with huge overheads of cost, time, and risk, or to abandon the potential innovation because of the associated cost, time, and risk factors. The Y2K problem was a classic example. It cost the world around $600 billion (Yuen & Mitchell, n.d.) and exhausted a considerable part of the world's professional resources, just to convert a two-digit representation of the calendar year to four digits[1], which enabled automation to deal with the new millennium.

As systems and processes became more integrated and tightly coupled, it became imperative to isolate and manage the effects of change. The strategy was to encapsulate densely clustered

Figure 1.1. The evolution of information technology



interactions into components, which were coupled loosely with other similar components to produce requisite behaviors and outcomes. These components became the parts of more integrated, more modular systems across larger scopes, which were more maintainable because the impact of change was better managed within modules. This approach required abstraction of information. Each step of the journey in Figure 1.1 not only made business more agile and scalable but also led to higher levels of abstraction. The levels before it did not disappear; rather they hid themselves behind more malleable constructs that became the primary interface between man and machine or machine and machine. This helped the system to become more agile.

As business processes became more tightly coupled with automation, the lack of agility in information systems became a serious bottleneck to product and process innovation. Several frameworks have attempted to solve this problem. Most have failed, or at best, have had very limited arguable success: Structured Programming, Reusable Code Libraries, Relational Databases, Expert Systems, Object Technology, CASE[2] tools, code generators, and CAPE[3] tools, to name a few. They failed because they did not adequately address the ripple effects of change—how business rules and knowledge may be represented so that we may change a rule once and send corresponding changes rippling across all the relevant business processes. To do this, we need ontology, a schema of interrelated meanings, which are derived from each other. Ontology is a study of the meanings of things. It was a philosophical concept that became concrete and computable and, in so doing, took computation into the plane of meanings (see Appendix IV). It is the next advancement in the evolution of automation (see Figure 1.1).

Currently, business rules are replicated in dissimilar formats in multiple, intermingled ways in multiple information systems and business processes. They must all be coordinated when any rule is changed. It makes change and innovation complex, perilous, and problematic to implement. This has been the most critical problem related to change.[4] Purely technical approaches have failed miserably. Despite some claims to the contrary, the problem was not resolved in the 1950s when computer professionals replaced the intertwined programming code of machine language with assembly languages, or in the 1960s when the next generation of these professionals replaced the cumbersome code of assembly languages with that of third generation languages like COBOL and FORTRAN. During the 1970s and 1980s, it was not solved either, when the expert systems, relational databases, and CASE tools

Related Content