NORTH-HOLLAND

Application of an Information Technology Model to Software Engineering Environments

Marvin Zelkowitz* and Barbara Cuthill

Computer Systems Laboratory, Nat'l Institute of Standards and Technology, Gaithersburg, Maryland

The Information Technology Engineering and Measurement (ITEM) Model has been developed to describe the information processing activities of an enterprise, both the automated tasks performed by computer and the manual processes performed by the information technology staff of an organization. In this article, the ITEM model is applied to the description of software engineering environments as extensions to two previously developed reference models, the NIST/ECMA framework (i.e., "toaster") model and the Project Support Environment reference model. The ability to measure the degree of automation within a process, the ability to define the complexity of a process, and the ability to measure technology transition via a concept called technological drift are all metrics that can evolve from this model.

1. INTRODUCTION

The ability to accurately model information technology use within an organization is taking on increased importance as computers take on increasingly central roles in an organization. We describe one model that can be used to model this technology and which provides the basis for use in roles such as requirements analysis for software development, quantitative evaluation of computer technology use, and traceability of software requirements and designs.

The concept of an environment within the software engineering field has grown to mean an infrastructure set of services (e.g., a file system) and a set of end user services (e.g., the set of tools to be used such as editors, compilers). Models like the NIST/ECMA frameworks model (i.e., the "toaster" model) (NIST, 1993a) provide a classification scheme for identifying the set of infrastructure services provided by the environment, and models like the Project Support Environment (PSE) model (Brown et al., 1993) provide for the necessary set of end user services.

Approaching this problem from the business engineering perspective, models like the Corporate Information Management (CIM) interface architecture (CIM, 1993) provide for the static description of an enterprise, from the highest level of industry standards down to the set of tools and files executing on a given desktop. With this model, functional areas (e.g., software development, accounting, weather forecasting) evolve into a series of applications (e.g., software design) that is implemented using the tools of the underlying platform.

Both of these approaches, the infrastructure-end user services of the PSE model and the seven layer CIM model, are somewhat static. End user services are, by definition, executable on an appropriate computer system. On the other hand, in the CIM model, it is assumed that the functional areas are implemented by applications at the next level. If a new tool is developed (e.g., a tool that does true software design), then software design, in the CIM model, moves from the functional level to become just an application.

This, however, becomes untractable. We would like to apply a given model to several enterprises in a consistent manner. If software design is a functional area in one organization, then it is a functional area in another, although the means to provide for that design may be different. In effect, the degree of automation of a given functional area is an attribute that undoubtedly will change from organization to organization.

Address correspondence to Dr. Marvin Zelkowitz, NIST NORTH, Room 517, Gaithersburg, MD 20899.

*Dr. Zelkowitz is also affiliated with the Computer Science Department at University of Maryland, College Park, MD 20742.

J. SYSTEMS SOFTWARE 1997; 37:27–40

655 Avenue of the Americas, New York, NY 10010

0164-1212/97/$17.00
PII S0164-1212(96)00045-3

The degree of automation is behind much of the current interest in software process engineering. Developing a software process provides for the steps which must be undertaken to complete an activity. This process could simply be the execution of a single program (e.g., compile Ada source file) or could involve a complex series of actions involving both programmer and computer (e.g., testing software by compiling, executing tests, checking output, and repairing errors if found). At each level of the enterprise, it is important that we be able to identify the process used to implement actions at that level, measure the degree that the process is automated, and to identify the interrelationships among the various components of the enterprise.

It is towards this goal that we have developed our Information Technology Engineering and Measurement (ITEM) model. We would like a single notation useful to model the actions of an enterprise, model the process of software development, and to understand the role of process engineering within this domain.

1.1. Overview of the Model

The ITEM model is an enterprise-wide service-based model that builds upon existing models of information technology use in organizations. *Enterprise-wide* means the major activities that drive the information management decisions of a large organization, such as a corporation or government agency. Use of such models is important to understand information technology use within an organization. "An enterprise [-wide] model is the essential ingredient of any architectural approach. This model shows both the data needed by the entire organization and the processes which manipulate that data" (Work and Balmforth, 1993). The ITEM model describes the enterprise's use of automation and processes as a sequence of layers. Each layer represents an abstract view of the enterprise's behavior at a defined level of detail.

Much of the structure is a merger of existing concepts mentioned earlier.

1. The CIM interface architecture
2. The NIST/ECMA frameworks model
3. The PSE model for end user services
4. The POSIX.0 open system environment reference model (IEEE, 1993) which provides a model of communication and interaction among environment components.

All of these are layered service-based models describing the functionality available at each level in the model. We add to this the concept of software process, such as the Quality Improvement Paradigm (QIP) model of Basili (Basili and Rombach, 1988), which places the product and measurement as primary drivers of an organization.

Organizations do not exist in a vacuum. We append to the model four stimuli to describe influences on an organization's information use (Figure 1).

- *Market forces* are external stimuli which affect the organization. These include consumer demand, resource availability, or government regulation.

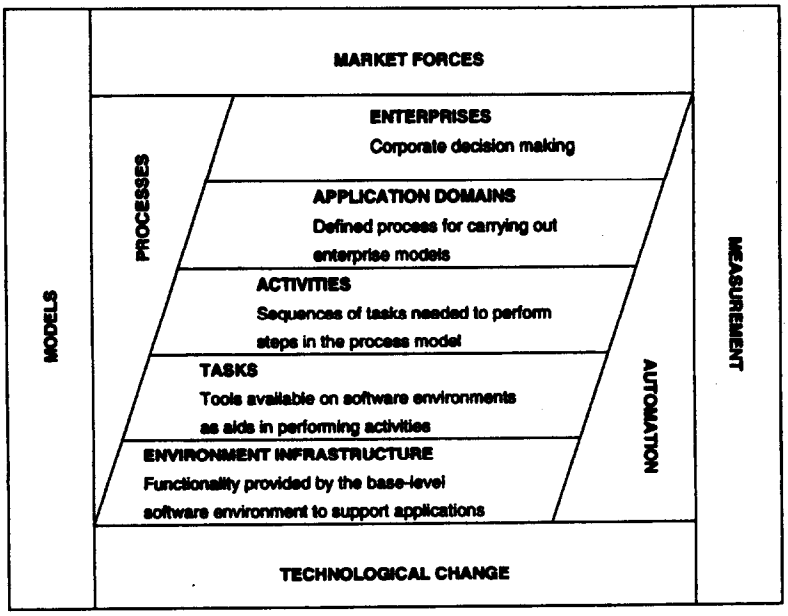

Figure 1. ITEM Model.

- *Technological changes* are the information resources available to the organization to solve the problems defined by the above market forces.
- *Models* are the abstractions needed to understand how the available technology can address changes imposed by market forces.
- *Measurement* determines how well the enterprise's information processes match the abstractions present in the models.

An organization uses the models to weigh the effects of market forces and technological changes on the organization. Measurement is important for determining success criteria for the process. Using these four components, the enterprise develops, tailors, and adopts strategies. The ITEM model views strategies as combinations of processes and the automated support for those processes. Here the ITEM model is described and then applied to the description of software engineering environments. A more complete description of the model is also available (Zelkowitz and Cuthill, 1994).

1.2. ITEM Model Levels

The model defines five information levels. The different levels describe the *processes* that the organization uses (e.g., the way the enterprise does its business) and the *automation* that the organization employs (e.g., the way in which it uses software and hardware information system components) allowing for the transfer of information among levels of the model. Notations such as IDEF0 (NIST, 1993b) may be used to model these attributes of an organization. The five levels are

1. *Enterprise*. This level defines organization policy and decision making. Major corporate decisions on organization policies are made at this level with little direct concern about information technology. These decisions include the selection of corporate strategies for manufacturing capabilities, outsourcing, and product development. Measures of success would include business metrics such as profit, market share, return on investment, etc.
2. *Application domain*. At this level methods for implementing enterprise decisions are developed. Examples of application domains include development of specific product lines, such as aircraft manufacturing or on-board computing in automobiles and business data processing for an organization.
3. *Activities*. Activities are sequences of steps needed to address enactment of services within an application domain. Activities represent complex series of interactions that aid in solving the problems within the application domain, as will be explained below.
4. *Tasks*. Tasks represent single steps needed to carry out an activity. This could represent editing a file, compiling a program or producing a design document. Each of these individual actions may be carried out using different methods by different enterprise components.
5. *Infrastructure*. Infrastructure represents indivisible components within an enterprise. Hardware technology like typewriters, fax machines, and telephones are infrastructure components. With computer software, basic functionality (e.g., data base access, communication processes) represent infrastructure.

The divisions separating these levels are arbitrary. Later we define these levels as the *abstraction level* of a process. One long-range goal of this work is to quantify more precisely the abstraction levels for the various processes in an organization.

A task is a single step, an activity is a set of such tasks and an application domain is a set of activities solving a larger problem. For example, although software quality assurance can be an activity (Brown et al., 1993), it could also be a task or an application domain in a different enterprise. An application domain (e.g., software engineering) may be considered an activity in a larger application domain (e.g., product development). In what follows, tasks will be sets of actions supported by a single tool or small tool set and application domains as larger product-oriented sets of activities that achieve organizational goals.

Over time, a process changes its relative complexity and level. As described in Section 3, increased automation simplifies an activity so that it becomes the task of running one tool. For example, generating and testing the code for a graphical user interface (GUI) now can consist of running one GUI building tool rather than using an editor, compiler, linker, debugger, and simulator to write, debug, and test the code. Alternatively, the enterprise may place an increased emphasis on some area requiring the elaboration of previously simple tasks into activities. For example, an enterprise-wide increased emphasis on quality may lead to the use of more testing tools and more elaborate recording and analysis of the products of those tools. Infrastructure, tasks, and activities are changing over time in response to changes in the market place and available technology, while application domains and enterprises are

more stable. What constitutes a complex activity or application domain may become simpler with increased automation or a simple task may become more complex with increased elaboration. We call this concept *technological drift* and describe it in greater detail later.

It should be made clear that the levels of the ITEM model are logical and not physical boundaries. Although business plan decisions at the enterprise level may not involve automation, the process model to define, implement, and monitor the plan may require considerable automation-data bases to store information, word processors to produce reports, spreadsheets for producing "what if" scenarios, decision support expert systems for analyzing strategies, etc.

1.3. Automation vs Process

Each level is concerned with the degree of automation and the set of processes needed to carry out the specifications of that level. The relationship between process and automation is key to differentiating this model from others. Based upon related work (Brown and Carney, 1995), processes *constrain* the set of services available to the enterprise, and these services *depend on* the set of mechanisms (i.e., automation). Looking at the inverse relation, mechanisms implement services that support processes.

At the lowest level, automation is prevalent with little concern for specific behaviors in an application domain. For example, a data base system, a word processor, or a spreadsheet are software technologies that are generally independent of the specifics of any application domain. Templates or schemas tailor a database to the application domain by defining the structure of its data. Thus, a database provides a high degree of independent automation with little attempt to influence the behavior of the user. This is comparable to the role of the telephone, fax machine, or typewriter.

As one goes up the levels of the ITEM model, this balance changes. For example, at the highest level (e.g., determining the business plan of the enterprise), the role of automation is relatively small with the major goal to define the process that will be the most likely to increase profits, increase market share, or develop quality products. Automation comes into play as one implements the defined models as more detailed sets of activities and tasks needing or utilizing automation support.

An alternative view of this hierarchy is to examine the translation of data (e.g., bits being transmitted or stored in a repository) to information (e.g., schedule of airline flights from Washington to Los Angeles) to knowledge (e.g., best alternative to arrive at Los Angeles airport at 9 pm with lowest cost flight). Infrastructure services (e.g., a word processing program, a telephone, a fax machine) process data without much interpretation in its use. At intermediate levels, the basic data is converted to information (e.g., collecting data on software development to determine product reliability, productivity, profitability). With this information, knowledge can be extracted (e.g., Is the waterfall life cycle more effective than the spiral model? Does concurrent engineering improve development attributes?) At present, automation effectively collects data, and individuals are needed to analyze that data and to process knowledge. Much current research in computer technology is in developing methods for extracting information from data and in analyzing information to determine knowledge.

Completion of an activity requires a combination of both process and automation support and may successfully be solved by various combinations of both. For example, the activity of sending a message to an individual at another location within the enterprise can be solved in multiple ways.

- *Write out information on a sheet of paper and deliver it to recipient.* No technology is needed and this can be viewed as a process solution to the problem without the need for automation.
- *Write out information on a sheet of paper and send fax.* This method uses no computer support and only minimal automation support (e.g., use of a fax machine).
- *Use a word processor to develop memo, print it, and then send copy via fax machine.* This uses a minimal amount of automation in developing the memo, and there is no integration step which combines the memo generation process (e.g., use of word processor) with the transmission process (e.g., use of fax machine).
- *Use a word processor to develop memo, and then use a command such as "send fax" to automatically have computer-installed fax hardware send the memo to the receiving site.* In this case, the word processor is integrated with the fax machine and automation support for this activity is very high.
- *Use word processor and "send fax" software to send fax directly to receiving computer.* In this case, no fax machine is used at all, and a direct computer-to-computer link is established.
- *Use a word processor and then send electronic mail to recipient.* This avoids the concept of a fax ma-

chine totally, and represents an integrated automated solution to the problem.

In all six cases the results are the same: The receiver at another location gets a memo; however, the process and the method of automation for achieving the goal differ. For example, solutions two through five (e.g., use of fax transmission) depend upon the telephone network to provide telecommunications support and the use of fax protocols for secure and correct transmission of the message. The electronic mail solution, while eliminating the direct use of telephone lines and fax protocols, assumes the existence of a more complex communications network linking the computers together (which may use telephone lines).

2. SOFTWARE ENGINEERING ENVIRONMENTS

Currently, most discussions of information technology used in such organization emphasize only one of the following perspectives.

1. *Automation support:* Environments supporting interoperable software tools procured from various sources are an important factor in improving effective use of computer technology. Considerable effort has focused on developing better ways of specifying the computer systems that a heterogeneous group of vendors can provide. This includes the hardware platform, software environment infrastructure, and tools for effectively using the enterprise's information.
2. *Processes:* The services implemented in information systems provide only part of the answer for increasing productivity. How one uses those resources, the process model that drives an enterprise's use of information, is often as important, if not more so, than the underlying computer resources.

Both concepts, the automated environment and the process, are important to effectively use information technology. Therefore, any information model should integrate both aspects of the enterprise.

2.1. Environment Infrastructure

At the base of the model is the automated computer system providing a core set of services for the automation of components of the adopted task, activity, application domain and enterprise processes. The NIST/ECMA SEE Framework Reference Model (NIST, 1993a) and systems like UNIX,[1] POSIX, Microsoft Windows, etc. describe models and products addressing information system infrastructure.

The infrastructure of information systems is undergoing a significant degree of standardization today. The frameworks reference model developed jointly by NIST and ECMA (NIST, 1993a) is typical of today's approach towards defining the software component of an environment infrastructure. In the NIST/ECMA frameworks reference model (the *framework RM*), an environment infrastructure or framework consists of 7 sets of basic framework services: (1) *Object management* services with data repository functions, (2) *Communication* services for passing information among environment components, (3) *Process management* services for building and using process models in the environment, (4) *User interface* services for permitting the user to communicate with components executing in the environment, (5) *Policy enforcement* services for instituting security and integrity constraints in the environment, (6) *Framework administration* services for maintaining the environment and tailoring it for individual use, and (7) *Operating system* services for implementing primitive functions that communicate with the underlying hardware platform.

The model is a service-based model in which each service describes an interface that supports some needed functionality. However, because this is an abstract description, the details of this functionality (i.e., its signature of input and output objects) is not specified. Edition 3 of the framework RM lists 66 such services grouped into these 7 categories. It is not an architecture because it does not constrain the system designer in implementing each service and in combining the services into components.

The framework RM is not a requirements document for an environment. Instead, it is a catalog of potential services; each framework implementation can be measured against the framework RM to understand which services are present and which are not. It is unlikely that any environment will implement all the services. For example, there may be several methods for communicating information among the processes in the environment—storing

[1]Certain commercial products are identified in this article to specify adequately the applicability of the model. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products are necessarily the best available for the purpose.

information in the object repository, using a message passing communications service, using a remote procedure call communications service, or sharing common data storage in memory. It is unlikely that any specific tool set will need all of these options. Security (i.e., Policy Enforcement Services of the framework RM) is an additional example where enforcement of security issues throughout an environment may seriously impact performance if such enforcement is not warranted (e.g., a stand-alone computer operating in a home office environment).

Many of today's software interface standards define services relative to the services in the framework RM. Standards and proposed standards like X-Windows and Motif provide the interface for the user services of the framework RM, several of the POSIX standards (e.g., 1003.1) and X/Open's Spec 1170 define interfaces for operating system services, ECMA's Portable Common Tool Environment (PCTE) provides many of the object management services, and Sun's Tooltalk, and HP's Broadcast Message Server (BMS) provide many of the communication functions needed in a distributed system.

There are products, whether standards-based or not, providing components of the environment infrastructure. Systems like IBM's OS/2, SUN's Solaris, IDE's Software through Pictures, Cadre's Teamwork, Microsoft's Windows NT, are all attempts at solving the environment infrastructure problem by defining the set of application program interfaces (APIs) needed to support one facet of tool interoperability. Given a standard set of APIs, various applications can more easily interoperate on the same environment platform with a relatively high degree of common look and feel and internal data consistency.

While many approaches to an environment infrastructure have been implemented or are under development, not all relevant problems have been solved. Several key issues still remain.

1. The foremost problem is effectively *integrating* the tools of an environment. This, described shortly, is the primary functionality needed for appropriate automation of the Activity level that will allow personnel to use collections of software products to solve a single application problem.
2. The framework RM provides the set of infrastructure services needed to support the tools for many of an enterprise's tasks. However, is the set of services provided by this document sufficient? For example, is "weather forecasting" an appropriate service for a software development environment? (e.g., Is predicting whether your employees will be able to show up for work during a storm an appropriate software engineering development capability? It certainly would be for a personnel application domain.) This is unclear.
3. The framework RM was developed within a certain context around 1990. As such, it has certain biases and limitations, among which include the following.

   - The framework RM was originally developed around a repository-based structure. As such, the set of communication services was originally very limited and, while subsequently expanded, still needs further development.
   - The repository design of the framework RM did not originally include object-oriented technology that evolved since the development of this model; additional service definitions to accommodate this technology may be needed.

4. The definition of other tools for specifying environment infrastructure is needed. These would include profiles of infrastructure standards and mechanisms for specifying the integration methods supported by the tools.

2.2. Tasks

Tasks represent single steps towards the completion of the defined activity. Each development activity is composed of a set of tasks for performing that activity. Tasks usually are, but are not required to be, automated or supported with specific automated tools that transforms inputs into outputs. For example, the activity of designing software within the waterfall life cycle (Royce, 1970) application domain process requires a coding task which may be performed manually and supported with a text editor or automated with a code generator. Combinations of tools or tool sets can provide assistance in the performance of individual tasks and activities.

The PSESWG Project Support Environment Reference Model (Brown et al., 1993) (called PSE RM) developed the concept of *end-user services* to describe these automated products which support development activities. The end-user services suggest a partial list of tasks for a software development enterprise.

For example, the PSE RM developed end-user services in four different application domains: Technical Engineering, Technical Management, Project Management, and Support Services. Each of these, in turn, is divided into the set of activities needed to support the processes engaged in specific tasks. For

example, the Technical Engineering services include System Engineering services, Software Engineering services, and Process Engineering services. The Software Engineering services include the more familiar tasks of requirements definition, design, coding, static analysis, testing, etc. Typically, a single product implements one of these fine-grain services. These are products available supporting or automating most of these services. For example, Table 1 shows that within the Software Engineering application domain, the following sample tools exist.

The application domain process model selected by the enterprise defines activities and tasks, the processes used to implement them, the services used to support the set of them, the tools used to implement them and the relationship among those tools. The application domain and the activity level process defines how the end-user services, and the tools implementing those services should be related in an environment. For example, PSE software engineering services include both a software verification service and a software testing service. How one implements each service and connects the implementations are decisions made locally to tailor the environment to support the enterprise's defined process.

While the PSE RM has defined a candidate set of end-user services for the systems and software engineering application domains, outside of the software development world, little progress has been made to develop catalogs of necessary end-user services in other domains.

There are few standards or standards activities related to defining tasks and end-user services. Programming language syntax and semantics have been standardized for most common languages (e.g., C, FORTRAN, Ada) and many data base languages do have standard interfaces (e.g., SQL). There are also industry "benchmarks" for testing some systems and standards like POSIX 1003.1 and several language standards do have test suites for testing conformance. ISO/IEC/JTC1/SC7 has defined software life-cycle processes and begun defining the tasks that make up these processes in areas like configuration management. Outside of these activities, there have been few efforts to define processes or compatible interfaces for end-user services.

Table 1.

Service	Software tool
Software Design	MarkV ObjectMaker
Software Simulation	Arcadia project's Chiron GUI builder
Software Generation	UNIX's YACC
Compilation	Compilers for C, FORTRAN, Ada,...
Software Static Analysis	NASA's Static Analysis Program (SAP)

2.3. Activities

In software development, software engineers must engineer software (e.g., specify and design it), build software (e.g., code and test it), certify software (e.g., verify, validate, perform quality assurance on it), and maintain software (e.g., fix errors and enhance it). Each of these activities supports a portion of the application domain. Each organization tailors its software development process into activities which address specific concerns in its own application domain model. Various process models are available for modeling activities, such as object oriented design or cleanroom software development, while others models handle configuration management, quality assurance, and specific development tasks like software coding, testing, etc.

Activities are defined sequences of tasks, supported by end-user services, which have a clear initiation point, progression and end point and have the goal of producing a product. This does not rule out iteration among tasks until conditions are met. The definition of an activity includes policies on when and how the activity can be initiated (preconditions on the activity), the circumstances of moving from one task to another (pre- and postconditions on each task) and how the activity can be concluded (postconditions on the activity).

The PSE RM defined activities as sequences of user actions supported by sets of defined services as opposed to tasks which are user actions supported by a single service. This mapping of activity to multiple services implies the need for multiple tools to complete that activity. For example, the PSE RM defined software quality assurance (SQA) as an activity that utilized the metrics, verification, testing, and configuration management end-user services of the existing PSE RM. SQA was not deemed to be a separate service (e.g., implemented as a single computer tool), but instead was a process composed as the enactment of several existing services.

Tool integration, mentioned above under environment infrastructure, is crucial to effectively provide automated support for activities. In order to automate activities with a tool set, there must be a seamless way to pass information and control among the tools. For example, the build software activity mentioned above generally uses the following tasks and tools.

- Use a **design compiler** to aid in the translation of design to source program text.

- Use a reuse library **browser** to identify any available components to reuse.
- Use an **editor** to enter source program text.
- Use a **compiler** to translate the text.
- Use a **linker** to assemble the separate program components into an executable version.
- Use a **static analyzer** to analyze the structure of the source program.
- Use a **metrics evaluator** for determining the complexity of the source program.
- Use a **verifier** to prove the correctness of the source program.
- Use a **source code debugger** to monitor program execution.
- Use a **testing tool** to execute and test the source program.
- Use a **configuration management tool** for maintaining libraries of source programs.

Two immediate observations from this list of tasks.

1. Today, although all of these tools would be useful, they cannot be built into a single software engineering environment economically. However, this set of tasks is necessary (whether computer-based or manual) for most software development today, and the proposed model in this article must be able to describe an environment containing all these tools and processes that would use the full tool set.
2. Information must be able to be passed among the set of listed tools: **editor, browser, design compiler, compiler, linker, static analyzer, debugger, metrics evaluator, verifier, testing tool,** and **configuration management tool.** This is the integration problem. While individual tools are relatively easy to build, accepted techniques for transferring data and control among tools need to be further studied and agreed upon mechanisms need to be developed.

There are a few standards, or standards activities, in this area. Those available tend to focus on software development or other types of product development. Standards like Microsoft's Object Linking and Embedding (OLE), Common Data Interchange Format (CDIF), Standard for the exchange of product model data (STEP) (Trapp, 1993), Electronic Design Interchange Format (EDIF) (Kahn and Goldman, 1992), Common Object Request Broker Architecture (CORBA) and PCTE are trying to address how heterogeneous tool sets may interoperate. Standards like IS 12207-1 Software Life Cycle Processes define processes and activities in areas such as software development, acquisition, and configuration management.

2.4. Application Domains

If software development is important to the enterprise, what is the application domain process for building software? Such processes as DOD-STD-2167A for defense system software development (DOD, 1988), cleanroom development for improved quality of the resulting program, and the spiral model of software development (Boehm, 1988) are possible choices. There are also processes like the SEI CMM (Paulk et al., 1993) for measuring and improving the development process. Measurements of the effectiveness of application domain processes to meet the enterprise level goals begin to address the "engineering" of information technology. Effective automated support for processes requires extensive tool interaction and customization in software environments.

Application domains are focus areas of the enterprise that result in identifiable products. However, "product" does not necessarily mean items sold by the enterprise. Products may be internally used, such as production of the payroll "product." An application domain process will consist of a sequence of activities with the identifiable product as a goal. Application domains can produce products which are feedback for redefining tasks and activities and for reentering the process. Application domain models define the mechanisms an organization uses to build its sets of activities and the organization's constraints on these activities. These activity definitions determine the organization's need for end-user services. The task interconnections in the activity imply the need for corresponding service interconnections implemented in some environment. Much of the current interest in process modeling addresses the set of processes needed to solve problems in some application domain. Process model activities in the software development domain include,

- *Defining the enterprise's Software Development models.* The process of developing software is certainly a major concern, and development models have been under study for many years. Most organizations use the so-called "waterfall model" and processes similar to DOD-STD-2167A standards for developing software. In this model, each phase of the development process must be com-

pleted before the next phase can begin. The process progresses from requirements to specification to design to code to test for the complete software system. The model is *product-based* because moving from one activity to the next generally requires the completion of a milestone product—for example, a design document before coding begins, source programs successfully completing unit test before integration testing, an integrated system passing integration tests before beginning software quality assurance validation, etc.

However, variants to the waterfall model exist. Boehm's spiral model emphasizes risk reduction and prototyping as drivers even though the ultimate development follows a waterfall-like process. Rather than view the development process as a progression of deliverable documents, development is viewed as successive prototypes needed to reduce the overall risk of building a product. The cleanroom process uses similar phases to the waterfall model; however, cleanroom places a greater emphasis in design verification with a corresponding lessening (if all goes well) of testing during the coding phase.

- *Quality improvement.* An important consideration is the improvement in the quality of software by improving the quality of the software development process. The assumption behind improving the development process is that improvement in the process will lead to improvement in the product. Two well-known examples of quality improvement processes follow.
  (1) The SEI Capability Maturity Model (CMM) is a process an organization can use to improve its software development activities. The CMM defines a process for an organization to follow to investigate its own development processes, to institute management controls and measurement guidelines, and to improve its understanding of how it does its business.
  (2) The NASA/GSFC Experience Factory (Basili et al., 1992) is a model that grew out of NASA-Goddard's Software Engineering Laboratory (SEL). This is a bottom-up approach where an organization tailors an existing development process for process improvement. Its ultimate goal, however, is similar to the CMM in getting an organization better prepared to develop software.

There is additional work in developing notations for designing development processes. Systems like Marvel, Process Weaver, and others, are all attempts to provide automated support for a defined process, therefore, encouraging its use.

2.5 Enterprise Models

An enterprise consists of the processes which define the functioning of the application domains that produce and direct the enterprise's products and services. Within the software development area, understanding the relationship of software to the organization's goals is a first step. Real-time embedded applications (e.g., computer controls in an automobile) reflect a different set of issues than a pure software development (e.g., producing a new spreadsheet program for use in a desktop workstation). Using methods such as sequential or "pipeline" processes are major enterprise-level decisions. Alternatives may consist of parallel or "concurrent" engineering approaches (Malone et al., 1993). How one implements such processes depends upon the specific application domain that the enterprise is concerned about.

The enterprise model also defines the policies used in executing the processes, the relationships among these processes, and the starting and termination of processes. Risk analysis is a major concern of business policy-making. This level is concerned with the policies of the enterprise's business plan. How does an organization define its development processes? Is the organization often involved in building many new and different products where the risk of an incorrect design is extremely high or is it involved in repeated development in similar applications? In the former case, a process like the spiral model will minimize the exposure to cost risks by forcing successive prototypes to deal with the unknowns of the development process. On the other hand, for the latter case, a more standard waterfall process may be less expensive since there is less risk of a faulty design.

Does a CMM or an Experience Factory process improvement approach fit better with other processes in the enterprise and with the enterprise goals? What are the risks in developing either model. Little has been done to develop a software environment supporting enterprise level processes, process management, and decision making.

3. MEASUREMENT

The ITEM model represents a classification of the processes undertaken by an enterprise in its use of information technology. However, as part of the evaluation of those processes we need to provide a

mechanism for measuring the use and effectiveness of information technology in pursuit of the goals of the enterprise. In this section, we discuss the external market forces and internal technological changes that have an effect upon the level of automation used by an enterprise.

We can initially address three measurement attributes as part of the model (Figure 2).

1. *Abstraction level* represents the hierarchical place that a given process has within the enterprise. We already described this in the previous section as a nominal measure representing an infrastructure, task, activity, application domain, or enterprise process. A refinement of this measure would be to a more quantitative concept to measure the *degree* of hierarchical level within the model. However, as of now we only define these five nominal levels.
2. *Automation level* represents the degree of automation a given process possesses. An automation level of 0% represents a purely manual process, while an automation level of 100% represents a purely automated process. An obvious goal of information technology is to increase the automation level of all processes.
3. *Process complexity* represents the complexity of performing a given process. For a given process, we would like as low a complexity as possible (i.e., *process simplicity*).

Previous work on the PSE, NIST/ECMA, and other reference models has discussed the level of services provided by an environment (i.e., what we call the abstraction level here). In this section, we discuss further the interaction between automation level and process complexity. Much current process research is implicitly concerned about these two measures, although such research has not explicitly discussed their characteristics.

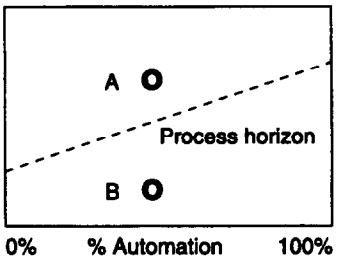

Figure 3. Process horizon.

3.1. Process Horizon

Our concept of process complexity is based upon two assumptions about the complexity of processes.

Assumption 1. *Process horizon:* For any automation level, there is a limit to the complexity where more complex processes are unstable (e.g., unreliable, incorrect). We an describe this by Figure 3. For any given level of automation, there is a level of complexity that represents the maximum process we can develop reliably. For example, process *A* represents an unstable process, and process *B* represents a more stable process. The dotted line represents our process horizon or the maximum complexity we can comprehend for that degree of automation. Because machines are more reliable than people for repetitive tasks, the slope of the process horizon is positive as more automation is introduced, as explained below.

This assumption is used constantly in software development. For example, consider the process of building programs in assembly language, a process that is mostly manual with a little degree of automation (i.e., use of the assembler). Process *A* might represent a large real-time operating system, a process that has frequently been shown to be error-prone and very time consuming (e.g., development

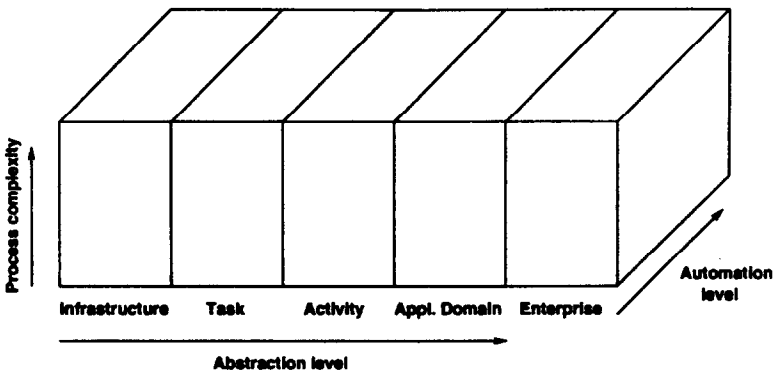

Figure 2. ITEM Measurement attributes.

J. SYSTEMS SOFTWARE
1997; 37:27-40

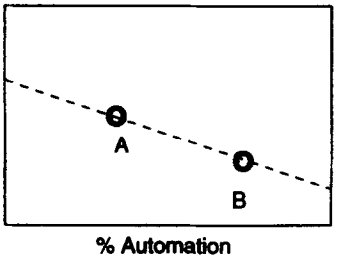

Figure 4. Equivalent complexity.

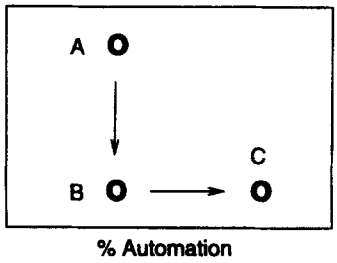

Figure 5. Process movement.

of OS/360 by IBM (Brooks, 1975)). Process *B* might represent a smaller program written in assembly language, a more manageable unit of work.

Assumption 2. A second assumption implicit in software development concerns the effects of increasing the degree of automation in a process: If a process becomes more automated, the process complexity decreases. Figure 4 represents the complexity of a process as it becomes more automated. As the computer takes on an increasing role, the complexity of the overall process decreases. In this figure, process *A* might represent our assembly-language operating system, while process *B* might represent the same operating system written in C, a simpler process than the original one. The assumption is that for a given complexity level, the more automated the process is, the more reliable it will become. This is based upon the observation that automated processes (even relatively unreliable ones) are more reliable than purely manual ones that depend upon people performing the actions.

Figure 5 represents the effects of both of these assumptions.

Corollary 1. For two processes of equal automation levels, choose the simpler process (e.g., Process *B* over process *A* in Figure 5).

Corollary 2. For two processes of the same complexity, choose the more automated process (e.g., Process *C* over process *B* in Figure 5). The assumption is that automation increases the reliability of the process.

3.2. Technology Transition

We can summarize the above discussion to provide a picture of technological innovation by Figure 6. Let us use the example of building an operating system, as given previously.

- A process is attempted that is above the process horizon (e.g., process *A* of building a complex operating system in assembly language).
- In order to be successful, a simpler process with the same degree of automation that is below the process horizon is attempted (e.g., process *B* is a simpler system built in assembly language).
- Later, as a result of automation, the same process can be achieved with lower complexity and increased automation (e.g., process *C* could represent the same system as process *B* written in C). In this case, we would state that process *C* is a *conforming process* to process *B*.
- As a result of the increased automation, the process that was above the process horizon is now suitably below this horizon and can reliably be implemented using this new process (e.g., Process *D* might represent the original complex operating system, now easily built in the C language).
- Because of the increased complexity allowed by the increased process horizon, an even more complex system than the original system may now be built (e.g., process *E* may represent a complex client/server distributed system built in C, which wouldn't have even been attempted with the original assembly-language process of process *A*).

This five-step process represents the general technology transfer process as applied to information technology. A process is too hard (i.e., above process horizon), it is simplified, a more automated process is found, and then it is discovered that with the new process even more complex processes can be built.

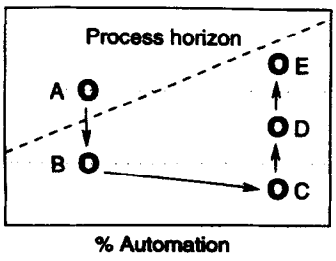

Figure 6. Technological innovation.

This process is continually repeated until the process is completely automated.

Corkscrew phenomenon. Of course, we have assumed in this discussion a single abstraction level slice of the ITEM model. In reality, increased automation tends to also increases the abstraction level. For example, use of C++ instead of C not only increases the degree of automation, but the introduction of objects and classes moves the abstraction level to the right somewhat in Figure 2 as the enhanced process builds more capabilities into the process. What we get is a kind of "corkscrew" path in three dimensions through Figure 2. Process *C* of Figure 6 represents a more automated version of the previously-implemented process *B*, while *D* and *E* represent enhanced capabilities not present in the original process. Hence, the new process *E* represents a higher abstraction level than the original process *A*.

One effect of this is that abstraction level is a property of an individual organization. What is an application domain process in one company may be only an activity or task within another, depending upon the automation and processes that are in place. Because abstraction level is a property of an individual organization, we can discuss both the abstraction level of a single company and the abstraction level within the particular industry of that company. We will do this by first introducing the concept of technological drift.

Technological drift. The discussion so far has assumed a rather static structure for an abstraction level. In reality, this is dynamic and changes over time, with a drift towards the left (that is called *technological drift*), which is shown in Figure 7. That is, over time, concepts that were considered advanced processes in an enterprise become more mundane and routine.

For example, in the 1950s, reading and writing to a file was a complex application program (in the application domain abstraction level). By the 1960s with the advent of access methods reading from a file became a rather simple task. Today, with windowing systems becoming prevalent, reading and writing is a simple infrastructure concept and even more complex actions such as scrolling, cut-and-paste, and menu buttons are becoming simple tasks.

The effects of technological drift on our process horizon concept is that as a process becomes automated (processes *A* through *E* of Figure 6), the effects of increased abstraction level is countered by this drift. So a successful company is one that manages to stay "in place" for its processes. For example, the complexity of software in C++ today is about at the same abstraction level as assembly code was in the early 1960s.

Because we say that process *C* of Figure 6 is a conforming process to *B*, the extension to processes *D* and *E* represent technological innovation. We say that process *E* is at the same abstraction level of the former process *B* and process *B* has drifted to a somewhat lower abstraction level.

What impact does this have on technology transition within a company? If a company "stays ahead of the curve," that is, it builds new automated processes faster than the technological drift across an industry, then it will raise the abstraction level of its processes compared to other companies, allow its personnel to consider more advanced concepts and be more attuned to the needs of the enterprise. Thus, a successful company will have its processes evolve faster than the technological drift to the left in Figure 7. On the other hand, if a company does not adapt to changing technology rapidly enough (e.g., staying with process *C* of Figure 6 instead of developing processes *D* or *E*), then its own abstraction level will drop relative to that of the industry, and it will be left behind and probably fail.

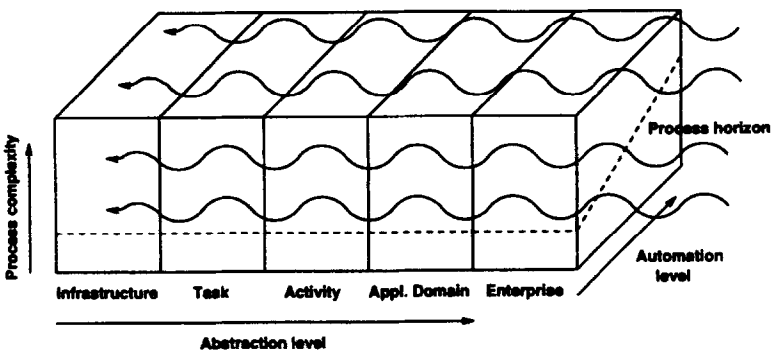

Figure 7. Technological drift.

So if we can compute an average or a representative abstraction level for an organization, we will say it is technologically successful if it is higher than the corresponding abstraction level for comparable processes within the industry. The U.S. automobile industry was a classic example of this. The use of automation and robots by Japanese manufacturers during the 1980s allowed Japanese cars to be more reliable than U.S. cars by using a more advanced development process based upon increased automation. It was only when more advanced technology was used in the late 1980s did U.S. manufacturers start to close this quality control gap.

However, what is still needed in this theory is a quantitative metric of process complexity. We need the analog of *function points* or *lines of code* for software products that would allow us to compare each of these complexities and to be able to fine-tune what we mean by an abstraction level. While we can say C++ is "a higher abstraction level" than C, the question is "how much higher?" How would we compare a Smalltalk program with C++, and C++ to Prolog, for example? All of these need further investigations at this time.

4. CONCLUSIONS

This article described the outline of the Information Technology Engineering and Measurement Model to understand how information systems are used in an organization. Understanding this as information technology evolves provides a mechanism of change as organizations evolve over the next decade.

The ITEM model was used to describe the components that go into developing integrated software engineering environments. This is an improvement over previous models in this area because

1. The model includes both computer-automated tasks and manual processes in its description. This is an improvement over previous models that emphasized one over the other.
2. The model describes organizations at multiple levels of detail. This article emphasized the software development components of an organization. In Zelkowitz and Cuthill (1994), the ITEM model is used to describe enterprise-wide information technology and to describe the processes that such an enterprise undergoes as its information technology needs change over time.
3. The model qualitatively can be used to describe technology transition and provides a mechanism for studying the evolution of an organization over time as technology changes within an industry.
4. We have the beginnings of a theory to quantitatively define what we mean by technology transition and a quantitative measure of process improvement.

Today, pieces of the ITEM model are well understood. The framework and the PSE RM provide good baselines for automation issues at the task and activity levels. Work on process modeling provides the basis for determining higher-level attributes, but there are many gaps (e.g., what are the various models at each level? How is the process horizon and technological drift applied across all levels and applied across an industry? What standards and products can be used to implement services at each level?) Current efforts are to determine how to extend the model and fill in these gaps.

REFERENCES

Basili, V. R. and Rombach, H. D., The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Trans. on Software Engineering* (14)10:758–773 (1988).

Basili, V. R., Caldiera, G., and Cantone, G., A Reference Architecture for the Component Factory, *ACM Trans. on Software Engineering and Methodology*, (1)1:53–80, 1992.

Boehm, B., A Spiral Model of Software Development and Enhancement, *IEEE Computer* (21)5:61–72 (May, 1988).

Brooks, F., *The Mythical Man-Month*, Addison-Wesley, Reading, Massachusetts, 1975.

Brown, A., and Carney, D., On the Necessary Conditions for the Composition of Integrated Software Engineering Environments, *Advances in Computers* 42, 1995.

Brown, A., Carney, D., Oberndorf, P., and Zelkowitz, M., (eds.), *Reference Model for Project Support Environments*, Version 2, NIST Special Publication 500-213, October, 1993.

Corporate Information Management: Process Improvement Methodology for DoD Functional Managers, Second Edition, D. Appleton Co., 1993.

Dept. of Defense, Military Standard: Defense System Software Development, DOD-STD-2167A, 29 February 1988.

IEEE, Draft Guide to POSIX: An Open Systems Environment, P1003.0, D16, August, 1993.

Kahn, H., and Goldman, R., The Electronic Design Interchange Format (EDIF): Present and Future, *29th ACM / IEEE Design Automation Conference*, 1992, pp. 666–671.

Malone, T., Crowston, K., Lee, J., and Pentland, B., Tools for Inventing Organizations: Toward a Handbook of Organizational Processes, in *Proc. of 2nd IEEE Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises*, 1993, pp. 72–82.

NIST, *Reference Model for Frameworks of Software Engineering Environments*, Edition 3, NIST Special Publication 500–211, August, 1993.

NIST, Integration Definition for Function Modeling (IDEF0), FIPS Pub 183, NIST, December, 1993.

Paulk, M. C., Curtis, B., Chrissis, M. B., and Weber, C. V., Capability Maturity Model for Software, Version 1.1, *IEEE Software*, 10(4) 18–27 (July, 1993).

Royce, W. W., Managing the Development of Large Software Systems: Concepts and Techniques, in *Proceedings Wescon*, IEEE Computer Society, 1970.

Trapp, G., The Emerging Step Standard for Product Model Data Exchange, *IEEE Computer*, 85–87 (February, 1993).

Work, B., and Balmforth, A., Using Abstractions to Build Standardized Components for Enterprise Models, in *Proc. 1993 IEEE Computer Society Software Engineering Standards Symp.*, 1993, pp. 154–162.

Zelkowitz, M., and Cuthill, B., Information Technology Engineering and Measurement Model, National Institute of Standards and Technology, Technical Report NISTIR-5522, November, 1994.