The reengineering of legacy systems is widely recognized as one of the most significant challenges to be faced by the software engineering community.

A legacy system is a technically obsolescent component of the infrastructure of a content management environment (Omnibus Lexicon Definition, http://www.fourthwavegroup.com/Publicx/1301w.htm). Legacy systems embody substantial corporate knowledge including requirements, design decisions and business rules. Databases, application programs and all of the other forms of hardware and software typically owned by companies, including mainframes, personal computers, terminals, networks and operating systems, constitute them.

Although the functionality delivered by a legacy system may be available from more modern technology, a migration to newer systems may be deterred by the possibility of service disruption during upgrading, or by the perceived difficulty in converting legacy content to new models and formats.

The need for reengineering legacy systems is thus implicitly contained in the definition given above, and is motivated by the desire to utilize more cost effective hardware or software platforms, to reduce the costs of maintenance (as the Y2K problem has taught) or to add significant new functionalities. The problem is widespread since it effects all kinds of organizations; failing to face it may hamper an organization’s attempts to remain competitive if not threaten its very existence on the market. Finally, it is a problem that may persist over time, as there seems to be no good reason for being confident that systems currently under development will not be tomorrow’s legacy systems.

Among the possible aspects that need to be taken into account while reengineering a legacy system, the software perspective represents the focal issue (and in fact reengineering is often treated as a synonym of software reengineering).

As with many new and evolving fields of research, the scientific community has yet to agree on a common taxonomy of terms with respect to software reengineering. In fact, although in 1992 the Joint Logistic Commanders Computer Resources Management group (JLC/CRM) authorized and sponsored a DoD policy workshop aimed to formally define a software reengineering terminology. As of today, there
is not even an agreement upon the spelling of reengineering (the most common being “re-engineering” and “reengineering”). Therefore, it is important to provide some basic definitions of the domain, in order to gain a common understanding of the terms and the keywords that will be used throughout this volume.

Software Reengineering may be defined as “the examination and the alteration of an existing subject system to reconstitute it in a new form.” On the other hand, it may also be defined as “the process of modifying the internal mechanisms of a system or program or the data structures of a system or program without changing its functionalities.” Whichever definition is adopted, the primary goal of software reengineering is to attain new levels of efficiency of the existing assets, without recurring to the development of new systems. Therefore, an important aspect that needs to be further explored is to define the extent to which reengineering is helpful and to identify some metrics, if any, that can be used to decide whether the option of rebuilding from scratch should be followed.

Regardless, all of the researchers in the field seem to agree on the fact that the process of software reengineering encompasses a combination of sub-processes such as code-reengineering, restructuring, redocumentation, retargeting, reverse and forward engineering. Some of these sub-processes are aimed at program understanding, as for instance restructuring, redocumentation and reverse engineering. The others are geared towards evolution, as for instance, code reengineering, retargeting, forward engineering.

Thus software reengineering may imply, among many other tasks, restructuring “spaghetti-like” code (code-reengineering); transforming the system representation from one form to another at the same relative level of abstraction, while preserving the external functional behavior (restructuring); producing support documentation and reformatting the systems’ source code listings (redocumentation); transporting and hosting or porting the existing system to a new configuration (retargeting); understanding, analyzing and abstracting the system to a new form at a higher abstraction level (reverse engineering), generating new source code from design information captured via previous reverse engineering activities (forward engineering).

These are just some of the topics that will be covered by the research contributions contained in this volume: a useful starting point for anyone interested in getting a deeper insight on software reengineering tools and techniques.

I would like to dedicate this book to my parents.

Sal Valenti
Università di Ancona, Italy,
September 18, 2001