Part IV

IT Development and Management

Chapter VII

IT Development

INTRODUCTION

In this chapter trends of IT-driven enterprise development are presented. These trends compete among themselves for supremacy. They do not create a well integrated set of techniques; vice-versa, this set is very eclectic and contains techniques very old and still applicable, like the System Development Life Cycle, and new ones, like Web technologies. These trends are extended into issues of an IT vision for the 21st century, IT skills, and computer controversies that may influence IT developers' awareness about how to pursue IT developmental projects.

IT CENTERS

The IT Centers' concept characterizes the main areas of tasks, knowledge and skills that are necessary to develop, operate, and manage EII. Table 7-1 illustrates each center's methodology, examples of projects, and examples of outcomes.

Table 7-1: The Characteristics of IT Centers

IT Center	Methodology	Projects	Outcomes
PLANNING	Information Engineering, System Engineering	Enterprise-wide Systems Configuration Management	Systems Federations Systems Integration Hardware/Network/Soft -ware Configurations
DEVELOPMENT	Information Engineering, Software Engineering	System Analysis and Design Business Process Reengineering Business Process Integration Workflow Integration B2B Integration e-Market Integration	Objects Applets Components Subsystems Systems
MAINTENANCE	Software Engineering	Code Improvements Legacy Systems Integration	Code List, Subroutines, Objects, Components
INFORMATION DATA	Help Desk Operations Management	Problem Solving Productivity, Security	Problem Solved Data, Information, Knowledge Processing
NETWORK	Topology Planning, Network Administration	Network Services Network Security Network Throughput Mobil Integration	LAN, MAN, WAN, GAN Private or Public

IT DEVELOPMENT CENTER

EII Development Methodologies

To develop an EII one can apply the following methodologies:

- **Information engineering** – to analyze information needs, integrate business and system strategies, and design logic of information systems.

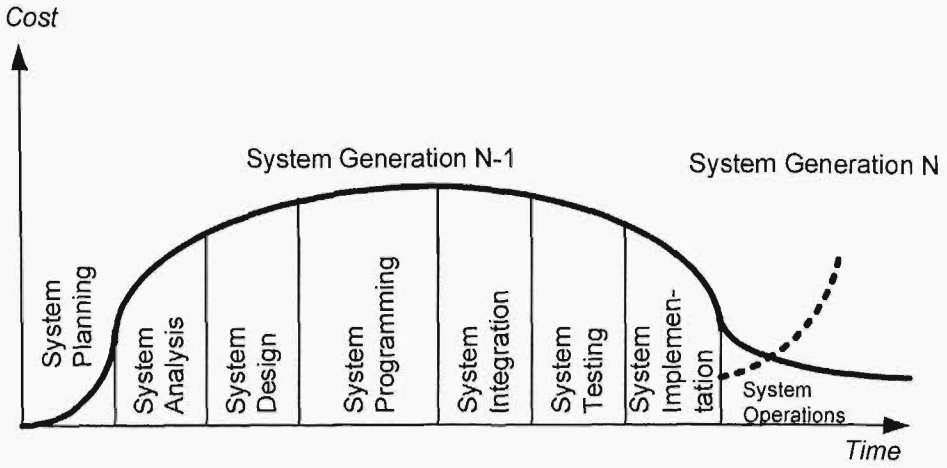
- **System engineering** – to select computer platforms, software packages, and computer/telecommunication networks, and design their configurations.

- **Software engineering** – to buy or design software systems and program their components.

The developmental activities are guided by the System Life Cycle as illustrated in Figure 7-1.

In the EII System Life Cycle, the stage of system construction is replaced by system programming in computer languages or automating editors (CASE-Computer Aided Software Engineering).

Figure 7-1: The EII System Life Cycle



EII Development Strategies

In designing an IS one can apply one of three strategies:

- Bottom-up strategy.
- Top-down strategy.
- Mixed strategy.

The bottom-up strategy used to be the most popular one. It is based on the so-called Application Portfolio Methodology (McFarlan, 1981), which contains only those system projects that are characterized by low risk and useful benefits coming from their implementation. However, this strategy encourages the development of "sure" applications such as "payroll," "inventory control," and "customer orders." This strategy fosters selection of "subsystems" based on the capabilities of individual applications. The application portfolio methodology does not consider holistic issues of Enterprise Information Infrastructure. Therefore this methodology has developed a lot of so-called "legacy systems" that are the subject of reengineering process projects.

The top-down strategy is based on the Federated Systems Methodology (Targowski, 1990), which accepts the premise that the number of information

systems/services and their subsystems in the enterprise is, to a certain degree, finite. Each system has its own generic architecture (see Part II of this book), which defines its major components and establishes their relationships formulated by the list of parts, sub-assemblies, and assemblies (called a Bill of System Processor). System federations are enterprise-wide driven, but systems and subsystems are business process-driven.

The mixed strategy is a combination of bottom-up and top-down strategies. The mix strategy is the most realistic strategy. The top-down strategy is the most idealistic strategy which can be applied in the development of a completely new enterprise environment. Such a situation is rare, because in the 1990's practice, we can always find an existing IS. These may be legacy but they are still functioning. In such a case the reengineering process will improve the legacy system according to the idealistic system, expressed under the form of the top-down strategy.

In the 1970's when the maintenance load at Data Processing departments increased from a modest 20% to 80%, many practitioners and theoreticians began looking for the cause behind this reality. Three different causes were discovered:

1. The linear system life-cycle presented a linear way of developing an IS. It was based on the premise that every next step may add new improvements; however, it was not the practice to return to the previous stage and introduce changes into a specification of a given IS. This caused many errors that later were discovered in the operational stage and required additional maintenance activities.

2. The linear system life-cycle was applied on the premise that a system designer can develop a correct IS for a given business process. In fact, this has never happened since this business process is not static, has changed and as a result, has required more maintenance tasks.

3. The linear system life-cycle was applied to the individual application system, without taking into account the complexity of the existing application portfolio, which had to be adapted or vice versa to a new IS. This adaptation required more and more maintenance tasks.

As a result of these false methodological premises, in the 1980's the backlog in system development reached from two to five years at major DP

departments. As a response to this backlog, several new methodological activities have been established in the IS developmental practice:

- System quality assurance was developed as a corporate set of policies containing standards for the system's life cycle stages.
- Quality control was introduced at the level of the individual IS developmental process, through the evaluation of every major step's internal solution and its influence upon the adjacent stages of the system's life cycle.
- The system prototyping stage was introduced to discover early errors in the system logic, programming or integration.
- Joint Application Development (JAD) methodology was introduced, which put together system designers and system users.
- Application software packages became more popular than the approach of designing a proprietary IS.

As a result of these new approaches, a "user friendly" software concept (menu-driven) was created and the help function was expanded, including the application of artificial intelligence that can create so-called "wizards" in customizing the user's needs.

STRATEGIC USE OF IT

In the 1960's and 1970's during the early stages of applying information systems, IT was treated as a means of automating clerical routines and reducing employment of clerks. In the 1980's with the emergence of microcomputers and its quiet, creeping revolution, information systems emerged from back offices to front offices and began to affect the competitive positions of businesses.

This type of information system was named strategic information systems (SIS). They differed from the internal organizational IS since they mostly were interfacing a given business with its customers, suppliers, and distributors. In other words, these IS are of an inter-organizational nature.

Parson (1987) identifies (based on Porter's 1980 theory) and this author (A. Targowski) exemplifies the impact of SIS on the business environment at the following levels:

1. The industry level:
   - A product/service can be enhanced by improving its functionality (embedded chips that make a product smart) and as a result, a given firm has a new competitive advantage among its competitors.
   - A product's life-cycle can be shortened by the application of CAD and CAM/CAP (Process) systems and new innovations can be implemented sooner, giving a firm a competitive advantage.
   - The speed of distribution can be accelerated by the system integration of a producer with a distributor through the Extranet, providing a firm with a new competitive advantage (for example, an electronic bookstore *www.amazon.com*).

2. The firm's inter-organizational level:
   - Buyers using SIS can influence a supplier's selection and their prices. An example is a system of searching electronic catalogs on the Internet.
   - Suppliers can reduce prices and increase customer satisfaction by the application of an agile factory, which supports mass customization.
   - New entrants using SIS, particularly in the area of e-commerce, can reduce entry barriers and limit entry deterrence.
   - Substitution of products and services represents opportunities for a firm or a consumer. An Internet service provider substitutes e-mail for snail mail or an e-retailer for a traditional retailer.
   - Rivalry – a firm having access to its own or commercial databases and knowledge bases may establish effective links within the industry and gain a better competitive position among the industry rivalries.

3. The firm's strategy level:
   - SIS can be applied to reduce the overall cost so a firm can gain in cost leadership within an industry. Such is the case of the Japanese auto companies that in the 1980's applied robots, just-in-time inventories, and strong quality controls and were selling cars at very competitive prices.

- SIS can be applied to achieve a firm's product/service differentiation among the industry's producers. For example, the electronic Amazon bookstore provides excellent customer service and traces customers' behavior (knowledge management SIS), so it can offer additional book choices when the customer orders a given book. Due to this service, the customer returns to Amazon to buy more books.

- SIS can be applied to focus on a particular market/product niche. For example, Microsoft gained its very high competitive advantage in the software industry because it specialized in a desktop operating system.

The above mentioned examples of SIS can provide a strategic position for a firm as long as its competitors do not implement similar systems.

In the 1990's the strategic perspective on information systems' role within enterprise operations recognized three trends:

1. The development of a singular SIS as it has been exemplified above.

2. The development of the strategic Enterprise Information Infrastructure architecture.

3. The development of strategic system-driven inter-organizational information systems and services.

The development of the strategic Enterprise Information Infrastructure architecture is based upon choice among media-oriented enterprise

Table 7-2: The Choice of a Mediated Enterprise Type

	Local	*National*	*International*	*Multi-domestic*	*Global*
Off-line	Yes				
On-line	Yes	Yes	Yes	Yes	
Integrated	Yes	Yes	Yes		Yes
Agile	Yes	Yes	Yes		
Informated	Yes	Yes	Yes	Yes	Yes
Communi-cated	Yes	Yes	Yes	Yes	Yes
Mobile	Yes	Yes	Yes	Yes	Yes
Electronic	Yes	Yes	Yes	Yes	Yes
Virtual	Yes	Yes	Yes	Yes	Yes

types. Table 7-2 identifies such choices for a given geography-oriented enterprise.

Once a mediated enterprise type has been chosen, its type characterizes the kind of an enterprise configuration, as it has been presented in Chapter 2.

DEVELOPMENT OF SUBROUTINE, OBJECT AND COMPONENT

Computer programming was first introduced by Ada Lowelace, a collaborator of Charles Babbage, who both developed the first programmable "analytical engine" in 1832. For the last 150 years programming techniques were aimed at the development of convenient instruction sets.

In the 1970's the *structured programming* technique was introduced to minimize the negative effects of "spaghetti" like program codes full of "go to" instructions that call for subroutines.

In the 1990's the *object programming* technique was introduced to make self-dependent code modules=objects more reusable. The term object refers to people, places, things, or transactions about which data is maintained. For example, CUSTOMER, STUDENT, and TAXPAYER are all examples of people objects; similarly, BUILDING, INVOICE, and REGISTRATION are examples of place objects, and transactions objects, respectively. New software solutions are collections of objects that incorporate both data structure and behavior which contains instructions for operating on data (*hire, fire, pay-dividend, open, close, change-job, change-address, close, hide, redisplay,* etc.). This is in contrast to conventional programming in which data structure and behavior are only loosely connected. Most object-oriented software solutions were developed with object-oriented programming languages such as Smalltalk, C++, Object COBOL, Visual Basic, and Java.

In the 2000's now emerges the *component programming* technique which relies on some object-oriented techniques, since a component can be perceived as a set of objects. However, a component is designed to be reused and customized without access or modification of the component's source code, unlike objects, which often come in the form of a class library and which are meant to be customized by subclassing the source code. On the other hand, components have varying granularity and can consist of only one class, a composite of many classes, or an entire application.

Most components cannot process code by themselves, but require a module called a "container," which provides an application context for one or

more components and which also secures control services for the components. The container operates as an operating system, which executes the code within the component. The first commercial component system MS COM was created by Microsoft, which incorporated it into its Windows as Object Linking and Embedding (OLE). OLE was designed to make it possible to embed modules from one program into another. For example, when a user of Word clicks on the Excel icon in the menu bar, this icon has the embedded container controlling access to a spreadsheet program. Object applications are the most successful in such cases as is shown in this example. In typical software programming, the reuse of objects is rather rare and in practice, this academic approach did not deliver solutions as was expected. The industrialized object programming led to the emergence of component programming, where objects became less universal and more application-centric.

Components are based on the premise that users can wrap a code module and create an interface (for example in a GUI style) that will respond to their messages (commands). Users are not involved in the manipulations of the internal code of a component. Just the interface intercepts commands directed to the component (in our example to Excel) and then does whatever is necessary to trigger that component's operations. In the object programming, objects must be programmed in the same language (e.g., C++ or Java) in order to interact between themselves. In component programming, two components can be programmed in different languages, since their interaction is secured by the container (interface). However, to ensure a smooth environment for different components' interaction, they should be developed according to standards of CORBA (Common Object Request Broker Architecture) with the Internet Inter ORB Protocol (IIOP) or COM/DNA (Microsoft's Component Object Model - Distributed InterNetwork Application Architecture), or in J2EE (Sun's Java 2 Enterprise Edition architecture, also called Enterprise Java Beans – EJB, where a bean is a kind of self-describing object). Sun's EJB was designed to interact with MS' COM/DNA; on the other hand EJB interacts with CORBA components, thus in such a manner all standardized components can interact among themselves.

MIDDLEWARE-DRIVEN INTERFACING

Middleware systems link a variety of different applications or their components from different computer/software platforms. Middleware keeps track of the locations of the software modules that need to link to each other,

and thus manages the actual exchange of information. Because of that, the location of modules is transparent for users. Each linked component must have a standardized interface.

The major types of middleware are as follows:

- **Database middleware (DBM)** – translates SQL requests from applications into the native tongue of the target database;
- **Transaction processing middleware (TPM)** – facilitates updating of multiple databases by a transaction-oriented application;
- **Remote Procedure Call (RPC) middleware** – allows an application executed on one computer platform to call a procedure and/or send data to an application running on another computer platform;
- **Message-oriented middleware (MOM)** – lets applications on different computing platforms (mostly in the client/server configuration) and networks exchange data reliably and securely. Messages are sent and received through an independent layer in the asynchronous mode, which is backed by the buffer capacity;
- **Distributed component middleware (DCM)** – moves messages between components of applications and provides services in security and transaction processing, applying RPC and MOM styles;
- **Distributed object middleware (DOM)** – moves objects of applications between applications and provides services in security and transaction processing, applying RPC and MOM styles;
- **Application server middleware (ASM)** – builds application servers in the n-tier server architecture (client-application-database-directory services-Internet-security) which provides such services as: transactions processing, persistence and data biding (data integrity in multi-user environment), security support, directories, and load balancing as well as the inclusion of all the above middleware types.

Middleware is primarily applied by companies implementing their own application integration. Those companies that buy complete software or outsource their information infrastructure used to ignore middleware applica-

tions. Now, with the emergence of object and component-based solutions, almost every computerized organization will sooner or later be using client software which has incorporated object/component solutions that must be linked with other object/components, even with those which are outsourced.

EAI – ENTERPRISE APPLICATIONS INTEGRATION

Integrating information across the enterprise should be at the top of a Chief Information Officer's (CIO) agenda. The information integration goal should be the linkage among business units, applications, data architectures, computer platforms, network topologies, websites, suppliers, customers, etc. An integration strategy can send a company one step forward or two steps backward.

EAI focuses on solving the integration of multiple applications that were independently developed within the enterprise, may use incompatible information technology, and may remain independently managed. For example, SAP applications can be integrated with Peoplesoft applications that are used in different locations, including different countries. EAI aims at the linkage between different application semantics in order to move information seamlessly between systems in short time frames. For instance, within a short transaction, information exchange takes place to support a discrete event, such as the addition of a customer in one application while automatically updating another.

EAI requires layers of data transformation, metadata administration, software adapters and connectors, network connectivity, and integration administration. The EAI integration scenario is about how information is updated between sources and targets within a given organization.

In an average corporation there are about 50 applications developed internally, installed by ERP software or delivered by the third-party, that should be integrated. Major ERP vendors publish application programming interface (API) to enable connectivity with the third-party applications.

The integration between two applications typically occurs at several levels concurrently. Table 7-3 illustrates seven different levels of integration; each level based on services provided by the lower levels.

In the industrial practice of applying EAI one can recognize four levels of possible solutions delivery: