A PROCESS MODEL APPLICABLE TO SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING*

Silvia T. Acuña†, Natalia Juristo††

† Facultad de Ciencias Forestales, Universidad Nacional de Santiago del Estero
Argentina
Fax: (54 85) 22 2595/24 1075
E-mail: silvac@unsere.edu.ar

††† Facultad de Informática, Universidad Politécnica de Madrid
Spain
Fax: (34 91) 336 74 12
E-mail: natalia@fi.upm.es

Correspondence to:
Natalia Juristo
Facultad de Informática, Universidad Politécnica de Madrid
Campus de Montegancedo, s/n, Boadilla del Monte
28660 Madrid, SPAIN
Phone: (34 1) 336 6922
Fax: (34 1) 336 74 12
E-mail: natalia@fi.upm.es

* **ACKNOWLEDGMENTS:**

This paper was prepared and written with the collaboration of CETTICO (Centre of Computing and Communications Technology Transfer, Spain).

† The work reported in this paper was completed during a sabbatical at the Universidad Politécnica de Madrid.

A PROCESS MODEL APPLICABLE TO SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING

ABSTRACT

Software engineering (SE) and knowledge engineering (KE) develop software systems using different construction process models. Because of the growing complexity of the problems to be solved by computers, the conventional systems (CS) and knowledge-based systems (KBS) software process is at present passing through a period of integration. In this paper, we propose a software process model applicable to both CS and KBS. The model designed is declarative, that is, it indicates what is done to build a software system. Its goal is to provide software and knowledge engineers with a techno-conceptual tool to develop systems comprising both traditional and knowledge-based software.

Keywords: Software process, software engineering, knowledge engineering.

1. INTRODUCTION

The convergence between SE and KE has slowly gestated at the level of methods, techniques, tools and procedures [1]. However, it is also important to consider the CS and KBS construction process. The experimental nature of KE goes hand in hand with a style of software development best characterized as *exploratory*, which has not been much studied in traditional SE [2]. It is conceivable that there may be mutual benefit to be gained from trying to synthesize SE and KE construction processes. This synthesis should lead to the definition of a software process model that could be used in both branches of engineering, but, above all, in integrated projects that contain conventional and knowledge-based software.

The description of the software construction process is a subject studied in SE for many years now. In 1991, the IEEE published a qualitative, informal and prescriptive model. Since then many other proposals have emerged [3] [4] [5] [6] [7] seeking to formalize and automate the construction process.

The situation in KE, however, is quite different. The issue of the technical activities to be performed to build a KBS was debated in the 80s [8] [9] [10] [11] [12], but KE has never taken an interest in fully defining all the activities to be performed when building a KBS, including management activities and support activities.

This paper seeks to propose an informal, qualitative process model. It is an original contribution especially to KE where there is no specific software process. Although our proposal overlaps

with SE, there are substantial differences from existing models: it attaches more importance to the early development phases (adding new activities to these phases).

This paper presents a qualitative, informal, prescriptive model, whose importance lies in the fact that it has adopted, integrated and unified the activities for constructing CS and for developing KBS. So, the process model proposed here can be followed for both building CS and developing KBS and is especially suited for integrated development projects. If consensus is reached on an integrated process model for SE and KE, a move could be made towards formally modelling and even automating that process. As Riddle said [13], "Developing a process model is a process of gradual accretion and structuring of information. Done totally logically, this is a process of gradual elaboration of what is known about the process. In a totally logical approach, one starts with a very general description and iteratively *binds* various pieces of the description to make it more and more specific". This paper addresses the first step specified by Riddle, namely, a general description, as this is the first attempt at describing a process that integrates SE and KE.

2. DISTINCTIONS BETWEEN LIFE CYCLE AND SOFTWARE PROCESS

Originally, the term life cycle was always employed. Even though the notion of software process was present in each and every software development effort, the software process was not clearly identified [14]. As software was studied in more depth, the software process acquired an identity in its own right and has been a subject of research and investigation in recent years. Many software process models have been designed for structuring, describing and prescribing the process of building a software system, for example [3] [4] [5] [6] [7].

The concepts of life cycle and process are so closely related that confusion often arises. Our view is as follows:

* *Life cycle*: all the **states** through which the software evolves. The life cycle centres on the product, defining the states through which the product passes from the start of construction (the initial state is the user need) until the software is in operation (this product state is the deployed system) and finally retired (the state is the retired system). The states through which a future software system passes until it reaches maturity are also referred to as products, as they can be viewed as the intermediate results of the construction project [15] [16]. To understand why the life cycle is product oriented, consider the following analogy: the human life cycle is infancy, childhood, adolescence, youth, adulthood, old age, that is, the changing states of human beings from when they come into until they leave the world.

  A *life cycle model* is an abstract representation of the software life cycle. In the case of the software product, there is no one life cycle. A product can pass through different states, depending on the specific circumstances of each project. For example, if the problem is well

defined and well understood and the user need is practically invariable, a *short* life cycle, such as requirements specification, design, software system [17], is likely to be sufficient. However, when we are up against a poorly defined and poorly understood problem and a highly volatile user need, we can hardly expect to output a full requirements specification at the start. In this case, we have to opt for alternative life cycles [18]. For example, a prototyping life cycle, where the first state will be a prototype; or an incremental life cycle, where the first state will be the specification of the system kernel, followed by the kernel design and finally implementation, then going back to specify the next system *increment* and so on. So, there are different life cycles for different project circumstances. Life cycle models describe or represent these different types of possible life cycles.

* *Software process*: a series of **activities** undertaken to develop and maintain software systems. That is, the software process centres on the construction process rather than on the product(s) outputted. An organization may define its own manner of producing software. However, certain activities are common to all software processes. A *software process model* is an abstract representation of the software process. Different process models may represent different points of view. For example, one model may define the agents involved in each activity, while another may centre on the relation between activities and which products are interchanged. That is, each model observes, focuses on or gives priority to particular points of such a complex world as software construction [19] [20]. Furthermore, there are a host of notations for representing process models, each especially suited for a particular process approach.

Table 1 summarizes the differences between life cycle and software process, explaining that these are two different, albeit closely related concept.

Bearing in mind the above definitions, this article centres on the software process and does not refer to the software life cycle. A common or integral software process model is proposed providing for the construction of systems composed of traditional and knowledge-based software.

The paper is organized as follows. After reviewing the common core of software construction in SE and KE in section 3, the activities which could be common to the construction of each software type are discussed in more detail in section 4. In section 5, we discuss model automation and the conclusions of the paper are given in section 6.

3. SE AND KE SOFTWARE PROCESS

In both SE and KE, the software process has a common objective: to build and maintain a software product that satisfies a need detected by a user. In both SE and KE, the software process is a modelling process that should unite or connect two different worlds: the problem world and the computer world. The integral software process model to be used in both disciplines, SE and KE, seeks to do just this: define a series of activities to be performed to produce software (whether conventional or knowledge-based).

According to Blum [21], software construction starts with the development of *problem models* which are then converted into *system models*, that is, implemented products (Figure 1). Blum employs the terms *conceptual model* and *formal model*. However, we decided to replace these terms by problem model and system model, respectively, as the terms originally used by Blum can lead to confusion if the conceptual models are represented in formal languages.

The problem model describes the user need, the problem to be solved by the future system and the problem-solving mode. This model is used by engineers to understand the domain, the user, the problem and its solution. On the other hand, the system model describes the software system, its structure and composition. This model is used by engineers to prescribe the construction of the software that is to solve the problem described and satisfy the user need [21]. So, software construction involves two classes of analysis. The first centres on the problem space (problem modelling) and the second focuses on the implementation space (system modelling). This view of software construction is valid for both KBS and CS construction.

SE has taken the view that needs could be accurately defined (that is, the requirements were closed) and that problem models were in fact quite close to system models. As we move towards more open problems (open both in initial understanding and to change over time), the gulf between the expression of the problem and the representation of the system that solves it widens. KE, on the other hand, addresses knowledge-dependent problem solving. KE seeks to replicate the expert problem-solving mechanism in the computer. This approach means that the problem modelling stage is a long process in KBS construction, because it is difficult to elicit the expert reasoning mechanism. This has led to the creation of expressive representation formalisms for the system models that can be transformed automatically into an implementation.

Growing software complexity calls for planning, monitoring and control, and evaluation of development projects. Therefore, SE considers all the management, support and development

processes involved in the project, from the identification of a need to the retirement of the software.

The traditional software process, as per IEEE standard 1074-1991, is composed of four main processes, each grouping a series of activities that are responsible for implementing their associated goals [22]. These processes are shown in Table 2. Although it has not been formally defined by any organization or document, the KE software process can be considered to be composed primarily of the processes shown in Table 2, as all of these activities appear in one way or another in the most well-known KE development methodologies [8] [9] [10] [11] [12].

Below we discuss which of the activities of SE could be applied in KE and vice versa. As we shall see, some SE activities are not explicitly set out in KE, although they are in fact often already performed as they are implicit in the KBS building methodologies. Table 3 shows the processes for building software developed by the two disciplines. An arrow at the end indicates that the process is likely to be exportable to the other discipline, whereas an arrow at the start and at the end means that the processes are complementary. Similar processes in the two disciplines are linked by a plain line.

3.1. What SE Software Process Activities Could Be Applied to KE?

The following SE processes have been neither defined nor described in KBS development: identification and selection of a life cycle model, project management, and configuration management, documentation and training. It is worth considering whether these SE process activities would be useful in KE.

- The *selection of a life cycle model* for a project enables software engineers to organize, plan, support, budget, programme and manage a software project, and prescribes what documents are to be produced. It is reasonable to assume that KBS projects also need to be organized, planned, supported, budgeted, programmed and managed, and therefore this process should be exported to KE. This means that the software life cycle for the CS construction process can be useful for the process of building KBS. Obviously, as KBS have their own particular characteristics, the range of life cycle models from which to choose to develop this type of systems would differ from those used for conventional software. Indeed, traditionally, the life cycle models most used in KE are incremental development and prototyping.
- Planning, monitoring and control, and evaluation of development projects is required in order to commercialize KBS, that is, controlled resource consumption and performance of commitments entered into with the customer is also of interest in this type of projects. Therefore, *project management* activities should be carried out in KBS construction and can be included in the integral software process.

- Moreover, a technically impeccable development is not sufficient in order to guarantee the success of a KBS, the user needs to be involved in the process for the system to be accepted in its environment and used efficiently. Therefore, all of the user-related integral activities – *documentation, training, evaluation*– should also be performed in KE.
- *Configuration management and quality management* in SE are a response to problems that arise when large systems are built. These problems can be expected to recur when KBS pass from programming-in-the-small to programming-in-the-large [23]. Therefore, these activities could improve the results of KBS construction.

The differences are bigger in Development-Oriented Processes, primarily on the following grounds [1]. Mainly because KE solves poorly structured and poorly defined problems, it is difficult to analyse and even more so to define requirements. In KBS development, problem understanding often continues throughout the entire life cycle. However, there is an increasing tendency to make distinctions between requirements that cannot be specified at the start of a KBS due to poor problem structuring and other requirements that are typical of any software system and can be specified [24].

On the other hand, as the complexity of the problems addressed by SE grows, it moves further and further away from the myth of closed requirements at the start of development. Therefore, one of the issues that used make SE and KE development processes irreconcilable (that is, starting the development process with closed requirements) is fading fast. Another point separating SE and KE processes is that there are far more problem analysis activities in KE than in SE. Again this is because the domains to which KE is applied are complex, the problems addressed are poorly structured and the problem-solving process is not well enough understood. The possible integration of the activities involved in the technical processes is considered below.

- *Requirements analysis and specification* as used in the traditional software life cycle are not applicable in KE. Nonetheless, as SE comes up against problems with requirements that are difficult to understand at the start of the project and change over time, strategies of analysis are being developed that are closer to those used in KE (discussions held with the user are being extended and given more weight, prototypes are being used, etc.) and are moving away from the traditional view of requirements being frozen after their specification, thus bringing the view of requirements in KE and SE closer together.
- The fundamental principles of conventional software *design* (modularity, abstraction, refinement, functional independence and information occultation) are applied to reduce complexity, facilitate implementation and changes, and thus assure product quality.

Therefore, good design guidelines must be followed in code design and programming, whatever the type of code developed [25].

3.2. What KE Software Process Activities Could Be Applied to SE?

Having discussed what SE software process activities can be applied in KE, let us review the software construction process in KE and discuss what activities could improve the SE software process.

- Once the problem has been defined or the need has been identified, a *feasibility study* is carried out in order to decide whether the problem can be addressed by KE. This special feasibility study emerges as a result of the need to ascertain which problems can be processed by KBS and which by classic software. These problems arise because KE techniques are alternatives to the classical approach to systems development, and a decision of this type needs to be made at the start of project development to ensure that there is agreement between the team of engineers and the customer. In an integral software process that can be used by software and knowledge engineers, after exploring the domain and gaining an understanding of the problem posed, it is essential to include both the cost/benefit analysis of alternative approaches to a particular project and activities that make up the special feasibility study in order to decide on the type of software solution required by the problem.
- Of the processes that make up the KBS construction process, the *knowledge acquisition and conceptualization processes* have no explicit equivalent in SE. Acquisition and conceptualization activities are performed because of the characteristics of the problems processed by KE: complex domains, non-algorithmic and non-explicit problem-solving processes. Compared with a traditional software engineer, a knowledge engineer needs a lot more time to understand the domain and the expert problem-solving process. As the expert has internalized this process and it is not explicit, elicitation is a demanding and difficult task. Additionally, once the knowledge engineer gains an insight into the expert's picture of the world and the pieces start to fit together, he/she must start to conceptualize all of the knowledge being elicited and make it explicit in a problem model.

  In view of the increasing complexity of the problems addressed by SE, there is a need to go deeper into domain and problem exploration and comprehension and, as a result, the software engineer must interact closely with the current problem solver to understand the problem-solving process and the application environment. System analysis requires intense communication between the customer and the analyst. The customer must understand the system objectives and be able to explain them clearly. The analyst must know what questions to ask, what advice to give and what investigations to undertake; in short, which is the best

way of extracting information. If communication breaks down, the success of the entire project is jeopardized. Although the domains traditionally addressed by SE are better known, the problems are easier to understand and the problem-solving processes are often explicit and known algorithms, now that SE is faced with the challenge of solving more complex, albeit algorithmic, problems, the time spent interacting with the user as opposed to with the computer will grow. Indeed, the activity known as requirements elicitation is becoming increasingly important within the analysis phase. So, in order to achieve action efficiency, we have to resort to information/knowledge acquisition and analysis activities that elicit and conceptualize the problem-solving knowledge and extract and model knowledge of the domain and the information, skills, behaviour, tasks and actions of the users. While no expert is involved in CS, the user from whom the information is extracted, who is highly skilled in performing his/her job, is really an expert in a broad sense. Although he/she does not use expertise, he/she has some specific knowledge as to his/her job in the organization and properly trained to perform the job efficiently. Experts and specialized users can be grouped under the term solvers.

The trend in SE is towards the need to solve poorly specified, increasingly complex problems with shallow knowledge of the problem domain. SE will be dominated by the following trends: open and dynamic requirements, reuse, integration and diverse computational models [26] [27] [21].

- *Formalization and implementation* activities are particular to KBS as they are responsible for representing the problem model outputted by conceptualization using formalisms proper to KE. It appears, therefore, that these activities cannot be exported to SE, although they should be addressed by the integral software process in some manner.

4. PROPOSAL FOR AN INTEGRAL SE AND KE SOFTWARE PROCESS

Based on the traditional SE processes [22] and the processes defined in Table 3 for knowledge-based software construction, an integral software process has been designed whose objective is to assist in the development of software in both SE and KE. An overall description of the process is given below, detailing the subprocesses and original activities or activities considered to be of more interest for the purposes of the integration sought after. The IEEE elects to represent the software process as a decomposition of the entire process (outside box) into subprocesses (project management, development-oriented processes and integral software support) and further into subsubprocesses. That is, the software process, as represented by the IEEE, is the decomposition of the entire software process into simpler processes.

As mentioned earlier, all software is an automated response to a real-world need; there is a process that transforms the need into a computerized solution; software development involves a series of activities that start, plan, manage and support the project and that develop and evaluate a series of products ending with the software system. Therefore, the common software process needs a subprocess that provides for the selection of a software life cycle model (SLCM), which will become its axis, and another three subprocesses: one that manages the project, another that models the products and a third that assists with modelling. We have named these subprocesses respectively: Software Life Cycle Model Process, Project Management Processes, Software Modelling Processes and Integral Project Support Processes.

In other words, the four basic processes in the IEEE standard 1074-1991 are also valid for KE. The first two processes keep the names allocated in the IEEE standard, as the name of the processes exactly matches the activities to be performed whether in SE or in KE or joint development. The latter two processes proposed by the IEEE have been renamed. Figure 3 shows the new software process proposed, which meets the needs for producing software systems that process data, information and knowledge.

The processes that are referred to as Development-Oriented Processes in the above-mentioned standard now form a group named Software Modelling Processes in the SE and KE process. This name was chosen because their goal is achieved and their job performed by advancing from one model to another, where program code is just a model of the detailed design model and system operating documentation is just a model of system and user behaviour, etc. The Modelling Processes are divided into Exploration Processes, Problem Understanding Processes, System Construction Processes and Utilization Processes that correspond to the exploratory, problem, system and utilization models, respectively. Thus, exploratory models are created for problem formulation and feasibility determination; problem models for need definition, requirements specification and knowledge determination; system models for design and implementation representation, and utilization models for software installation, use, retirement and maintenance.

The processes that assist in Software Modelling Processes and are necessary for successfully completing the activities in the software project are referred to as Integral Project Support Processes: *integral* because they are applied throughout the entire software construction process and to all of the aspects involved in each subprocess and *support* because they support reliable software construction (Verification and Validation Process, Configuration Management Process) and ensure that it is used to full capacity (Training Process, Documentation Process). Additionally, the product should be developed in constant interaction with users (Information

and Knowledge Acquisition Process). Integral Support Processes are an essential aid throughout the entire process of quality software construction. They are activated by Project Management Processes and Software Modelling Processes and by the Integral Support Processes themselves. They are divided into Quality Protection Processes, including Verification and Validation and Configuration Processes, Technology Transfer Processes, including Documentation and Training Processes, and, finally, the Information and Knowledge Acquisition Process.

As the activity input and output definition is complete and robust, certain processes specified by IEEE standard 1074-1991 have been exported to the proposed model, although some change name. Other processes are given the same name as in the standard but include new activities. Three processes are completely new as compared with the standard: the Domain Study Process and the Knowledge Analysis Process within the Software Modelling Processes and the Information and Knowledge Acquisition Process within the Integral Project Support Processes.

So, the main differences from IEEE standard 1074-1991 are the Software Modelling Processes and the Integral Project Support Processes as shown in figures 5 and 6. Below, we review these two processes showing how the processes and activities are defined so as to be applicable to both classical software and knowledge-based software development.

4.1. Software Modelling Processes

The development of software is basically a modelling process, involving the construction of a series of intermediate models at different levels of abstraction. Modelling enables the software/knowledge engineer to handle the complexity of a software system mainly using the strategy of “divide and conquer”.

As shown in Figure 1, the essential and characteristic job of the modelling processes can be said to be to build different types of models: descriptive (for the problem model) and prescriptive (for the system model). The problem model corresponds to the Problem Understanding Processes and the system model to the System Construction Processes. Each model undergoes successive reviews until it resembles the real thing as acceptably as possible. In the case of the problem model, the *real thing* is the user domain, need, problem and solution; in the case of the system model, the *real thing* is the system to be produced.

The processes of Exploration, Problem Understanding, System Construction and Utilization, into which the Software Modelling Processes and their respective subprocesses are divided, are discussed below. Figure 5 shows these processes in the proposed model as compared with the processes defined in IEEE standard 1074-1991.

4.1.1. Exploration processes

These comprise the study, analysis and diagnosis of the situation, which provides for appropriate problem recognition and formulation, precise delimitation and definition of the need for the computerized solution and determination of the feasibility of the system to be developed.

Generally, a problem arises when somebody perceives a difficulty or thinks the current state of affairs could be improved. When a problem has been detected, the next step in problem identification will be to formulate or conceptualize the problem adequately. Once the aspects of the problem that is to be solved have been precisely defined and they have been precisely and clearly described, that is, once the problem has been formulated, problem analysis continues in order to identify the idea or need for the system to be developed, whether this is a new application or a change of all or part of an existing application, and to formulate potential solutions, considering their limitations and benefits.

Accordingly, Exploration Processes involve two processes: a) the Domain Study Process, which is required in complex software systems development and b) the Feasibility Study Process that corresponds to the Concept Exploration Process of IEEE standard 1074-1991, and also analyses how well the problem domain fits in with KE methods. This study will output the type of solution: buy, reuse, develop algorithmic software, develop knowledge-based software or others.

4.1.1.1. DOMAIN STUDY PROCESS

This process seeks to gain a perspective and a mental comprehension of a given phenomenon in the real world. It usually starts with the engineer familiarizing him/herself with the problem that is to be studied, which includes processing the bibliography on the matter, specialist and expert opinions, personal experiences, etc., and this is done by means of the Information and Knowledge Acquisition Process. The engineer needs to understand the problem domain and the general terminology used in order to form a preliminary or mental map of the problem domain and acquire the vocabulary needed to dialogue with the user.

Once this has been done, the aspects of the problem to be solved are defined as accurately and described as clearly as possible. Throughout this process, an effort must be made to achieve maximum conciseness, clarity and precision as regards the problem studied, considering its objectives, the scope of the problem domain and the relations between the subproblems of which it is composed. If the software is to be intelligent, this process is also oriented towards getting an overall understanding of the human expert’s task. It seeks to understand and make explicit both domain and expert knowledge.

The following must be identified and analysed in the Domain Study Process: a) the goals of the user's organization; b) the organization's problems; c) functions of the future solution; d) solution environment; e) tasks requiring expert knowledge, and f) user tasks, in order to establish the current status of the organization and define the domain of the problem to be solved. This process outputs: (a) a *Current Status Model* that includes a list of user organization objectives with the current status and desired priority, a description of the organization's problems, the current sequence of operations performed, identification of system users and their requirements, how experienced the solver and user is and his/her role, and (b) a *Problem Domain Model* that includes the main concepts and relations between these concepts, functionality and specification of the operating environment, showing the relations between the problem domain and its environment (users, other software, physical systems, etc.) and what data, information or knowledge will be interchanged.

The importance of the Domain Study is directly proportional to the complexity of the problems processed.

4.1.1.2. FEASIBILITY STUDY PROCESS

Once the problem has been studied and defined, this process triggers the modelling effort with the identification of an idea or need for a system to be developed, whether it is a new application or a change of all or part of an existing application. It includes the identification of an idea or need, the formulation of potential solutions, their evaluation (feasibility study) and refinement at the system level [22]. If all or part of the alternative to be developed is a KBS, it should be determined beforehand whether the solution is feasible applying KE techniques and methods [28] [29]. Once system constraints have been established, the *Statement of Need* for the system to be developed is generated, which fires the Problem Understanding Processes and feeds the Project Management Processes. The Statement of Need is the document that constitutes the basis of all later engineering work.

The Feasibility Study Process describes the need and the solution to be implemented (software solution) in the Statement of Need. It includes descriptions of the application domain, needs and expectations selected, the characteristics of the software and its impact on the organization. It is important because it supplies timely and suitable information to the management levels for making the decision to start software solution construction.

4.1.2. Problem understanding processes

Problem understanding processes comprise conceptual analysis and modelling that provide for problem definition, an understanding of the relevant knowledge relations and problem-solving processes, and specification of software requirements.

They involve the Environment Analysis, Knowledge Analysis and Requirements Analysis Processes that strongly interact with the Information and Knowledge Acquisition Process in order to get the necessary and relevant information and knowledge about the environment, knowledge and requirements models, respectively. The Knowledge Analysis Process constitutes an essential process in KBS construction. The Environment Analysis Process and the Requirements Analysis Process are based on the System Allocation Process and the Requirements Process, respectively, in IEEE standard 1074-1991. However, some changes need to be made, especially to the Requirements Process, to account for the fact that not all the requirements can be defined and frozen at the start of the KE project. As mentioned earlier, as classic software applications become more complex and move into new domains, this need to process open requirements is increasing in SE. As the integral software process proposed here does not determine any order for process performance, the engineer will *navigate* from one of the three conceptualization processes to another, where one process will support the others and none of the processes will be concluded until all three are complete, producing: adequate requirements, an adequate knowledge model and an adequate architecture.

The three conceptual models (Environment, Knowledge and Requirements) describe how the solvers, users and software and knowledge engineers view the environment, the relations between relevant concepts and software product requirements. It is important for all of the viewpoints to converge in a single representation of the system. This representation shows what the software is to do and when and how it is to do it and what knowledge it is to use.

At the end of these processes, we will get a description of the relationships between the software and its environment, a description of domain knowledge, a list of all the selected needs and expectations and a list of the software product characteristics.

4.1.2.1. ENVIRONMENT ANALYSIS PROCESS

This process is performed to set the software in its external environment. It is especially valid when the software is to be embedded in a bigger system.

The Statement of Need is the basis for environment analysis, identifying the inputs, required outputs and full system functions. A *Functional Description of the System* and the *System*

Architecture are specified. This process of environment definition concludes with the *Functional Software Requirements Specification, Functional Hardware Requirements Specification* and *System Interface Specification* [22].

The most important functions of the system to be built are specified in the software system requirements in order to define what the software is to do to achieve each objective and make it useful for the system end users. These functions are described in general terms and refined throughout the project until exact functionality is obtained. If the software to be built has an intelligent component, the outputs of this process will be defined at a higher level than for systems that have no such component. However, as this process interacts with the Knowledge Analysis Process, these outputs will be refined and, ultimately, it will be possible to specify system requirements, before moving on to the System Construction Processes.

4.1.2.2. KNOWLEDGE ANALYSIS PROCESS

This process is designed to organize all of the concepts, relations, inferences, strategies, tasks, etc., that are elicited from the Information and Knowledge Acquisition Process in order to understand, adapt and later model the domain and problem solver behaviour. Accordingly, the Knowledge Analysis Process seeks to define all the existing concepts, attributes and functions, which generates a static and dynamic knowledge structure, enabling the engineer to represent his/her understanding of solver knowledge and the solver to identify conceptual errors on the part of the engineer; this structure is referred to as a knowledge model. This involves a two-stage process of structuring the knowledge acquired: Analysis and Synthesis. The type of activity to be performed as part of this process is outlined below. In order to be as generic as possible, we will refer to the models obtained as the static model (or structural model) and the dynamic model (or functional model). These generic names can be specified, depending on the approach taken to building the system, as procedure-oriented, object-oriented, reasoning-oriented, etc.

During the *knowledge analysis* activity, the knowledge obtained during knowledge acquisition is analysed at three levels:

1. *Strategic Knowledge* is knowledge of the steps of the task performed by the expert/solver and the software flow control. It answers the question “what are the steps to solve the problem?”

   To conduct this analysis, the sequence of actions performed by the solver that are to be executed by the system are first identified. The substeps are then identified. Finally, when the subtasks can be broken down no further, they are defined. All this should be reflected in a representation –flow charts, hierarchical trees, etc.– that describes the problem-solving process.

2. *Tactical Knowledge* is a lower level of knowledge. It determines how to perform each task identified in the strategic knowledge. It answers the question “how is each step performed?” Or “what has to be done in each step?”

   The analysis of this knowledge must produce detailed definitions of each step executed to solve the problem; for this purpose, we must start by identifying all the data needed in each step. Then the individual steps are defined. These should map out the overall problem-solving structure, such as: reaching conclusions and reactions to new information. Different representations can be used to represent this knowledge, such as: decision tables, pseudorules, decision trees, formulae, etc. If the system has no intelligent component, tactical knowledge may correspond to the algorithm of each unit or module.
3. *Factual Knowledge*: This is what the system will know about the domain, and information that will be obtained by the system about the case in question when performing the task, that is, the facts taken as a starting-point by the system and the facts arrived at by the system during and after execution.

   To conduct this analysis, the information collected about each attribute has to be organized within a written definition of the attribute, then the attributes that are important for the application are classified. Later, the facts about the application area, relating concepts, processes or entities that are independent of each particular case, are organized and, finally, the interrelations between the concepts and other entities identified are defined.

For *knowledge synthesis*, the above results are used to define two models: the static (or structural) model and the dynamic (or functional) model.

– *Static Model*: This model will contain domain information: concepts, their definitions, relations between concepts, attributes, their values and the tasks or activities in which each concept participates.
– *Dynamic Model*: This model will record the tasks to be performed by the system, the inputs that activate these tasks, the results obtained and task execution control.

In sum, the engineer can be said to have concluded domain analysis when he/she has defined the domain concepts, relations, attributes, values and functions. It is then that he/she understands the problem and how it is solved. Therefore, the engineer now understands what the software system is to do and how it is to do it, and can, therefore, go ahead with its construction, passing through the system construction processes.

4.1.2.3. REQUIREMENTS ANALYSIS PROCESS

This process includes iterative activities targeting the development of software requirements (functional, performance and interface) which started in the Environment Analysis Process.

Software Requirements Specification (SRS). It is the document outputted at the end of this process. Whereas the Environment Analysis Process centres on the description of the domain (what there is in the domain and how domain tasks are performed), Requirements Analysis focuses on the user (what users expect of the system, what they want it to do, etc.). Note that these are two complementary views, and both are necessary in the construction of any software system type.

In the case of systems with open requirements, such as KBS, the *a priori* determination of functional requirements is confined to the specification of minimum high-level functional requirements. These are gradually refined during the requirements analysis process considering the conceptualizations output in the Knowledge Analysis Processes and even in the Preliminary Design Process, until an exact and clear set of functional requirements, which provide for system validation in line with the acceptability criteria involved, is agreed *a posteriori*. This process is semi-open, perfectible and iterative as is characteristic of software requirements formulation and evaluation.

4.1.3. System construction processes

System Construction Processes seek to represent the intuitive elements addressed in the Problem Understanding Processes by means of formal languages. The objective is to develop a system model that corresponds with the problem model. They involve the Design Process, the Implementation Process and Integration Process. Faced with the problem world (user domain) and the computer world, the System Construction Processes are where the engineer considers the computer (implementation domain) for the first time in software development. The Design Process has the same name as in IEEE standard 1074-1991, but includes additional activities essential for KBS construction, such as: the selection of formalisms suited for representing knowledge and inference techniques appropriate for reasoning with that knowledge. The Integration Process and Implementation Process are equivalent to the Implementation Process set out in the standard, except that, in KBS development, they output the knowledge base code and the inference engine and must also load the knowledge base, which involves transferring the formalized knowledge to the knowledge base structure. The increasing complexity of software systems means that component integration is becoming more important. This is why system integration is considered as a separate activity from implementation.

In the System Construction Processes, we move from a qualitative to a quantitative model that represents system structure and behaviour. The system models, which determine the criteria of correctness, are built as part of these processes and the models are programmed, thus producing a software model that must be correct with respect to the system model.

The objectives of the System Construction Processes are to produce a detailed system model (Design Process) and translate that representation into an implementation in a programming language (Implementation Process and Integration Process). The process by which the detailed model is developed combines: intuition and criteria based on experience in building similar systems, a set of principles and/or heuristics that guide the manner in which the model is developed, a set of criteria for ascertaining quality and a process of iteration that finally leads to a representation of the final design.

For KBS, the output of this activity is the logical representation of the solver's knowledge depending on implementation issues; that is, that the knowledge model (defined in the Knowledge Analysis Process) is expressed formally within the framework suggested by the tool or by the programming language.

4.1.4. Utilization processes

These are processes that are to be performed to install, operate, support, maintain and retire a software product. The engineer moves from building system models that conclude with the software to constructing utilization models; that is, from activities that produce the software to activities that provide for its use.

Accordingly, the Utilization Process involves the Installation and Acceptance Process, the Operation and Support Process, the Retirement Process and the Maintenance Process. These processes are equivalent to the processes defined in IEEE standard 1074-1991, as they have to be performed in KE as well.

4.2. Integral Project Support Processes

Figure 6 shows the subprocesses making up the Integral Project Support Processes of the proposed model, which are equivalent to IEEE Integral Processes, except for the manner in which they are divided and the addition of a new and essential process: Information and Knowledge Acquisition Process. As these support processes include fundamental activities for ensuring that the system built is reliable and that it is developed and used to full capacity and in close interaction with users and solvers, they are divided into: Quality Protection Processes (Verification and Validation Process and Configuration Process), Technology Transfer Processes (Training Process and Documentation Process) and the Information and Knowledge Acquisition Process, and constitute the basis for achieving higher end user satisfaction and overall software product and process quality.

4.2.1. Quality protection processes

Software quality assurance is a “protective activity” that is applied to each software process activity. Software quality assurance combines procedures for the effective application of methods and tools, formal technical reviews, testing techniques and strategies, changes in control procedures, standard enforcement procedures and measurement and information mechanisms. So, the Verification and Validation Process and the Configuration Process are performed on the basis of software quality and constitute the Quality Protection Processes. The two processes are defined in the IEEE standard [22]. The Verification and Validation Process is extended to include techniques that can be used on KBS, providing for verification of the lexical, logical and contextual coherence of the knowledge base and KBS validation using test cases and parallel testing.

4.2.2. Technology transfer processes

Any technological system requires suitable transfer for proper deployment and routine use. Indeed, it is not the same thing for the builders to use the system as it is for the users to use the system. The only way of eliminating these differences is by meticulous technology transfer, mainly including system documentation and user training. So, the Documentation and Training Processes are responsible for technology transfer, thus providing for proper insertion of the software into the organization. Both processes are defined in the IEEE standard [22]. Technology transfer includes many more processes than just documentation and training [30]. Software developers must be convinced that they are also responsible for these transfer activities. When this responsibility is assumed, other activities ought perhaps to be included in the Technology Transfer Processes.

4.2.3. Information and knowledge acquisition process

This is the process of gathering information from several sources to build a software system. The word *information* is considered in its broadest sense, that is, data, information as such and knowledge. This process is taken from KBS software processes, where it is as important as analysis and design and involves interaction with the expert. This task most certainly came to take such an important place in KBS development because of the highly complex problems addressed by KE. This complexity made it very difficult for engineers to understand the domain and, especially, the expert reasoning and problem-solving process. As the problems processed by SE are becoming more complex and it is moving into new domains, we thought it advisable to export this activity from KE to SE.

The Information and Knowledge Acquisition Process is mainly invoked by the Software Modelling Processes (Exploration, Problem Understanding, System Construction and Utilization Processes) and takes place in parallel, supplying each process with the information as required. This process is planned and performed to acquire the information and knowledge required to understand the domain, the problem and the problem-solving process when building software systems. This process involves all the activities that plan, execute and evaluate the acquisition sessions and gather, analyse and classify the information and knowledge extracted.

Each session is performed according to any of the existing techniques [28] [31]. The information and knowledge obtained are classified as terms, concepts, relations, procedures, algorithms, inferences, etc., in the *Acquisition Report* which feeds the Modelling Process that invoked this process.

Obviously, iterativeness is an intrinsic feature of this process, as sessions have to be repeated over again until a satisfactory level of information is gained.

5. CONCLUSIONS

The paths open to SE and KE lead towards joint development, enabling the construction of systems that include traditional and knowledge-based software. Having reached this point, software developers will need a software process model for rational construction of software that meets the needs of open requirements and complex and dynamic software. The Integral Software Process Model proposed is a potential solution to this problem, as the different processes comprising the common software process discussed here are valid for both types of software.

A descriptive process model has been presented as a first step towards a prescriptive and automated process model. To achieve the second step it is necessary to reach general agreement about which subprocesses and activities make up a software process that describes CS and KBS construction. This was the objective of our paper. If software is developed according to this model, the process will be different because it includes new activities. For example, a new subprocess emerges, called information acquisition and conceptualization, which, apart from a user requirements specification, outputs problem models of the user's problem and how the user solves it (as is done in KE). With regard to KE, new processes emerge for KBS development, such as management processes, life cycle selection, configuration management, etc., activities not contained in any of the methodologies that describe KBS construction.

The type of system that best fits our process are integrated or mixed systems that have traditional and knowledge-based subsystems. Using this model, it will no longer be necessary to employ different process models for CS and KBS, which made them difficult to manage.

With the common software process, which provides for modelling complex software in relation to its dynamic environment, software and knowledge engineers will construct conventional systems and knowledge-based systems in an integral manner. The common process considers all the processes involved in the software life cycle, from problem identification through to software retirement in the framework of software management, monitoring and control, knowledge acquisition and evaluation. It contributes three original processes as compared with IEEE standard 1074-1991: the Domain Study Process, the Knowledge Analysis Process and the Information and Knowledge Acquisition Process.

The Integral Software Process Model addresses SE and KE convergence in terms of integral software development processes and activities at the level of conceptual justification and it gives conventional and knowledge-based system developers a conceptual tool to help them to produce quality software. The model proposed is a useful tool for modelling the construction of flexible and reusable, evolutionary software systems that process data, information and knowledge, which facilitates the development of systems combining intelligent software with traditional software. This paper ushers in a new form of dialogue between software and knowledge engineers, experts and users which will provide for progress in the implementation of individual integral and interdisciplinary projects as a result of the interest and experience shared.

The integral software model developed provides a visualization of the structure of the common software process, a comparison of the hypotheses formulated in the software construction models by the software and knowledge engineer, a deepening of the knowledge on which activities are difficult to perform in the construction process and an explanation of the activity of complex software problem solving. Studies that fully describe the process activities and their logical relations need to be conducted, SE methods and techniques need to be adapted for KE and KE methods and techniques for SE, and the proposed model needs to be validated in order to improve model capabilities.

REFERENCES

1. F. Alonso, N. Juristo and J. Pazos, "Trends in life-cycle models for SE and KE: proposal for a spiral-conical life-cycle model approach", *Int. J. Software Engineering and Knowledge Engineering*. **5**, 3 (1995) 445-465.
2. L. Ford, "Artificial intelligence and software engineering: a tutorial introduction to their relationship", *Artificial Intelligence Review*. **1**, 4 (1987) 255-273.
3. J. Lonchamp, K. Benali, J. C. Derniame and C. Godart. "Towards assisted software engineering environments", *Information and Software Technology*. **33**, 8 (October 1991) 581-593.
4. W. Deiters and V. Gruhn. "Software process analysis based on FUNSOFT nets", *Systems Analysis Modelling Simulation*. **8**, 4-5 (1991) 315-325.
5. M. H. Penedo and C. Shu. "Acquiring experiences with the modelling and implementation of the project life-cycle process: the PMDB work", *Soft. Eng. J*. **6**, 5 (September 1991) 259-274.
6. M. I. Kellner. "Software process modelling support for management planning and control", in *Proc. 1st Int. Conf. Software Process*. Redondo Beach, California, USA (21-26 October 1991) 8-28.
7. S. C. Bandinelli, A. Fuggetta and C. Ghezzi. "Software process model evolution in the SPADE environment", *IEEE Trans. Soft. Eng*. **19**, 12 (December 1993) 1128-1144.
8. M. A. Carrico, J. E. Girard and J. P. Jones, *Building Knowledge Systems: Developing & Managing Rule-based Applications*, McGraw-Hill, 1989.
9. B. J. Wielinga, A. Th. Schrieber and P. de Greef, *KADS: Synthesis.* Document Y3, Project ESPRIT KADS, Amsterdam University, 1989.
10. R. Alberico and M. Micco, *Expert Systems for Reference and Information Retrieval*, Great Britain: Mockler Corporation, 1990.
11. P. Harmon and B. Sawyer, *Creating Expert Systems for Business and Industry*, New York, NY: John Wiley & Sons, 1990.
12. J. S. Edwards, *Building Knowledge-based Systems, Towards a Methodology*, Biddles, Ltd., 1991.
13. W. E. Riddle, *Fundamental Process Modeling Concepts*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.
14. B. I. Blum, *Software Engineering: A holistic view*,, Oxford University Press, 1992.
15. W. Scacchi, "Models of software evolution: life cycle and process", *SEI Curriculum Modules*, SEI-CM-10-1.0, Carnegie Mellon University, 1987.

16. M. Dowson, B. Nejmeh and W. Riddle, Fundamental Software Process Concepts, in *Proc. First European Workshop on Software Process Modeling*, AICA Press, 1991.
17. W. W. Royce, *Managing the Development of Large Software Systems*, IEEE WESCON, 1970, 1-9. Currently available in Proceedings ICESE9, IEEE/ACM, New York, 1987.
18. B. W. Boehm, "A Spiral Model of Software Development and Enhancement", *Computer*. (May 1988) 61-76.
19. M. Dowson, B. Nejmeh and W. Riddle, Concepts for Process Definition and Support, in *Proc. Sixth Intern. Software Process Workshop*, Rock Mountain Inst. of Software Engineering, October 1990.
20. P. Feiler and W. Humphrey, Software Process Development and Enactment: Concepts and Definitions, Proc. Second Intern. Conf. on the Software Process, Rock Mountain Inst. of Software Engineering, February 1993.
21. B. I. Blum, *Beyond Programming: To a new era of design*, New York, Oxford University Press, 1996.
22. *IEEE Standard for Developing Software Life Cycle Processes*, IEEE Standard 1074-1991.
23. M. Shaw, "Prospects for an engineering discipline of software", *IEEE Software,* 7, 6 (November, 1990) pp. 15-24.
24. J. Rushby, "Quality Measures and assurance for AI software", *NASA Contractor Report 4187*, 1988.
25. R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 3rd Ed., McGraw-Hill, New York, 1992.
26. F. P. Brooks, "No Silver Bullet", *Computer*. **20**, 4 (1987) 10-19.
27. L. A., Belady, "From SE to KE: The Shape of the Software Industry in the 1990's", *International Journal of Software Engineering and Knowledge Engineering*. **1** (1991) 1-8.
28. A. Gómez, N. Juristo, C. Montes and J. Pazos, *Ingeniería del Conocimiento: Diseño y Construcción de Sistemas Expertos*, CEURA, Madrid, Spain, 1997.
29. J. Slagle and M. Wick, "A Method of Evaluating Candidate Expert System Applications, *AI Magazine* (Winter 1988) 44-53.
30. P. Fowler, I. García Martín, N. Juristo, L. Levin and J. L. Morant, "An Expert System In the Domain of Software Technology Transfer", *Expert Systems with Applications*, **12**, 3 (April, 1997), 275-300.
31. A. C. Scott, J. E. Clayton and E. L. Gibson, *A Practical Guide to Knowledge Acquisition*, Addison-Wesley, Reading, MA, 1991.

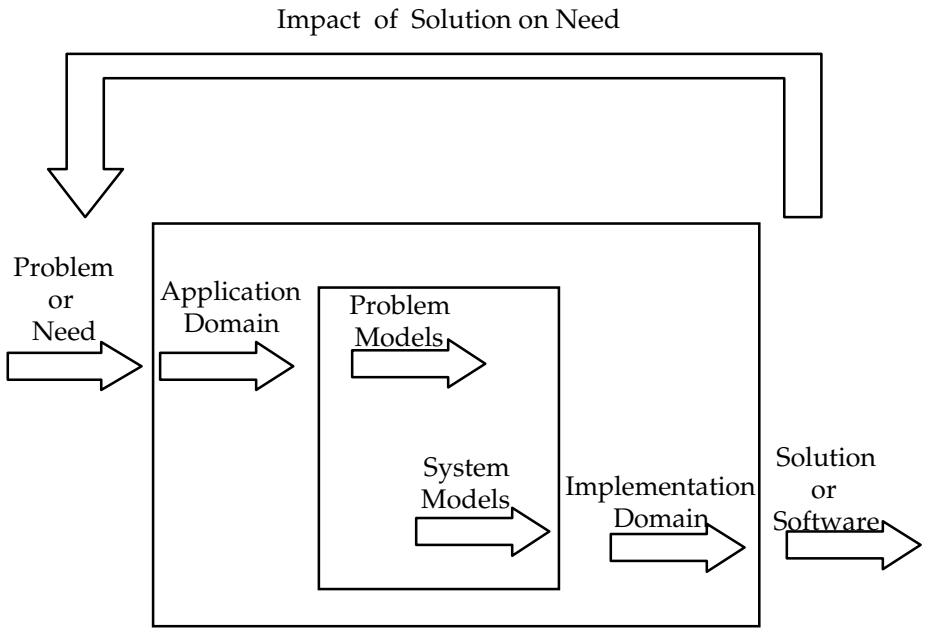

Impact of Solution on Need
Problem
or
Need
Application
Domain
Problem
Models
System
Models
Implementation
Domain
Solution
or
Software

Figure 1

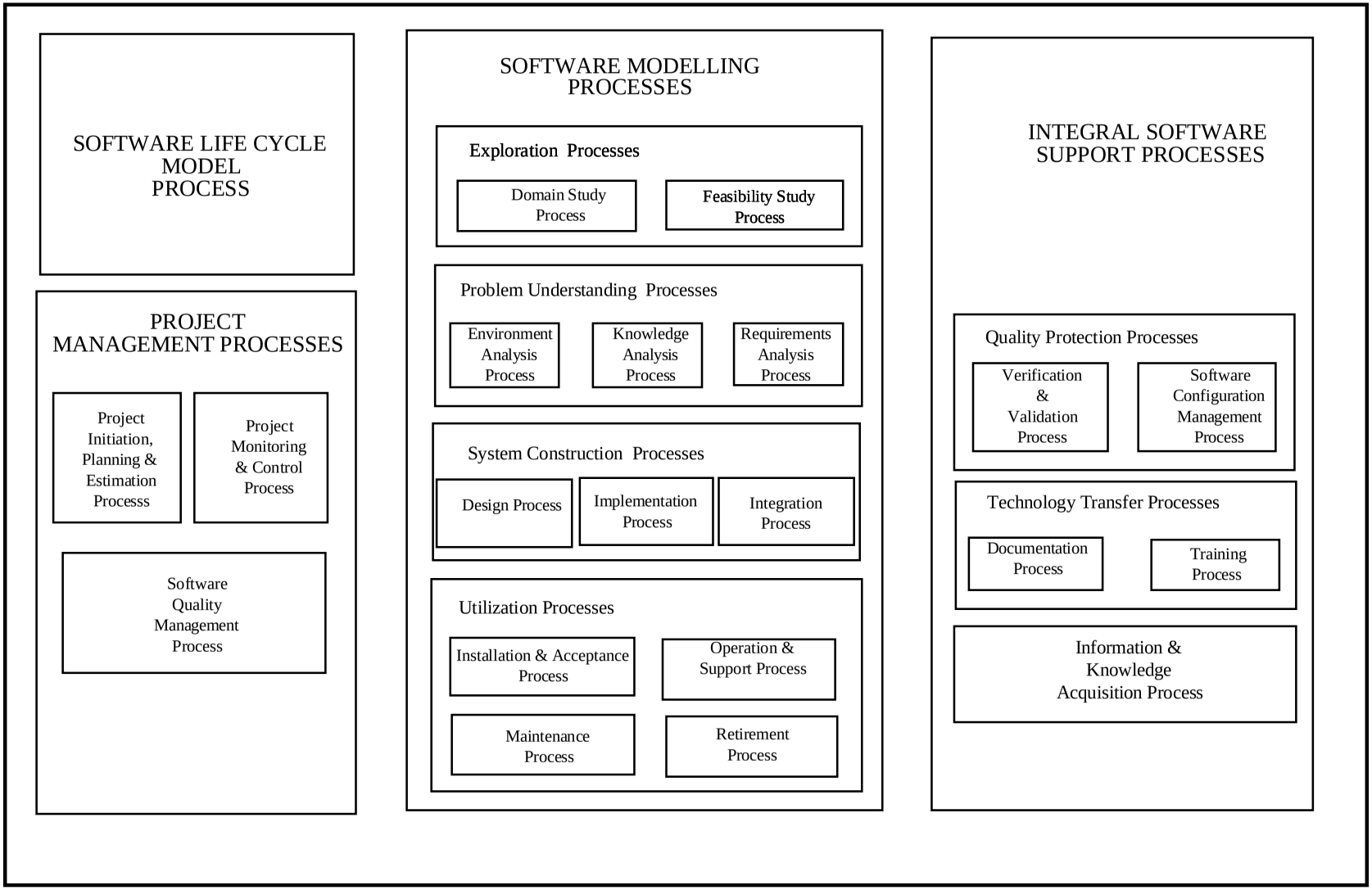
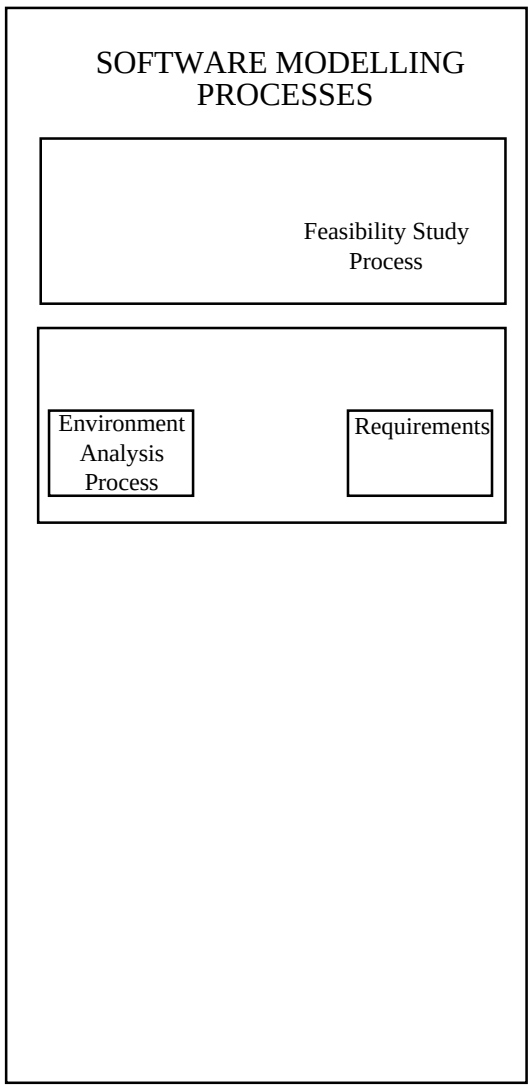

SOFTWARE LIFE CYCLE MODEL PROCESS
PROJECT MANAGEMENT PROCESSES
Project Initiation, Planning & Estimation Processs
Project Monitoring & Control Process
Software Quality Management Process
SOFTWARE MODELLING PROCESSES
Exploration Processes
Domain Study Process
Feasibility Study Process
Problem Understanding Processes
Environment Analysis Process
Knowledge Analysis Process
Requirements Analysis Process
System Construction Processes
Design Process
Implementation Process
Integration Process
Utilization Processes
Installation & Acceptance Process
Operation & Support Process
Maintenance Process
Retirement Process
INTEGRAL SOFTWARE SUPPORT PROCESSES
Quality Protection Processes
Verification & Validation Process
Software Configuration Management Process
Technology Transfer Processes
Documentation Process
Training Process
Information & Knowledge Acquisition Process

Figure 2


SOFTWARE MODELLING PROCESSES
Feasibility Study Process
Environment Analysis Process
Requirements

ERROR: stackunderflow
OFFENDING COMMAND: rlineto

STACK: