Handbook of Software Engineering and Knowledge Engineering
Vol. 0, No. 0 (2000) 000-000
© World Scientific Publishing Company

THE SOFTWARE PROCESS: MODELLING, EVALUATION AND IMPROVEMENT

SILVIA T. ACUÑA[1], ANGÉLICA DE ANTONIO[2], XAVIER FERRÉ[2], MARTA LÓPEZ[3] and LUIS MATÉ[2]

[1]Departamento de Informática
Universidad Nacional de Santiago del Estero
Avenida Belgrano (S) 1912
4200 Santiago del Estero
Argentina
silvac@unse.edu.ar

[2]Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo
28660 Boadilla del Monte
Spain
{angelica, xavier, jlmate}@fi.upm.es

[3]Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
USA
mlopez@sei.cmu.edu

Two different research fields work on the issue of software process: software process modelling on one hand and software process evaluation and improvement on the other. In this paper, the most relevant results of both approaches are presented and, despite they have evolved independently, the relation between them is highlighted. Software process modelling tries to capture the main characteristics of the set of activities performed to obtain a software product, and a variety of models have been created for this purpose. A process model can be used either to represent the existing process in an organisation, or to define a recommended software process. Software process evaluation assesses the quality of the software process used in a software development organisation, being the SCE and ISO/IEC 15504 the two most commonly used evaluation methods. Software process evaluation can be the starting point of a software process improvement effort. This effort aims to direct the organisation's current practices to a state where the software process is continuously evaluated and improved.

Keywords: software process, software process modelling, software process evaluation, software process improvement.

1. Introduction

The software process is a critical factor for delivering quality software systems, as it aims to manage and transform the user need into a software product that meets this need. In this context, software process means the set of activities required to produce a software system, executed by a group of people organised according to a given organisational structure and counting on the support of techno-conceptual tools. General concepts about software process are presented in this chapter and some specific approaches are highlighted.

The ultimate goal of research into the software process is to improve software development practice through [87]: a) better ways of organisational design at the level of individual processes and the organisation as a whole b) accompanying innovations in technological support. Hence, there are two lines of research into the software process.

1. *Software process modelling* describes the creation of software development process models. A software process model is an abstract representation of the architecture, design or definition of the software process [50]. Each representation describes, at different detail levels, an organisation of the elements of a finished, ongoing or proposed process and it provides a definition of the process to be used for evaluation and improvement. A process model can be analysed, validated and simulated, if executable. The process models are used mainly for software process control (evaluation and improvement) in an organisation, but they can be also used for experimenting with software process theory and to ease process automation.

2. *Software process evaluation and improvement* by means of which to judge and decide on the quality of the object under analysis or the evaluand, that is, the software process of a given organisation, and propose a process improvement strategy. The efforts of the scientific community in this field have led to quite a number of maturity models and standards, such as ISO 9000 [78][79][120], CMM (Capability Maturity Model)[108], ISO/IEC 15504 [77] and Bootstrap [91]. All these models have two goals: a) determine the aspects for improvement in a software development organisation; and b) reach an agreement of what a good process is. This goal stems from the very nature of the evaluation process, as it is essential to use a reference model or yardstick against which to compare the software process of the organisation under evaluation. Therefore, it involves modelling the above process by identifying what sorts of activities have to be carried out by an organisation to assure the quality of the production process and, ultimately, the end product. The existing methods include models focused on the management process, encompassing quality management [112]. Methods of this sort evaluate the maturity of the software process, that is, how well defined, managed, measured and controlled a given process is and determine an organisation's software production capability. The findings of this evaluation can include a software process improvement strategy, as the reference standard is really an evolutionary framework for improving management activities. This group includes the CMM-based appraisals (SCE (Software Capability Evaluation) and CBA IPI (CMM Based Appraisal for Internal Process Improvement)) [108] and methods like Bootstrap [91].

These are the two aspects of the software development process that are going to be addressed in this paper (sections 3 and 4 deal with software process modelling, sections 5 and 6 with evaluation and section 7 with improvement). Firstly, some basic concepts related to this subject are presented in section 2.

2. Software Process Basics

2.1. *What is the Software Process?*

Originally, the term life cycle was employed in every software development project [39]. Even though the notion of software process was present in these projects, the concept of software process was not clearly identified [16][99]. As software and software construction were further investigated, the software process acquired an identity in its own right and has been the subject of research and investigation in recent years [52][65]. The concepts of life cycle and process are so closely related that confusion often arises, calling for a clarification of these concepts. The view taken of these concepts is as follows [2].

1. *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) [119][43]. 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*, waterfall-like life cycle is likely to be sufficient. However, when we come 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 *longer* and more complex life cycles, like the spiral life cycle [18]; prototyping, where the first state will be a prototype [100] [58]; 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 [66]. So, there are different life cycles for different project circumstances. A method for selecting a model from several alternative life cycle models is presented in [6]. Descriptions of the life cycle models that represent the main life cycles are given in [119][29][111][110]. Software engineering and knowledge engineering life cycle models are compared in [135] and software engineering models in [38][29].

2. *Software process*: a partially ordered set of **activities** undertaken to manage, develop and maintain software systems, that is, the software process centres on the construction process rather than on the product(s) output. The definition of a software process usually specifies the actors executing the activities, their roles and the artefacts produced. An organisation can define its own way to produce software. However, some activities are common to all software processes. The set of processes that form a software process should be called subprocesses, but we will follow in this chapter the IEEE's terminology, calling them processes too. A *software process model* is an abstract representation of the software process. The two key international standards that prescribe processes for developing and maintaining software are IEEE 1974-1991 [75] and ISO/IEC 12207 [81]. Both standards determine a set of essential, albeit unordered activities, which have to be completed to obtain a software product. They do not prescribe a specific life cycle. Each organisation that uses the standard must instantiate the prescribed process as a specific process. ISO/IEC 12207 presents a process of adaptation that defines the activities required to tailor the standard to an individual software project. A variety of software process models have been designed to structure, describe and prescribe the software system construction process. These models are considered in section 4.

This paper deals with the software process and not the life cycle.

2.2. *Software Engineering and Knowledge Engineering Software Process*

The prescription of the software construction process is a subject that has been studied in software engineering for quite some years now. The IEEE published a qualitative, informal and prescriptive model in 1991. As mentioned above, many other proposals have emerged since then, seeking to formalise and automate the construction process. The situation in knowledge engineering, however, is quite a different kettle of fish. The issue of the technical activities to be performed to build a Knowledge-Based System (KBS) was debated in the 80s [25][134][5][62][45], but knowledge engineering has never taken an interest in fully defining all the activities to be performed to build a knowledge-based system, including management and support activities.

Although knowledge engineering has not focused efforts on outputting a full definition of its development process, it is true that the process definitions made in software engineering are not so far removed. This is because the process models are highly abstract, thus hiding differences in the development techniques required to build different types of software. The software process defines how the development is organised, managed, measured, supported and improved, irrespective of the techniques and methods used for development [41]. Therefore, even admitting the underlying differences between traditional and knowledge-based software, the work carried out in software engineering on software process definition, modelling, evaluation and improvement can be interesting for knowledge engineering.

[2] includes a discussion of the software engineering activities applicable to knowledge engineering and vice versa, and it also includes a software process model to be used in both disciplines. This model prescribes a set of processes, activities, products, roles, and abilities of the people participating in the development process of both traditional and knowledge-based systems.

2.3. *Software Process Modelling, Evaluation and Improvement*

Today, there are three main actions that can be taken with respect to the software process: define or model, evaluate and improve. The *definition of the software process* refers to the definition of the processes as models, plus any optional automated support available for modelling and for executing the models during the software process. Finkelstein et al. [52] define a process model as the description of a process expressed in a suitable process modelling language. There are other possible uses of software process models that will not be considered, such as the introduction of a new process in an organisation and personnel training/motivation.

Curtis et al. present some of the specific goals and benefits of modelling the software process [36].

1. *Ease of understanding and communication*: requiring a process model containing enough information for its representation. It formalises the process, thus providing a basis for training.
2. *Process management support and control*: requiring a project-specific software process and monitoring, management and co-ordination.
3. *Provision for automated orientations for process performance*: requiring an effective software development environment, providing user orientations, instructions and reference material.
4. *Provision for automated execution support*: requiring automated process parts, co-operative work support, a compilation of metrics and process integrity assurance.
5. *Process improvement support*: requiring the reuse of well-defined and effective software processes, the comparison of alternative processes and process development support.

Different process models can define different points of view. For example, one model may define the agents involved in each activity, while another may centre on the relationship between activities. There is a model that addresses the organisational culture and focuses on the behavioural capabilities or duties of the roles involved in the software process [3][4]. This means that each model observes, focuses on or gives priority to particular points of such a complex world as software construction [42][50]. A model is always an abstraction of the reality and, as such, represents a partial and simplified description of the reality; that is, not all the parts or aspects of the process are taken into account in the model. Generally, a process model can be divided into several sub-models expressing different viewpoints or perspectives. Additionally, there is a variety of notations for defining the process models as described in section 3.2. The results achieved so far in software process modelling research are reviewed in [7].

Software process evaluation involves analysing the activities carried out in an organisation to produce software. The ultimate goal of process evaluation is to improve software production. Development process evaluation and improvement works under the hypothesis that the quality of the software product is determined by the quality of its development process. This strategy is equivalent to the one implemented in other branches of engineering and industries, where the quality of the resulting product is increased by controlling the process used in its production [70].

Software process evaluation methods introduced innovative concepts that have changed the way in which software production activities are perceived. The new process standards and maturity models developed for use as yardsticks in the assessments are the most important innovation. However, agreement has not been reached on what a good process is and what characteristics of the process are to be evaluated.

Software process improvement studies the way of improving the software development practices in an organisation, once software process evaluation has made clear what the current state of the process is. Software process improvement is not planned in a single step to excellence, but it is performed gradually by transitions between specific maturity levels. Due to the unstable nature of software development technologies, it is necessary to continuously evaluate and improve the software process. Thus, a capable and mature software development organisation institutionalises the improvement effort.

Software process research is comprehensively reviewed in [41], while [57] presents an overall critical evaluation and possible directions for future research.

3. Concepts and Approaches of Software Process Modelling

Firstly, we are going to deal with the basic process-related elements, as well as their possible interrelationships. Then the different approaches to modelling from the viewpoint of the information they can address, and the diverse process environments will be presented.

3.1. *Basic Process Modelling Elements*

The software process basically consists of two interrelated processes: the production process and the management process. The first is related to the construction and maintenance of the software product, whereas the second is responsible for estimating, planning and controlling the necessary resources (people, time, technological, ...) to be able to carry out and control the production process. This control makes it possible to generate information about the production process, which can be used later on by the management process with a view to improving the software process and increasing the quality of the developed software. There are two main submodels in process modelling: the management process model and the production process model [41].

Different elements of a process, for example, activities, products (artefacts), resources (personnel and tools) and roles, can be modelled [68]. There are several classifications concerning the main elements of a process model [94][50][31]. The elements most commonly modelled are presented below [14][52][99][56]. Figure 1 shows these elements and their interrelationships. Other possible elements like project/organisation, workspace (repository), user viewpoint, model of co-operation, versioning, transactions, quality, etc. will not be considered.

- *Agent* or *Actor*: is an entity that executes a process. Actors can be divided into two groups as regards their close relationship with the computer: a) human actors, who are the people who develop software or are involved in the software process and are possibly organised as teams; b) system actors or system tools, which are the computer software or hardware components. An actor is characterised by the properties of its role and its availabilities. An actor, person or system can play several roles, which are composed of consistent sets of activities. A role can be judged by several co-operative actors.
- *Role*: describes a set of agent or group responsibilities, rights and skills required to perform a specific software process activity. This is an association between agents and activities in terms of a defined set of responsibilities executed by agents.
- *Activity*: is the stage of a process that produces externally visible changes of state in the software product. An activity can have an input, an output and some intermediate results, generically termed products (they can be software products, requirements specification documents, database schemata, etc.). The activity includes and implements procedures, rules, policies and goals to generate and modify a set of given artefacts. This activity can be performed by a human agent or using a tool. The activities can be divided into more elementary activities; that is, there are elementary or compound activities. The activities can be organised in networks with both horizontal (chained) and vertical (decomposition) dimensions. The activities are associated with roles, other activities and artefacts.
- *Artefact* or *Product*: is the (sub)product and the “raw material” of a process. An artefact produced by a process can be used later as raw material for the same or another process to produce other artefacts. An artefact or software product is developed and maintained in a process. An artefact can have a long

lifetime, and there may be different versions of each artefact as the software process evolves. The (sub)products can be created, accessed or modified during a process activity. A set of software artefacts to be delivered to a user is named software product. Therefore, a relationship "composed of" can appear between the products.

- *Event*: is a noteworthy occurrence happening at a specific moment in time. Event-based models provide a different view (not represented in Figure 1) of the activities by which the dynamic nature of the process is made evident.

The relationships between these elements, such as activity/role/actor interaction, activity-activity interaction, product-activity interaction, product-product interaction, are described in [14].

Definitions of the process modelling concepts are found in [30][94] and the main software process models that use these concepts are described in [52].

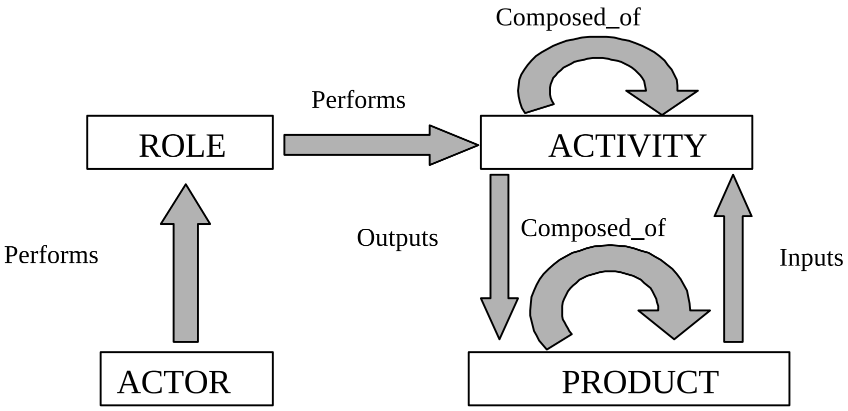


Figure 1 Basic process modelling components

3.2. *Modelling Approaches*

There are different types of process modelling. Processes can be modelled at different levels of abstraction (for example, generic models versus tailored models) and they can also be modelled with different goals (descriptive models versus prescriptive models). The distinguishing features of these models have been described in [70][96][99][110]. The types of information in a process model can be structured from different viewpoints. Curtis presents the following list of *information perspectives* commonly found in the literature [36]:

- *Functional*: represents which process elements are being implemented and which information entity flows are important for the above process elements.
- *Behavioural*: represents when (that is, sequentiality) and under which conditions the process elements are implemented.
- *Organisational*: represents where and by whom in the organisation the process elements are implemented.
- *Informative*: represents the information entities output or manipulated by a process, including their structure and relationships.

The models are built according to the languages, abstractions or formalisms created for representing the specific information about these characteristics. The most used language in practice is (structured) natural language due to its flexibility. Table 1 presents a list of representation abstractions (excluding natural language) organised according to the above-mentioned viewpoints. These abstractions are the most popular in software process research. None of these abstractions covers all the classes of information. Therefore, most of the models found in the literature present languages based on more than one of these abstractions. These models consider the need to integrate multiple representation paradigms (that is, different process models), although this generally makes the model more complex to define [132].

Table 1 Perspectives for process information and applicable language bases

Language Base	Information Perspectives
Procedural programming language [114]	Functional Behavioural Informative
Systems analysis and design, including data flow diagram [55] and structured analysis and design technique (SADT) [102]	Functional Organisational Informative
Artificial intelligence languages and approaches, including rules and pre-/post-conditions [12]	Functional Behavioural
Events and triggers // Control flow [52]	Behavioural
State transition and Petri nets [40][11] // Statecharts [88][89][61][113]	Functional Behavioural Organisational
Functional languages // Formal languages [36][68]	Functional
Data modelling, including entity-relationship diagrams, structured data and relationship declarations [109]	Informative
Object modelling, including types of classes and instances, hierarchy and inheritance [49]	Organisational Informative
Quantitative modelling, including quantitative operational research and systems dynamic techniques [1]	Behavioural
Precedence networks, including actor dependency modelling [138][20]	Behavioural Organisational

On the one hand, as shown in Table 1, a wide variety of notations have been used to model processes. A text that describes different notations for modelling the process is [128]. A good overview over the concepts used in some of the various modelling notations mentioned is given in [116]. There are no data on which of the available notations are more useful or easier to use under given conditions.

On the other hand, as mentioned above, a variety of multi-paradigm approaches have been proposed for modelling software processes. These approaches have two main characteristics. Firstly, excepting mainly Petri net-based approaches, they all use textual representations. Secondly, most of them are already at a programming language level of abstraction. Thus, they make it difficult to model a problematic situation directly, whereas they force the engineer or process manager to take into account many technical aspects of the modelling formalism [49].

3.3. *Process Environments*

Software process models can exploit contributions , concepts and factors from three interrelated environments: organisational, cultural and scientific/technological.

- The *Organisational Environment* reflects if the model deals with organisational issues such as the organisational culture, behaviour, the design and evolution of the organisation, and the abilities of the people and organisations involved.
- The *Cultural Environment* of a model deals with the three essential characteristics of the organisational culture: creative ability, social interaction and flexibility with the environment. *Creative ability* is about the development of new organisational and software process models. *Social interaction* is based on the relation between the different reasoning structures one can found in each discipline, in each profession and even in each person. *Flexibility with the environment* considers the organisation's position regarding socio-cultural and scientific/technological environments and generic software process models, where social interaction and creative ability dimensions are developed.
- The *Scientific/Technological Environment* of a model shows that it points towards the tools, infrastructure, software environments and methodologies to be used for software production and also for software process management, improvement and control.

Viewing software development as a process has significantly helped identify the different dimensions of software development and the problems that need to be addressed in order to establish effective practices. This vision has evolved from "software processes are software too" [106] to "software processes are processes too" [57]. Indeed, addressing the problems and issues of software development is not just a matter of introducing some effective tool and software environment. It is not sufficient to select a reasonable lifecycle strategy either. Rather, we must pay attention to the complex interrelation of a number of organisational, cultural, and scientific/technological factors.

4. Overview of Software Process Models

In this section the main software process models are presented, divided according to their approach into descriptive and prescriptive.

In case the reader is interested just in basic modelling concepts and not in particular models, he or she can skip this section.

4.1. *Descriptive Models*

The goal of descriptive modelling is to make a process now used in an organisation explicit, or to represent a proposed process so as to predict some process characteristics [99]. Software process models that go by the name of *descriptive models* answer the questions "how is software now developed?" or "how has software been developed?" [94]. A descriptive model can uncover many previously hidden problems in the organisation's software development process, and knowing what is actually being done is the first step to be able to improve it. McChesney [99] divides this group of models into two main categories (informal descriptive models and formal descriptive models), depending on the goal of the descriptions.

The goal of *informal descriptive models* is simply to provide an informal and qualitative model. Examples of informal models are the process cycle model [96]; the stratified behaviour model [35]; Bendifallah & Scacchi's model [15]; and Cain and Coplien's model [22]. *Formal descriptive models* of the software process can be related to process assessment, process improvement and/or process prediction. This category includes: the Systems Dynamics Based model [1]; the FUNSOFT network-based model [40]; the PRISM change model [98]; the STATEMATE model [89]; the TAME model [13]; the Amadeus model [124]; the actor dependency model [138]; and the Wolf & Rosenblum's event-based model [136].

Each of these classes are addressed below from the viewpoint of model criteria, representation criteria and methodological criteria in order to identify the elements that represent the software process and, therefore, model the process in question. Figure 2 lists some of the most important descriptive models [92][8][99], belonging to each of these classes. These models are characterised in Table 2 according to the above-mentioned criteria. If they meet a table characteristic, the respective table cell is marked with X; otherwise the space is left blank.

The stratified behaviour model and the process cycle model are qualitative and are orthogonal to existing process models. Although they are extremely informal and have no modelling procedure, they do address the question of the human agents involved in the software process. With regard to formal models, Madhavji provides a methodological perspective of process cycle evolution [96], proposes the PRISM methodology [97] for building and adapting generic software process models and provides a process-centred environment for managing changes in the PRISM model [98]. Basili and Rombach [13] formalise the Goal-Question-Metric paradigm for the following tasks: characterisation of the current state of the project environment and planning, construction, analysis, learning and project feedback. Each of these tasks is addressed in the TAME model from the constructive and analytical viewpoint; that is, the model integrates methods and tools to build the products (constructive viewpoint) and methods and tools to analyse the construction processes and products output (analytical viewpoint). Additionally, STATEMATE covers the four information perspectives, as it integrates three graphical representation languages (state diagrams, activities diagrams, modules diagrams) [89] to analyse and predict the behaviour of the process models.

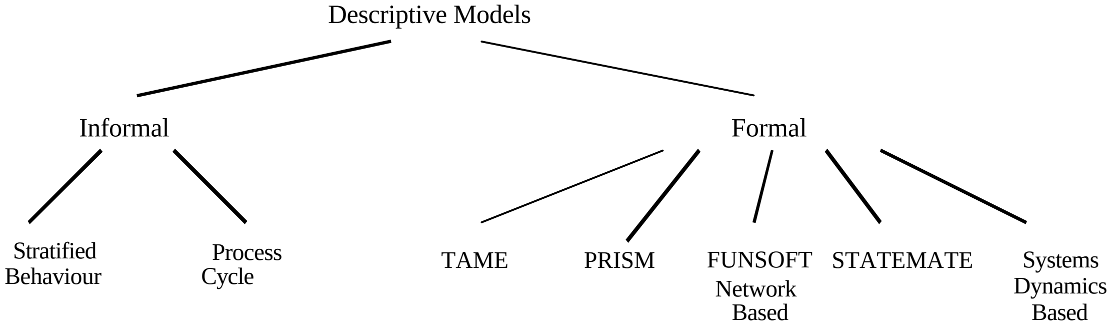


Figure 2 Descriptive models

As shown in Table 2, the FUNSOFT network-based model focuses on a formal and graphic notation. It addresses neither organisational issues nor aspects related to the people involved in the process, it merely deals with the scientific/technological environment, which is reflected under the column labelled *Process environments addressed by the model*. The systems dynamics approach to process modelling is used as a means for simulating proposed processes, predicting dynamic software process behaviour and reporting the administration or management of the right policies and procedures. Abdel-Hamid and Madnick [1] develop a comprehensive automated systems dynamics model of the software development process. It is the only descriptive model that includes both the organisational environment and people's creative ability, which is reflected under the columns labelled *Process environments addressed by the model* and *Cultural environment*.

However, as shown in Table 2, none of these models examine all the components of software process models that we consider to be desirable, which is evidenced under the column labelled *Procedure coverage (partial)*. Sufficient model coverage specifies whether the procedure proposed by each model, if any, covers all the objects and characteristics of the software process model we consider desirable; that is, whether the content of the model considers all the environments, processes, activities, agents, products, roles and relationships between the elements involved in the process. With respect to the environments, sufficient coverage indicates whether the procedure models the organisational, cultural and scientific/technological environments with their related aspects. As far as processes and activities are concerned, it denotes whether the procedure models organisational, management, technical and process support processes and their respective activities. With regard to people and their roles, it shows whether the procedure models the roles of designer, manager and developer (including software engineer, planning engineer, monitoring and control engineer, analyst, designer, knowledge engineer) and the capabilities of the people who play these roles. With reference to products, sufficient coverage specifies whether the procedure models all the documents generated by the activities of each process.

Table 2 Characterisation of descriptive models

MODEL	***Model Criteria***										***Representation Criteria***									***Methodological Criteria***						
	Process elements represented by the model					Process environments addressed by the model					Information perspectives				Notation characteristics					Modelling procedure		Procedure coverage		Procedure definition		
							cultural								from the viewpoint of information quality			from the viewpoint of formal notation								
	agent	activity	artefact	role	event	organisational	creative ability	social interaction	environment flexibility	scientific / technological	functional	behavioural	organisational	informative	informal	formal	automated	text	graphic	non-developed	developed	partial	sufficient	undefined	semi-defined	defined
Stratified Behaviour	X					X							X		X					X						
Process Cycle	X	X		X						X		X	X		X					X						
TAME	X	X	X			X				X	X	X		X		X		X			X	X			X	
PRISM	X	X	X			X				X	X		X	X			X	X	X		X	X			X	
FUNSOFT Network Based	X	X	X		X					X	X	X	X			X			X	X						
STATEMATE	X	X	X		X	X				X	X	X	X	X		X			X		X	X		X		
Systems Dynamics Based	X	X				X	X			X		X					X	X			X	X		X		

Furthermore, none of the models considered provides a series of fully defined activities for process modelling, as shown under the column labelled *Procedure definition (defined)*. There are two models, TAME and PRISM, with a semi-defined modelling procedure, as shown under *Procedure modelling (developed)* and *Procedure definition (semi-defined)* columns and the column labelled *From the viewpoint of information quality (formal and automated)*. Finally, the models are found to provide nowhere near enough information for guiding the engineer in the process of building descriptive process models.

4.2. *Prescriptive Models*

Prescriptive models are the outputs of prescriptive modelling. The goal of prescriptive modelling is to define the required or recommended means of executing the process [99]. The software process models that go by the name of *prescriptive models* answer the question, “how should the software be developed?”

[94]. McChesney [99] divides this group of models into two categories (manual prescriptive models and automated prescriptive models), depending on the goal of the prescriptions.

Manual prescriptive models can be standards, methodologies and methods centred on management, development, evaluation and software life cycle and organisational life cycle support processes. The models belonging to this category include:

- Traditional structured methodologies.
- Object-oriented methodologies.
- Knowledge engineering methodologies.
- Organisational design methodologies, which are considered as some of the most representative in the field of socio-cultural systems [53][115].
- Software life cycle development process standards (like IEEE 1074-1991 and ISO 12207).
- Software process evaluation standard- or model-based methods.

The *automated prescriptive models* perform activities related to assistance, support, management and/or computer-assisted software production techniques. The models belonging to this category include: ALF [23]; IPSE 2.5 [133]; Marvel [84]; PMDB [109]; SOCCA [49]; SPADE [10]; TRIAD [114]; Conversation Builder [85]; EPOS [32]; GRAPPLE [67]; HFSP [86]; MERLIN [83]; and the Unified Software Development Process or Unified Process [82].

These models are computerised specifications of software process standards. Their main aim is to act as a guide for the software modelling process; that is, they are directed at aiding process agents by mechanically interpreting software process models [95]. The automated prescriptive models can be divided into two categories, depending on the guiding criterion selected to address software process modelling: a) activity-oriented models and b) people-oriented models.

Activity-oriented models focus on the functions, activities of and information about parts of the management, development and software life-cycle support processes. These include TRIAD, Marvel, IPSE 2.5 and SPADE, for example. People-oriented models focus on the specifications of the people involved in the software process and their relationships. They include SOCCA, ALF, PMDB and the Unified Process. People are the least formalised factor in existing software process models. Their importance, however, is patent [127]: their behaviour is non-deterministic and subjective and has a decisive impact on the results of software production, which is a basically an intellectual and social activity [59]. Additionally, the non-specification of human resources means that the process does not reflect the actual status of the software process of the modelled organisation, having the added risk of processes not suited to the capability of the organisation's human resources being executed [123][138][48].

As mentioned above, a variety of multi-paradigm software process modelling approaches have been proposed. Most focus on one main paradigm, like rules (Marvel, for example), imperative programs (TRIAD/CML, for example), activity-oriented programs (IPSE 2.5) or Petri nets (SPADE, for example). The integration of formalisms to describe both the software process and the members of the process and their relationships, which explicitly includes the human components and any interaction in which they are involved, has been proposed by approaches such as SOCCA, ALF and PMDB. The chapter by Kneuper in this handbook deals with this issue, considering the process models as support for developers. The complexity of the resulting software, which over-stretches both the technical and organisational management, can be better dealt with in this manner.

Each of these classes is addressed below from the viewpoint of model criteria, representation criteria and methodological criteria in order to identify the elements that represent the software process and, therefore, model the process in question. The prescriptive models considered are shown in Figure 3. Just two process definition standards were considered for manual models. The most important automated models were chosen [92][8][95][99][57]. These prescriptive models are characterised according to the above-mentioned criteria in Table 3. If they meet a table characteristic, the respective table cell is marked with X, otherwise the space is left blank.

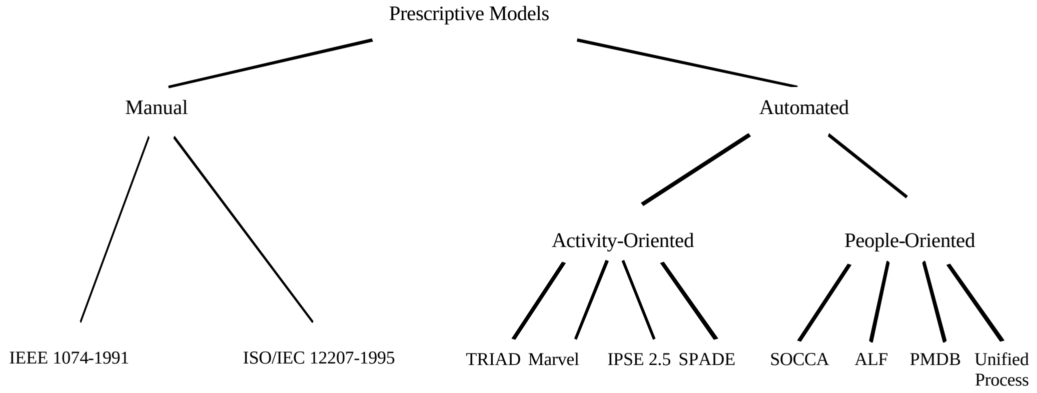


Figure 3 Prescriptive models

In relation to the manual prescriptive models, the software life cycle process standards are informal and consider only the functional perspective of the information in the model, as shown under the columns labelled *Notation characteristics - From the viewpoint of the information quality* (informal) and *Information perspectives* (functional).

Table 3 Characterisation of prescriptive models

MODEL	***Model Criteria***										***Representation Criteria***									***Methodological Criteria***						
	Process elements represented by the model					Process environments addressed by the model					Information perspectives				Notation characteristics					Mode-lling proce-dure		Procedure coverage		Procedure definition		
							cultural								from the viewpoint of information quality			from the view-point of formal notation								
	agent	activity	artefact	role	event	organisational	creative ability	social interaction	environment flexibility	scientific / technological	functional	behavioural	organisational	informative	informal	formal	automated	text	graphic	non-developed	developed	partial	sufficient	undefined	semi-defined	defined
IEEE 1074 - 1991		X	X							X	X				X					X						
ISO/IEC 12207 - 1995	X	X	X			X	X			X	X				X						X	X			X	
TRIAD	X	X		X						X	X	X		X			X	X		X						
Marvel	X	X	X							X	X	X					X	X		X						
IPSE 2.5	X	X	X	X						X			X	X		X		X		X						
SPADE	X	X	X		X					X	X	X	X				X		X	X						
SOCCA	X	X	X	X		X				X	X	X	X	X		X		X	X		X	X				X
ALF	X	X	X		X	X				X	X	X	X	X			X	X		X						
PMDB	X		X			X				X			X	X			X	X		X						
Unified Process	X	X	X	X	X	X				X	X	X	X	X		X		X	X		X	X				X

The table shows that the representations of prescriptive software process models, except the Unified Process, have focused on three elementary process features: the activity, the artefact and the agent (human and computerised) [99]. However, other characteristics, like human and organisational roles, have been empirically proven to have a big impact on the production process [17][34][63][60][131][72][126]. Most of the existing software process models deal with roles and related issues partially [93][52], while the organisation (organisational behaviour, culture and individual abilities) is considered as separate from the characteristics applied for modelling the software process [103] or it is ignored [49]. This is because the organisation is the software process environment and is not, therefore, explicitly modelled. Therefore, these software process models are not integral and joint modelling approaches to the technical and organisational characteristics of the software process.

According to Table 3, the most applicable models are SOCCA and the Unified Process, as the models formally address human resources and the software process interactions in which they are involved and specify a procedure for building and adapting generic software process models, as reflected under the columns labelled *Modelling procedure (developed) and Procedure definition (defined)*. However, the proposed guides do not cover all the desired concepts, objects, characteristics and software process environments nor is there a fully comprehensive and formal modelling process [95], as observed under the column *Procedure coverage(partial)*.

Owing to the complexity of and the requirements for modelling the software process, the best thing appears to be for each low-level modelling or software process implementation to be derived from a high-level specification step, where all the information perspectives can be modelled: a) separately and b) as integrated components of a full specification. Such a modular specification reduces the complexity of the modelling activity and increases the possibility of software process model change and evolution. SOCCA, ALF and PMDB consider these aspects, generally taken into account from the software engineering viewpoint. The Unified Process is iterative, incremental and it is centred in the software architecture. This design based in the software architecture serves as a solid basis over which to plan and manage the development of component-based software. These are the four most advanced and complete models. They model the software project life cycle process and investigate the use of a process model, in this case SOCCA, ALF, PMDB or the Unified Process, as a conceptual and user interaction model and as a framework for integrating tools and/or agents and roles. The approach is incremental, as the whole life cycle is long and complex. Special emphasis is placed on process formalisation, large project support, generalisation and extendibility. The Unified Process includes a formal prescription of the process, where all the process elements are specified using the UML (Unified Modeling Language). Additionally, as mentioned above, SOCCA and the Unified Process give directions on an aspect that is often neglected in software engineering: the problem of the specification under development having to describe not only the technical but also the human parts of the software process, or better still, the human resources, the software process and all classes of interaction between the different non-human and human parts. However, SOCCA and the Unified Process do not deal with the cultural software process modelling questions, absent as well in the other models analysed or considered informally in the socio-cultural models [53][115], as observed under the *creative capability, social interaction and environment flexibility* columns of the *Cultural environment* of the process addressed by the model. Table 3 shows that the organisational, cultural and scientific/technological environments are not integrated in the models in question (as shown under the columns of the same name).

Finally, Table 3 evidences one of the main limitations of existing models: the lack of proposed guides for identifying and defining all the essential organisational and software process conceptual modelling process components. In other words, it confirms that all the prescriptive models considered lack a *defined* *and fully comprehensive software process modelling procedure*. This procedure should provide a formally integrated model of both the software processes and the organisational environments [49][23][103][132].

5. Software Process Evaluation

There are now several software process evaluation methods, each of which is associated with a standard or maturity model used as a reference model or yardstick in the appraisal. The descriptions of these evaluation methods found in the literature tend to match the original definition given by the method developers. However, we will describe the software process evaluation methods according to a set of six basic components, common to any type of evaluation irrespective of the discipline, field or evaluand considered. These components are derived from Evaluation Theory and the progress made in other non-software fields concerning evaluation [122][121][137]. These components are described in subsection 5.2, after reviewing current software process evaluation methods (subsection 5.1). Finally, the components of an evaluation will be used to describe the two most commonly used methods in section 6: SCE [21] and ISO/IEC 15504[1] [77].

5.1. *Review of Evaluation Methods*

Organisations that want to improve their software process now have the possibility of choosing from several software process evaluation methods. This choice will depend on the model or standard to be used as a yardstick for evaluating and improving the software process applied in the organisation. Nowadays, there are a plenty of standards and models [125]. However, not all the standards/models were generated for evaluation purposes, which is why they do not feed any method of evaluation. An example of this are the standards mentioned in section 4.2, IEEE 1074-1991 and ISO 12207. On the other hand, we can find models applied in evaluations, like the Software Capability Maturity Model (CMM hereinafter) [108] which is used as a yardstick in two appraisal methods: Software Capability Evaluation (SCE) [21] and CMM-Based Appraisal for Internal Process Improvement [44]. Indeed, process modelling and evaluation came into being independently and ran parallel, without much (if any) interaction. They are, however, two sides of the same coin and two steps taken on the same object: the software process.

Focusing exclusively on software process evaluation, Figure 4 shows a classification of the best-known software process evaluation methods. Depending on the yardstick for application, organisations can use one of the developed evaluation/assessment methods or prepare the evaluation components according to the requirements for performing an assessment and developing a compatible reference model described in ISO/IEC standard 15504 (formerly known as SPICE, Software Process Improvement and Capability dEtermination). A distinction can be made between the developed evaluation/assessment methods, depending on the type of yardstick used. Hence, ISO 9000 Certification and TickIT [129] are audits of the organisation’s quality system based on the ISO 9000 series, and specifically ISO 9001 and/or ISO 9000-3. Evaluations of this kind are always run by a team from outside the organisation for the purpose of certifying that the organisation’s quality system meets the requirements specified in these standards, applying the guidelines for auditing quality systems described in ISO standard 10011 [76]. On the other hand, if the organisation wants to use a maturity model as its software process improvement guide, it can use the CMM-based appraisals CBA IPI; or BOOTSTRAP [91], a software process assessment and improvement method developed by a European research project within the ESPRIT programme.

[1] To date, ISO 15504 is a Technical Report of type 2. This implies that it is still in draft form. Only Technical Reports of type 3 can be considered as International Standards.

One requirement of most of the methods shown in Figure 4 is that the team of evaluators is composed of authorised evaluators from outside the organisation whose software process or quality system is to be evaluated. However, software process improvement efforts can be based on the findings of an internal self-assessment (for example, an internal CBA IPI) or on the results of an in-house instantiation of the assessment process according to ISO/IEC 15504. Therefore, the set of evaluation methods shown in Figure 4 are the seed that organisations can use to start to improve their software process on the basis of the results yielded by an evaluation applying a given yardstick. Some of these evaluation methods can be used with other yardsticks. The most noteworthy case is Trillium [130], a telecommunications product development and support capability model that can be used in a capability evaluation, performed according to recognised auditing practices such as ISO 10011; in a capability joint-assessment; and in a capability self-assessment, following the Software Process Assessment method (SPA), the forerunner of the CBA IPI [105]. After having outlined the set of evaluation components, the SCE method and ISO 15504 standard will be described in the next section to illustrate the efforts made in the evaluation area.

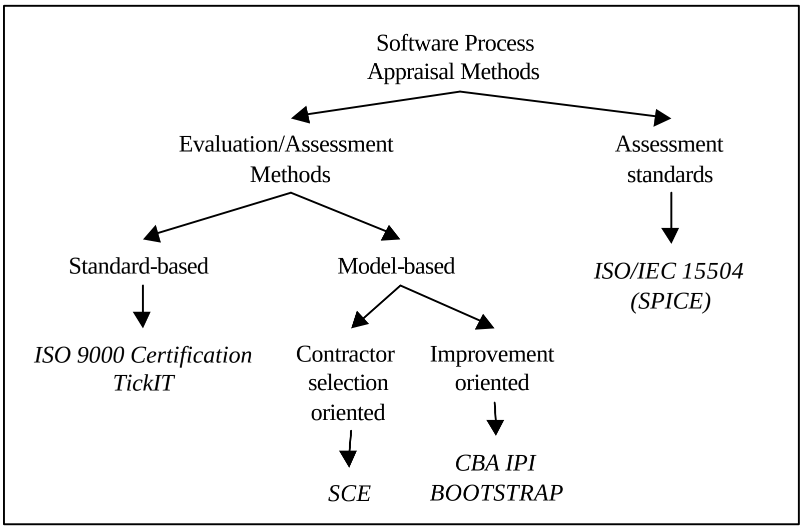

Software Process Appraisal Methods
Evaluation/Assessment Methods
Assessment standards
Standard-based
Model-based
ISO/IEC 15504 (SPICE)
ISO 9000 Certification TickIT
Contractor selection oriented
Improvement oriented
SCE
CBA IPI BOOTSTRAP

Figure 4 Software process evaluation methods

However, irrespective of the selected evaluation method, the output of an evaluation is not a single digit to label an organisation at a specific maturity level: the most relevant output is the discovery of the problems with the current process and the recommendations to fix them [139]. This is the ideal purpose of the software process evaluation and improvement activities. However, this ideal purpose can be sometimes distorted because some managers think about evaluation only as a way to label the organisation within a maturity level but without involving in a real software process improvement (SPI) effort. Although it is possible to initiate an SPI program without management commitment, the success of the program will depend on the commitment of the managers. As a consequence, the organisation's management plays an important role: its attitude will motivate the organisation's professionals to accept the changes to the software process and assure that the necessary resources, time and personnel are available to undertake the evaluation and improvement efforts. A software process evaluation is a critical process but it is not the only activity for enhancing the way software products are constructed.

5.2. *Basic Components of an Evaluation*

The basic components of an evaluation can be applied: a) to design an evaluation method, developing all the components involved in the evaluation; b) to develop and/or adapt these components in each instantiation of the evaluation method when it is run. The basic components of an evaluation are defined as follows.

- Target or evaluand: the object under evaluation.
- Criteria: the characteristics of the target for assessment.
- Yardstick: the ideal target against which the real target is to be compared.
- Gathering techniques: the techniques needed to gather data or information for each criterion under analysis. This information will be used by the synthesis techniques to judge each criterion.
- Synthesis techniques: techniques used to organise and synthesise the information obtained with the gathering techniques and judge each criterion comparing the data obtained with the yardstick. The output of these techniques is the results of the evaluation.
- Evaluation process: series of activities and tasks by means of which an evaluation is performed.

These components are closely interrelated, as shown in Figure 5. A software process evaluation method should identify and explicitly delimit the target in question to find out the primary software process factors to be considered in the evaluation (process, technological resources, etc.). A specific set of characteristics, or evaluation criteria, should be identified to make it easier to adapt the yardstick, if necessary, and select the gathering techniques to be applied. Also, explicit definition and development of the synthesis techniques is an aid for minimising the variability in the final results of the evaluation. All these components are linked by the evaluation process, because the activities and tasks describe the order of application of the preceding components. Also, the evaluation process includes tasks focused on: the definition of the scope and extent of the evaluation; and the development of the criteria, yardstick and techniques. When some of these components are already developed, it is necessary to analyse if they have to be adapted or not. All software process evaluation methods apply these components, although some are described implicitly: normally target delimitation and criteria identification are implicit in the yardstick or reference model/standard used, and the gathering and synthesis techniques are described in the activities and tasks in which they have to be developed and applied. However, if the evaluation components used in the software process evaluation methods are explicitly identified, it will be possible to find out how formally described the method is, how well developed each component is and how much effort it will take to make the evaluators apply the method. Thus, if the method does not provide the gathering techniques, the evaluation team will have to develop these techniques before visiting the organisation whose software process is to be evaluated.

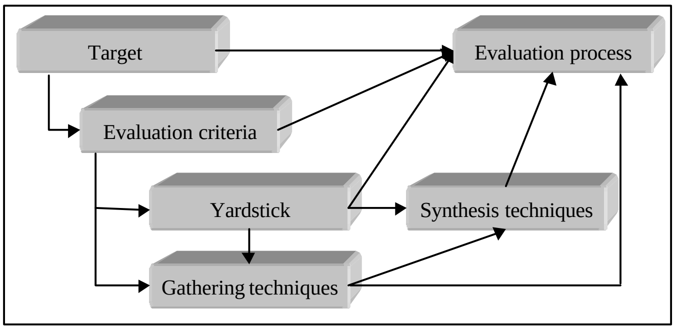


Figure 5 Components of an evaluation and their interrelationships

5.3. *Software Process Evaluation and Knowledge Engineering*

Software process evaluation is an activity performed in software engineering since the late 80´s, having a great influence in this community because an evaluation is an aid to establish management processes in an organisation and to institutionalise a documented, controlled and matured way to produce software. And vice versa, organisations can enhance the yardsticks to apply in the evaluation; for example, including new processes needed to control other processes or develop the software product. Nevertheless, there is no correspondence of this relevant process in the knowledge engineering community. In knowledge engineering we can find diverse methodologies but there are not software process definitions of management and support processes similar as those applied in software engineering. In this sense, knowledge engineering can be considered more immature than software engineering because it has no way to manage the basic knowledge engineering activities.

However, it could be possible to apply the evaluation in knowledge engineering taking into account the hypothesis and evaluation methods considered in software engineering. We can evaluate the knowledge engineering process under the hypothesis that the quality of the software product obtained is determined by the quality of its development process. This implies the identification and description of the management processes needed to control the basic knowledge engineering activities. But to develop a complete evaluation method, we have to elaborate all the evaluation components: with regard the evaluation process, the main processes and activities performed in the software engineering field could be applicable, but the target, criteria, yardstick, and gathering and synthesis techniques have to be developed. Analogous to software engineering, it could be possible to define a specific knowledge engineering maturity model with KPAs (see section 6.1 below) associated with each maturity level. However, there is no experience with the application of management processes in knowledge engineering and consequently, although we can try to apply software engineering management activities in knowledge engineering field, it could be better to discuss first and pilot the potential application of the different software engineering processes, their implications in the knowledge engineering basic activities, the potential alternatives practices to be considered, synthesis techniques to apply, and so on. But despite the great work needed to implement and institutionalise all these management processes, the effort would be worthwhile whether the evaluation could help to mature the knowledge engineering field.

6. Process Evaluation Methods

The two most popular process evaluation methods, SCE and ISO/IEC 15504, are presented in this section, along with related methods. If the reader is not interested in specific evaluation methods, he or she can skip this section.

6.1. *Software Capability Evaluation (SCE) and other CMM-Based Methods*

SCE is the method developed by the Software Engineering Institute (SEI) for evaluating the software process of an organisation for the purpose of using the results of the evaluation for: supplier selection, as a discriminator to select suppliers; process monitoring, between the sponsoring organisation and the development organisation; and internal evaluation, to provide independent evaluations of the internal processes applied in one organisation. An external group applies the SCE in all the above cases. An SCE evaluation focuses on the analysis of the implementation and institutionalisation of certain key process areas by analysing if the organisation’s software processes satisfy the requirements specified in the yardstick. The SCE evaluation process is composed of three phases: plan and preparation; conduct evaluation (during the visit of the evaluation team to the evaluated organisation); and report results.

The *target* considered in a SCE evaluation is delimited by the yardstick applied: SCE uses the CMM as the reference model. The general target of SCE evaluations is the software process, defined as a set of activities, methods, practices and transformations that people use to develop and maintain software and associated products. In particular, the SCE's target is a set of processes (or key process areas, KPAs) divided into three categories: organisational processes, which contain a set of KPAs focused on software process issues and organisational management; project processes, which include a set of KPAs focused on software projects management, such as project planning and tracking; and engineering processes, or the set of KPAs which provides product building operational support, such as requirements management, product engineering or peer reviews. Although the CMM refers to the technical production processes, these are not evaluated in a SCE evaluation.

The CMM provides requirements that good processes will meet [125]. These requirements are organised in a structure based on the concept of software process maturity, defined as the extent to which a specific process is explicitly defined, managed, measured, controlled and effective [108]. Based on this concept, the CMM describes five maturity levels that define an ordinal scale corresponding to the evolution of the software process from an "ad hoc" (unpredictable and unmanaged), through an intuitive (basic management of the processes and capability of repetition of some projects), qualitative (well-defined and institutionalised processes) and quantitative (measured and controlled processes), up to the optimised state in which the processes are continuously improved thanks to their quantitative control. The five maturity levels and the KPAs associated with each level, except the first level that has no KPAs, are shown in Figure 6.

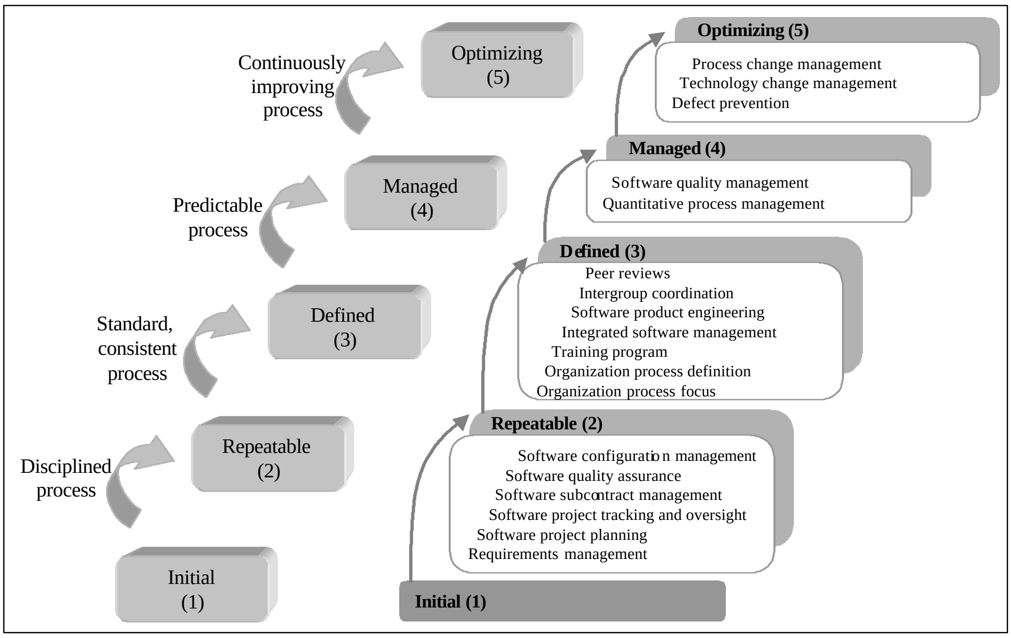
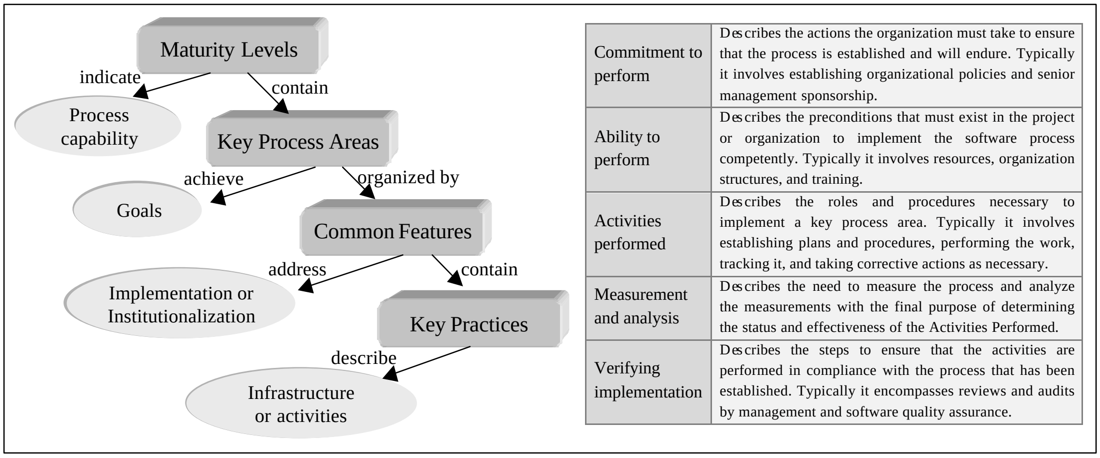

Continuously improving process
Optimizing (5)
Predictable process
Managed (4)
Standard, consistent process
Defined (3)
Disciplined process
Repeatable (2)
Initial (1)
Optimizing (5)
Process change management
Technology change management
Defect prevention
Managed (4)
Software quality management
Quantitative process management
Defined (3)
Peer reviews
Intergroup coordination
Software product engineering
Integrated software management
Training program
Organization process definition
Organization process focus
Repeatable (2)
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight
Software project planning
Requirements management
Initial (1)

Figure 6 Target of a SCE evaluation: KPAs associated with different levels of maturity

All the characteristics of each process that will be analysed make up the *evaluation criteria*. These criteria are implicit in the structure of the maturity levels and in the description of each KPA. The structure of the CMM at each maturity level is shown on the left-hand side of Figure 7. Each KPA is described in terms of the key practices (KP) organised by a set of common features. The common features, described on the right-hand side of Figure 7, provide a structure for describing each KPA, using the same general criteria. The specific criteria will depend on each KPA, as each KPA will necessarily have to deal with all the characteristics described in the common features. Therefore, the set of generic criteria (common features and characteristics considered in each of these categories) of an SCE evaluation can be obtained taking the general description of the CMM. The CMM model, shaped according to Evaluation Theory, is detailed below.

As mentioned above, the yardstick or reference model of a SCE evaluation is the CMM. This maturity model contains the description of the goals of each KPA and all the KPs of each KPA under consideration, apart from the subpractices and supplementary information which usually accompanies some KPs. Table 4 shows an extract from the model for the KPA Software Quality Assurance (maturity level 2) and the common feature Ability to Perform. The specific criterion under consideration is associated with each KP in the left-hand column of Table 4. The information obtained from the application of the gathering techniques will be compared against the CMM (the reference model). This maturity model will be applied to analyse the software process applied in the organisation as a whole: the evaluation team evaluates the organisation's software project development practices against the KPAs in order to determine whether the organisation follows a stable, predictable software process. The results are always related to the organisation, not to a specific project or projects.



Commitment to perform	Describes the actions the organization must take to ensure that the process is established and will endure. Typically it involves establishing organizational policies and senior management sponsorship.
Ability to perform	Describes the preconditions that must exist in the project or organization to implement the software process competently. Typically it involves resources, organization structures, and training.
Activities performed	Describes the roles and procedures necessary to implement a key process area. Typically it involves establishing plans and procedures, performing the work, tracking it, and taking corrective actions as necessary.
Measurement and analysis	Describes the need to measure the process and analyze the measurements with the final purpose of determining the status and effectiveness of the Activities Performed.
Verifying implementation	Describes the steps to ensure that the activities are performed in compliance with the process that has been established. Typically it encompasses reviews and audits by management and software quality assurance.

Figure 7 CMM structure and general criteria of a SCE evaluation: Common features

The *gathering techniques* applied in a SCE evaluation are interviews, document review and presentations. These techniques are applied during the visit to the organisation to gather the information required to judge the target.

Table 4 Yardstick of a SCE evaluation: Example of Key practices for a KPA considered in the CMM

Maturity Level: 2. ***Key Process Area***: Software Quality Assurance *Common Feature*: Ability to perform	
Criteria	Key practices
Responsibility	A group that is responsible for co-ordinating and implementing SQA for the project (i.e., the SQA group) exists.
Resources and funding	Adequate resources and funding are provided for performing the SQA activities.
Training	Members of the SQA group are trained to perform their SQA activities.
Orientation	The members of the software project receive orientation on the role, responsibilities, authority and value of the SQA group.

Interviews can be used to find out how the processes are implemented in practice and show the extent to which processes are internalised and understood by the staff. There are two types of interviews:

exploratory, used to gather information about the actual processes practised and guide the team to the supporting documentation; and consolidation, focusing on corroboration and clarification of evidence.

Document review provides objective evidence of the processes used. This technique is complementary to the above and is based on the review of product artefacts (both intermediate and end products) obtained in the execution of a process to determine the extent of implementation across a site.

Finally, presentations provide both additional data (when the organisation presents to the team and when participants react to team presentations) and validation of data (when participants react to preliminary observations of the SCE final results).

An important aspect is data consolidation, necessary for organising the information obtained after applying the gathering techniques and combining it into a manageable summary of data; determining whether or not the information provides a sufficient basis for making judgements concerning an organisation's process capability; and, if not, determining any revisions that should be made to the gathering techniques to obtain the additional information required to make judgements.

However, to be able to apply the gathering techniques, the evaluation must be delimited during a previous "plan and prepare" phase. Scope (how much of the reference model will be investigated) and coverage (amount of detail with which data within the scope must be collected) must be defined, and the areas in need of more detailed analysis must be also determined by applying instruments like the maturity, project, and organisation questionnaires, among others [140]. The maturity questionnaire was the main gathering technique in the first version of the SCE method [69]. As of SCE version 1.5, the maturity questionnaire is no longer a gathering technique, because data from the questionnaire could not be used as part of the consolidation and judgement process [21].

The information obtained after applying the gathering techniques will be used to get the results of the evaluation, by applying the *synthesis techniques*. These techniques output the analysis and judgements made taking into account the validated observations (findings) collected by the gathering techniques. In general, the rating process always proceeds in a bottom up manner: key practices and common features will be rated first; goals are then rated, and the results are rolled up into KPA and maturity level ratings. However, rating the key practices, common features and maturity level are optional outputs of the evaluation. The decision to rate these components of the CMM must be made during the evaluation planning activity. A typical output of an SCE evaluation only includes the ratings for the goals and KPAs. To judge each KPA's goal satisfaction in an organisation, both the activities (implementation) and other common features (institutionalisation) must be satisfied. All of the goals of a KPA must be achieved in order for the KPA to be satisfied, and all KPAs within a maturity level and each lower maturity level must be rated to determine whether the maturity level has been attained. The rating values used to rate each reference model component (KP, common features, goals, KPAs) are: satisfied, not satisfied, not applicable and not rated. A "satisfied" rating is assigned if a reference model component is implemented and institutionalised either as defined in the CMM or with an adequate alternative. A reference model component will be rated as "not satisfied" when there are significant weaknesses in the implementation or institutionalisation of the item, as defined in the CMM, or no adequate alternative is in place. When the reference model component does not apply in the organisation's environment, it will be rated as "not applicable". Lastly, a component is "not rated" if the evaluation data do not meet coverage requirements or the item is outside the appraisal scope. All the ratings will be used to develop the final results of an SCE evaluation: strengths, weaknesses, improvement activities, and graphical representations of the rating values for each KPA and maturity level.

Finally, the SCE evaluation process indicates when to apply and/or use the preceding components of an evaluation. Figure 8 shows the phases and main activities to be carried out in a SCE evaluation. The "Plan and prepare for evaluation" phase encompasses the activities required to delimit the evaluation, select the evaluation team and prepare for data collection; all these activities will be completed before visiting the organisation. The “Conduct evaluation” phase and “Deliver final findings” activity, focused on the application of the gathering and synthesis techniques, will be performed during this visit. The visit ends when the results of the evaluation are obtained. Later, the evaluation team will produce the reports and support follow-on activities.

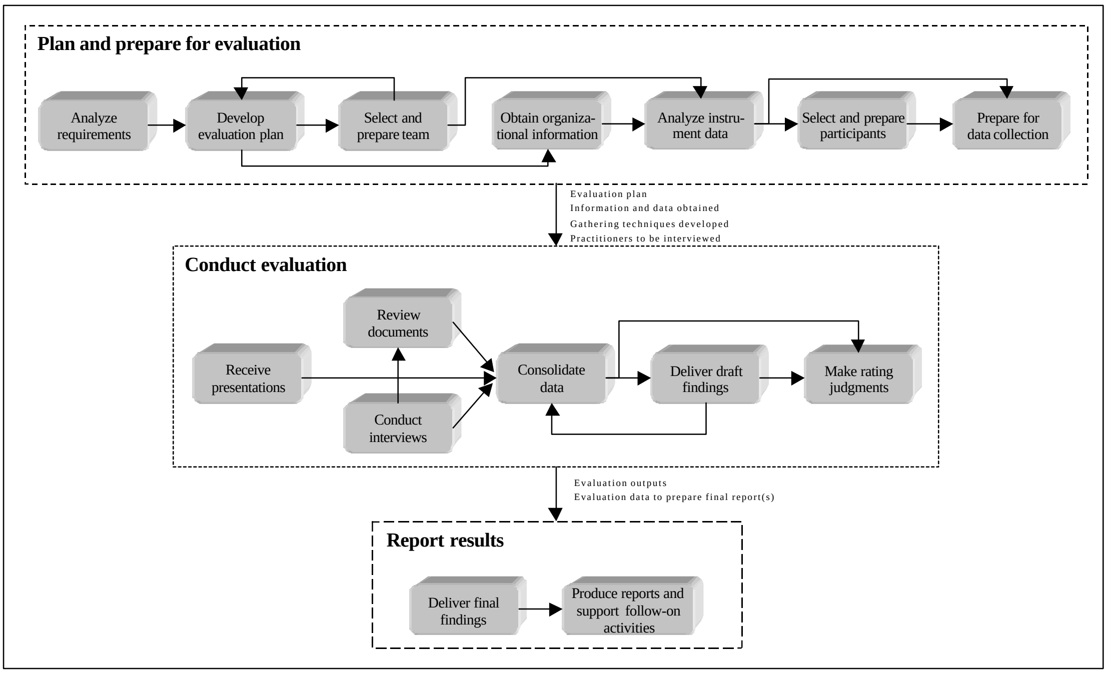

Plan and prepare for evaluation
Analyze requirements
Develop evaluation plan
Select and prepare team
Obtain organizational information
Analyze instrument data
Select and prepare participants
Prepare for data collection
Evaluation plan
Information and data obtained
Gathering techniques developed
Practitioners to be interviewed
Conduct evaluation
Review documents
Receive presentations
Conduct interviews
Consolidate data
Deliver draft findings
Make rating judgments
Evaluation outputs
Evaluation data to prepare final report(s)
Report results
Deliver final findings
Produce reports and support follow-on activities

Figure 8 SCE evaluation process

The components of the evaluation of a CBA IPI assessment are very similar to a SCE evaluation. The main differences lie in the orientation of the appraisal, as it is an improvement-oriented assessment; the formation of the assessment team, which will be either a mixed or an internal team, formed by professionals from the evaluated organisation; and the allotted time, as a SCE takes on average some 176 hours, approximately, including the four-day visit to the evaluated organisation [21], whereas as CBA IPI is run in a time frame of 4 months [44]. The CBA IPI assessments and CMM maturity model are used by organisations from all over the world to assess their software processes. The results obtained from CMM-based appraisals can be used to verify the underlying hypothesis for developing CMM and its associated evaluation and improvement methods [64].

Diverse criticisms and comments about CMM and its related methods have been made since their introduction at the end of the 80s. For example, Bollinger and McGowan [19] criticize the gathering and synthesis techniques used in the first versions of the evaluation methods based on the CMM; they criticize the application of a questionnaire with just two possible answers (“Yes” or “No”) and the simplicity of the algorithm to obtain the final value of the organisation’s maturity level. Bach [9] centers his analysis on the CMM model emphasis: its focus on process (ignoring people) and its lack of theoretical basis. The CMM authors’ rebuttals to Bollinger’s and Bach’s criticisms can be found in [71] and [37], respectively. Some other criticisms are focused on the evaluation process; for example, Saiedian and Kuzara highlight the lack of rigorous definition of the process applied; the results variability; and the non-comparability of the findings of more than one assessment, even when applying the same method [117]. Other criticisms and comments can be found in [24][26][33][104].

The recent release of the CMMI models [27] [28] has been accompanied by a new assessment method, the SCAMPI method (Standard CMMI Assessment Method for Process Improvement) [141], based on the CBA IPI V1.1 and the Electronic Industries Alliance/Interim Standard (EIA/IS) 731.2 Appraisal Method [142]. One of the driving forces in the definition of SCAMPI has been emphasizing the confidence in the appraisal findings despite the cost of the assessment.

6.2. *ISO/IEC 15504*

ISO/IEC 15504 is a framework for assessing an organisation's software processes for the purpose of using the results for process improvement or process capability determination [77]. This framework defines a reference model of processes and process capability that forms the basis for any model to be used for the (singular, no plural) purpose of process assessment [46]. The composition of the assessment team will depend on the purpose and specific circumstances of the assessment. ISO/IEC 15504 does not provide a fully developed assessment process, it sets the minimum set of requirements for performing an assessment, thus increasing the likelihood that results are objective, impartial, consistent, repeatable, representative of the processes assessed and comparable with the results of future assessments. Similarly, the ISO/IEC 15504 reference model specifies the requirements to be met in order for a model(s) used in an assessment to be compatible with the reference model of the standard. The assessors can use other standards and/or models for developing the assessment model to be applied in a specific assessment of the software process of a particular organisation.

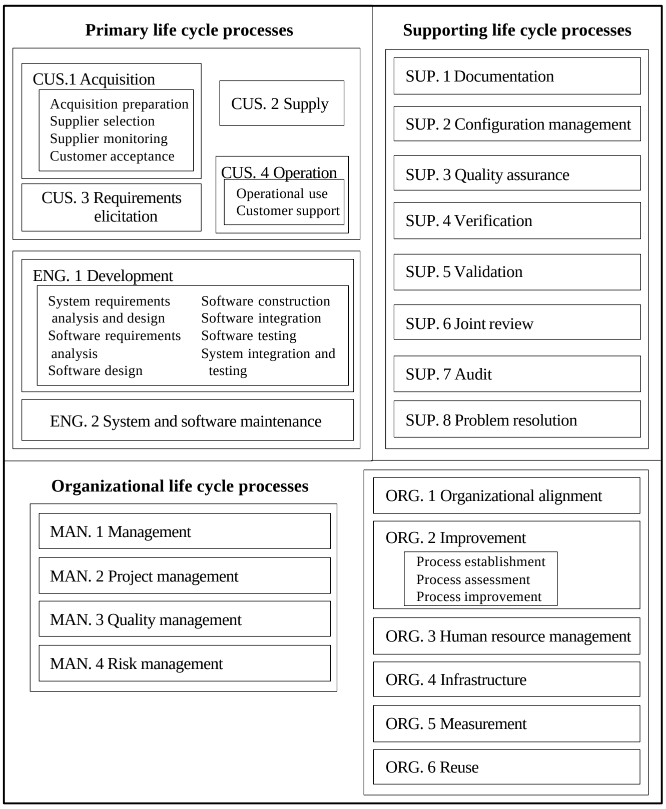


Figure 9 Target of ISO/IEC 15504

The *target* of the ISO/IEC 15504 is the software process and specifically a set of processes strongly related to those defined in ISO 12207. The total set of processes are divided into five categories (customer-supplier, engineering, support, management and organisation), which are grouped into three life cycle processes: primary, supporting and organisation, shown in Figure 9. The primary life cycle process includes the categories: customer-supplier, which contains the set of processes that directly affect the customer, support development and transition of the software to the customer and provide for the correct operation and use of the software product and/or service; and engineering, or the set processes that directly specify, implement or maintain the software product, its relation to the system and its customer documentation. On the other hand, the supporting life cycle process is composed of the support category, which contains the set of processes that can be employed by any of the other processes (including other supporting processes) at various points in the software life cycle. Finally, the organisational life cycle process is composed of two categories: a) management, or the set of processes which contain practices of a generic nature that can be used by anyone who manages any type of project or process within a software life cycle; and b) organisation, which includes the set of processes that establish the business goals of the organisation and develop process, product and resource assets, which, when used by the projects in the organisation, will help the organisation to achieve its business goals.

The ISO/IEC 15504 *evaluation criteria* are several measurable characteristics defined to analyse any of the processes identified in the target. Each criterion is associated with one of the six process capability levels that define a scale for a well-defined route for improvement for each individual process. Table 5 shows the different levels, general criteria associated with each level (denoted as process attributes in ISO/IEC 15504) and the description of each criterion. However, these general criteria have to be broken down into specific criteria or indicators when running an assessment. An indicator is defined as an objective attribute or characteristic of a practice or work product that supports the judgement of the extent of achievement of a particular process attribute. For example, the specific criteria for process performance are: identify input and output work products; ensure that the scope of work is identified for process execution and for the work products to be used and produced by the process; and ensure that basic practices are implemented, producing work products which support achievement of the defined process outcomes.

Table 5 Process capability levels and criteria associated with each level

Levels	*General criteria*	*Definition*
0 – Incomplete	–	–
1 – Performed	Process performance	The extent to which the process achieves the process outcomes by transforming identifiable input work products to produce identifiable output work products.
2 – Managed	Performance management	The extent to which the performance of the process is managed to produce work products that meet the defined objectives.
	Work product management	The extent to which the performance of the process is managed to produce work products that are appropriately documented, controlled, and verified.
3 – Established	Process definition	The extent to which the performance of the process uses a process definition based upon a standard process to achieve the process outcomes.
	Process resource	The extent to which the process draws upon suitable resources (for example, human resources and process infrastructure) that is appropriately allocated to deploy the defined process.
4 – Predictable	Measurement	The extent to which product and process goals and measures are used to ensure that performance of the process supports the achievement of the defined goals in support of the relevant business goals.
	Process control	The extent to which the process is controlled through the collection, analysis and use of product and process measures to correct, when necessary, the performance of the process to achieve the defined product and process goals.

Levels	*General criteria*	*Definition*
5 – Optimising	Process change	The extent to which changes to the definition, management and performance of the process are controlled to achieve the relevant business goals of the organisation.
	Continuous improvement	The extent to which changes to the process are identified and implemented to ensure continuous improvement in the fulfilment of the relevant business goals of the organisation.

ISO/IEC 15504 does not provide a *yardstick* that is directly applicable in an evaluation. The standard only provides a reference model that defines what processes can be considered in the evaluation, with the definition of each process purpose (denoted as process dimension and corresponding to the target); and the capability levels and process attributes (denoted as capability dimension and corresponding to criteria). The yardstick for application (denoted as assessment model) has to be developed on the basis of the above two components (target and criteria) and the structure defined by ISO/IEC 15504, as shown in Figure 10, to run a software process assessment[2]. For this purpose, the assessors will have to complete the definition of each process for assessment, including a set of base practices for each process, providing a definition of the tasks and activities needed to accomplish the process purpose and fulfil the process outcomes; a number of input and output work products, associated with each process; and characteristics associated with each work product. Figure 11 includes an example of an instantiation of the yardstick for the "Software design process", showing the reference model definitions in italics. The base practices, process inputs and outputs and characteristics of an output (high level software design) are shown under the process purpose. The yardstick to be applied in a specific assessment would be formed by this detailed description for all the processes to be assessed.

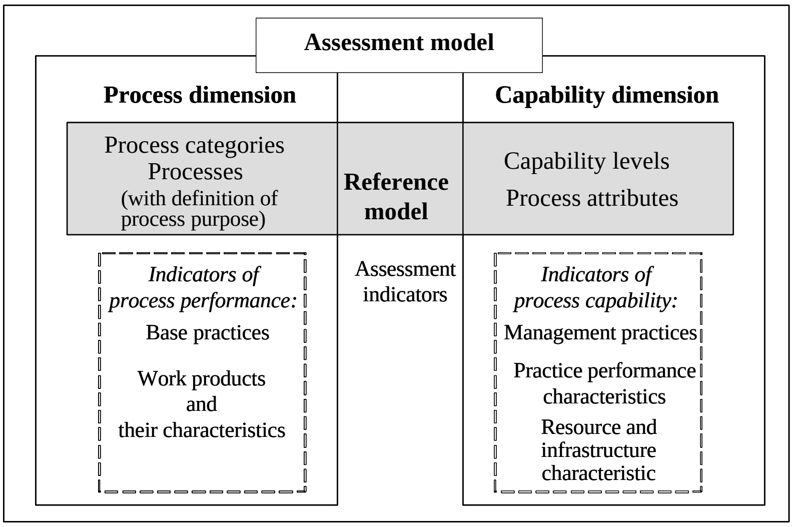

Assessment model
Process dimension
Capability dimension
Process categories
Processes
(with definition of process purpose)
Reference model
Capability levels
Process attributes
Indicators of process performance:
Base practices
Work products and their characteristics
Assessment indicators
Indicators of process capability:
Management practices
Practice performance characteristics
Resource and infrastructure characteristic

Figure 10 Structure of the reference and assessment model

ISO/IEC 15504 does not provide a developed set of *gathering techniques* to be applied in the assessment. It merely specifies that "the documented assessment process should provide guidance on data collection mechanisms, such as interview techniques and document reviewing instruments", that the

[2] The reference model is described in a normative part of the Technical Report (Part 2. A reference model for processes and process capability).

An example of an assessment model can be found in part 5 (An assessment model and indicator guidance), which is informative.

information gathered must be validated, and that validated data should sufficiently cover the assessment scope.

ENG. 1.3. Software design process
Component process of ENG. 1 Development process

The purpose of the Software Design Process is to define a design for the software that implements the requirements and can be tested against them. As a result of succesful implementation of the process:

- *An architectural design, will be developed that describes the major software components that will implement the software requirements.*
- *Internal and external interfaces of each software component will be defined.*
- *A detailed design will be developed that describes software units that can be built and tested.*
- *Consistency will be established between software requirements and software designs.*

Base practices:

ENG.1.3.BP1. Develop software architectural design. Transform the software requirements into a software architecture that describes the top-level structure and identifies its major components.

ENG.1.3.BP2. Design interfaces. Develop and document a design for the external and internal interfaces.

ENG.1.3.BP3. Verify the software design. Verify that the software design satisfies related software requirements.

ENG.1.3.BP4. Develop detailed design. Decompose the top level design into a detailed design for each software component. The software components are refined into lower levels containing software units. The result of this base practice is a documented software design document, which describes the position of each document unit in the software architectures.

NOTE: The detailed design includes the specification of interfaces between the software units.

ENG.1.3.BP5. Establish traceability. Establish traceability between the software requirements and the software design.

ENG.1.3. Software design process. Associated Work Products	
Input	Output
2) Life cycle model	54) High level software design
33) Reuse strategy/plan	55) Low level software design
52) Requirement specification (software)	58) Traceability record/mapping
53) System design/architecture	63) Unit test strategy/plan
59) Test strategy/plan	101) Database design
	105) Customer documentation

54) High level software design
- Describes the overall software structure - Identifies the required software components - Identifies the relationship between software components. - Consideration is given to: · Any required software performance characteristics · Any required software interfaces · Any required security characteristics · Any database design requirements · Any required error handling & recovery attributes

Figure 11 Example of an instantiation of the ISO/IEC 15504 yardstick

Synthesis techniques correspond to the process rating for each criterion (process attribute). The rating scale applied is a percentage scale from zero to one hundred that represents the extent of achievement of the criteria. This scale has the following values: a) not achieved (N, 0% - 15%), which means that there is little or no evidence of achievement of the defined attribute in the assessed process; b) partially achieved (P, 16% - 50%), to indicate that there is evidence of a sound systematic approach to and achievement of the defined attribute in the assessed process; c) largely achieved (L, 51% - 85%), when there is evidence of a sound systematic approach to and significant achievement of the defined attribute in the assessed process; and d) fully achieved (F, 85% - 100%), when there is evidence of a complete and systematic approach to and full achievement of the defined attribute in the assessed process, and therefore no significant weaknesses exist across the defined organisational unit. The set of the criteria rated for each process forms the process profile for that process. These values yield the capability level associated with each assessed process, taking into account that the value of the criteria associated with the above n-1 levels must be "F" (Fully achieved) and the value of the criteria for the level n must be "L" (Largely achieved) or "F" to satisfy a capability level n. The final result of the assessment will generally contain the set of process profiles for all assessed processes and the capability levels associated with the assessed processes.

With regard to the *assessment process*, ISO/IEC 15504 only specifies the requirements to be met by a software process assessment to ensure that the assessment output is self-consistent and provides evidence to substantiate the ratings [47]. The assessors will have to develop and document the process to be applied which must contain at least the activities shown in Table 6.

Table 6 Assessment process of ISO/IEC 15504

1. Planning	A plan for the assessment shall be developed and documented, specifying at least: 1) The required inputs defined in the standard (assessment purpose, scope, constraints and model(s) used, among others). 2) The activities to be performed in conducting the assessment. 3) The resources and schedule assigned to these activities. 4) The selection and defined responsibilities of the assessors and organisation's participants in the assessment. 5) The criteria for verification of the performance of the requirements, and 6) A description of the planned assessment outputs.
2. Data collection	Data required for evaluating the processes within the scope of the assessment shall be collected in a systematic and ordered manner, applying at least the following: 1) The strategy and techniques for the selection, collection and analysis of data and justification of the ratings shall be explicitly identified and shall be demonstrable. 2) Correspondence shall be established between the organisational unit's processes specified in the assessment scope through the compatible model(s) used for assessment to the processes defined in the reference model. 3) Each process identified in the assessment scope shall be assessed on the basis of objective evidence. 4) The objective evidence gathered for each attribute for each process assessed shall be sufficient to meet the assessment purpose and scope. 5) Objective evidence, based on the indicators, that supports the assessor's judgement of process attribute ratings shall be recorded and maintained to provide the basis for verification of the ratings.
3. Data validation	The data collected shall be validated. Actions shall be taken to ensure that the validated data sufficiently cover the assessment scope.
4. Process rating	A rating shall be assigned based on validated data for each process attribute 1) The set of process attribute ratings shall be recorded as the process profile for the defined organisation unit. 2) In order to provide the basis for repeatability across assessments, the defined set or assessment indicators in the compatible model(s) shall be used during the assessment to support the assessors' judgement in rating process attributes. 3) The decision-making process (e.g., consensus of the assessment team or majority vote) that is used to derive rating judgements shall be recorded.
5. Reporting	The assessment results, including at least the outputs specified in the standard (data, inputs, objective evidence gathered and set of process profiles, among others) shall be documented and reported to the Assessment Sponsor.

7. Software Process Improvement

Software process evaluation should be only the first step for an organisation on its way to raising maturity levels. As soon as the current state of the organisation is known it is possible to plan and start the transition to the desired state. But, what is the desired state? Looking, for example, at the SEI's CMM maturity levels, an organisation that continuously improves its software processes is considered to be at the highest maturity level. So, the goal of an organisation should be to institutionalise the improvement effort.

A capable and mature software development organisation is one that is never completely satisfied with the actual results, no matter how good they are; one that is able to recognise the limitations and pitfalls of its software processes as they are actually defined and implemented; one that is able to identify candidate changes to the software processes, or technologies to be adopted, and to evaluate their possible impact on the organisation's performance; and one that is able to manage the adoption of the new technologies and the process changes so as to make the most of them at minimum cost. Therefore, software process improvement is a way of life in a capable and mature organisation.

What is the next step for an organisation that has gone through a software process evaluation and the results show that they are still far from the desired state? First, they should realise that improvement is a long and difficult job, and it has been proven that it is better to do it incrementally. This means that they should set priorities among the many different goals, and they should select a limited set of goals for the first increment. The prioritisation and selection of improvement goals is one of the most important and critical issues.

Looking at the most popular frameworks for the evaluation of software processes, ISO/IEC 15504 and the SEI's CMM, there is a fundamental difference in the way they help to set improvement goals. While ISO/IEC 15504 deals with processes separately, the CMM organises the processes in a levelled representation that defines a reasonable sequence for improvement. As mentioned above, each level in the CMM groups a set of KPAs, and each KPA groups a set of related activities that, when performed collectively, help to achieve goals that are considered important for establishing the capability of the software process at the given maturity level. If an organisation is, for instance, at level 2, the CMM recommends that their immediate improvement efforts should be directed towards level 3 KPAs. ISO/IEC 15504, on the other hand, provides a detailed snapshot of the current state of each process through process profiles, but makes no suggestions as to how to improve on that state.

The new integrated version of the CMM, the CMMI, which has been recently released by the SEI [27][28], aims to combine both approaches and provides two alternative representations, a continuous and a staged representation. The staged representation clearly orders the maturity levels, while the continuous representation focuses on separate process areas and provides achievement profiles (a list of process areas and their respective capability levels) as a result of the evaluation. Using the continuous approach, the improvement should be planned as target staging, that is, a sequence of target profiles to be achieved by the organisation. An equivalence has been defined between the maturity levels of the staged representation and some target profiles. Thus, the continuous approach allows organisations to set their own improvement goals more flexibly, while staging is a more guided and restricted approach, although it allows for benchmarking between organisations or projects.

Complementary to the CMM, the SEI has defined an improvement model called IDEAL [101] that divides the improvement process into five stages: Initiation, Diagnosing, Establishing, Acting and Leveraging. The initiation stage is prior to software process evaluation. It is the stage at which the senior management first realises the need to improve the organisation's software processes and sets the context for and makes a commitment to the improvement program. Sustained senior management commitment is vital for the success of the improvement program. The elementary infrastructure for improvement is established at this stage, including a Management Steering Group (MSG) and a Software Engineering Process Group (SEPG).

The concept of SEPG has been used extensively by improving organisations and has proven to be extremely useful for sustaining the support and commitment of a continuously changing and unstable organisation to the Software Process Improvement (SPI) program. This group is responsible for co-ordinating and nurturing the activities of the different improvement teams that work simultaneously on specific improvements. It is also responsible for creating and maintaining a repository of software process assets. This repository should contain the organisation's standard software processes, guidelines and criteria for tailoring the standard processes for individual projects, the description of possible life cycles for software development, a software process library and a software process database, which contains measurements and data about the observed performance of defined processes on given projects. A post-mortem analysis of projects that have failed can shed new light over the weaknesses of current software processes. Thus, maintaining a software process database is fundamental for the identification of possible improvements. A mature organisation with a repository of standard processes has captured and made explicit the organisational best practices, and therefore it is able to maintain a uniform level of performance independently of the performance of the individuals.

Coming back to the IDEAL model, the Diagnosing stage is the one that is most related to the evaluation of the current software processes. The objective of this stage is to set a baseline for the current state. This baseline will help identify the deficiencies to be overcome and, later on, to measure the degree of improvement that has been achieved.

The next step is to Establish the goals for the improvement program. The MSG should produce a Strategic Action Plan for the next three to five years, setting the short and long-term goals, according to the baseline and the organisation's business needs. A Tactical Action Plan is also sketched for the first improvement cycle and organisational arrangement are made, including one or more Technical Working Groups (TWG). Each TWG will be in charge of building and implementing solutions for one of the selected improvement areas.

During the Acting stage, the TWG searches for solutions to the problems identified at baseline. The main steps to be followed are: a) analyse in detail the current or "as is" state; b) define the desired or "to be" state; c) evaluate possible solutions and select the most appropriate one; d) launch one or more pilot projects and refine the solution; e) plan the institutionalisation process; and f) wrap up and dissolve the team. The roll-on and institutionalisation of solutions across the organisation is the responsibility of the MSG and the SEPG.

The current improvement cycle finishes with a Leveraging stage in which the organisation analyses what happened in this cycle and prepares for the next cycle, revising the goals of the improvement effort and verifying that the necessary commitment and resources are still there. The results of the improvement are compared to the baseline and communicated throughout the organisation, and the improvement process itself is improved for the next iteration.

Other improvement models and solutions have also been defined, like the Business Improvement Guides (BIGs) developed by the European Software Institute (ESI) [118][107][51] or the Process Improvement Guide (PIG) developed by the ISO/IEC 15504 project [80] (called "Software Process Assessment – Part 7: Guide for use in process improvement", working draft V1.00). The methodology proposed in the PIG comprises the following steps: examine organisation's needs; initiate process improvement; prepare and conduct process assessment; analyse results and derive action plan; implement improvements; confirm the improvement; sustain improvement gains; and monitor performance. It is also a cyclic model, and the approach is very similar to that of IDEAL, although the PIG is much less detailed than IDEAL. The most noteworthy difference from IDEAL is perhaps the "sustain improvement gains" step that recognises the complexity of deploying and institutionalising the improved process throughout the organisation.

The ESI's BIG guides are also based on ISO/IEC 15504. However, they are specially oriented towards small and medium-sized enterprises (SMEs), and their objective is to help SMEs to develop their own improvement plans with limited resources. Starting from a generic business need, the ESI derives a staged model from the continuous ISO/IEC 15504 model, by selecting the most important ISO/IEC 15504 processes for that business need and defining an appropriate capability level for each process. The staged model is included within an improvement guide, together with the steps to be taken to get a customised staged model for a specific company. Three BIGs have been developed. The business needs they address are: reducing Time To Market (TTM), attaining ISO 9001 certification, and achieving repeatability of project results through CMM level 2 activities. The BIG project started in 1997 and is still in progress.

Software process improvement in SMEs deserves a special mention. A SPI program is typically very high cost, and costs for a small organisation cannot be expected to be cut in proportion to the smaller number of affected employees, making SPI almost prohibitively expensive. Moreover, the cultural issues in a small organisation are usually completely different. As Kelly [90] states, small organisations tend to be creative, dynamic and innovative, and their software engineers are used to being involved in all aspects of the software engineering process. SPI is perceived as leading to increased bureaucracy and decreased freedom. Kelly advises getting as many people as possible involved in the SPI effort, because solutions imposed by a small group might not be accepted.

The cultural issues in both SMEs and in big companies are crucial to the success of the improvement effort. [54] suggests that SPI should be dealt with like a technology transfer problem. Studying the affected people and the culture of the organisation may be as important in the adoption of a new technology or process as having in-depth knowledge of the new technology. The ISO/IEC 15504 PIG also stresses the importance of cultural issues and identifies some of the most significant questions, like staff motivation, senior management leadership, middle management commitment, confidentiality of the assessment results, an appropriate recognition and reward system, education and training and communication of progress.

While recognizing the importance of cultural and people-related issues, all of the above mentioned models and methods for software process improvement have an organisational focus. They are top-down approaches that require the commitment of top-level management and the establishment of an organizational infrastructure for process improvement. A radically different approach was proposed by Watts S. Humphrey in 1995 with his Personal Software Process (PSP) [72][73]. The PSP is a bottom-up approach focused on the individual software engineers, that provides them with a framework to measure and analyse their current software development processes and to identify and implement process improvements for a better individual performance.

The PSP is organized as a course with fifteen lessons and ten programming exercises. Starting with the establishment of a baseline of the current process (called PSP0), six additional processes are progressively introduced and applied, with incremental improvements.

The rationale behind PSP is that the quality of a software system is determined by the quality of its worst component, and the quality of a software component is determined by the quality of the engineer that developed it. Therefore, by improving the “quality” of individual software engineers an improvement in the quality of software systems can be expected. Two fundamental improvement areas are considered in PSP: the estimation and planning of individual work and the quality of software products.

Filling the gap between the CMM (an organization-centred approach) and the PSP (an individual-centred approach) the Team Software Process (TSP) [74] came to address the software process improvement problem at the team level. The goal of the TSP is to help in the establishment of self-directed teams that are able to plan and track their own work, to set goals and that own their processes and plans, with a strong motivation for maintaining top-level performance.

Despite the many approaches to software process improvement, there are no magic recipes. Improvement is a long and hard effort and any organization that embarks on a SPI program should realise that the main benefits will be perceived in the long term.

8. Summary

We have tried to offer in a single chapter a joint view of research on software process modelling with research on evaluation and improvement. Process modelling serves the purpose of defining the current process under evaluation, or defining the desired process to attain in an improvement effort. In fact, the first level of the maturity levels considered in process evaluation, is usually called "defined". Anyway, process definition is not an easy task, as the diversity of process models presented in section 4 emphasizes. Despite these difficulties, software process evaluation and improvement offer a way to work on and improve the software process in a software development organisation, even if it is required a substantial and sustained effort in order to reach and maintain a high maturity level in terms of software process practices.

References

1. T. Abdel-Hamid, S. Madnick, Software Project Dynamics: An Integrated Approach, (Prentice-Hall, 1991).
2. S. T. Acuña, M. Lopez, N. Juristo, A. Moreno, A process model applicable to software engineering and knowledge engineering, *International Journal of Software Engineering and Knowledge Engineering,* **9**, **5** (1999) 663-687.
3. S. T. Acuña, G. Barchini, M. Sosa, A culture-centered multilevel software process cycle model, *Proceedings of the 22nd International Conference on Software Engineering* (June 2000).
4. S. T. Acuña, G. Barchini, C. Laserre, A. Silva, M. Sosa, V. Quincoces, Software engineering and knowledge engineering software process: Formalizing the who's who, *Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineeering* (July 2000) 221-230.
5. R. Alberico, M. Micco, Expert Systems for Reference and Information Retrieval. (Mockler Corporation, 1990).
6. L. Alexander, A. Davis, Criteria for selecting software process models, *Proceedings of COMPSAC'91* (1991) 521-528.
7. V. Ambriola, R. Conradi, A. Fuggetta, Assessing process-centered software engineering environments, *ACM Transactions on Software Engineering and Methodology* **6**, **3** (1997) 283-328.
8. P. Armenise, S. Bandinelli, C. Ghezzi, A. Morzenti, A survey and assessment of software process representation formalisms, *International Journal of Software Engineering and Knowledge Engineering* **3**, **3** (1993) 401-426.
9. J. Bach, The Immaturity of the CMM, *American Programmer* **7**, **9** (September 1994) 13-18.
10. S. C. Bandinelli, A. Fuggetta, C. Ghezzi, L. Lavazza, SPADE: An environment for software process analysis, design, and enactment, In Software Process Modelling and Technology chapter 9. (Research Studies Press, 1994) 223-247.
11. S. Bandinelli, A. Fuggetta, L. Lavazza, M. Loi, G. Picco, Modeling and improving an industrial software process, *IEEE Transactions on Software Engineering* **21**, **5** (1995) 440-454.
12. N. Barghouti, D. Rosenblum, D. Belanger, C. Alliegro, Two case studies in modeling real, corporate processes, In Software Process – Improvement and Practice, (Pilot Issue, 1995) 17-32.
13. V. R. Basili, H. D. Rombach, The TAME project: Towards improvement-oriented software environments, *IEEE Transactions on Software Engineering* **14**, **6** (June 1988) 758-773.
14. K. Benali, J. C. Derniame, Software processes modeling: what, who, and when, *Proceedings of the Second European Workshop on Software Process Technology* (September 1992).
15. S. Bendifallah, W. Scacchi, Work structures and shifts: An empirical analysis of software specification teamwork, *Proceedings of the 11th International Conference on Software Engineering* (May 1989) 260-270.
16. B. I. Blum, Software Engineering: A Holistic View. (Oxford University Press, 1992).
17. B. W. Boehm, Software Engineering Economics. (Prentice-Hall, Englewood Cliffs, 1981).
18. B. W. Boehm, A spiral model of software development and enhancement, *Computer* (May 1988) 61-72.
19. T.B. Bollinger, C. McGowan, A Critical Look at Software Capability Evaluations, *IEEE Software* **8**, **4** (July 1991) 25-41.
20. L. Briand, W. Melo, C. Seaman, V. Basili, Characterizing and assessing a large-scale software maintenance organization, *Proceedings of the 17th International Conference on Software Engineering* (1995).
21. P. Byrnes, M. Phillips, Software Capability Evaluation. Version 3.0. Method Description, Technical Report CMU/SEI-96-TR-002, (Carnegie Mellon University, Pittsburgh, 1996).

22. B. G. Cain, J. O. Coplien, A role-based empirical process modelling environment, *Proceedings of the Second International Conference on Software Process* (February 1993) 125-133.
23. G. Canals, N. Boudjlida, J. C. Derniame, C. Godart, J. Lonchamp, ALF: A framework for building process-centred software engineering environments, In Software Process Modelling and Technology chapter 7. (Research Studies Press, 1994) 153-185.
24. D.N. Card, Understanding Process Improvement, *IEEE Software* **8**, **4** (July 1991) 102-103.
25. M. A. Carrico, J. E. Girard, J. P. Jones, Building Knowledge Systems: Developing & Managing Rule-based Applications. (McGraw-Hill, 1989).
26. F. Cattaneo, A. Fuggetta, L. Lavazza. An Experience in Process Assessment, *Proceedings of the 17th International Conference on Software Engineering*, (ACM Press, 1995) 115-121.
27. CMMI Product Development Team. CMMI for Systems Engineering/ Software Engineering/ Integrated Product and Process Development, Version 1.01 (CMMI-SE/SW/IPPD, V1.01) Staged Representation. Technical Report CMU/SEI-2000-TR-030 ESC-TR-2000-095. (Carnegie Mellon University, Pittsburgh, 2000).
28. CMMI Product Development Team. CMMI for Systems Engineering/ Software Engineering/ Integrated Product and Process Development, Version 1.01 (CMMI-SE/SW/IPPD, V1.01) Continuous Representation. Technical Report CMU/SEI-2000-TR-031 ESC-TR-2000-096. (Carnegie Mellon University, Pittsburgh, 2000).
29. E. Comer, Alternative software life cycle models, In Software Engineering (IEEE CS Press, 1997).
30. R. Conradi, C. Fernström, A. Fuggetta, R. Snowdon, Towards a reference framework for process concepts, *Proceedings of the Second European Workshop on Software Process Technology* (September 1992) 3-17.
31. R. Conradi, C. Fernström, A. Fuggetta, Concepts for evolving software processes, In Software Process Modelling and Technology chapter 2. (Research Studies Press, 1994) 9-31.
32. R. Conradi, M. Hagaseth, Jens-Otto Larsen, M. N. Nguyen, B. P. Munch, P. H. Westby, W. Zhu, M. L. Jaccheri, C. Liu, EPOS: Object-oriented cooperative process modelling, In Software Process Modelling and Technology chapter 3. (Research Studies Press, 1994) 33-70.
33. D.A. Cook, Confusing Process and Product: Why the Quality is not There Yet, *CrossTalk* (July 1999) 27-29.
34. B. Curtis, Human Factors in Software Development. (IEEE Computer Society, 1985).
35. B. Curtis, H. Krasner, N. Iscoe, A field study of the software design process for large systems, *Communications of the ACM* **31**, **11** (November 1988) 1268-1287.
36. B. Curtis, M. Kellner, J. Over, Process modeling, *Communications of the ACM* **35**, **9** (September 1992) 75-90.
37. B. Curtis, A Mature View of the CMM, *American Programmer,* **7**, **9** (September 1994) 19-28.
38. Davis, E. Bersoff, E. Comer, A strategy for comparing alternative software development life cycle models, *IEEE Transactions on Software Engineering* **14**, **10** (1988) 1453-1461.
39. Davis, Software Specification. Events, Objects and Functions. (Prentice Hall, 1993).
40. W. Deiters, V. Gruhn, Software process analysis based on FUNSOFT nets, *Systems Analysis Modelling Simulation* **8**, **4-5** (1991) 315-325.
41. J. C. Derniame, B. A. Kaba, D. Wastell, Software Process: Principles, Methodology and Technology. Lecture Notes in Computer Science 1500. (Springer, 1999).
42. M. Dowson, B. Nejmeh, W. Riddle, Concepts for process definition and support, *Proceedings of the Sixth International Software Process Workshop* (October 1990).
43. M. Dowson, B. Nejmeh, W. Riddle, Fundamental software process concepts, *Proceedings of the First European Workshop on Software Process Modeling* (May 1991).
44. D.K. Dunaway, S. Masters, CMM^{SM}-Based Appraisal for Internal Process Improvement (CBA IPI): Method Description, Technical Report CMU/SEI-96-TR-007, (Carnegie Mellon University, Pittsburgh, 1996).
45. J. S. Edwards, Building Knowledge-based Systems, Towards a Methodology. (Biddles Ltd., 1991).
46. K. E. El Emam, J-N Drouin, W. Melo, SPICE: The Theory and Practice of Software Process Improvement. (IEEE Computer Society Press, 1998).
47. K. El Emam, N.H. Madhavji, Elements of Software Process Assessment & Improvement. (IEEE Computer Society Press, 1999).

48. E. Ellmer, Extending process-centered environments with organizational competence, In Lecture Notes in Computer Science, Software Process Technology: Proceedings of the 5th European Workshop 1149 (Springer-Verlag, 1996) 271-275.
49. G. Engels, L. Groenewegen, SOCCA: Specifications of coordinated and cooperative activities, In Software Process Modelling and Technology chapter 4. (Research Studies Press, 1994) 71-102.
50. P. H. Feiler, W. S. Humphrey, Software process development and enactment: Concepts and definitions, *Proceedings of the Second International Conference on Software Process* (February 1993) 28-40.
51. P. Ferrer, J.M. Sanz, E. Gallo, M. Vergara, G. Satriani, Business Improvement Guide: BIG-CMM – Part A; User Manual, Technical Report, ESI-1999-TR-007, (European Software Institute, February 1999).
52. Finkelstein, J. Kramer, B. Nuseibeh, Software Process Modelling and Technology. (Research Studies Press, 1994).
53. R. Flood, M. Jackson, Creative Problem Solving: Total Systems Intervention. (John Wiley & Sons, 1991).
54. P. Fowler, M. Patrick, Transition Packages for Expediting Technology Adoption: The Prototype Requirements Management Transition Package, SEI Technical Report CMU/SEI-98-TR-004, (Carnegie Mellon University, Pittsburgh, 1998).
55. D. Frailey, Defining a corporate-wide software process, *Proceedings of the First International Conference on the Software Process* (1991) 113-121.
56. Fuggetta, A. Wolf, Software Process. (John Wiley & Sons, 1996).
57. Fuggetta, Software process: A roadmap, In The Future of Software Engineering, Ed. A. Finkelstein, (ACM Press, 2000) 27-34.
58. H. Gomaa, The impact of rapid prototyping on specifying user requirements, *ACM Software Engineering Notes* **8**, **2** (1983) 17-28.
59. V. Gruhn, Software processes are social processes, Technical Report University of Dortmund. (February 1992).
60. R. Guindon, B. Curtis, Control of cognitive processes during design: What tools would support software designers?, *Proceedings of the CHI'88, Human Factors in Computing Systems*, (1988) 263-268.
61. D. Harel, M. Politi, Modeling Reactive Systems with Statecharts: The Statemate Approach. (McGraw-Hill, 1998).
62. P. Harmon, B. Sawyer, Creating Expert Systems for Business and Industry. (John Wiley & Sons, 1990).
63. R. Hastie, Experimental evidence on group accuracy, In Information Processing and Group Decision-Making, Ed. G. Owen, B. Grofman, (JAI Press, 1987) 129-157.
64. J. Herbsleb, D. Zubrow, D. Goldenson, W. Hayes, M. Paulk, Software Quality and the Capability Maturity Model, *Communications of the ACM* **40**, **6** (1997) 30-40.
65. D. S. Hinley, Software evolution management: A process-oriented perspective, Information and Software Technology **38**, **11** (November 1996) 723-730.
66. E. Hirsch, Evolutionary acquisition of command and control systems, *Program Manager* (November-December 1985) 18-22.
67. K. E. Huff, V. R. Lesser, A plan-based intelligent assistant that supports the software development process, *ACM SIGSOFT Software Engineering Notes* **13**, **5** (November 1988) 97-106.
68. K. Huff, Software process modeling, In Software Process. (John Wiley & Sons, 1996).
69. W. Humphrey, W. Sweet, A Method for Assessing the Software Engineering Capability of Contractors, Technical Report CMU/SEI-87-TR-023, (Carnegie Mellon University, Pittsburgh, 1987).
70. W. Humphrey, Managing the Software Process. (Addison Wesley, 1989).
71. W. Humphrey, B. Curtis, Comments on 'A Critical Look'. *IEEE Software* (July 1991) 42-46.
72. W. Humphrey, A Discipline for Software Engineering. (Addison Wesley, 1995).
73. W.S. Humphrey, Introduction to the Personal Software Process, SEI Series in Software Engineering, (Addison-Wesley, 1997).
74. W.S. Humphrey, Three Dimensions of Process Improvement. Part III: The Team Software Process, *Crosstalk*, April 1998
75. IEEE Standard for Developing Software Life Cycle Processes, IEEE Standard 1074-1991.

76. ISO 10011. Guidelines for auditing quality systems, 1994.
77. ISO/IEC TR 15504. Information Technology – Software process assessment. International Organization for Standardization, International Electrotechnical Commission, 1998. http://wwwsel.iit.nrc.ca/spice
78. ISO, ISO 9001:1994. Quality systems. Model for quality assurance in design, development, production, installation and servicing. (International Organisation for Standardization, ISO, 1994).
79. ISO, ISO 9000-3:1997. Quality management and quality assurance standards. Part 3: Guidelines for the application of ISO 9001: 1994 to the development, supply, installation and maintenance of computer software. (International Organisation for Standardization, ISO, 1997).
80. SPICE project web site. http://www-sqi.cit.gu.edu.au/spice
81. ISO/IEC International Standard: Information Technology. Software Life Cycle Processes, ISO/IEC Standard 12207-1995.
82. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process. (Addison Wesley, 1999).
83. G. Junkermann, B. Peuschel, W. Schäfer, S. Wolf, MERLIN: Supporting cooperation in software development through a knowledge-based environment, In Software Process Modelling and Technology chapter 5. (Research Studies Press, 1994) 103-129.
84. G. E. Kaiser, P. H. Feiler, S. S. Popovich, Intelligent assistance for software development and maintenance, *IEEE Software* (May 1988) 40-49.
85. S. M. Kaplan, W. J. Tolone, A. M. Carroll, D. P. Bogia, C. Bignoli, Supporting collaborative software development with Conversation-Builder, *Proceedings of the 5th ACM SIGSOFT Symp. Software Development Environments, ACM SIGSOFT Software Engineering Notes* **17**, **5** (December 1992) 11-20.
86. T. Katayama, A hierarchical and functional software process description and its enaction, *Proceedings of the 11th International Conference on Software Engineering* (May 1989) 343-352.
87. P. Kawalek, D. G. Wastell, Organisational design for software development: a cybernetic perspective, In Lecture Notes in Computer Science, Software Process Technology: Proceedings of the 5th European Workshop 1149 (Springer-Verlag, 1996) 258-270.
88. M. Kellner, G. Hansen, Software process modeling: A case study, *Proceedings of the 22nd International Conference on the System Sciences* (1989).
89. M. I. Kellner, Software process modelling support for management planning and control, *Proceedings of the First International Conference on Software Process* (October 1991) 8-28.
90. D.P. Kelly, B. Culleton, Process Improvement for Small Organizations, *IEEE Computer,* **32**, **10** (October 1999) 41-47.
91. P. Kuvaja, J. Similä, L. Krzanik, A. Bicego, G. Koch, S. Saukkonen, Software Process Assessment and Improvement. The BOOTSTRAP Approach. (Blackwell Publishers, 1994).
92. J. Lonchamp, K. Benali, C. Godart, J. C. Derniame, Modeling and enacting software processes: an analysis, *Proceedings of the 14th Annual International Computer Software and Applications Conference* (October-November 1990) 727-736.
93. J. Lonchamp, Supporting social interaction activities of software processes, *Proceedings of the Second European Workshop on Software Process Technology, EWSPT'92* (September 1992) 34-54.
94. J. Lonchamp, A structured conceptual and terminological framework for software process engineering, *Proceedings of the Second International Conference on Software Process* (February 1993) 41-53.
95. J. Lonchamp, An assessment exercise, In Software Process Modelling and Technology chapter 13. (Research Studies Press, 1994) 335-356.
96. N. H. Madhavji, The process cycle, *Software Engineering Journal* **6**, **5** (September 1991) 234-242.
97. N. H. Madhavji, W. Schäfer, Prism – Methodology and process-oriented environment, *IEEE Transactions on Software Engineering* **17**, **12** (December 1991) 1270-1283.
98. N. H. Madhavji, Environment evolution: the Prism model of changes, *IEEE Transaction on Software Engineering* **18**, **5** (May 1992) 380-392.
99. R. McChesney, Toward a classification scheme for software process modelling approaches, *Information and Software Technology* **37**, **7** (1995) 363-374.

100. D. McCracken, M. Jackson, Life cycle concept considered harmful, *ACM SIGSOFT Software Engineering Notes* **7**, **2** (April 1982) 29-32.
101. McFeeley, IDEAL: A User’s Guide for Software Process Improvement, Handbook CMU/SEI-96-HB-001, (Carnegie Mellon University, Pittsburgh, February 1996).
102. McGowan, S. Bohner, Model based process assessments, *Proceedings of the 15th International Conference on Software Engineering* (1993) 202-211.
103. Sang-Yoon Min, Doo-Hwan Bae, MAM nets: A Petri-net based approach to software process modeling, analysis and management, *Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering* (June 1997) 78-86.
104. O’Connell, H. Saiedian, Can you trust Software Capability Evaluations?, *IEEE Computer* **33**, **2** (February 2000) 28-35.
105. T. Olson, W. Humphrey, D. Kitson, Conducting SEI-Assisted Software Process Assessments, Technical Report CMU/SEI-89-TR-007, (Carnegie Mellon University, Pittsburgh, 1989).
106. L. J. Osterweil, Software processes are software too, *Proceedings of the 9th International Conference on Software Engineering* (1987).
107. Ostolaza, E. Gallo, M.L. Escalante, G. Benguria, Business Improvement Guide: BIG-TTM Version 1.0, Technical Report, ESI-1999-TR-012, (European Software Institute, February 1999).
108. M. C. Paulk, C. V. Weber, M. B. Chrissis, The Capability Maturity Model: Guidelines for Improving the Software Process. (Addison-Wesley, 1995).
109. M. H. Penedo, C. Shu. Acquiring experiences with the modelling and implementation of the project life-cycle process: the PMDB work, *Software Engineering Journal* **6**, **5** (September 1991) 259-274.
110. S-L. Pfleeger, Software Engineering: Theory and Practice. (Prentice-Hall, 1998).
111. R. Pressman, Software Engineering: A Practitioner’s Approach. (McGraw-Hill, 1997).
112. Rae, P. Robert, H-L. Hausen, Software evaluation for certification. Principles, practice and legal liability. International Software Quality Assurance Series. (McGraw-Hill, 1995).
113. D. Raffo, M. Kellner, Modeling software processes quantitatively and evaluating the performance of process alternatives, In Elements of Software Process Assessment and Improvement, Eds. K. El Emam, N. Madhavji, (IEEE CS Press, 1999).
114. J. Ramanathan, S. Sarkar, Providing customized assistance for software lifecycle approaches, *IEEE Transactions on Software Engineering* **14**, **6** (June 1988) 749-757.
115. R. A. Rodríguez Ulloa, Strategic Management and Control Processes in the Peruvian Culture: The Systemic Methodology for Strategic Management (SM: SM). Ph.D Thesis, University of Lancaster, Lancaster. (1991).
116. H. D. Rombach, M. Verlage, Directions in software process research, *Advances in Computers* 41 (1995) 1-63.
117. H. Saiedian, R. Kuzara, SEI Capability Maturity Model’s Impact on Contractors, *IEEE Computer* **8**, **1** (January 1995) 16-26.
118. Satriani, A. Andrés, M.L. Escalante, L. Marcaida, Business Improvement Guide: SPICE – ISO9001 Version 1.0, Technical Report, ESI-1998-TR-007, (European Software Institute, May 1998).
119. W. Scacchi, Models of software evolution: life cycle and process, Carnegie Mellon University. Software Engineering Institute. SEI Curriculum Modules, SEI-CM-10-1.0. (1987).
120. H. Schmauch, ISO 9000 for Software Developers. (ASQC Quality Press, 1995).
121. M. Scriven, Evaluation Thesaurus. (Sage Publications, 1991).
122. M. Scriven, Evaluation Core Courses Notes, Claremont Graduate University, 2000. http://eval.cgu.edu/lectures/lecturen.htm
123. B. Seaman, V. R. Basili, OPT: An approach to organizational and process improvement, *Proceedings of AAAI Symposium on Computational Organizational Design* (March 1994).
124. R. W. Selby, A. A. Porter, D. C. Schmidt, J. Berney, Metric-driven analysis and feedback systems for enabling empirically guided software development, *Proceedings of the 13th International Conference on Software Engineering* (May 1991) 288-298.

125. S. A. Sheard, The frameworks quagmire, a brief look, *Proceedings of the 7th Annual International INCOSE Symposium (INCOSE´97)* (August 1997) 3-7. http://stsc.hill.af.mil/CrossTalk/1997/sep/frameworks.asp http://www.software.org/quagmire/frampapr

126. K. Sherdil, N. H. Madhavji, Human-oriented improvement in the software process, In Lecture Notes in Computer Science, Software Process Technology: Proceedings of the 5th European Workshop 1149 (Springer-Verlag, 1996) 145-166.

127. Sommerville, T. Rodden, Human, social and organisational influences on the software process, Lancaster University. Computing Department. Cooperative Systems Engineering Group. Technical Report: CSEG/2/1995. (1995) 1-21.

128. SPC, Process Definition and Modeling Guidebook. Software Productivity Consortium, SPC-92041-CMC. (1992).

129. The TickIT Guide (4.0). A Guide to Software Quality Management System Construction and Certification to ISO 9001, (British Standards Institution, 1998).

130. Trillium. Model for Telecom Product Development and Support Process Capability. Version 3.0. (Bell Canada, 1994).

131. J. D. Valett, F. E. McGarry, A summnary of software measurement experiences in the Software Engineering Laboratory, *Journal of Systems and Software* **9**, **2** (1989) 137-148.

132. M. de Vasconcelos Jr., C. M. L. Werner, Software development process reuse based on patterns, *Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering* (June 1997) 97-104.

133. B. Warboys, The IPSE 2.5 project: Process modelling as the basis for a support environment, *Proceedings of the First International Conference on System Development Environments and Factories* (May 1989).

134. B. J. Wielinga, A. Th. Schrieber, P. de Greef, KADS: Synthesis, Document Y3, Project ESPRIT KADS, (Amsterdam University, 1989).

135. M. Wilson, D. Duce, D. Simpson, Life cycles in software and knowledge engineering: a comparative review, *The Knowledge Engineering Review* **4**, **3** (1989) 189-204.

136. L. Wolf, D. S. Rosenblum, A study in software process data capture and analysis, *Proceedings of the Second International Conference on Software Process* (February 1993) 115-124.

137. Worthen, J. Sanders, J. Fitzpatrick, Program Evaluation. Alternative approaches and practical guidelines. (Addison Wesley Longman, 1997).

138. S. K. Yu, J. Mylopoulos, Understanding ‘‘why’’ in software process modelling, analysis, and design, *Proceedings of the 16th International Conference on Software Engineering*, IEEE Computer Society (May 1994) 1-10.

139. S. Zahran, Software Process Improvement. Practical guidelines of business success. (Addison Wesley Longman, 1998).

140. Zubrow, W. Hayes, J. Siegel, D. Goldenson, Maturity Questionnaire, Special Report: CMU/SEI-94-SR-7, (Carnegie Mellon University, Pittsburgh, 1994).

141. CMMI Product Development Team. SCAMPI V1.0. Standard CMMI SM Assessment Method for Process Improvement: Method Description, Version 1.0. Technical Report CMU/SEI-2000-TR-009 ESC-TR-2000-009. (Carnegie Mellon University, Pittsburgh, 2000).

142. Electronic Industries Association. Systems Engineering Capability Model EIA/IS 731, (Electronic Industries Association, Washington, D.C, 1998). http://www.geia.org/eoc/G47/page6.htm