CHAPTER 1

Introduction

This chapter takes a look back at systems analysis and design as a discipline that emerged primarily from another related discipline, computer science. It is argued here that systems analysis and design's origins led to the current orientation in university circles toward in-house automation as an approach to organizational improvement using information systems. This, in turn, led to a corresponding neglect of business process redesign and other alternatives, such as customization of off-the-shelf packages and business process outsourcing. This chapter calls for a change in the practice and teaching of systems analysis and design—a call that this book aims at answering.

Computer Science as a New Discipline

Even though computers have been around since the 1940s, the birth of computer science as an independent discipline dates back only to the 1960s. The foundations of this new discipline came from the related fields of mathematics and electrical engineering. Mathematics was the source of one of the key ideas that led to the development of the computer, which is that data can be represented in a binary way by sequences of zeroes and ones (called bits). Electrical engineering provided the basic technologies necessary to support the development of electronic devices and circuits that would store and operate on bits.

In the years that followed its emergence as an independent discipline, computer science was primarily concerned with the development and teaching of ideas aimed at the solution of early problems associated with large computers, called mainframes. With the increasing use of mainframes in organizations, one of the key problems at the time was the reduction of the cost of using mainframes. One solution was to allow more than one user to exploit a mainframe's computing power and storage resources at the same time. Another was to make it easier for users to provide sequences of instructions to be executed by mainframes. This context provided the impetus for the development of a variety of operating systems and programming languages.

Many subdisciplines have sprung up from the computer science discipline. Those subdisciplines specialized in certain areas such as operating systems, computer programming, computer architectures, artificial intelligence, database design, computer networking, language processing, and data encryption—just to name a few, some with broader scopes than others. The many developments in connection with computer programming are particularly relevant for our understanding of the emergence and current state of the discipline of systems analysis and design.

In the absence of higher-level programming languages, early computer programmers had to write instructions for the execution of computer-based tasks using what is known as "machine language." This fundamentally numeric language can be understood and executed directly by a computer, without the need for any conversion or translation. However, machine language is difficult for humans to use. It consists of sequences of zeroes and ones representing operation codes and computer memory addresses.

Because of the difficulties associated with the use of machine language, assembly language was devised to provide computer programmers with slightly greater convenience. Assembly language enabled computer programmers to express machine language instructions in alphabetic symbols, also called mnemonic codes (e.g., AD, SUB, OR), instead of sequences of bits. Assembly language was easier to use than machine language but still far removed from the way humans normally communicate with each other.

Machine and assembly languages are often referred to, respectively, as first- and second-generation computer languages. The birth of third-generation computer languages begins with FORTRAN (a name that stands for Formula Translation), which many believe to have been invented in 1956. FORTRAN was developed mainly to be used by scientists and mathematicians, and its notation was similar to mathematical notations. This presented some difficulty for it to be used in commercial applications, which in turn led to the development a few years later (in about 1960) of another programming language, known as COBOL (an acronym that stands for Common Business-Oriented Language). Later, yet other programming languages were developed, some more user-friendly than others, including the widely popular BASIC (Beginner's All-Purpose Symbolic Instruction Code), as well as the also popular Pascal and C programming languages.

As the use of programming languages increased, so did the complexity of programming projects. This ushered a new discipline into existence, generally called software engineering. This new discipline was primarily concerned with tools and techniques aimed at improving productivity and quality in large computer programming projects. In the late 1970s, Tom DeMarco, one of the pioneers in the fields of software engineering and systems analysis and design, wrote a seminal book in which he argued in favor of structured methods for software development. The methods proposed by DeMarco included several ideas on how to effectively document the systems development process so that the correction of software errors (a.k.a. bugs) could be accomplished in a timely and cost-effective way.

DeMarco's ideas found a willing audience since it was clear then that most computer systems ended up having many bugs when they were finally delivered to the users. That was particularly true for large computer systems. The correction of those bugs was seen as a part of what became known as maintenance costs of computer systems. Maintenance costs have always been quite high, sometimes as high as 80% or more of all of the computer systems development costs.

In the 1980s, a new set of computer tools emerged to automate key tasks associated with programming projects, which tended to reduce maintenance costs by reducing the need to develop computer code. Those tools were generally referred to as CASE (computer-aided software engineering) tools.

The Origins of Systems Analysis and Design

J. Daniel Couger authored a seminal article in 1973 that discussed the origins of systems analysis and design. The article was titled "Evolution of Business System Analysis Techniques" and was published in the journal *Computing Surveys.* According to Couger, the origins of systems analysis and design date back to the 1900s, when Frederick Taylor developed techniques for business process flow analysis. However, Couger also recognized that Taylor's techniques placed emphasis on the flow of materials through business process, a general trend that persisted until the 1950s, when techniques aimed at the analysis of business processes and related design of computer-based systems to automate those processes began to appear.

Other authors argue that the origins of systems analysis and design are more recent, and rooted in the need to cope with computer system complexity. The argument goes more or less like this. As computer programs have become more complex, so has the realization that advanced specification, careful planning, and the generation of related documentation were "necessary evils" to ensure the success of complex computer systems development projects. And so systems analysis and design was born, at least as we know it today. That is, systems analysis and design emerged from the need to perform certain activities around, and particularly prior to, the steps involved in developing a computer system using software engineering tools and techniques.

Why refer to tasks such as advanced specification, careful planning, and the generation of related documentation as "necessary evils"? The reason is how those activities are likely to be perceived by people who write computer code for a living—something that many of the early software engineers did, day in and day out. From a computer programmer's point of view, those tasks are, well, fairly boring. Many of those who have been full-time computer programmers for a few years view computer programming as a fun, albeit somewhat solitary, activity. It is safe to say that most programmers prefer to start programming right away when a business process–related problem is posed to them. Specification, planning, and documentation are not nearly as fun as computer programming, at least not in the eyes of the people who ultimately develop software.

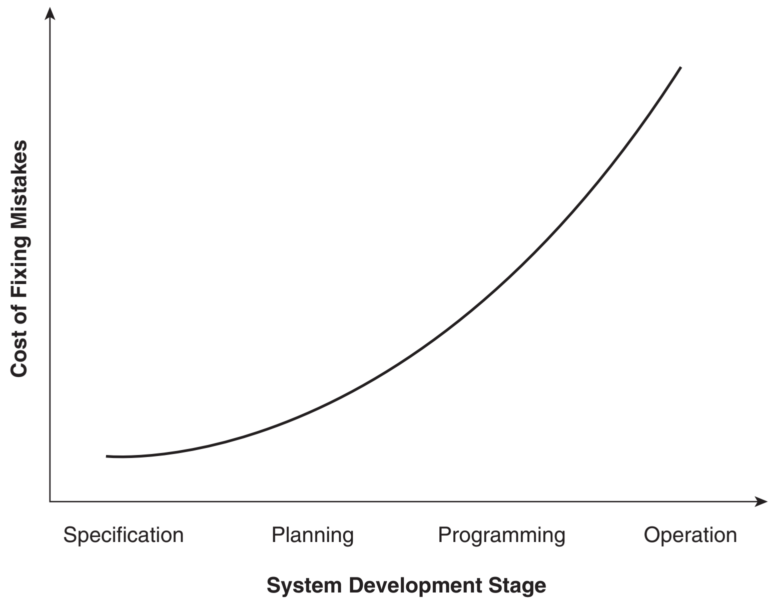

Figure 1.1 Costs of Fixing Bugs at Different System Development Stages

One of the key reasons why preparatory tasks such as specification, planning, and documentation became very important in complex software development projects was the recognition that the cost of fixing mistakes increases sharply as a project moves from preparatory tasks to system implementation and operation (or use). This is illustrated in Figure 1.1, where the relationship between the cost of fixing mistakes and the stage of a computer system development project is shown as being an exponential relationship. Most of the traditional literature on systems analysis and design is consistent with this depiction of the relationship.

Figure 1.1 also suggests that for the majority of systems development projects, particularly the ones where not a lot of emphasis is placed on pre-programming activities, most of the costs tend to be associated with fixing problems in the systems after they are being used. That is, most of the costs tend to be maintenance costs that are incurred after the system is under operation.

The above conclusion is consistent with a notion that has often been called by software developers the "software crisis," which many argue persists to this day. The idea behind the software crisis is that the quality of most of the computer systems developed is way below what it could be if proper precautions were taken. However, it has been difficult to identify exactly what can be done to provide a solution to the crisis. One of those who pointed out the existence of a software crisis in a high-profile manner was Roger Pressman, a software engineering and systems analysis and design pioneer and author of the seminal book *Software Engineering: A Practitioner's Approach.*

The software crisis is characterized by a few key statistics that stand out among others. Only about a third of all system development projects are considered successful by the users of the system. Only about half of all the system functions incorporated during development are utilized by the final users of the system; examples of system functions would be a file encryption option of an accounting system and a charting feature of a sales forecasting system. More than two thirds of all system development costs are maintenance costs; that is, those costs are associated with fixing problems in systems that are already under operation.

The Software Development Life Cycle Model

Early ideas in connection with systems analysis and design led to the development of the waterfall model of systems development, which is also known as the software development life cycle model. It is called the "waterfall" model because its graphical representation reminded its developers of a waterfall. Some may say that this is a little bit of a stretch, but with some effort, one can indeed see some similarity with a waterfall (see Figure 1.2). While sometimes depicted in different ways by different authors, it can be generally seen as comprising the following main steps: requirements definition; analysis; design; specification; database analysis; database design and specification; coding, testing, and implementation; and operation and maintenance. Table 1.1 shows a brief description of each of these steps.

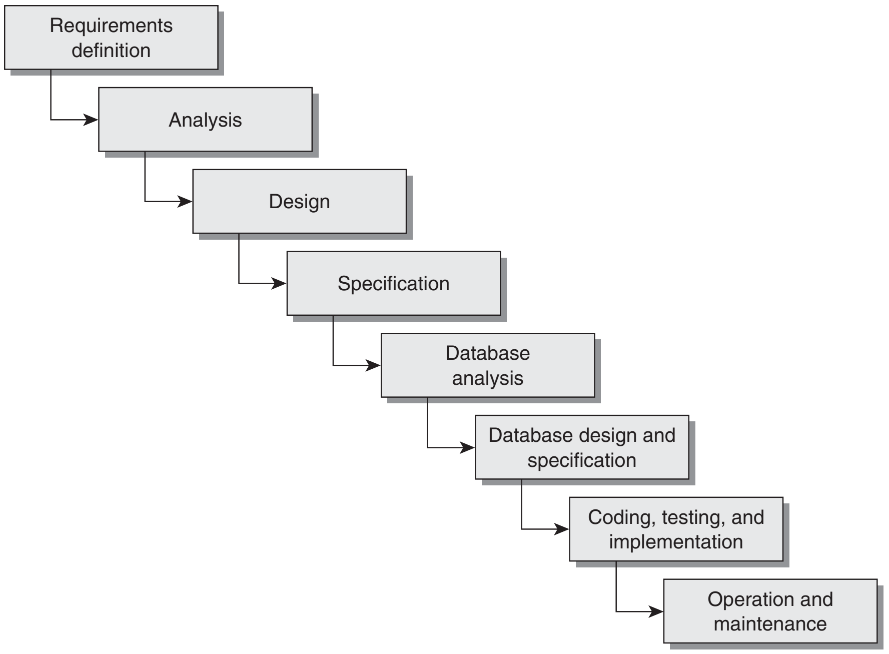
Requirements definition
Analysis
Design
Specification
Database analysis
Database design and specification
Coding, testing, and implementation
Operation and maintenance

Figure 1.2 The Waterfall, or Software Development Life Cycle, Model

Table 1.1 Software Development Life Cycle Steps

Step	*Brief Description*
Requirements definition	Interviews are conducted, and a textual description of a business process and related problems are generated.
Analysis	The business process is graphically represented through data flow diagrams. Data flow diagrams are standardized representations of the flow of data in a business process.
Design	A computer program depiction called a structured chart is developed. Structured charts are standardized representations of the flow of data between computer program modules.
Specification	A high-level specification of the computer program modules, using structured English, is developed. Structured English is a midpoint representation that looks like a computer program, but without all the details that a computer program must have to be understood by a computer program compiler or interpreter (e.g., program lines do not have to end with a specific symbol such as the ";" used to signal the end of a line of code in the C programming language).
Database analysis	Entity-relationship diagrams are developed. Entity-relationship diagrams represent the different structured files in a database and their relationships with each other.
Database design and specification	Database files (also called tables) are designed and specified, which essentially means that the structure of the tables, the names of their fields, and the data types of those fields (e.g., numeric, date) are specified in detail. Also in this step, the data tables are normalized—a task whereby duplication of data in different files of the database is minimized.
Coding, testing, and implementation	The computer program is finally implemented, often using computer-aided software engineering tools and/or computer programming environments.
Operation and maintenance	The computer system is used, and errors in the system are corrected. Also in this step, new features are added to the computer system due to business process–related changes brought about by new market demands.

As it can be seen in Table 1.1, the software development life cycle model of systems analysis and design is aimed at automating business processes. So much so that it starts with the generation of a textual description of a business process and related problems. The automation through a computer program is aimed at solving the problems identified and associated with the business process in question. However, the software development life cycle model does not address the issue of business process redesign—an omission that has led to criticism from systems analysts and information technology managers. The problem with this lack of concern with business process redesign is that often computer systems end up being used to automate disorganized and inefficient business processes, something that could be reasonably avoided if a business process redesign step were incorporated into the software development life cycle. This theme will be picked up later in this chapter and also in other chapters of this book.

The software development life cycle description provided in Table 1.1 is somewhat database oriented. That is, it assumes that the computer program generated will be used primarily to maintain a database. This is why the steps that follow the "specification" step are called "database analysis" and "database design and specification." The reason for this focus on databases is that most commercial systems follow it. On the other hand, this does not mean that all projects conducted according to the software development life cycle will be database oriented. So, there may be variations in the software development life cycle from project to project.

Also, certain steps of the software development life cycle, as described in Table 1.1, are somewhat dated. Good examples are the "design" step, which has traditionally employed structured charts; and the "specification" step, which has traditionally made use of structured English. Structured charts usually show how computer program modules relate to each other. Structured English is a computer program specification that uses English, looks a lot like the Pascal programming language, and is produced as an intermediate step before software coding. Systems analysts have traditionally utilized the design and specification steps to communicate to computer programmers some of the details of a computer system, as far as its code is concerned, without the systems analysts having to actually write any computer code.

The design and specification steps of the software development life cycle lost much of their usefulness to systems analysts and computer programmers as a result of the development of computer-aided software engineering tools. Those tools generate computer programs automatically, and often transparently, based on higher-level representation of the computer system. That is, when using computer-aided software engineering tools, the computer programmer may never even see the underlying computer program, as he or she may design the system by manipulating only high-level elements such as icons and diagrams.

Computer-aided software engineering tools have been evolving over the years, leading to what some refer to as computer-aided *systems* engineering tools. Often, the same acronym, namely CASE, is used for both types of tools. This may lead to some confusion because computer-aided systems engineering tools tend to support substantially more systems analysis and design steps than computer-aided software engineering tools. For example, computer-aided systems engineering tools may incorporate project management features such as visualization and tracking of systems analysis and design tasks through Gantt charts. Such charts allow a project to be depicted as a series of interrelated tasks, which are usually shown on the chart as horizontal bars.

The waterfall, or software development life cycle, model was developed based on early and pioneering ideas in connection with systems analysis and design. Later, an alternative to the software development life cycle model was proposed. That alternative model became generally known as object-oriented analysis and design, which followed in the footsteps of the highly popular notion of object-oriented programming.

Object-oriented programming is a generic term used to refer to computer software development approaches in which computer program modules are developed as part of what are known as software objects. The central component of a software object is its data definition, which usually includes a set of variables. A distinctive characteristic of object-oriented programming is that only special program modules can modify the content of the variables that make up a software object. Those program modules share one special characteristic, which is that they are part of the software object whose variables they can modify. A widely popular object-oriented programming language is C++, which itself is an adaptation for the also popular structured programming language C.

Apparently due to the success of object-oriented programming, the 1990s saw the emergence of object-oriented analysis and design as a widely touted approach for business process analysis, as well as computer systems design and specification. Object-oriented analysis and design (discussed in more detail later in this book) has apparently been presented as an approach that allows computer systems developers to move swiftly from analysis and design to coding using object-oriented programming techniques. And it seems that object-oriented analysis and design has been reasonably triumphant in connection with that arguably narrow goal. However, unlike its object-oriented programming cousin, object-oriented analysis and design has been only modestly successful from a more general perspective.

The Industry–University Gap

Mathematicians, engineers, and computer scientists have been a key force behind the technological achievements from the development of the first modern computer to the emergence of the Internet. And those technological developments have deeply affected organizations, large and small, for profit and not. This has led to the view that organizations need mathematicians, engineers, and computer scientists to run their information technology operations. This view is also reflected in how universities have designed their information technology courses and academic program offerings over the years.

As discussed before in this chapter, systems analysis and design emerged within the context above as an attempt to bring organization and planning to the craft of computer systems development. As such, systems analysis and design has often been seen as a management-oriented discipline inserted into computer science programs. And, since computer science programs have tended to be more technology oriented than business oriented, it has been a natural consequence that the initial emphasis of systems analysis and design has been on the application of computing technologies to automate business processes.

Moreover, the association with computer science programs has led many systems analysis and design instructors, sometimes due to sheer pressure from technophile students interested in learning how to build computer applications, to place strong emphasis on the development of computer applications.

The above state of affairs has led to a number of problems, of which two are particularly relevant for the arguments presented in this book, to some extent because they have been exacerbated by universities and their difficulty adapting to change driven by industry needs.

Among the problems mentioned, the first that comes to mind is that current approaches to systems analysis and design, because of their inherited emphasis on the application of computing technologies to automate business processes, do not address the issue of business process redesign. The second problem is that the strong emphasis on the development of computer applications in current systems analysis and design has led many to be blindsided by what has become a reality in the 1990s and 2000s. That reality is that the vast majority of information technology–enabled projects are implemented through either customization of off-the-shelf computer packages or business process outsourcing.

A good example of customization of off-the-shelf packages (i.e., software packages that can be purchased in a box or as part of an Internet download package) has been the enterprise systems craze of the 1990s. Several Fortune 500 companies purchased and installed large systems (e.g., SAP/R3) to automate many of their activities in an integrated way. In many cases, the activities automated cut across the entire supply chain of the company, going from order taking to product delivery.

Many information technology executives firmly believe in the following motto: If you can buy a software solution, don't even think about building it yourself. There are exceptions to the rule of thumb implied by the motto, which will be discussed later in this book. Nevertheless, it is pretty clear that the proliferation of software development companies in the United States and around the world, particularly since the 1980s, almost guarantees that just about any software need has an off-the-shelf software package developed to address it.

A good example of business process outsourcing is the hiring, again by several Fortune 500 companies, of outside information technology service providers to run help desk operations (e.g., support to office computer applications) for those Fortune 500 companies. These providers of help desk support are sometimes located in different countries and accessible through a toll-free number. Their telephone support, in connection with generic applications such as word processing and spreadsheets, is sometimes better than the support previously provided internally by the companies that outsourced the support.

The advent of the Internet made it increasingly easier to implement business process outsourcing in connection with a variety of support areas. For example, many companies routinely outsource the hosting of their Web sites to other companies, usually known as Internet service providers (or ISPs). Another example of business process outsourcing enabled by the Internet is the farming out by several companies of the software asset management process to specialized service providers, which conduct regular software audits on the companies' application servers through the Internet. Software asset management is the business process whereby companies keep track of their installed software applications and respective licenses. This tracking is aimed at helping them avoid heavy fines in the event that any of their installed software is pirated.

The Need for a New Approach

Systems analysis and design is often seen as a course on how to design computing solutions to business problems. Whether the problems are fictitious or not, this emphasis on the development of computer applications is inconsistent with two key realities. The first is that business process redesign should precede computer-based automation; in fact, in some cases, only business process redesign, without any computer-based automation, leads to significant productivity gains for organizations. The second is that emphasizing computer application development, as possibly the main component of systems analysis and design, creates a narrow perception that is inconsistent with the reality that the vast majority of computer technology–enabled projects are implemented through either customization of off-the-shelf computer packages or business process outsourcing.

This book's solution to the notorious difficulty in teaching systems analysis and design is a simple one. It essentially entails teaching systems analysis and design with an emphasis on business process redesign and with an eye on industry trends and practical needs of organizations.

The above orientation is well aligned with the widespread adoption of enterprise systems since the 1990s. Those systems are often called "enterprise resource planning," or ERP, systems. The reason why the above orientation is well aligned with the widespread adoption of enterprise systems is that it in fact addresses a key problem almost always faced by organizations deploying those systems. Although enterprise systems can lead to major gains in productivity, by integrating information from different organization areas (e.g., sales, production, accounts receivable), they often require that key organizational processes be redesigned before they are deployed. Many examples exist in the business literature of organizations that have spent millions of dollars implementing enterprise systems, only to see those implementations fail due to incompatibilities between the organizational processes and the functionality provided by the enterprise systems.

The organizational revolution brought about by the advent of the Internet also supports the view that systems analysis and design should be taught and practiced with an emphasis on business process redesign. The Internet allows the streamlining of key business processes, such as order and production processes, so as to bring the customer much closer to the organization. It also enables the dramatic reduction of traditional supply-chain costs and order-to-delivery times, as exemplified by companies such as Dell and Amazon.com. However, those improvements can only be realized when traditional business processes are redesigned to take advantage of the new public information-sharing infrastructure provided by the Internet.

Summary and Concluding Remarks

This chapter looks at how systems analysis and design evolved as a discipline. Such evolution is complex and arguably closely intertwined with the evolution of computing. A key argument is put forth, namely that the evolution of systems analysis and design led the discipline to inherit a preference toward business process automation through in-house systems development as a way to improve organizations through the use of information technology. This has arguably led to a corresponding neglect of business process redesign and other alternatives, such as customization of off-the-shelf packages and business process outsourcing. The problem, as more and more organizations are finding out, is that those alternatives are often preferable to business process automation through in-house systems development.

This chapter resorts to some history to justify a key argument for a transformation of systems analysis and design, which is perhaps long overdue. Essentially, the argument is that modern systems analysis and design should be practiced and taught with an emphasis on business process redesign. This chapter calls for a related change in the practice and teaching of systems analysis and design, a change for which this book is presented as an initial answer, though a partial one that needs to be complemented by broader initiatives. Among those broader initiatives are accreditation standards for educational institutions, firms, and individuals selling systems analysis and design training and services—a theme that is picked up later in this book.

The aforementioned discussion is seen as important for the purposes of this book for two reasons. The first reason is that it gives the reader an overview of how systems analysis and design has evolved, which provides the basis on which the reader can understand how systems analysis and design is perceived by many today. The second reason is that the discussion provides the basis on which the reader can understand where the author is coming from, so to speak, which hopefully will clarify from the outset why the book covers the topics that it does, and why it covers those topics in the way that it does.

Review Questions

1. Computer science is not:
   (a) An independent discipline that dates back to the 1960s.
   (b) A discipline that emerged from the need to use assembly language.
   (c) A discipline that is taught in universities.
   (d) A discipline that encompasses several other important subdisciplines.

2. Systems analysis and design has:
   (a) Traditionally placed little emphasis on business process redesign.
   (b) Traditionally been hailed as the most exciting computer-related discipline by students.
   (c) Often been associated with the marketing discipline, particularly in schools of business.
   (d) Originated the discipline of object-oriented programming, through the language C++.

3. It is incorrect to say that the software development life cycle is:
   (a) An approach to systems analysis and design.
   (b) Seen as an abstraction resembling a waterfall-like structure.
   (c) An object-oriented methodology.
   (d) A byproduct of earlier ideas, including software engineering ideas.

4. Which of the following activities is not part of one of the steps in the traditional software development life cycle?
   (a) Generating textual description of a business process and related problems.
   (b) Graphically representing a business process through data flow diagrams.
   (c) Developing a high-level specification of the computer program modules, using structured English.
   (d) Developing a set of object-oriented representations of a business process.

5. It is incorrect to say that:
   (a) Systems analysis and design has traditionally been seen as a software development-oriented discipline.
   (b) Object-oriented programming has been very successful.
   (c) Systems analysis and design has traditionally placed little emphasis on business process redesign.
   (d) The software development life cycle disappeared in the 1990s.

Discussion Questions

1. Let's assume that medical doctors developed systems analysis and design primarily to address their professional needs. What would their version of systems analysis and design look like? Illustrate your answer through the development of a software development scenario where it is clear that the needs addressed are related to the medical profession.

2. As mentioned earlier in this chapter, the software development life cycle usually involves the design and specification of one or more databases. This implies that databases are a fundamental component of most (if not all) computer systems. Is this really the case today? Come up with a business software development scenario that illustrates a possible project in which database design and specification plays a minor role (if any). The scenario developed by you should be as realistic as possible.