A software process model for business reengineering

A. T. Berztiss[1,2] *and J. A. Bubenko, jr.*[2,3]

[1] *Department of Computer Science, University of Pittsburgh Pittsburgh, PA 15260, USA;*
e-mail: alpha@cs.pitt.edu; fax: +412-624-8854

[2] *SYSLAB, University of Stockholm, Sweden*

[3] *SISU, Stockholm, Sweden*

Abstract

A major component of any business reengineering effort is the identification of business processes, and the development of software to support these processes. The development of the software is itself a process, commonly called the software process. One reason for reengineering a business is to decentralize its mode of operation, or to make a decentralized mode more effective. We contend that a properly defined general software process model is essential for the development of support software for a reengineered decentralized enterprise. We have developed a sixteen-step plan for business reengineering, and an enterprise model composed of eight submodels. In this paper we bring together the enterprise model, relevant steps of the business reengineering plan, and the Capability Maturity Model of the Software Engineering Institute to define a software process model for business reengineering.

Keywords

Business process, business reengineering, capability maturity model, enterprise model, process model, software process

1 INTRODUCTION

A reengineered business is characterized by a set of well-defined business processes that support the goals of the organization (Davenport, 1993; Hammer and Champy, 1993; Johansson *et al*, 1993). The purpose of this paper is to outline a software development process designed for the introduction of software support for the business processes. Although our work is primarily concerned with requirements engineering for so-called information systems, it is relevant to the development of embedded control systems as well. We see in fact no real difference between the two kinds of systems. Both interact with their environments, and the form an interaction takes is nearly always influenced by the contents of an information base. Moreover, advanced information systems, particularly systems that contain expert systems, often exercise autonomous control over key activities of an enterprise. Control is particularly important in information systems for decentralized organizations. It is therefore appropriate to refer to the software systems with which we will be concerned as information-control systems.

In (Berztiss, 1995) we have proposed a sixteen-step process for business reengineering, where the steps do not necessarily follow the sequence given below:

1. Commitment by top management
2. Selection of the executive and operational managers of reengineering
3. Selection of reengineering teams
4. Initial education phase
5. Identification of the processes of the organization
6. Specification of each process
7. Development of alternative implementation plans for each process
8. Cost-benefit study for each plan and selection of an appropriate plan
9. Infrastructure definition based on the selected plans
10. Setting of priorities for implementation of processes and infrastructure
11. Second education phase
12. Pilot implementation of one or two processes
13. Reexamination of the reengineering effort
14. Implementation of the remaining processes
15. Continuing automation of processes
16. Automatic linkages to the environment

Our main concern here will be with Steps 5-9, which will be defined in some detail in Section 2. In (Berztiss, 1995) we point out that the Information Systems Department (ISD) of an organization is well positioned to provide major support for any business reengineering effort - by being outside traditional departments, such as sales, marketing, and personnel, ISD personnel can view their organization with neutral eyes. More importantly, the ISD personnel knows much about the organization, but it must be realized that there is also much it does not know. The missing knowledge has to be supplied by business domain experts. Moreover, the changes to be implemented will be much more radical than the changes the ISD has had to deal with in the past. In fact, in order to deal with these changes, the processes of Information Systems have to be reengineered first of all - many ISDs operate without a properly defined software development process.

In order carry out effectively the support role we envisage for the ISD, it should have reached at least Level 2 of the SEI Software Maturity Model (SMM), which is described in (Humphrey, 1989; Paulk *et al*, 1993; Paulk *et al*, 1993a). The relevance of the model to business reengineering is discussed in Section 3. In particular, the model requires that an organization develop its software by means of a well-understood process that can be subjected to measurements. Our aim here is to arrive at a definition of such a "reengineered" process for business reengineering.

Having the technical competence defined by Level 2 of the SMM is a necessary prerequisite for the effectiveness of an ISD in business reengineering, but it is not sufficient. There also has to be strong cooperation between the technical personnel of the ISD and management representatives. Management has the twofold responsibility of articulating the strategic goals of the business reengineering effort, and of expressing explicitly the business rules by which the organization operates. The need for continuous attunement of the technical aspects of process definition to management views requires a rather complex model of the software development process, based on a model of the enterprise that is being reengineered. In Section 4 we outline the SISU (Swedish Institute of Systems Development) enterprise model, and in Section 5 move from this enterprise model to a software process model. In Section 6 our process models are combined into an abstract process model for business reengineering, and in Section 7 the abstract process model is converted into an operational prototype-based model. In Section 8 we advance some conclusions from this work.

2 SOFTWARE SUPPORT FOR BUSINESS REENGINEERING

In this section we shall consider in detail Steps 5-9 of our business reengineering process. The business processes are to be regarded as essentially independent of each other in the sense that each process is supported by its own information base, and the only way a process can change the information base of another process is by sending a message requesting the other process to carry out the change. This is in keeping with the concept of ownership of information by a process (Davenport, 1993), and is essential for effective decentralization. Moreover, unless the total information base is partitioned in such a way, the software modules corresponding to the processes are subject to common coupling, which is an undesirably tight mode of module coupling (Constantine and Yourdon, 1979). Of course, in a centralized system, the individual information bases can be combined into a relational data base, but each table is to be owned by a specific process, and the authority for making changes to the table is to be retained by this process.

Process identification

The number of processes with which a company is concerned typically varies between ten and twenty (Davenport, 1993). Conventionally a process is seen as the movement of a piece of work from site to site, where the sites are under control of different departments. Not only is a process defined in terms of a number of sites, but a site may serve more than one process, e.g., a site that checks the credit rating of a customer would normally be used both by order processing and by credit issuance. Therefore the actual identification of processes can be more difficult than it appears.

Process specification
In this step a process is separated into component tasks, each task examined to determine whether it is actually needed, unnecessary tasks eliminated, and the remaining tasks restructured. In the restructured process specification there has to be a clear indication of the order in which tasks are to be performed, and which tasks can be performed in parallel. We recommend that specifications be formal, but supplemented with diagrams to help achieve intuitive understanding of the processes. The specification of a process should be sufficiently general to allow a fair degree of implementation freedom. In particular, it should allow a decentralized implementation. Although process identification and process specification are usually regarded as separate steps, taken one after the other, sometimes it helps to consider them as a single amalgamated step. Note that the initial assignment of the activities of an organization to processes may have to be modified once specification of the processes begins.

Implementation alternatives
Bridge designers have the standard practice of presenting a client with several preliminary designs, and the designer and client decide which design to carry forward (Spector and Gifford, 1986). The same approach can be used with a business process, but in this step alternative designs are merely listed, with no decision as to which design is to be adopted. It needs to be remembered that any specification, be it expressed as diagrams or text, should leave the implementer with some choice. Thus it can be left open whether a function value is to be obtained by interaction with a sensor, looking up a table, or having a function subprogram compute it. It may also be left open at what site a particular task is to be performed, e.g., is order processing to be done at the main office or at a warehouse.

Cost-benefit analysis
The costs for each of the implementation alternatives are established. This is relatively easy, but it is not easy to determine corresponding benefits because of the difficulty of assigning a definite cash value to an intangible benefit such as customer good will. If one of the alternatives is an improvement over another in every aspect, then we know which to choose, but more commonly we have to balance both higher cost and higher intangible benefits of one alternative against both lower cost and lower benefits of another alternative.

Infrastructure definition
The infrastructure consists of an information system and a communication system, such as a set of LANs and a WAN. Although a distributed system may present more problems than a centralized system, the concept of information being owned by a process reduces their effect. In this step the infrastructure costs are determined for the total set of processes under the implementation alternative selected for each process. The processes, which we have been considering in isolation, now become integrated. It is possible that a different alternative is now chosen for some of the processes in order to reduce infrastructure costs. One objective is to reduce the volume of data flow - some thoughts on how to achieve this can be found in (Berztiss, 1993).

3 THE SEI CAPABILITY MATURITY MODEL

The Software Engineering Institute (SEI) has defined five software process maturity levels. At the initial level (Level 1) there is no process maturity at all - the "process" is characterized by lack of planning. Much software for the support of reengineered processes is being developed by organizations at Level 1, and the unreliability of their services and products is hurting the reputation of business reengineering.

To arrive at the lowest respectable maturity level (Level 2), the software process has to become repeatable. When presented with two development tasks in the same domain, both of the same size and complexity, an organization at Level 1 can be expected to come up with significantly different cost and schedule estimates. At Level 2 the estimates should not differ by much. In order to reach the repeatable level an organization has to maintain extensive statistics relating to its past projects. At Level 3 the process becomes defined. To reach this level, an organization has to be able to transform a generic process structure into a detailed definition of the actual process that is to produce a specific software system.

The next step is to arrive at reliable cost and schedule estimates for all projects. Achievement of this aim makes the process managed. This is Level 4, and reliable cost and schedule estimates are now available for any system development task, even when the organization has not been developing software for the given application domain in the past. In other words, all software development projects can be subjected to adequate management cost controls. The transition to this level is achieved by defining a set of standardized tasks, establishing a costing procedure for each task, and defining the development of a specific software system in terms of the standardized tasks. At the highest level, the software process has become optimized. In order to reach this level, an organization should have automated the collection of process data that are used to improve the software process. For example, these data can indicate that some of the standardized software process tasks are bottlenecks, or that their costs should be reduced.

Each level is characterized by a set of key process areas. For Level 2 there are six such areas: requirements management; software project planning; software project tracking and oversight; software subcontract management; software quality assurance; software configuration management. We claim that an ISD cannot provide effective support for business reengineering unless it has mastered these key areas. It is not easy to isolate all the reasons why many business reengineering initiatives have failed, but the absence of required capabilities in the ISD is often to blame. We shall now elaborate on the key process areas of Level 2.

Requirements management is to achieve a common understanding between a sponsor of a software product and the ISD that the software product will indeed have the properties that the sponsor expects it to have. This means that an agreement is to be reached between the sponsor and the ISD, that the agreement be fully documented, and that changes to the requirements document be made only with the approval of both sponsor and ISD. This agreement is to be the basis for a software project plan. As part of planning, time and resource estimates for the project have to be established, and a schedule of activities has to be drawn up. Sponsors are not to be confused with users. Sponsors view the software as a means of satisfying broad business objectives; users will be interacting with the software on a day-to-day basis.

Software tracking and oversight is then to check that the actual software development process adheres to the plan. Such tracking is to extend to the performance of subcontractors. The activities involved in software subcontractor management consist of selection of subcontractors, specification of contract obligations, and monitoring of the performance of subcontractors.

Software quality assurance activities consist of reviews and audits of software products and of the software process. The aim is to ensure that the products and processes comply with standards and guidelines, and that management is appraised of the results of the reviews and audits. The purpose of software configuration management is to improve software quality by integrity maintenance throughout the development of a software system. It ensures that all representations of the same version are consistent, e.g., that an information-control system is consistent with its functional requirements, or that two different implementations of the same facility are equivalent. As an example of the latter consider a function that returns the sunrise at Trondheim for any given day in the year 1996. This could be implemented as an algorithm that computes the time of sunrise, or as a table with 366 entries. The requirements for a system may stipulate that both implementations be provided, and configuration management must then ensure that substituting one implementation for the other will make no observable difference to a user.

4 AN ENTERPRISE MODEL

Here we introduce an enterprise model developed at SISU as part of the ESPRIT Project F3 ("From Fuzzy to Formal") - see Bubenko (1993), and Bubenko *et al* (1993). It emphasizes the separation of concerns during development of information systems, and in this it is related to the modeling of requirements in terms of four modeling "worlds" under the ESPRIT project DAIDA (Jarke *et al*, 1992), and the NATURE project (Jarke *et al*, 1993).

The F3 assumption is that the development of a system requirements specification is helped by a "population" of a number of sub-models. These sub-models describe the kinds of interrelated "documents" (or sub-documents) to be developed. By following a set of "good" sub-models, the risk to develop incomplete or inconsistent specifications can be reduced. By "good sub-models" we mean models that separate concerns in a way that improves clarity, as well as productivity and the quality of work products. Experience has shown that the use of an "objectives-model-driven approach" may lead to improved understanding of the problem domain by decision makers, definers of requirements, system developers, and "end-users". The economic value of improved communication and understanding, and for minimizing rework in later stages of the information systems development process has been observed elsewhere as well (Curtis *et al*, 1988). We propose to structure a requirements specification in an number of "submodels", shown in Figure 1.

In the objectives submodel of a requirements specification we describe, in a structured way, the "why" component of a requirements document. Goals for a particular enterprise or an activity are stated, and their relationships analyzed. This may relate to existing processes that are to be left intact, processes that are to be modified, or new processes that are to be designed. In the formulation of goals reference is made to problems and their causes, opportunities, and development actions. These supporting components explain why certain goals have been formulated, and are therefore important parts of the objectives submodel. The objectives submodel is a graph with the above types of components as nodes, connected by relationships of type "motivates" or "influences". The motivates relationship is here seen as a refinement relationship (e.g. a goal is refined in a number of sub-goals). The influences relationship is a relationship indicating positive or negative influences between components of the objectives submodel. These relationships can be given different levels of "strengths", e.g. low, medium or high.

The concepts submodel is used to define the "ontology" of the "universe of discourse" that concerns us, i.e. the set of object types, relationships, and object properties of the application we are talking about. For developing a functional information systems specification, the concepts submodel will define the "things and relationships" about which information must be represented in the information system, and the rules of behavior of the information system, implemented in the processes of the system. The concept submodel will, moreover, serve as a dictionary of user and customer defined concepts, and conceptual structures to be used to better understand other parts of a requirements specification.

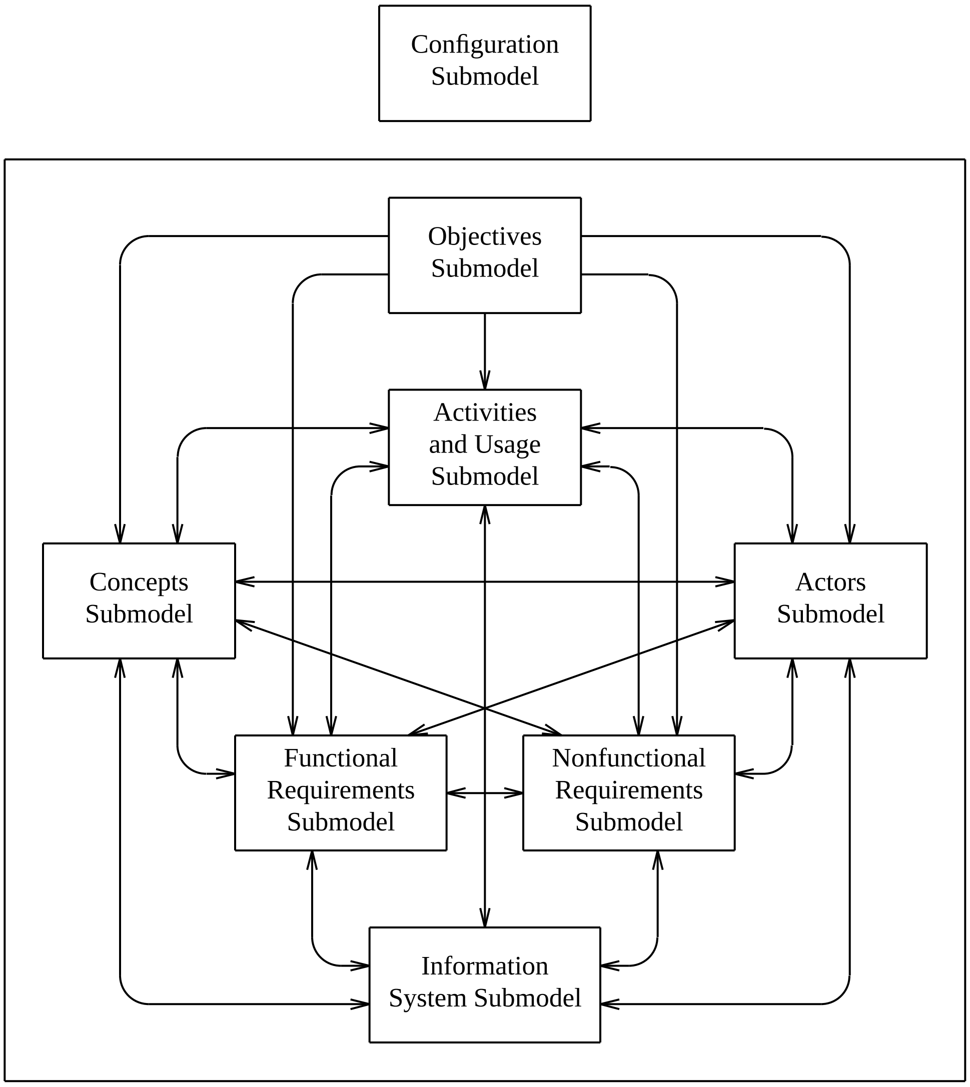

Figure 1 An enterprise model

The actors submodel is used to define the set of actors of the activity under consideration (individuals, groups, job-roles/positions, organizational units, machines, etc.), and their interrelationships, such as part-of, reports-to, etc. This submodel includes links to the other submodels, e.g., it identifies the "stake-holder" of a particular goal, the actor responsible for managing an activity according to a particular goal, etc.

In the activities and usage submodel, a particular existing process, a process to be modified, or a new process is defined and described from the point of view of activities and tasks, and the information and material flows between them. Describing, for instance, human-computer interaction is part of this submodel. Clearly, components of this submodel are motivated by components of the objectives submodel, they are performed using or referring to components of the concept submodel, and they are typically carried out by components (resources) of the actors submodel. In this submodel we also define business rules, which may be static or dynamic. These rules define permissible system states as well as permissible state changes. Information system requirements, related to the above models, are described by the following two submodels.

The functional requirements submodel elaborates the specific objectives and requirements that are put on the information system to support the activities and objectives defined. Typically it will indicate needs for establishing objects (or relationships, attributes), deleting objects, modifying objects, and the need for checking, querying and browsing objects and relationships.

The non-functional (NF) requirements submodel is primarily related to the activity and usage model, and indirectly to the objectives submodel, as activities normally are motivated by the objectives submodel. NF requirements concern components of the information systems submodel, i.e. information subsystems. NF requirements can be of several different kinds, such as standards for interfacing other systems, restrictions concerning use of hardware and/or software, accuracy of information, timeliness of information, security, access-rights, economic limits of the development project, etc.

The information system is described conceptually by the information systems submodel. By this we mean the complete, formal, functional specification of the information system. It is the task of the IS designer to develop IS designs and descriptions such that designated activities and processes in the activities and usage submodel, as well as the functional and NF-requirements are supported (to a greater or lesser degree).

In order to manage the modeling process and its results, we need the configuration submodel. This is a submodel, orthogonal to the other submodels. The main concepts of this submodel are version and release. We need this dimension for the reason that systems, be they organizational or computer-based, continuously evolve and change. For instance, the concepts submodel, or parts thereof, will change at times. New versions of these submodels will be developed, based on other earlier versions. On the other hand, we cannot disregard such historical information, as some actors may be still using it. Developing a new version of some submodel will most likely also cause needs for change in the other submodels. For instance, a new objectives submodel description will certainly affect and require a reconsideration of all other submodels. To manage "configurations" of descriptions is an important mission of an enterprise modeling facility.

What we have above is basically a set of interrelated descriptions or documents, each constructed with a predefined set of "permitted" types of components and component relationships. Depending on the purpose of the modeling exercise, we may, for a particular modeling job, need only a subset of these submodels. We may think of several other purposes for modeling. One such is the redesign of organizational processes, without assuming any supporting computer system to be developed. Another, quite typical purpose in practical situations, could be to reconsider the organization’s goals, objectives, mission, etc., and then study how an existing information system actually supports the new set of goals. This, of course, is part of business reengineering.

5 FROM ENTERPRISE MODEL TO PROCESS MODEL

Figure 1 indicates how the seven submodels that comprise the actual enterprise model are interrelated, and clearly shows the special importance of the objectives submodel - this submodel provides the impetus for everything else. We shall now derive a process model from the enterprise model.

In English there are six types of wh- questions - these questions start with why, what, who, which, when, and where. In addition we have questions that start with how. A process model is to provide answers to these questions. In terms of Figure 1, the objectives submodel responds to the why-part, the two requirements submodels relate to the what-part, the actors submodel responds to the who-part, and the information systems submodel to the how-part. The remaining three types of questions all relate to the activities and usage submodel - this submodel includes business rules, and we are concerned with which business rule is to be applied at a given time (when) and place (where).

This leaves the concepts submodel unaccounted for, and rightly so. It does not belong to a process structure. It provides a vocabulary and static structure that indicates how the terms of the vocabulary are related. Also, answering some of the how-part and all of the who-part can be delayed. We note again that at an abstract level it makes no difference how a value is obtained - it could be looked up in a table, computed by an algorithm, indicated by a sensor, or supplied by a user. Generalizing this somewhat, a process step may start out as a manual activity, become partially supported by a computer-based decision support system, and then become fully automated as an expert system. To take a concrete example, the processing of a purchase order may be initially performed by a number of employees in several departments, then, as process structure replaces departmental structure, by several employees belonging to the same process team, then by a single member of the process team, and ultimately by a software system operating on its own. These transitions may take place long after the initial process model for order processing has been formulated. Thus the actors submodel is not of primary importance in setting up an abstract process model. This view is supported by case studies in (Davenport, 1993; Hammer and Champy, 1993; Johansson *et al*, 1993), and case studies reported in *Computerworld* suggest that the failures of some business reengineering initiatives may be attributed to a preoccupation with information technology and role assignments rather than with analysis of the essence of business processes. However, in some contexts, early emphasis may have to be put precisely on the actors (Yu and Mylopoulos, 1994). Note that the distribution of tasks to different decentralized sites belongs to the activities and usage submodel.

Figure 2 shows a process model based on the enterprise model of Figure 1, but the process model takes into account the remarks made above. Two of the labels have been changed: we have generalized the model to information-control systems, and it has been made explicit that configuration control is to deal with both configurations and versions. The actual process is defined as a sequence of five submodels: an objectives submodel feeds an activities and usage submodel in the sense that the enterprise objectives motivate and determine activities and usage within the enterprise, the latter suggest functional and nonfunctional requirements for an enterprise information-control system, and the actual system is based on these requirements. This sequence is the core of the process. Although Figure 2 presents the submodels as an ordered sequence, there is much feedback and backtracking while the submodels are being developed. First, even when an orderly phased development plan is aimed for, it may be found that the initial objectives are unrealistic. Second, concurrent engineering encourages parallel development of the submodels. Some discussion of concurrent engineering of information-control systems can be found in (Berztiss, 1993a).

The core submodels depend on the two submodels that we have put outside the core. As mentioned above, the concepts submodel provides a vocabulary for use by the core submodels, and the actors submodel allocates roles to different operators within the enterprise. Briefly expressed, actors relate to a process, and concepts to the product of the process, but they do not have to be made part of a process while the process is still in an abstract specification stage.

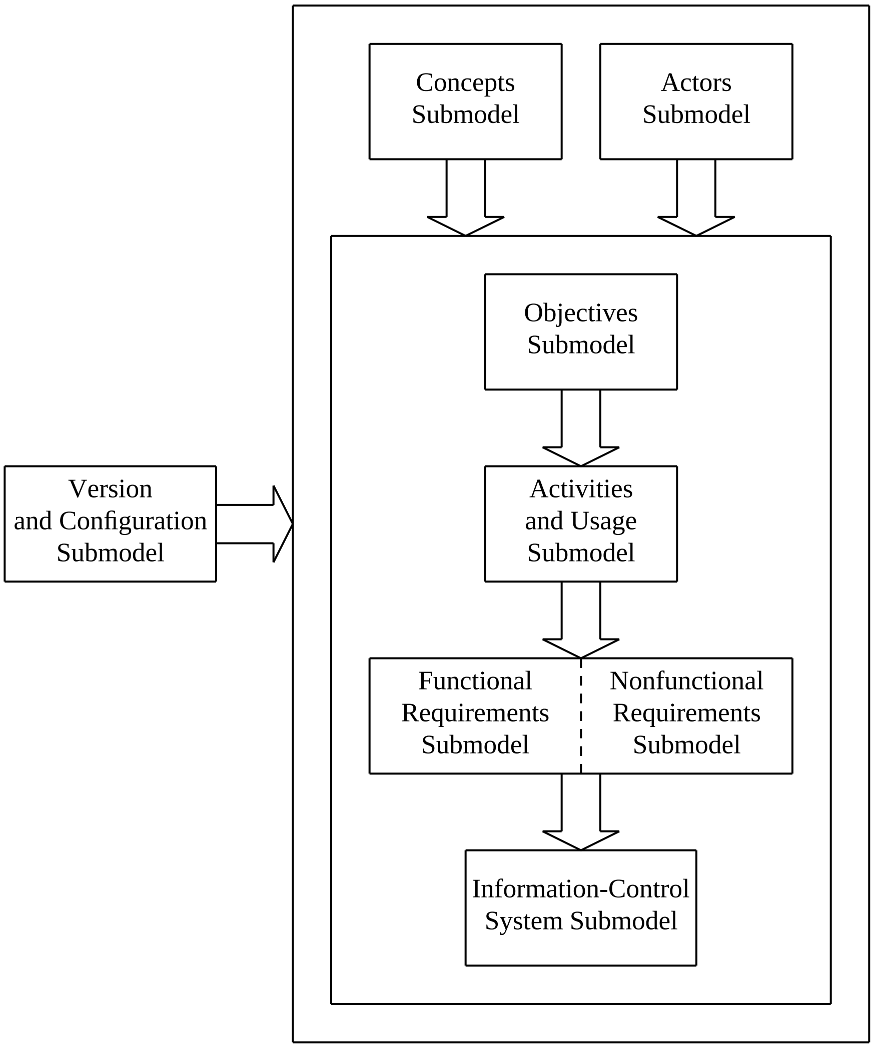

Figure 2 An abstract software system development model

We have to keep in mind that that we are dealing with three types of processes. The first is an enterprise analysis process that determines which of the operations of the enterprise are to be handled by software, and where they are to be handled. The second is a software process that is to develop information-control software systems. These information-control systems are processes of the third kind, and each such process is a product of the software process. All the information-control systems of an enterprise constitute the software support base for this enterprise.

6 A SOFTWARE PROCESS MODEL FOR BUSINESS REENGINEERING

We now bring together our three threads: the software process model (SPM), the business reengineering plan (BRP), and the key process areas for Level 2 of the CMM (CMM2). We propose four activity groups - a WHY-group, a WHAT-group, a HOW-group, and a WHO-group. Submodels of SPM, activities of BRP, and key process areas of CMM2 are to be allocated to these groups. The contributions of the three approaches to modeling the software process for business reengineering are shown in Figure 3.

WHY-group. We allocate to this group the objectives submodel of SPM and the process identification step of BRP. The CMM is not concerned with enterprise objectives at any of its levels.

WHAT-group. Requirements management of CMM2 belongs to this group, and so does process specification of BRP. We noted in Section 5 that the activities and usage submodel indicates when and where particular business rules are to be applied. This is elaborated into requirements, so that the activities and usage submodel and both the requirements submodels of SPM belong to this group.

HOW-group. Under BRP several implementation alternatives are proposed and evaluated in the implementation alternatives and cost-benefit analysis steps. Both these steps belong to the how-group because they lead to the selection of a particular alternative for actual implementation. We noted that software project planning is part of CMM2. This key process area deals with time and resource estimation, and a schedule of activities. For accurate estimation of the time and resource consumption by the project, and also by the product, there has to exist a software design. Such a design is to be produced as part of software planning of CMM2 and this design is the information-control system submodel of SPM.

WHO-group. Only the SPM explicitly addresses the actors issue. Under the other software development approaches the allocation of tasks to actors is postponed, but there are some parts of these approaches that relate to actors. Thus, under BRP, the infrastructure definition gives an indication of how actors are to communicate. Three key process areas of CMM2 can be allocated to this group. Software project tracking and oversight, software subcontract management, and software quality assurance all relate to the performance of actors, and whether a software process will be successful depends in a given instance on how well the tasks of the process have been allocated.

This still leaves us with the concepts submodel and the versions and configurations submodel of SPM, and software configuration management under CMM2. It is important to realize that a conceptual framework is an essential part of software development, although it does not belong to the software process as such. The management of versions and configurations is also outside the software process, but this may change as this topic keeps gaining in importance, particularly when configuration management is coupled to the maintenance of a software reuse library.

	BRP	SPM	CMM2
WHY?	Process Identification	Objectives Submodel	
WHAT?	Process Specification	Activities and Usage Submodel + Functional Requirements Submodel + Nonfunctional Requirements Submodel	Requirements Management
HOW?	Implementation Alternatives + Cost-Benefit Analysis	Information-Control System Submodel	Software Project Planning
WHO?	Infrastructure Definition	Actors Submodel	Software Project Tracking and Oversight + Software Subcontract Management + Software Quality Assurance
OTHER		Version and Configuration Submodel + Concepts Submodel	Software Configuration Management

Figure 3 Contributions to a software process model

It is interesting to note that CMM2 pays no attention to the determination of the objectives of an enterprise. The closest it comes to it is in Level 3. There the key area Integrated Software Management refers to the integration of management activities with software engineering methods into a coherent generic software process. Then, in adopting the generic process to a specific project reference needs to be made to the business environment.

7 PROTOTYPE-BASED PROCESS MODEL

We call software systems "sponsor-defined" when the requirements for such software are for the most part provided by business domain experts outside the ISD. All process software systems are in this category, and they cannot be frozen while the understanding of the business aspects of the enterprise by the ISD continues to be corrected by business domain experts. This is particularly important for business reengineering. In terms of Figure 3, the WHY-part is wholly the province of business experts or sponsors, the HOW-part wholly the province of information systems experts, and the two groups of experts have to define the WHAT-part in close cooperation.

Three major problems that beset development of sponsor-defined systems have been identified (Curtis *et al*, 1988). Besides the inadequate application domain knowledge by developers, there is the related problem that requirements keep fluctuating or contain contradictions. The third problem is that software developers are often victims of communication and coordination breakdowns. All three problems are amplified when radical business reengineering of a decentralized organization is being undertaken. The harmful effects of the three problems can be reduced by prototyping. Although exceptional designers are quite good at acquiring domain knowledge in other ways, even they would be helped by a user-developer dialogue supported by a prototype. The information from users is particularly useful when it is in the form of scenarios of system use. Such interaction between users and developers reduces external communication barriers, and thus helps stabilize requirements, but internal barriers too are lowered when all members of the development team have the same prototype system to refer to.

Briefly, the main purpose of a prototype process system is to allow users to establish that the developers have reached a conceptually sound understanding of the process, and to allow developers to determine that user expectations have been met. To this end a prototype is to provide a user interface and main system components in preliminary versions. We suggest that a prototype be built early, that it be adapted to user needs, and that the suitably modified prototype be converted into an efficient implementation, preferably by automatic transformations. Figure 4 is an idealized and grossly simplified overview of this approach.

In Figure 4 the "Function allocation" box represents the assignment of process functions to people, software, and hardware. One result of the allocation is an informal software requirements statement that is formalized and structured in the "Requirements documentation" box. The initial input to the "Function allocation" box is from the environment, which is a collective term for various sources of information, including the system sponsors. Some changes to requirements also originate in the environment, and lead to a changed requirements document. Such changes differ from changes suggested by users as part of the prototype test. Let us emphasize again that while sponsors view the software as as a means of satisfying broad business objectives, users will be interacting with the software on a day-to-day basis. Sponsor participation in the tests is recommended. Such participation is recognized by the arrow leading back to the "Environment" box from the two tests - this emphasizes that the prototyping process may lead to a modification of initial business objectives - the better understanding of business processes that prototyping provides can influence basic business strategy.

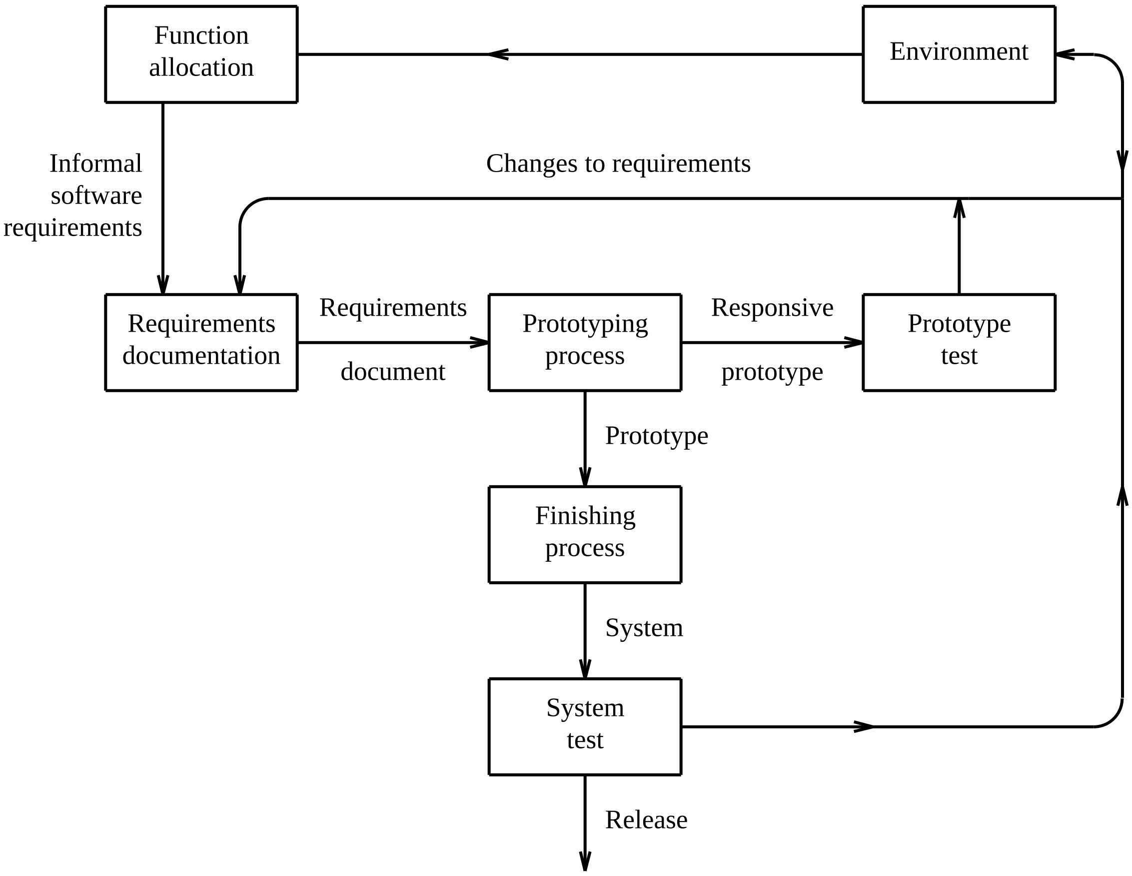

Figure 4 Prototyping-based software process

Referring from Figure 4 back to Figure 3, we note that the "Environment" box of Figure 4 stands for the WHY-part of Figure 3, the "Requirements documentation" box for the WHAT-part, "Function allocation" for the WHO-part, and the rest of the process for the HOW-part. However, when function allocation is related to the actors submodel, an inconsistency appears to arise between Figures 2 and 4. In Figure 2 the actors submodel is placed outside the core of the process, and with reference to the model of Figure 2 it was suggested that allocation of roles to actors can be delayed. In Figure 4 the "Function allocation" box starts off the process. The apparent inconsistency arises because Figure 2 represents an abstract model, but Figure 4 relates to a concrete software process. While actors need not be of primary concern in the abstract model, a concrete implementation cannot begin before the roles assigned to software have been fixed. Roles can be reassigned at a later stage, but an initial assignment as in Figure 4 is essential. Figure 4 can be regarded as an elaboration of Figure 2 - the objectives and the activities and usage submodels of Figure 2 have become the "Environment" box in Figure 4, the actors submodel has become the "Function allocation" box, and the rest of Figure 4 deals essentially with the conversion of requirements into an information-control system, first as a prototype, and then as a released system.

The requirements document is submitted to a prototyping process, a prototype emerges from this process, and the prototype is tested. There can be great variability in what the

"Prototyping process" and the "Prototype test" boxes contain, and how the two are related. For example, the "Requirements document" may contain an executable specification, i.e., a computer program, in which case there is no need for a separate prototyping process. In some software development processes the prototype test would be prototype execution with user participation. In others, certain properties of the software system could actually be proven. The "Requirements documentation" - "Prototyping process" - "Prototype test" loop is iterated until the prototype passes an acceptance review, which we regard to be a part of "Prototyping process". Testing exposes errors and some oversights, and this leads to changes in the requirements document. Requirements may also undergo modifications because of real world changes, such as the elimination of certain tax deductions. These changes originate in the "Environment" box.

The accepted prototype moves through the "Finishing process" box in which the prototype is turned into a deliverable system. This may mean the integration of prototypes of several subsystems into a composite system, and addition of ancillary features, such as a help system. We assume that another technical review is part of this process. The composite system is then put through a system test. An alternative is to start with the prototype of a kernel subsystem, perform system tests on this subsystem, and then add prototypes of other subsystems, performing system tests after each addition. In any case, the full transition from the WHAT to the HOW of Section 6 is completed in the "Finishing process" box. At this point the WHO-questions need to be fully answered as well.

An important characteristic of prototypes of real-time systems, particularly in a decentralized environment, is that they rarely provide reliable time estimates. This means that a system that has passed its prototype tests may still fail in the field because its responses are too slow. If the "System test" of Figure 4 detects this, then there are three options. First, faster hardware can be introduced. Second, the "Finishing process" can be tuned to improve the program it produces. Third, the specifications can be modified to sacrifice some other properties in favor of improved response times. We hope, though, that early application of performance engineering techniques - see, e.g., (Smith, 1990) - will have given an accurate estimate of the performance of the system as part of "Requirements documentation".

The prototyping-based development plan of Figure 4 is at best a generic scheme. Much detail has to be added to define the development plan for a particular software system. The ability to accept a planned process at all, to adapt a generic process structure to actual needs, and to continue to improve the mechanics of how the adaptation is done defines the process maturity level of an organization, as discussed in Section 3.

8 CONCLUSIONS

The comparison of three process views as carried out in Section 6 has shown incompleteness in all three views. At no level is the CMM concerned with the objectives that cause an organization to introduce a new process or reengineer a business process. Neither BRP nor SPM pay much attention to how the software process is to be managed, although Step 2 of the BRP is concerned with the selection of managers for the reengineering initiative. The BRP contains no explicit steps in which actors would be allocated to tasks. Although the detailed outline of the CMM (Paulk *et al*, 1993a) does discuss software configuration management in terms of a software repository, there is no explicit consideration of the repository as a source of software for reuse. Reuse is not considered in SPM either, and BRP does not even consider version and configuration management. These omissions illustrate the benefits that derive from a systematic comparison of several approaches that have the same objective.

Omission of a step need not be a deficiency, but may indicate differences in philosophy. Systematic comparison makes sure that an omission is not an oversight, but a deliberate decision. For example, while under SPM the actor submodel is defined early, the high level of abstraction at which BRP is kept for as long as possible implies that role assignment to actors is delayed until process implementation.

This suggests that the business reengineering process is to be highly flexible. The differences in basic philosophy are not arbitrary, but depend on the organizational culture of the enterprise being reengineered, particularly in the effect this culture has on reengineering teams. There is no single right approach. In some instances management has to select the teams, set an agenda for each team, and closely supervise the teams. In such a situation actor selection is an important early step, but actor selection need not be emphasized as a separate step when the teams are self-selecting and self-managing units comprising both management and technical personnel. To take another example, reuse of existing software is not particularly important when radical business reengineering takes place - after all, the purpose of business reengineering is to bring about significant changes in both business practices and in the software that supports the practices. When an organization already operates according to the principles that business reengineering aims to introduce, the reengineering of a particular business process may necessitate only minor changes, and maximal reuse of the existing software is then a worthwhile objective.

We note again that the composite process model we have considered is very general. Thus it applies equally well to the introduction of a new process, redesign of an existing process, or the radical reorganization of all the activities of an enterprise that business reengineering brings about. The main principle of business reengineering is to keep asking why a particular task is to be performed. This question needs to be put whenever software is being developed.

Moreover, we emphasize abstraction, which means that in early stages of system development it need not make any difference whether a system will be used in a centralized or a decentralized setting. Even the prototype of a process system that is ultimately to be distributed over several geographically separated sites can be developed at a single site. Partly this independence of the implementation mode has been brought about by technology - the transfer of a file between the United States and Norway does not take much longer than the transfer of the file between two users in the same building. Our design methodology also contributes to the independence - careful consideration of how the business processes are defined, and the association of each process with its own information base effectively decouple the processes from each other.

9 REFERENCES

Berztiss, A.T. (1993) Information transfer for decision support in distributed administrative systems, in *Decision Support in Public Administration* (eds. P.W.G. Bots, H.G. Sol, R.Traunmuller), North-Holland, 3-15.

Berztiss, A. (1993a) Concurrent engineering of information systems, in *Proc. IFIP WG8.1 Working Conf. on Information System Development Processes* (eds. N. Prakash, C.

Rolland, B. Pernici), North-Holland, 311-324.

Berztiss, A.T. (1995) *Software methods for business reengineering*. Springer-Verlag.

Bubenko, J.A. (1993) Extending the scope of information modelling, in *Proc. Fourth International Workshop on the Deductive Approach to Information Systems and Databases* (ed. A. Olive), Departament de Llenguatges i Sistemes Informatics of the Universitat Politecnica de Catalunya, Barcelona, 73-97.

Bubenko, J.A., Rolland, C., Loucopoulos, P., and DeAntonellis, V. (1994) Facilitating "fuzzy to formal" requirements modelling, in *Proc. IEEE Internat. Conf. on Requirements Eng.*.

Constantine, L.L., and Yourdon, E. (1979) *Structured design*. Prentice-Hall.

Curtis, B., Krasner, H., and Iscoe, N. (1988) A field study of the software design process for large systems. *Communications of the ACM,* **31**, 1268-1287.

Davenport, T.H. (1993) *Process innovation: reengineering work through information technology*. Harvard Business School Press.

Hammer, M., and Champy, J. (1993) *Reengineering the corporation: a manifesto for business revolution*. Harper Business.

Humphrey, W.S. (1989) *Managing the software process*. Addison-Wesley.

Jarke, M., Mylopoulos, J., Schmidt, J.W., and Vassiliou, Y. (1992) DAIDA: an environment for evolving information systems. *ACM Transactions on Information Systems,* **10**, 1-50.

Jarke, M., Bubenko, J.A., Rolland, C., Sutcliffe, A., and Vassiliou, Y. (1993) Theories underlying requirements engineering: an overview of NATURE at genesis, in *Proc. IEEE Symposium on Requirements Engineering, RE'93*.

Johansson, H.J., McHugh, P., Pendlebury, A.J., and Wheeler, W.A. (1993) *Business Process Reengineering: Breakpoint Strategies for Reengineering*. Wiley.

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

Paulk, M.C., Weber, C., Garcia, S., Chrissis, M.B., and Bush, M. (1993a), Key practices of the Capability Maturity Model Version 1.1. SEI Report CMU/SEI-93-TR-25, Software Engineering Institute of Carnegie-Mellon University.

Smith, C.U. (1990) *Performance Engineering of Software Systems*. Addison-Wesley.

Spector, A., and Gifford, D. (1986) A computer science perspective on bridge design. *Communications of the ACM,* **29**, 268-283.

Yu, E. and Mylopoulos, J. (1994) From E-R to "A-R" - modelling strategic actor relationships for business process reengineering, in *Proc. 13th Internat. Conference Entity-Relationship Approach* (LNCS No. 881, ed. P. Loucopoulos), Springer-Verlag, 548-565.