Tailoring the Software Process Model to Project requirements

Osama Eljabiri and Fadi P. Deek

College of Computing Sciences
New Jersey Institute of Technology
Newark, NJ 07102

Abstract

Software process modeling has undergone extensive changes in the last three decades, impacting process' structure, degree of control, degree of visualization, degree of automation and integration. This evolution resulted in a diversity of models that tended to suit software project requirements from different angles. This paper investigates the main streams of process models in an effort to build taxonomy of super classes of process models and their interrelationships. This investigation involved a literature survey of both existing process models and process modeling assessments. Based on this taxonomy an eight-step evaluation process was carried out. By utilizing a manual and an automated decision tables, a final assessment was presented. According to project management literature, a set of decision criteria was then established to facilitate tailorability among process models. The paper concludes with the optimized selection of process models most successful in meeting a variety of software projects requirements. It was also concluded that process diversity could be the rationale for integration rather than differentiation as to capture projects needs in adaptive and flexible combinations. Finally, several frameworks were presented to address similarities, differences, comparisons, and evolutionary aspects among software process models.

Keywords:

Process modeling, process diversity, process assessment, process models evolution, software development, requirements engineering, project management, tailorability.

1. Introduction

Examining the software engineering literature reveals a wealth of software-development approaches that have been introduced in the last three decades. Many vary by titles, structures, rationales, degree of real-world mapping and the extent of how these models reflect organizational strategic goals. Moreover, there are a variety of methods to classifying these approaches and to applying them to various business requirements. This can be attributed to several factors including diverse experiences and interdisciplinary backgrounds among software engineers, degree of problem

complexity, organizational goals, availability of technology, cognitive styles and approach to problem solving. In fact, human cognitive styles and the diversity of their experiences are often reflected in many disciplines and practices. Take management information systems, for example, where the interaction of different cognitive styles, skills, knowledge and activities involved in the development process can lead to better approaches to address information systems and perhaps solve their problems more creatively and efficiently [1]. Bottom-up, reverse engineering of models [2] and even tools can reveal novel ideas that may be helpful when attempting to explore the relations among a range of models in any discipline that could lead to establishing better framework of understanding. Consequently, this paper is an effort to examine the distinct aspects of the many approaches in the software process models to provide an in-depth understanding of the process and produce a criteria for evaluating these models and their suitability for meeting different project requirements.

The paper starts by a review of the literature on software process modeling. An eight-step process evaluation technique is then introduced, the process assessment literature is reviewed, and frameworks of both differences and similarities in process models approaches are established, based on the eight-step evaluation technique, to derive a taxonomy of process models categorization. Next, a decision criteria is established to serve as a guide for making appropriate model selection from among alternatives, based on the characteristics of process model and the needs of software projects. This goal for this criteria is to facilitate tailorability in the process "that allows it to be altered or adapted to suit a set of special needs or purposes" [3].

For the software process model to be tailorable to its users needs, it is necessary to develop an understanding of two important issues: software process modeling and users needs. First, it is important to understand the process and study how software development models impact software products. This requires surveying the literature on process models, studying the evolution of these models and their relationships. It is also important to assess the goals of these software process models, which is essential to evaluating them and identifying their differences. Second, it is also

important to understand software projects requirements. This is a relevant dimension in defining an evaluation method for process models and in determining criteria for matching project needs with process models.

2. Literature Review

The software development life cycle (SDLC) directs the journey that leads software developers from problems to solutions and encapsulates a "process" and a "model" for the development of software systems. Software development practices demand a process to transform ideas and problem requirements into solution specification and results. According to Merriam-Webster dictionary, a *process* is "a series of actions or operations directed toward a particular result" or a "natural phenomenon marked by gradual changes that lead toward a particular result". Process is also defined as "a sequence of steps performed for a given purpose" [IEEE-STD-610] [4]. Thus, in software development, a process is a sequence of transformation activities based on a set of problem requirements to produce a software application. Such software process requires a set of activities, methods, practices, and transformations that can be used to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals) [4]. It is important to note that some software is developed without the direction of a formal process, perhaps implying that a process may not have to be invented from scratch [5]. Accordingly, we can infer that a certain implicit process is there, whether it is well defined or not. However, an undefined process, although it exists, cannot be controlled, simply because it is not possible to control what is ignored. Another definition for the software processes is "the technical and management framework established for applying tools, methods, and people to the software task" [6].

2.1 The Software Development Process

The reliability and consistency of the software process definition is dependent on the organization’s maturity [4] as the software process needs to continuously be improved through evaluating quantitative data to guide the incorporation of new technologies and the enhancement of process structure [7]. Also, the software process is characterized by its complexity, dynamic

behavior, multidimensional activities, and by its difficulty to be grasped [7]. It might be discrete or continuous, sequential or non-sequential, hierarchical or distributed, based on human action or routine activity [8]. Because processes are cross-functional or cross-departmental they provide a description of organizational operations that does not only determine who is doing what but also how and when is this work accomplished. These value-added processes represent the core of understanding business process reengineering as well [9].

Software processes are complex activities that affect critical parameters such as final product quality and costs [10]. Therefore, process control is significant to assure software product quality, though not the only important element in software engineering [11]. Process control extends beyond understanding what is taking place, making preventive maintenance and corrective actions to include quality assurance and forecasting. Controlling and guiding a process toward desired outcome are obvious goals in order to solve real world software problems. This involves controlling people, time, resources, and risk to make software production a feasible task. Additionally, process control is relevant to product metrics, as defining software product metrics is an important method to achieve efficient control [3]. These metrics are dependent on project goals, a concept that could be addressed in a formalized manner such as "Tailoring a Measurement Environment (TAME)" concept encompassing the goal/question/metric approach (GQM). Obviously, process models will be affected by the mechanism of the control techniques.

Some control strategies and techniques in the software process overlap with those in project management, with similar terminologies being used in the literature [73]. It is actually important to notice the interdisciplinary nature of software process control as it incorporates such areas as management, finance, marketing, mathematics, statistics, engineering, psychology as well as other disciplines [74]. A narrow focus on software process control is problematic. For instance, inadequate attention to managerial aspects of software development has resulted in some failures in software products [12]. Similarly, economics is another areas with considerable impact on software development control as it addresses project feasibility, cost estimation, risk assessment, productivity,

and planning. The integrating of economics with process modeling provides an evaluation framework that takes into consideration both technical and human aspects of obtaining the best possible information processing services despite the restrictions of limited resources [13].

The next dimension to address in the process-modeling concept is models and modeling. Whether adopting the managerial approach or the technical approach in modeling software processes [10], a generic definition of a model is: an abstract representation of reality to reduce the complexity of understanding or interacting with a phenomenon by eliminating details that does not influence its relevant behavior [14]. This definition is suited to our process modeling analysis in this paper, as models reflect what their creators believe is important in understanding or predicting the phenomena modeled. A reverse engineering approach, which will be used here, can reveal the concepts or philosophies behind existing process models.

The nature of interactivity among process phases is another classification factor. Control could be achieved by means of "stepwise refinement" through a project control list [15]. This iterative enhancement approach is one dimension in classifying the software process models. The relationship among process activities has a significant influence on building a understanding of process modeling. This has to do with process discovery by means of exploring recurring process patterns as process nature impact process modeling. Some relevant techniques are data mining, reverse engineering and grammar inference [16].

Methodology adopted is also another factor to consider. For example, an object-oriented methodology has a different impact on software development than a process-oriented methodology. Although both of these methodologies have conceptual similarities in the earlier phases of process model, they vary as implementation related factors are considered and techniques or representational constructs are utilized [17]. However, some models [18] adopt an integrated approach of structured oriented methodologies for the requirements phase and object oriented for the design and implementation phases.

Problem complexity is an important factor to consider. The larger the project is the more risk will be involved. Working with a small system is a different experience than working with large ones, as modularization will not generate reliability without tailored approaches and techniques [19]. Process modeling is extremely influenced by this finding. Initially the evolution of process models started by the code and fix model [20], which fits the solution into the problem instead of drawing solutions from well-defined problems. Evolutionary development is based on stages that consist of expanding increments where "the directions of evolution are determined by operational experience" [20].

2.2 Process Diversity

A large collection of software development models has been produced in the last four decades. Some approaches are sequential and so each phase in the process is dependent on the previous one. Other approaches are iterative where the product has to be re-produced several times until a final version is achieved [15]. Control in the sequential approaches is based on the phases and their milestones. With the iterative approaches, control is based on rapidly developed high-speed versions of the product or, as in the iterative enhancement technique, on implementation of parts of the product that starts with a simple initial implementation of a subset of the problem and iteratively enhances existing versions until the full system is implemented [15].

Pressman [11] presented a comprehensive review of software process models that included the following: linear (classical waterfall), prototyping, rapid application development (RAD), incremental, spiral, component assembly, and concurrent development models. Pressman placed the incremental, spiral, component assembly, and concurrent development models in one category based on their evolutionary nature and separated the prototyping model in its own class.

Somerville's [21] classification includes four main categories: the waterfall, the evolutionary development, the formal transformation, and system assembly from reusable components. Somerville divided prototyping into two main classes: exploratory programming, which is mostly used for the development of artificial intelligent systems and throw-away prototyping, which

concentrates on experimenting with those parts of the customer requirements that are poorly understood. Somerville also discusses the reuse-oriented approach.

Behforooz and Hudson [22] distinguish between system development life cycle models (SDLC) and Software development life cycle (SWDLC). Behforooz and Hudson defined SWDLC as an abstract representation of how software is developed which consists of series of sequential or concurrent steps or phases in the software development process. They defined the following models in SWDLC: generic waterfall, DOD, NASA, rapid throwaway prototype, incremental development, evolutionary prototype, reuse and automated development, resource and schedule driven model and Cleanroom models. Behforooz and Hudson defined process model [22] as a sequence of distinct steps or phases in the development of a system and considered all process models virtually versions of the waterfall model. They also introduced the (management information systems) MIS-oriented process model. Although there is a strong similarity between the general process model and the MIS oriented one, the MIS-process model has more focus on business information systems and project management, feasibility study, planning, and other managerial issues that are more business-oriented rather than the technical-oriented.

The models covered by these three sources have many things in common including naming, definitions and similar phases such as analysis, design, coding, testing and maintenance. Several approaches, belonging to the new generation of object-oriented models, were later introduced. The unified software development model [23], a use-case driven, architecture centric, iterative and incremental process introduces new phases: inception, elaboration, construction and transition. The five workflows phases (requirements, analysis, design, implementation and testing) take place over the four new phases adopted in this process model [23]. While the unified process model is user-oriented and strongly influenced by unified modeling language (UML) techniques and requirements management, another management-oriented approach, dynamic modeling [12], was introduced yet influenced by the concepts of software economics. This approach recognizes the importance of managerial considerations and addresses the cost of quality assurance procedures as well. The IBM

Cleanroom method [24], a team approach to software engineering in which quality control of work is ensured by small team review and use of the formal methods in all phases, is a combination of managerial-based approaches and formal-based methods addressed by Somerville [21].

Due to it dynamic behavior, flexibility, adaptability and reusability object-oriented approaches can serve process model in more than providing a standard model. Process models built in object-oriented techniques can be easily modified, extended and viewed at appropriate levels of abstraction. Their application areas include "development of an abstract theory of software process, formal methods of software process, definition and analysis, simulation of software process and the development of automated enactment support" [25].

Although object-oriented methodologies have proven to be advantageous in process modeling, SOFL (structured -object-oriented-formal language) [18] is an approach that shows how integration between structured and object-oriented methodologies can add more value to a process model. This approach has also combined between static and dynamic modeling. These integrations aimed to develop a process model that over come formal methods problems, which limited their usage in the industry.

Addressing risk was one of the big motives in the evolution of process models over the years. Risk can be defined as a state or property of a development task or environment, which, if ignored, will increase the likelihood of project failure [26]. Indeed introducing risk-driven process models was a significant jump in process modeling after a huge library of models based on document –driven or code-driven approaches as “the evolving risk driven approach provided a new framework for guiding the software process [20]**.** This new model was named the spiral model, which was claimed to be fully adaptable to the full range of software project situations and flexible to accommodate a high dynamic range of technical alternatives and user objectives. However, it needs further calibration to be fully usable in all situations [20]. Boehm’s list of risk-related items in software developments became very popular and widely adopted. However, they were oriented to large software systems, including some multi-items that need to be further decomposed, having a

project management flavor, and lacking some theoretical foundation [26]. While Boehm seems to be the first to introduce risk components in his spiral models, some previous models attempted to address this issue more implicitly.

Using Process models in combinations might have good effects if integrated efficiently. Although the spiral model was initiated independently and focused on risk management, it was incorporated with several other process models. It was integrated with prototyping and component assembly models to produce more successful models. Using the spiral model in conjunction with the prototyping model can have a positive effect on risk reduction. Moreover, integrating formal methods with prototyping can have a great influence on prototyping quality [18]. Indeed prototyping can be used as a generic tool in the software development process. Not only prototyping can be integrated with other process models, but also it can help evaluating specific phases such as the requirements phase or even assessing the efficiency of the whole development cycle by means of utilizing it as an experimental tool. In this regard, prototyping can be used as a mechanism in monitoring software processes before investing a great deal of efforts and resources [27]. The spiral model can even be used as a process model generator [28]. In other words it can work as an enabler based on a software process model decision table so it can assist the selection decision more efficiently.

Although the previous resources seem to have a comprehensive coverage of process models, many other approaches were actually overlooked by these literature surveys. A good example of that is the commercial off-the-shelf (COTS) approach, which has gained attention recently. COTS components can be a complete application, an application generator, a problem-oriented language, or a framework in which specific applications are addressed by parameter choices [29]. Integrating COTS with the different phases of the process model might result in an enhanced development process framework [30] or even a life cycle model [31].

Process models that are application-based or domain –specific are another example of what have been overlooked. For instance, web development life cycle, which belongs to the sub-

software engineering area, recently referred to as web engineering is gaining an increasing interest among software engineering conferences. In order to develop and maintain web sites in a cost-efficient way throughout their entire life cycle, sophisticated methods and tools have to be deployed [32].

A third overlooked category is the technology-enabled approaches such as the workflow-based process model and the reengineering process model. Workflow applications are information systems in which work is coordinated by a workflow management system. The people dimension is crucial in designing these applications because of their variety. The basic phases of developing these applications are: information gathering, business process modeling, and workflow modeling phase. Several studies have been done so far in this area but they lacked comprehensibility. Some of these studies aimed at providing reference models for software engineering and business process reengineering [33]. Weske et al. introduced a reference model for workflow application development processes (WADP) in which a generic model was provided to avoid a number of problems related to workflow projects. However, it was suggested that there is no substitute for knowledgeable managers, skilled developers and efficient users. It was suggested that tailoring this reference model to each individual project requires more efforts to be made. This model consists of six phases: survey, design, system selection, implementation, test and operational phase. This line of reasoning was based on a number of case studies and a two-dimensional framework was proposed to tailor the model to more specific needs. The workflow experience and problem complexity were the two dimensions this framework relied upon [33].

Reengineering process model is an approach based on business metrics of cost, time and risk reduction after a dramatic change in existing processes, which should generate breakthroughs in the final products. Although it has been borrowed from the business literature (Business process reengineering - BPR), it was totally based on IT and software systems as enablers in establishing successful projects .The BPR influenced the software process modeling literature and some initial reengineering process models were introduced accordingly. We consider this approach as an

overlooked one, not because it was not addressed but because it was not included in its anticipated location among process models. This might be attributed to its lateness in introduction, lack of adoption or perhaps categorizing it as a technique that might be integrated with other process modeling approaches

According to Somerville [21], software reengineering has three main phases : defining existing system, understanding and transformation , and reengineering system. This means “taking existing legacy systems and re-implementing them to make them more maintainable. As part of this reengineering process, the system may be re-documented or restructured or even retranslated to a more modern programming language “ or implemented in a different architectural platform or data will be “migrated to a different database management system” [21]. Similarly but with a different process model, Pressman [11] introduced software reengineering process model in six main phases that work together in cyclical iterative fashion: inventory analysis, document restructuring, reverse engineering , code restructuring data restructuring and forward engineering . However, both authors emphasized on the importance of automatic techniques to make these models, if applied, cost - effective.

Technology-enabled models include models based on automation by means of CASE tools. While traditional environments were supported by loosely coupled CASE tools that assist independently each phase of the life cycle, more sophisticated architectures were lately introduced to provide a mechanism to ensure that tools are used properly and a user interface that can monitor the actions of project members and coordinate their activities in an integrated manner [34]. This architecture is rule - based with an artificial intelligence approach. The TAME process modeling approach represents an outstanding step to integrate process modeling with product metrics along with the automation capability of CASE tools in a more comprehensive framework [3].

Integrating experimental data with CASE tools can even make the process model much more efficient by means of data collection and building a knowledge base throughout the development process. This new approach has been introduced through the CASE methodology

where CASE and experiment work in conjunction towards the software production goal as assessing software product metrics will be more efficient with the statistical analysis based on experiments accompanied by the high degree of CASE automation [35]**.** The flow of events represented by the cleanroom development increment life cycle based on formal techniques can be categorized in the same class [36]. Integrating good practices has influenced the software process models towards continuous improvement in an evolutionary cycle. This can be seen through models such as the capability maturity model, the bootstrap model, the spice model and other process improvement models [37].

Human factors in developing process models have been somehow overlooked in the previous resources. These factors were indeed impeded implicitly here and there in previous process models but had attracted more attention recently. Scientists who are urging consideration of these factors in modeling software processes are emphasizing that pure technology can never provide a profound model. Whether approaching this issue from the managerial prospective [12] or the cognitive psychological prospective [38], problem solving cannot be achieved efficiently without adopting adequate strategies that are based on correct understanding of humans and their real needs. Hence, Leveson [38] stated, "Our representations of problems have an important effect on our problem-solving ability and the strategies we use ". Clearly, the whole process model is a sort of problem representation. Furthermore, another human-oriented approach is the Japanese version of continuous process improvement (Kaizen) that introduced a strategy for quality enhancement based firstly on human resources as the most important company asset [7].

Aiming to enhance software usability from a user-oriented prospective particularly in the area of user interface design, behavioral approaches have influenced process modeling as well [39]. According to Chase et al, developing a model of behavioral representation techniques involve three dimensions scope, content and requirements. Scope indicates the activities within interface development process that may utilize the technique. Content stands for the interaction components being represented using the technique including user definition, cognitive processes, main-line

action task, feedback display, etc. Requirements stand for the qualities of the representation including facility and expressiveness [39]. Clearly, human centered specification reflects the interdisciplinary impacts of cognitive engineering, a term used to denote the combination of ideas from systems engineering, cognitive psychology and human factors as for their capabilities and limitations of the human element in complex systems [38].

3.- Analysis

3.1 The Evaluation Process

The main goal of this paper is to tailor process models to projects requirements. In order to achieve a thoughtful decision regarding evaluation of the alternatives, an eight-step process is adopted. This process involves the following steps:

1- Extraction: This step implies extracting a list of process models in the literature based on our survey.

2- Assessment: This step will include studying some of the techniques in the literature that aimed to assess process models generally and specifically.

3- Grouping: This step involves discovering the similarities, differences, relations and rationales among this process models to create taxonomy of meaningful grouping and objective classification. This will be based on our two surveys of process models and evaluation efforts in the literature.

4- Representation: Based on this taxonomy and the evaluation literature, a chart of features, and advantages will be established to address process models capabilities.

5- Factor analysis: Identifying the dimensions of software projects requirements and needs from relevant literature.

6- Criteria Identification: Extracting relevant criteria items from these dimensions to be used in the final evaluation process. This step involves setting the *measurement tools* for each dimension. All attributes (criteria dimensions) are analyzed and broken down into more detailed sub-aspects (measurements). Based on the analysis of project requirements, the

percentage weights are established by answering the question of how important a given aspect is.

7- Technique application: Applying the scores-and-weights evaluation technique that allows scoring attributes of alternatives and assigning our own weights to these attributes in order to arrive at a comparative figure of merit.

8- Conclusion: Conducting the evaluation via a numerical evaluation matrix and providing our recommendation.

3.2- Evaluation procedure:

3.2.1- List of process models

Surveyed process models in our literature included the following models:

1. Water fall or Linear sequential model	2. Prototyping model which also includes: Exploratory programming, throw-away prototyping
3. Evolutionary models	4. Incremental and iterative models
5. V-shaped model	6. Spiral model
7. MIS-oriented model	8. 4GT - based models
9. Rapid application development (RAD) model	10. The TAME process modeling approach
11. CASE-tools based models	12. General Object-oriented process models
13. Unified Software Development process (UML) model	14. Component assembly model
15. Assembly from reusable components model	16. Dynamic (management-oriented) model
17. Behavioral model	18. Commercial –of- the- shelf (COTS) process model

19. Formal – based models	20. Cleanroom (IBM) model
21. Concurrent development model	22. Web-based (Web engineering) models
23. Reengineering –based models	24. Process improvement models
25. Department of defense (DOD) model	26. NASA model
27. Operational specification model	
28. Resource and schedule driven model	

3.2.2- Assessment study

Evaluating capabilities of process models is an essential part of any tailoring process. However, this evaluation could be generic (useful for all models), or specific (designed to evaluate a specific model). Firstly, general process modeling evaluation literature will be reviewed. This generic understanding will contribute to build a platform for specific evaluations. Secondly, this paper will discuss some of the extracted models above. This discussion will evaluate the most popular models in terms of their advantages and disadvantages or strengths and weaknesses.

3.2.2.1- Generic process model assessment:

The idea of one-model-fits-all -projects might be hard to imagine in software process modeling, as there are many angles that are difficult to capture in just one picture [40]. However, this difficulty did not eliminate the development of some process models that tended to capture the benefits of previous models in a unified manner. Humphrey [41] stated, “Since the software engineering process used for a specific project should reflect its particular needs, a framework is needed to provide consistency between projects”. Liu et al [40] argued against unification in models and propose important features that every successful model should have. These features are: the ability of a process model to describe the development process as a design process; addressing the parallel processing in large scale projects; mapping the diverse set of conditions that must exist pre-

activities; ability to debug the process by locating failed activities and resources; ability to allocate sufficient resources for each activity in the development project.

Indeed, many were against structuring and managing the process due to the overwhelming differences among projects, firms and cultures. Blackburn et al [42] argued against this assumption, as worldwide similarities in management of the process are apparent than the differences. In fact, many efforts have been exerted to establish customized solutions based on existing process models. Although few of them worked toward tailorability or matching process models to project needs, many of these efforts tended to provide evaluation criteria or metrics, ways of evolution and improvement, taxonomies or unified frameworks, as well as supporting tools and environments. However, some other efforts were dedicated to address general process description or abstraction rather than evaluating existing process models by means of constructing a process conceptual framework. These later efforts tended to serve as a common basis of process representation or transformation in order to assist in developing, reviewing and improving process models [43]. Clearly, understanding these diverse efforts contribute to our goal in building a more comprehensive taxonomy of process models.

Basically process models are used to enable effective communication, facilitate process reuse, support process evolution and facilitate process management [6]. Humphrey and Kellner [6] suggested that in order to evaluate the effectiveness a process model, we should consider the following criteria:

1- Its ability to represent the real world and the way the work is actually done
2- Its ability to provide flexible, understandable, and powerful framework for representing and improving the software development process.
3- Its ability to be refinable to any required level of detail or further specification.

Curtis, Kellner And Over [14] pointed to the following five basic uses of process models. These uses can be considered as evaluation criteria for process models as well.

1- Facilitating human understanding and communication

2- Support process improvement

3- Support process management

4- Automate process guidance

5- Automate execution support

Sutton [8] indicated that in order for a process model to be effective it should exhibit multi-dimensional characteristics including the following:

1- Depth and ability for decomposition to capture every detail of the work to be performed.

2- Full coverage of all activities of the software life cycle (length).

3- Reflect the actual distributed nature of the process including both sequential and parallel processing

4- Ability to combine interdisciplinary models from several related areas (i.e.: project management, configuration management, software evaluation and acquisition) in a single system development.

Madhavji et al [44] proposed the following procedure to elicit and evaluate process models:

1- Understand the organizational environment (organizational issues, process issues, project issues)

2- Define objectives including model-oriented objectives and project oriented objectives

3- Plan the elicitation strategy

4- Develop process models.

5- Validate process models

6- Analyze process models

7- Analyze process models

8- Post analysis

9- Packaging

According to Madhavji [45], the two main goals of using software process models are:

1- Producing software of high quality

2- Producing software that meets budget and time requirements by means of automated tools.

Although Khalifa and Verner [46] focused on waterfall and prototype models in their empirical study, they emphasized on significant factors that drive the usage of specific process models. These drivers are depth and breadth of use, and facilitating conditions (i.e.: size of development team, organizational support, and speed of adopting new methodologies). According to Boehm et al [28] critical process drivers are : requirements growth , understanding of requirements, degree of need for robustness, available technology , architecture understanding. They used these drivers as a baseline for software process model elicitation procedure. Blackburn et al [42] suggested five factors as the most influential drivers in the development process. These factors are development time, project characteristics, team size, and allocation of time in project stages, development language selection. His approach was based on the strong correlation between process optimization and software product metrics from a project management prospective. Madhavji [45] indicated that known life cycle models do have many benefits including process-understanding enhancement, global activities determination, quality improvement, cost reduction, methods and tools effectiveness and stakeholders' satisfaction Moreover, these approaches tackled the problem of managing resources including time and manpower by means of estimation. Furthermore, these approaches can provide predictive capability regarding primary performance measures and capture the variability and uncertainty associated with the software development process [47].

However, these models lack comprehension, detailed description and tailorability to project changing needs. These life cycle models had more focus on product engineering rather than showing many elemental process building blocks essential in project management and control [14]. Krasner [48] criticized these models by their tendency to " focus on series of artifacts that exist at the end of phases of the life cycles rather than on the processes that are conducted to create the artifacts". According to Madhavji, these traditional process models resulted in low software process maturity and difficulties in managing and controlling software processes [45].

In addition, their over-reliance on the waterfall model inherited its negative consequences such as the enforcement of the one way development by managers, inhibiting the creativity based on design/requirements tradeoffs, and corrupting the measurement and tracking system in processes [6]. Moreover, the conventional life cycles added so much documentation without any added value [48]. Kellner and Humphrey attributed these problems in conventional process models to inaccurate representation of the behavioral aspect of what is really going on due to their high sensitivity to task sequence. While Boehm [49] associated these problems with lack of user-interface prototyping, fixed requirements, inflexible point solutions, high–risk downstream capabilities and off-target initial release. According to Boehm these problems "led to the development of alternative process models such as risk-driven, reuse-driven, legacy-driven, demonstration-driven, design-to-cot driven, incremental as well as hybrids of any of these with the waterfall or evolutionary development models". Ropponen et al [26] elaborated on risk management to include the following components, which should be considered in process models assessments:

1- Scheduling and timing risks
2- System functionality risks
3- Subcontracting risks
4- Requirements management risks
5- Resources usage and performance risks
6- Personal management risks

Madhavji [45] proposed a solution that combines process detailed understanding and process support to address change in a process-centered software environment context. Madhavji identified five different perspectives in which a process model could be elicited from and can be viewed in terms of its static or dynamic properties. These perspectives are process steps, artifacts, roles, and resources and constrains [50]. Raffo and Martin [47] analysis recognized two major approaches in software development: process models and system dynamics. They emphasized on the importance of

system dynamics in developing intuition about the project behaviors under different management polices and variety of alternatives. According to Raffo ad Martin, this will be more powerful when utilizing simulation techniques. Although it was Abdel-Hamid and Madnick [12] who efficiently used system dynamics to model project risks such as delays, pressure, and unknown problems at different project levels, Raffo and Martin expanded this idea by using a continuous simulation framework [47]. There work is inline with the process improvement paradigm inspired by the CMM model [47], [4]. Boehm considered Abdel-Hamid and Madnick model a realistic contribution to quantitative models of software project dynamics. However, he argued about the lack of quantitative models of software life cycle evolution [13]. Clearly, both project management and software economics dimensions are gaining more attention as crucial aspects for process model assessment.

Other assessment research efforts focused on classifying process models in a taxonomy fashion. Blum [51] introduced a matrix that organized the development methods with respect to their focus of interest (the problem or the product) and their form of representation (conceptual or formal). Boehm [52] addressed an interesting issue in software projects where a combination of product, process, property and success models might be adopted, which ends up with clashes and conflicts. He proposed taxonomy of model clashes in an effort to resolve them.

As stated in our literature survey, rather than the fixed conventional process models, a trend of process improvement has evolved throughout the last decade. Accordingly, Bandinelli et al [7] "there has been an increasing interest in the development of frameworks and guidelines to support the evaluation of software process maturity and to identify strategies and key areas of improvement". Basili and Rombach [3] proposed the improvement-oriented (TAME) process model based on their GQM (goal –question-metrics) approach that incorporated components for: characterization of the current status of a project environment, the planning for improvement integrated into the execution of the projects, the execution of the construction and analysis of projects, the recording of experience into an experience base, and the distribution throughout the model within and across the components as information from a component to another. According to Basili et al, this components integration

distinguished this improvement model from traditional process models that address only a subset of the individual components of this model. Even the recently developed process models were not able to "completely integrate all their individual components in a systematic way that would permit sound learning and feedback for the purpose of project control and improvement of corporate experience". Learning from the SEI – CMM, Bootstrap, Kaizen, QIP, SPMS, Raytheon, Corning Kellner and Hansen and others experiences, Bandinelli et al [7] developed a feedback loop model for software development organizations to address the problems of discrepancies, poor description, poor comprehension and poor visibility and traceability among the four process forms (i.e.: the desired process, the perceived process, the observed process and the actual process). This model was the baseline for their further experiments to improve process maturity. We can infer two things from these experiences:

1- The process model is no longer that fixed model which is supposed to fit in a fixed problem definition. It has a dynamic nature that evolves as a function of time and responds to the dynamic changes in the problem itself until it reaches the maturity level.

2- Capturing the real world situation (the actual process) was and still is the most significant issue in assessing process models. The more close a representation is to the real world, the more effective and efficient it will be.

Kadary et al [53] raised some important questions about the need and the possibility of a generic paradigm for software life cycles, aspects of this generic model, and its role in industrial practices. In fact, answering this question is not an easy task not only because the problem is not yet well structured, but also because we can think of many alternatives that need further testing and assessment. Consequently, general assessments can be classified into the following categories:

1- Metrics-Oriented Assessments: These assessments tended to frame or synthesize processes and provide standards and metrics for further process enhancements and evaluations as in [6], [8], [14]. These metrics were in forms of factors, drivers or goals as in [28], [45], [46], [42]. Some relevant assessments suggested elicitation procedures or plans as well as in [10], [44].

2- Taxonomy or unified model driven Assessments that surveyed as many models as possible in an effort to build taxonomy [51], or achieve comprehensive conclusions toward a unified process model [23] based on a broad collection and in breadth understanding of process models.

3- Process improvement Assessments: These assessments assumed that existing models are not quite sufficient and they need improvement and new architectures as in [7], [3], [54], [55]. The maturity capability model (CMM) has been the official platform for this whole approach in addition to the many efforts to integrate it with ISO9000 standards. Some of these assessments focused on dramatic change rather than incremental development.

4- Tool support and software environment based Assessments: These assessments incorporated automated tools in process modeling. Some proposed frameworks for process model generation [28]. These approaches had more focus on software development environments. This category includes tool support for that aimed to build more sophisticated process models based on CASE tools and automation [5], [34]. However, we should admit that there is a lot of overlap between this category of assessments and the process improvement category.

3.2.2.2- Specific Process models assessment:

In this section selected process models are reviewed for specific assessments that were applied to them throughout the literature. This selection is based on their representation of the main steams in software process modeling.

4.2.2.2.1- Waterfall model:

As stated in our literature survey, the waterfall model was one of the very first and most influential process models. The waterfall model has played a significant role in process modeling evolution over the decades, as it has become the basis for most software acquisition standards [20]. In fact, it was another improved version of the earliest process model named nine-phase stage-wise model [45]. While the stage-wise model was a one directional linear model, the waterfall maintained the sequential linear nature but with bi-directional relations between the development stages. These

bi-directional relations served as feedback loop, which provided developers with more control over the software process. Moreover, the waterfall model introduced the primary idea of prototyping [45]. The waterfall did well in partitioning the business problem into digestible pieces especially when dealing with complexity or large systems. It was a highly influential refinement of the stagewise model as it recognized the feedback loops and had an initial incorporation of the prototyping in the software life cycle [20]. Also, it is used extensively for it is convenience in schedule and quality control at each process completion [56].

In 1992, the German Ministry of Defense introduced a modified version of waterfall named the V-shaped model. This model has more focus on validation and verification procedures by means of testing activities associated with analysis and design phases and reveals the iteration and rework that are hidden in the waterfall description [45]. However, waterfall is lacking risk assessment, very slow, and not adequate for object-oriented environment. That doesn't imply those objects oriented life cycles are always better. According to some experimental studies [57]**,** this is dependent on the problem type and development team experience. Imposing a project management structure was another drawback of the waterfall model. Furthermore, the waterfall model did not provide a guide for activity transformation among phases, which negatively impacts the capability to handle changes occurring during the development process. Another criticism of the to the waterfall model was its way of viewing the development process as a manufacturing process rather than a dynamic problem solving process that evolves over the time back-and-forth in a learning manner [58]. The bi-directional nature of waterfall phases was not quite sufficient to address this issue as it depends on developers' feedback rather then user involvement. Other problems of the waterfall approach include: "lack of addressing pervasiveness of changes in software development, unrealistic linear description of software processes in real world, difficulty in accommodating advanced languages or recent development, insufficient detail to support process optimization" [6]. As Boehm [49] indicated, the waterfall's millstones did not fit an increasing number of project situations. Although the waterfall model has many drawbacks, it is still the super class of many process-modeling

approaches in software engineering. The idea of decomposition and the sequential step-by-step approach in tackling business problems addressed by the waterfall model can be expanded or enhanced. However, they are difficult to be totally replaced as essential aspects in managing the increasing complexity in software projects.

3.2.2.2.2- The Prototyping Model:

Prototyping was the second most influential technique in process modeling as it was adopted –whether implicitly or explicitly- in almost every process model after the waterfall. Indeed it was even a visualized extension to the feedback bi-directional control in the waterfall itself as the later had an initial incorporation of prototyping [20]. Although there is no unique definition for software or information systems prototype [59], [60], we can recognize three significant characteristics of it: it is temporary, it is fast and it is a visual representation of the proposed system. It is also based on an evolutionary view of software development [60]. Prototyping has been often associated with the evolutionary development model. Also, the operational specification model suggested by Zave can be considered as a variation of prototyping [58]. Major benefits from prototyping includes: ability to extract meaningful feedback from users early in the development process, providing a common baseline for users and developers to identify problems and opportunities, motivating users involvement, establishing better relationship between users and developers [60]. Furthermore, though it is perceived to be more expensive, prototyping addressed some of the limitations of the waterfall such as semi and non-structured requirements [46].

However, prototyping has major shortcomings as well including: overestimation that can oversell the software product, difficulty of management and control, difficulty in working with large systems, difficulty in maintaining user enthusiasm [59]. Moreover, several studies indicated that prototyping does not offer any support for structuring the software development process but was typically used as integrated part of conventional software development life cycles [60]. However, Pfleeger [58] argued about the ability of prototyping to be itself the basis of an effective process

model and proposed a complete prototyping model from system requirements to the finally delivered system with iterative loops of lists of revisions among process main phases.

According to Lichter et al, it might be necessary to use a good mixture of presentation prototypes, prototypes proper, breadboards, and pilot systems for a successful system development. In fact, prototyping played several roles in process modeling. On the one hand it was a partial or a whole solution for process modeling as in [58], [59], [60] .On the other hand it was a tool for assessment, evaluation, monitoring or experimental studies [27] for software process models.

3.2.2.2.3- Spiral Model:

The popular spiral model has heavy reliance on prototyping [56] and software engineering economics [13], as it is mainly a risk - driven process model [20]. Boehm integrated all the previous process models (waterfall, evolutionary, incremental, transform) into his spiral model based on project-customized needs in an effort to maximize benefits and reduce uncertainty. However, he used these previous models as tools (i.e.: utilized just when needed) in his typical cycle rather than adopting the whole approach in each model. In addition, his model was user-sensitive as he exhibited that through iterative cycles of validation and verification. Basically there are 4 types of rounds in the spiral model. Round 0 is the feasibility study round; Round 1 is the concept of operation round, Round 2 is the top-level requirements specifications round; the succeeding rounds.

According to Boehm, advantages of the spiral model include utilization of all the advantages of existing process models and overcoming process models difficulties by practical focus on risk-management. Also it is highly flexible, adaptable and designed for customization [20]. Boehm pointed out that projects, which fully used the system, increased their productivity at least 50 percent. However, he discussed some of the potential difficulties encountering his model including: matching it to the world of contract software acquisition, its reliance on risk-assessment experiences, the need for further elaboration of spiral model steps.

In an effort to resolve model clashes and conflicts, Boehm [52] expanded his spiral model to another version named “win-win spiral model”. In this version of spiral model Boehm used a

stakeholder win-win approach to determine the objectives constraints and alternatives for each cycle of the spiral. In addition, he used a set of life cycle anchor points as critical management decision points. This version was integrated in a more advanced approach to address software critical milestones (in lifecycle objectives, lifecycle architecture and initial operational capability). This integrated win-win spiral model was successfully applied to the DoD's project named (STARS) in an effort to solve its risk problems [49]. This approach incorporated a customized mixture of software process models (waterfall, evolutionary, incremental, spiral and COTS) to suit different needs in software projects.

In [28] Boehm et al used the spiral model as a framework for eliciting or generating adequate process models based on five main drivers discussed in the generic assessment section.

3.2.2.2.4- Iterative and incremental models

Both iterative and incremental (sometimes called phased development [61] models have one goal in common. This goal is reducing the cycle time of the development process. However, they are different in terms of their ways of partitioning the development work. On the one hand, the incremental model is based on building parts of the system in each release until the final system is completed. However, the Ada process model extended this discipline into three dimensions: subsystem increments, build increments, and component increments as this model is supposed to deal with large systems [62]. On the other hand, in the iterative model the whole system is developed all in the first release but improved iteratively in each of the following releases until achieving the most optimized system [58]. According to Basili et al "At any given point in the process, a project control list acts a measure of the distance between the current and the final implementation" [15].

While Graham [61] considered evolutionary development is as a type of incremental development, Pressman [11] classified incremental development as a subclass of the evolutionary approach.

Obviously, prototyping can play a significant role in incremental and iterative development techniques. Moreover, these methods have much overlap with RAD (rapid application development)

as the later is sharing the same goal in reducing process cycle time. Indeed, both RAD and prototyping can be used as tools with the support of CASE tools toward more efficient incremental and iterative process models. Furthermore, the iterative approach is the strategic framework for the unified process model suggested by UML-Object oriented pioneers at Rational Rose Corporation and many of the software process improvement models as well. Due to their incremental or iterative nature, these process models are also adopted explicitly by the spiral model where risk is reduced as each increment or iteration is reviewed and enhanced.

Clearly, these two process models are inline with project needs in terms of cycle time reduction. This can create early markets, fix unanticipated problems quickly, train users in parallel with software improvement, and partition the work of the development team more efficiently [58]. Advantages of incremental development includes also improved team morale, early solution for implementation problems, reduced risk of disaster, improved maintenance, control of over-engineering, measurement of productivity, estimation feedback and smoother staffing requirements. According to Graham [61] problems with the incremental models include hardware related problems, life cycle problems, management problems, financial and contractual problems and user developer relationship problems. Adopting incremental approaches require dealing with a great deal with uncertainty, mastering configuration management, organizational culture change. It should be also empathized that this approach is typically a way to manage complicity in large systems.

3.2.2.2.5-Object oriented models:

Indeed object-oriented process modeling is represented in two forms. The first form is applying traditional process models to object oriented methodologies. This form has very little or almost no impact on process modeling. In other words, its is similar to put new wines in old bottles. The second form is modeling process life cycles by means of object-oriented methodologies.

Examples of this form are components-assembly model, the unified software development model and assembly from reusable components. Another approach was proposed by Riley [25] based on the DRAGOON language. His approach involves the following step-by-step procedure:

1- "Develop object oriented relationship model

2- Develop DRAGOON specification for each class

3- Develop object-behavior models for DRAGOON

4- Develop object-interaction models to analyze the overall process and revise if necessary".

Riley pointed out to the ability of object-orientation to a sound abstract theory in process modeling. He emphasized on the feasibility of an object –oriented approaches in contrast to previous functional approaches, considering Fusion's additional advantages when applied to process modeling.

3.2.3 Grouping

The more the software projects requirements are increasing, the more demands are generated for adequate solutions. However, not every process model discussed before was an individual solution to a specific problem. Although several factors contribute to the formation or development of these process models, they all aimed to common goals and shared characteristics regardless of their degrees of success or achievement. These common goals and characteristics of process models could be summarized as follows:

1- Significant Role of requirements engineering: All of the previous approaches were attempting to provide a solution from a relatively well-defined problem with the exception of the early code-and-fix approach. However, there were different levels and degrees of problem specification. In other words, all the process models are means of problem solving.

2- Influence of waterfall model: All the approaches adopted the four common stages of software development (analysis-design-code-maintenance) whether implicitly or explicitly in a sort of sequence. This implies the influence of the waterfall model regardless of the degree of linearity adopted [20].

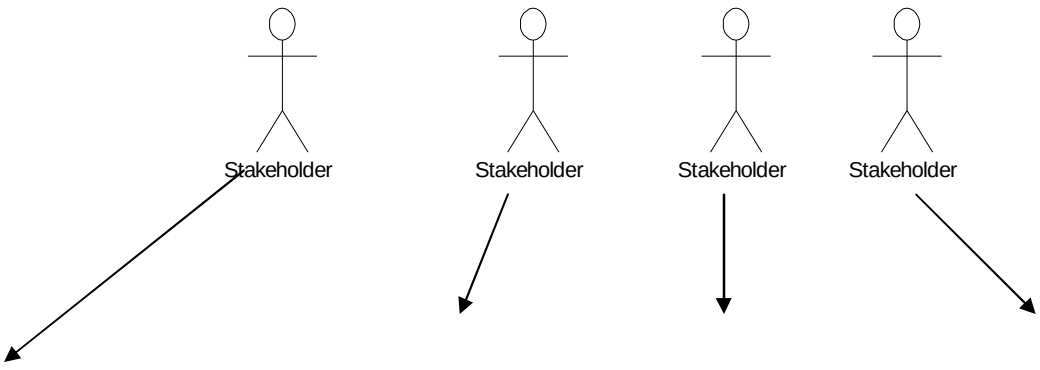
3-Reliance on documentation: All the approaches relied on documentation and artifacts as the main tool to assure quality, planning, monitoring and tracability. This reliance is negatively correlated

with the degree of automation and usage of CASE tools. Ironically, many of these CASE tools are deployed to create more documentation as well.

4- Stakeholders involvement: All the approaches attempts to control the software development process in order to achieve a valid and verified software product that meets stakeholders' requirements. However, the degree of user involvement and the human side impact varied considerably among approaches. This characteristic is inline with the requirements engineering role in process models

5- The project management dimension: All the process models are forms of managing project complexity more efficiently. The final goal was to produce a software solution that is cost-effective, profitable, achieving customer satisfaction and quality desired.

6- Financial Goals: Obviously, the most significant goal of all process models is financial success in terms of profit maximization, cost reduction [13] or customer satisfaction. A traditional way to express financial goals in the software engineering literature is to address them in terms of meeting deadlines, within the budget, and utilizing resources in an efficient manner [40]. Indeed, problems addressed in requirements engineering are in a way or another business problems and solving them efficiently implies adding economic value to both the software product and the organization.



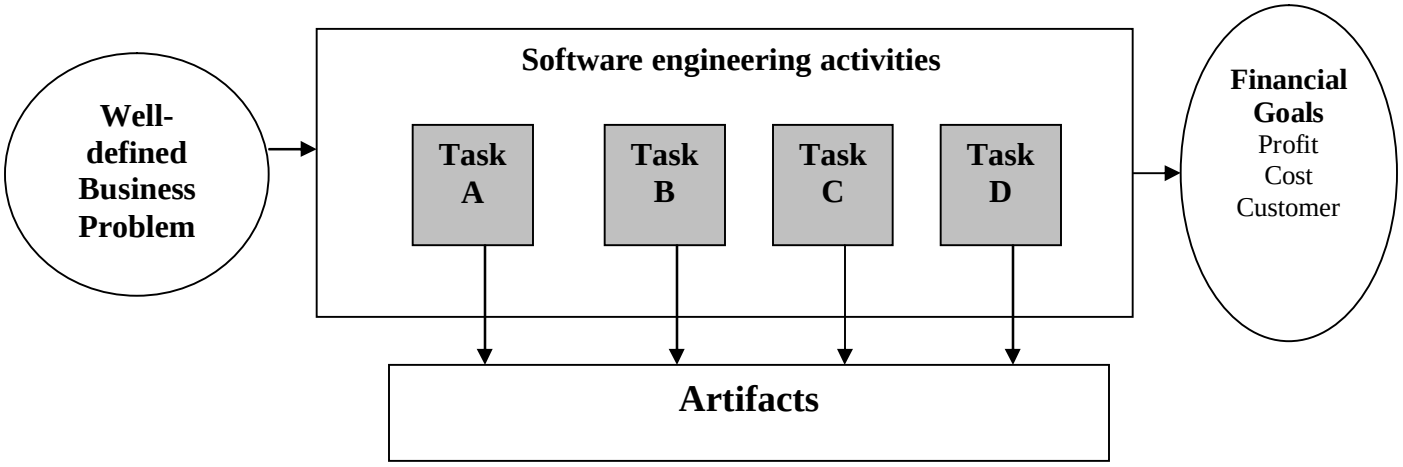


Figure 1. A framework of common characteristics among process models

Figure 1 demonstrates a framework of common characteristics among process models. In this framework, a well-defined problem represents the significant role of requirements engineering input in the development process, which remained constant in terms of existence in all software process models. Indeed, problem definition distinguishes the "after" software engineering age from the "before" age where code and fix approaches took place rather than well-engineered solution from well-understood problems. Moreover, this element reflects the increasing influence of user involvement, as problems are user problems whether he is an internal or an external user.

On the one hand, financial goals in Figure 1 represent the crucial outcome anticipated from the software development process, as there is no need for a software product that is not proven to be economically feasible. Although feasibility extends beyond the financial scope, cost-reduction and business benefits are considered to be the most common and appropriate measurements for effective software production. On the other hand, financial goal indicates the importance of project management aspect in process modeling as efficient utilization of resources is the output of effectively managing software projects as well as an important measurement of financial goals.

The third necessary element in this common framework is stakeholders playing a variety of important roles throughout the development process. Stakeholders could be direct or indirect users of the software product, people who influence the decision of determining system requirements, or developers and staff members involved in the development process This element represents the

people dimension in the development process. As mapped in Figure 1, this dimension is influential in every phase of the development process and it is shared among all the process modeling approaches as people cannot be totally eliminated from the software development process no mater what level of automation is incorporated.

The fourth element of this common software process-modeling framework of software development is the artifacts and deliverables as documentation was an essential strategy in all software process models. However, the more sophisticated CASE tools are involved, the less manual deliverables are needed. For example, in the IBM cleanroom method, automatic transformation takes place among process phases based on mathematical-driven specification languages rather than depending on manual artifacts. This significantly reduces documentation but cannot entirely eliminate it.

The fifth and final element is the tasks that should be followed to achieve a feasible solution from a well-engineered problem. Typically, analysis-design-coding-maintenance concept-wise is a shared sequence among most process models in terms of tasks performed even in the most iterative ones. However, several ways of naming and variations of detailed decompositions distinguish process models from each other. Furthermore, the way these tasks/phases interact is another dimension of process models diversity in terms of tasks interactivity and relationships. These relationships vary from sequence to iteration, from functionally independent to entirely transformational, and from static to dynamic. In other words, these variations in this common framework of similarities are the most important source of variance and diversity among software process models.

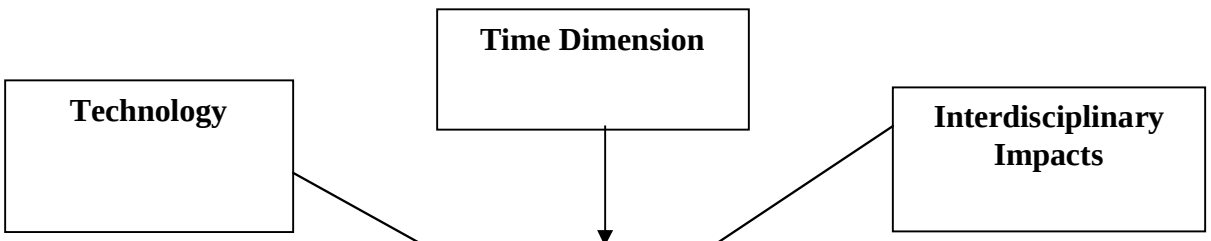
Despite of these commonalities, these process modeling approaches were neither introduced in the same period of time nor belonged to same schools or researchers, nor based on same prospective or the same availability of enabling technologies. Moreover, these approaches were not facing the same nature of problems. Therefore, from our previous survey we can infer that many of

the differences among previous software process models solutions can be attributed to one or more of the following influencing factors:

- Time and previous work influence: Some models were function of time and normal evolution in theory. Several models were function of each other as well .In other words, they were trying to solve problems in previous solutions or build on previous solutions.
- Technology: Some models were functions of technology advancements over the years. For example RAD approach was motivated by the introduction of CASE tools and 4GT techniques. Without the Internet explosion we won't be able to see web development approaches. Artificial intelligence capabilities had its impact as well on some of these approaches.
- Interdisciplinary impacts: Some models were results of more interdisciplinary effects including physiological [38], managerial [12] and financial considerations as in [26], [20], [13].
- Methodology and problem solving approach: Several software process models were reflections of problem solvers' approaches rather than problem solutions. Undoubtedly, what methodology a problem solver adopts, will impact his approach in modeling the solution. On the one hand, some approaches were based on structured-oriented methodologies, while others were based on object-oriented methodologies. On the other hand, some of the approaches were based on linear thinking while others were of iterative nature. In addition, some process models emphasize on concurrent workflows, while others adopt sequential workflows.
- Problem Domain: Some of these approaches were domain-specific while others were generic.
- Problem nature: The nature of the problem addressed was another factor. This includes three aspects of business problems: size, structure and complexity.

- Large and small: While problems associated with large systems have triggered some approaches [19], other small-scale projects required solutions that are more scalable to suit their needs.
- Problem structure: The degree of problem structure is another sub-factor. Moving bottom- up through the popular business inter-organizational pyramid, the problem solver might face problems from the very structured (operational level) to semi-structured (Middle management level) and end up with the unstructured (Top management / strategic level). Obviously, this has to do with problem uncertainty and equivocality as well [63].
- Problem Added-Complexity: Although problem structure and size are very associated to problem complexity, there are other software-related and organization-related elements that can add to complexity as well. There is a positive correlation between organization complexity and the impact of technical change [64].

- Behavioral Considerations: These considerations were the primary rationale for integrating system dynamics in process modeling. Process models that lack these considerations were more static in their structure [12], [39].
- Critical Factors: Process models approaches were tackling problems from different angles based on how critical a factor is in the approach. One critical factor is tasks, as some models focused on task decomposition in a way to solve the business problem. A second critical factor is people as some approaches were people centered by providing solutions to staff management in projects [12]. A third critical factor is tools as some approaches were entirely based on enabling software technology. Other critical factors played significant roles as well. Differences in process models are mapped diagrammatically as shown in Figure 2:



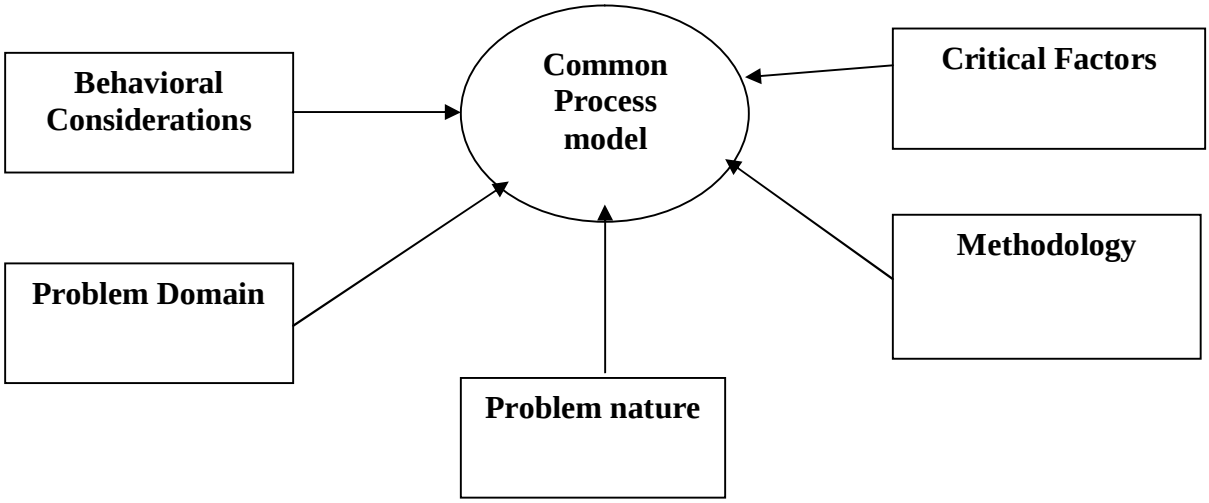


Figure 2. The context diagram of software process modeling

The above context diagram of software process modeling shows eight of the most important factors impacting process modeling diversity as explored in the literature survey. The time dimension implies the evolution of software process models as a function of time while the interdisciplinary impact points out how several sciences and disciplines have influenced the development of software process models. Based on the literature survey, the time dimension has also triggered the change in technology, methodology and nature of business problems, which strongly impacted process models diversity as well. Therefore, these three factors are mapped in the context diagrams. According to literature, behavioral considerations were the source of variation of more recent process models. However, some process models were tailored to respond to some specific domain problems such as IBM, NASA, and DOD models, which is the rationale for considering problem domain as another source of process diversity. Finally, many process models were focused on one or more critical factors as the main motives for developing such models. For instance, risk management was the major motive for developing the spiral model. For this reason, these motives are considered as a diversity factor. Consequently, and based on the context diagram shown in Figure 2, a comparison table is developed as follows:

Process Model	Time Dimension (Evolution in goals)	Methodology	Technology	Critical Factors	Interdisciplinary Impacts	Behavioral Considerations	Problem nature	Application Domain
Waterfall	Solving stage-wise problems	Sequential and structured-oriented	Not- critical	Tasks	None	None	Large scale Projects	General
Prototyping Model	Overcoming late implementations in long cycles	Iterative	Can accelerate The process	User feedback	Psycho.	None	Small scale projects but can be integrated with other large-scale oriented models	General but more successful with artificial intelligence systems and user interface design
Evolutionary models	Overcoming sequential thinking	Iterative or incremental	Can accelerate The process	User feedback	Psychol.	None	Relatively small systems	General but more successful with artificial intelligence systems and user interface design
Incremental and iterative models	Overcoming sequential thinking	Iterative or incremental	Can accelerate The process	User feedback	None	None	Initial Shortage of resources and predicted technical risks	General
V-shaped model	Modified version of waterfall with more focus on quality assurance	Sequential	Not- critical	Tasks , where testing is related to analysis and design	None	None	Large scale Projects	General
Spiral model	Addressing risk assessment overlooked in previous models	Iterative with risk metrics	Recent automated tools are proposed for model generation	Risk Management	Economics	High user interaction specially in the win-win version	Mainly Large scale projects with high degree of uncertain	General
MIS-oriented model	Addressing time management and cost-benefit analysis more in depth (More business oriented than other models)	Sequential	Can be significantly optimized by CASE tools	Projects management.	MIS	None	Large and complex	Business information systems
4GT - based models	Function of available state-of-the-art Technologies	Automatic transformation And CASE tools	Totally dependent on software automation and process technology	Specification languages	AI	None	Used for both small and large systems but require more design considerations for large systems	Recently becoming able to address most software application categories
Process Model	**Time Dimension (Evolution in goals)**	**Methodology**	**Technology**	**Critical Factors**	**Interdisciplinary Impacts**	**Behavioral Considerations**	**Problem nature**	**Application Domain**
Rapid	High-speed	Rapid Linear	Can have	Cycle Time	None	None	Good for	

application development	adaptation of the waterfall model	sequential development and Reuse	great influence	reduction and reusable program components			small systems but need sufficient human resources for large scalable systems	Some times not appropriate for high performance systems, high technical risks or when a system cannot be properly modularized
TAME	Improvement-oriented software development model	goal –question – metrics (GQM)	Initial prototypes	Feedback and measurements	None	High User involvement	More focus on tailorability for different Project requirements	General
CASE-tools based models Or automated development models	Supportive to several other models	Using waterfall with CASE tools support	Dependent on CASE tools	Software CASE tools	AI	None	None	General
Object-oriented process models	Overcoming structured-oriented problems	Object-oriented techniques and reusability	Can be extremely improved by CASE tools	Class objects components	None	None	Large and small systems	(More generic) Ability to work with cross-platform applications
Unified Software Development process	Capturing advantages and overcoming disadvantages in all previous models	Object-oriented based on UML And iterative modeling	Rationale rose ready – made software	UML approach	Economic and management considerations	None	Large and small systems	General
Component assembly model	Utilizing Software reuse advantages overcoming problems in structured paradigms	Object-oriented methodology and spiral model incorporation	Can be extremely improved by CASE tools	Class objects components	None	None	Large and small systems	(More generic) Ability to work with cross-platform applications
Assembly from reusable components model	A Japanese version of components assembly	Object-oriented from existing parts of the system		Existing system components	None	None	Large and small systems	General
Dynamic (management-oriented) model	Heavy focus on managerial considerations	System dynamics	Should be supported by software to capture links and quantitative descriptions due to high degree of complexity	Process Simulation	Management	Crucial specially with human resources	More adequate for Large systems	General
Behavioral models		System dynamics			Management			
Commercial-of-the-shelf "COTS"	Utilizing ready-made software solutions	Efficient Outsourcing and reusability to build cost-effective applications	Can be very effective	Ready-made reused applications	Economics	None	Might be difficult to manage change in complex environments which need high degree of flexibility and customization	Dependent on availability
Formal – based models	Focusing on accuracy and reducing ambiguity incompleteness and inconsistency	Mathematical Transformation	Highly dependent on Software automat.	Mathematical Specification	Math	Primarily, none	Complex systems with sufficient resources	Depending on level of staff training, available time and money, and

	for efficient verification							types of customers
Cleanroom (IBM) model	Focusing on accuracy and reducing ambiguity incompleteness and inconsistency	Mathematical Transformation	Highly dependent on Software automation	Specification Language	Math	Primarily, none	Complex systems with sufficient resources	IBM but can be generalized
Concurrent development model	Capturing the richness of concurrency that exists across various project activities	Activity analysis with state identification		Activity status	Computer Engineering	None	Systems with concurrency and/or networking-architectures	General But more likely in client-server applications
Web-based (Web engineering) models	Response to internet requirements	More dependent on object – oriented modeling	CASE tools can be highly efficient when incorporated.	Web elements	None	None	Large and small web-systems	Web applications
Reengineering –based models	Dramatic changes over existing systems	Business Process-oriented utilizing reverse engineering techniques	IT is crucial	IT and human resources	Modern business	Can have significant influence	Complex and large systems	General but more likely with legacy systems with many problems
Process improvement models	Assessing and improving software product quality	Mainly CMM and ISO standards	Becoming strongly correlated with software automation	Customer satisfaction	Industrial engineering and marketing	Play important role	Large systems	General
Department of defense (DOD) model	A modified version of waterfall model	Sequential problem solving	None	Tasks with PDR Formal Reviews	None	None	Large systems	Department of defense
NASA model	Waterfall structure with slight difference in naming	Sequential problem solving	None	Tasks with function configuration audit	None	None	Large systems	NASA
Operational specification model	Another version of prototyping	Iterative	None	Early user involvement	None	High user involvement	Large and small	General
Resource and schedule driven model	Based on waterfall with very little formality and driven by schedule	Sequential problem solving	None	Tasks with certification testing incorporation	None	None	Large systems	General

3.2.3.1- Process Models Classification

Based on the comparison table illustrated in the last section, we conclude that the following classes can capture the variety of process models in terms of shared characteristics and major differences. This is a primary taxonomy toward creating a comprehensive framework for software process modeling in general.

1. Linear task-oriented models: Sequential problem-solving approach applied generally on large-scale projects where activities decomposition is the core of this class. Long-term delivery is another characteristic for this class with the exception of rapid development models where models are maintaining sequential approaches but designed to deliver software

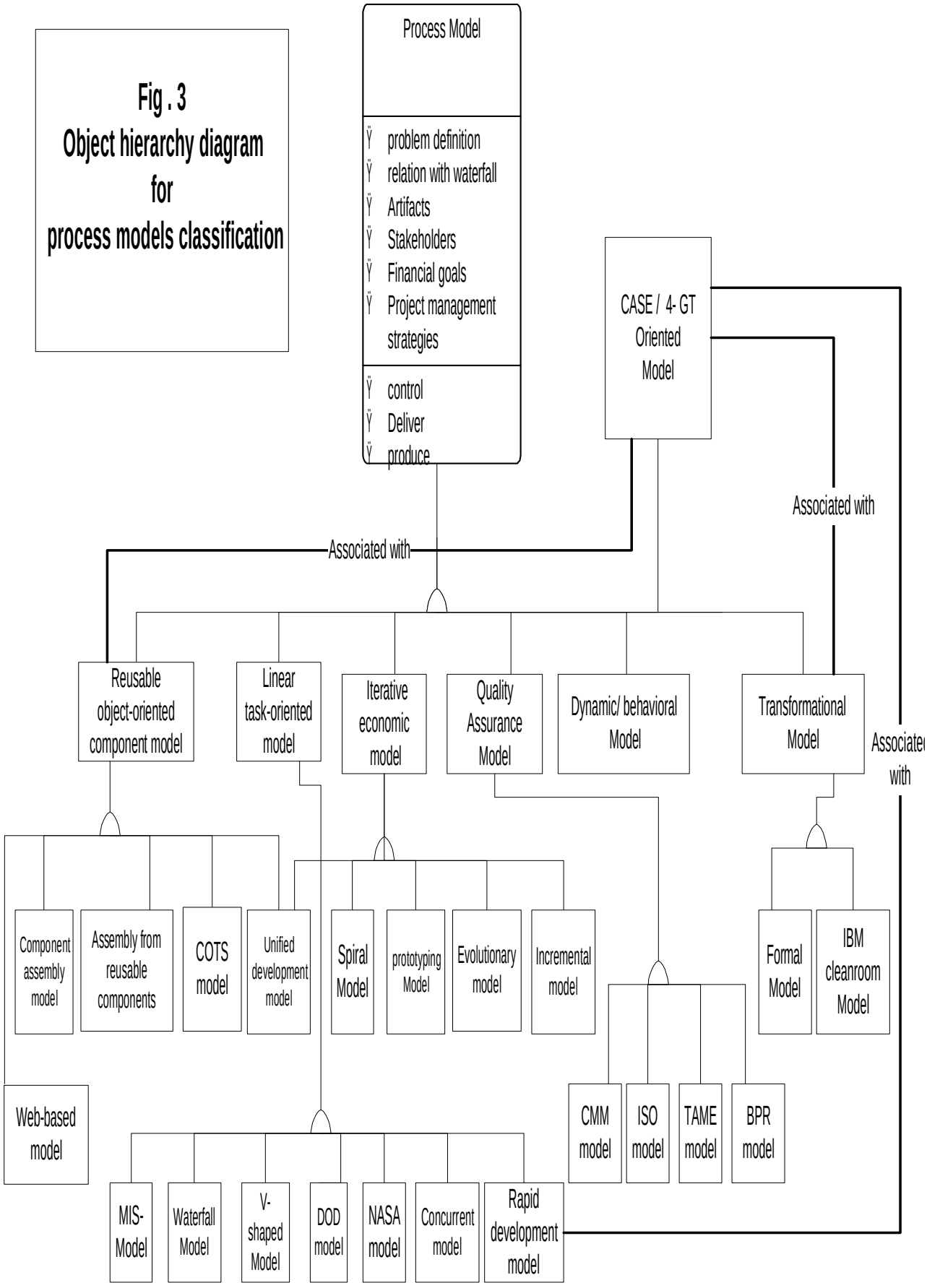
products much more rapidly than the other subclasses. Members of this category include: *Waterfall, V-shaped, MIS –Oriented, DOD, NASA, Concurrent, RAD and resource schedule models.*

2. Reusable object-oriented components models: Characterized by combinations of multiple process models and based on reusable components. Members of this category include: *Component assembly, assembly from reusable components, COTS, unified development and web-based models.* However, unified development model also multiple inherits from iterative modeling as well.
3. Quality assurance models: Typically focusing on process improvement in terms of CMM or ISO standards. Many subclasses in this category are associated with CASE tools, software automation and IT advancements Members of this category include: *capability maturity model (CMM), IS09000, TAME model and business process engineering (BPR) models*.
4. Dynamic Models: Behavioral and managerial considerations are crucial for this category. Heavy emphasis is on control by means of real world visualization and simulation. Therefore, Software automation can have a considerable effect on the efficiency of these models. Members of this category include: *Abdel-Hamid model and behavioral models.*
5. Iterative economic models: Economic considerations in terms of risk management and user's early inputs are major factors in these models. Members of this category include: *incremental model, prototyping model, evolutionary model, operational specification model, and spiral model in its different versions. The unified model is part of this group in terms of its highly iterative manner.*
6. Transformational models: Math and specification languages for later software automation distinguish this category. However, it is still limited due lack of human resources and expensive application. Members of this category include: *Formal model and IBM cleanroom model.*

7. 4-GT Models: Although this category does not have pure members, it is associated with several other members in other categories as it enables other process models to work more efficiently. Indeed, some of the other process models are totally dependent on these highly automated techniques provided by 4-GT environment. Members associated to this category include: *object-oriented models, RAD models and transformational models.*

Figure 3 demonstrates this classification in a class hierarchy diagram of process models based on Coad-Yourdan notations. This taxonomy is the platform for our later analysis in order to take adequate decisions and define profound criteria.

Fig . 3
Object hierarchy diagram
for
process models classification



3.4- Understanding project requirements:

Making an educated selection the many process models alternatives requires us to have thorough understanding of software project problems. In other words, software project are reflections of business problems in a way or another. Software project must be inline with organization goals and they are influenced by organization culture, politics and procedures including psychological, social and organizational processes at different levels [65]. More importantly software projects are no longer just affected by internal factors. Seeking the competitive advantage, market dominance, strategic opportunities are becoming more crucial than ever before [66]. One important factor in business problems is the project size as many previous efforts were trying to address large-scale projects similar to the way small-scale projects are addressed. Indeed, reliability of our software systems is a function of our recognition that building large-scale systems is a separate activity that has its own requirements [19]. Clearly, project size is associated often with problem complexity [67]. The more complex are organizations the more ill structured are their problems [68] and more hard to take are their decisions**.** On the one hand, their technical requirements in terms of information systems become more difficult to address. On the other hand, information technology might enable a complex organization to redesign its business processes so that it can manage complexity more effectively [69]. However, complexity can make software processes more difficult to handle for several reasons:

Firstly, the complexity of an organization increases the degree of ambiguity and equivocality in many of its operations [63]. Many organizations won't invest sufficient money to carry out a well representative analysis. Therefore, requirements specification becomes less concise and accurate. Implementing a system based on such a poor analysis will raise many failure issues as well as lack of compatibility with the diverse and contradicted types of needs. Raising analysis and feasibility budget for thorough determination of requirements might bring another dimension of complexity to the original problem. Secondly, the impact of technology is faced by more

people-determined resistance in the complex organization [70]. This happens not only because the system was not well engineered according to accurate requirements, but also due to a combination of social, psychological and political factors found in a complex organization. Finally, there is obviously a different rate of growth between a complex organization and information technology. While IT is growing very fast on an exponential curve, complex organizations grow relatively very slowly. Subsequently, digesting technical change becomes a serious challenge for many people and departments. This factor has several ingredients such as adaptability, training, upgradability and maintainability. Therefore, a negative correlation is most likely to take place with the case of the complex organization due to the above reasons as the more complex the organization, the less likely the impact of technical change will be influential [64]. Moreover, the more complexity exists in a project the more uncertainty is there. Controlling uncertainties is another crucial issue that should be considered in any solution. CASE tools have proven to be significantly efficient in reducing uncertainties as they can address the dynamic behavior of software systems and help in controlling users feedbacks [71].

As complexity increases uncertainty it also increases the exposure to risk. Although managers might be confident of their initial feasibility studies to process with a software project, they cannot avoid the significance of risk management. Determining risk different components is the most crucial step toward actively structuring and managing the risk of any large project [72].

Applegate et al [67] identified degree of structure, company-relative technology and project size as the most critical factors to assess project implementation risk. According to Applegate et al., one of the good techniques to manage risk is the portfolio approach. This approach is based on the diversification concept. Diversification stands here for the extent of how the several ongoing projects in a corporation are negatively correlated in terms of their risk measurements such as standard deviation and variance. The more negatively correlated are the projects the more likely that their risk magnitudes will cancel each other. In fact, the portfolio approach in managing risk among IS projects is borrowed from the portfolio management theory in finance.

Applegate et elaborated that in order to assess risk we should be able to identify its consequences such as: The outputs of the project are different than the anticipated ones; Actual Costs are higher than expected cots ;Time of implementation is higher than the expected time ;Technical performance is below the expected performance ;The system is incompatible with the non-functional hardware and software requirements.

According to Royce [73] some of the critical dimensions that should be considered in software projects when tailoring a process framework to create a practical process implementation are: Scale, stakeholder cohesion or contention, process flexibility or rigor, process maturity, architectural risk and domain experience. Blackburn et al. [41] suggested that influential factors in software projects include: project duration, size in terms of lines of codes, number of modules, newness of the project and team size during the project.

3.4.1- Process models evaluation procedure against general projects requirements:

3.4.1.1- identifying the relevant criteria items and their measurements:

From the requirements definition above, we are now able to extract the dimensions that are more significant to our evaluation criteria. Each dimension will be associated with its measurement tools. In the evaluation matrix, dimensions will serve as attributes and measurements tools will be their sub- attributes.

3.4.1.2- Conducting the evaluation

3.4.1.2.1- Establishing initial evaluation matrix

In the *quantitative scales* we will use the following table as an initial guide in structuring the evaluation criteria, sub-criteria and options according to the *L/M/H* approach where L indicates LOW, and M indicates MIDIUM, and H indicates HIGH .The following evaluation matrix represents the approximate results that each of the process model classes (categories) has achieved in our test. The 4-GT class is eliminated, as it represents supportive technology rather than pure process models.

This initial guide is then utilized in a more sophisticated CASE tool named DECIDE-RIGHT version 1.2 to map weights-and-scores in a more efficient manner .The scores assigned in this CASE tool are based on how much each of the criteria items has been addressed or applied in the process models class in the available literature or some well-known case studies, practices or experiences in respect of their real world implementations. However, in some cases where less knowledge is available, the published theories or techniques for some of these models have served as the basis of our inferences.

Attribute	Sub-attributes	Linear Models	Iterative Models	Object-oriented Models	Transformational Models	Dynamic Models	Quality Assurance Models
Problem Nature	Project size	M	H	H	M	H	H
	Project structure	M	H	H	M	H	H
	Project added-complexity	L	M	M	L	H	M
	Problem domain	H	L	L	H	L	L
Financial goals	Cost reduction	L	M	H	L	H	M
	Profit maximization	N/A	N/A	N/A	N/A	N/A	N/A
	Risk management	L	H	M	L	M	M
	Efficient utilization of resources	L	M	M	L	H	M
	Competitive advantage and market share	L	M	M	M	M	H
Capability Of control	Testing	H	H	H	H	H	H
	Feedback control	M	H	H	H	M	H
	Continuous improvement	L	M	M	L	M	H
	Visualization and simulation	L	L	M	L	H	L
	Dramatic Change	L	L	L	L	L	H

Customization	Flexibility To make changes	L	H	H	H	H	H
	Adaptable to available resources	L	M	M	L	H	M
	Adjustable to problem nature	L	M	H	M	M	M

3.4.1.2.2- Applying weights-scores method via CASE tool technology

The next step is to utilize this evaluation matrix structure in providing the sub-criteria items to the selected CASE (Decide-Right) tool and weigh them based on their importance project-wise. Then the six main alternatives/options are rated in terms of more accurate scores rather than the L/M/H approach based on the previous cited knowledge.

3.4.1.2.3- Extracting final CASE tool report

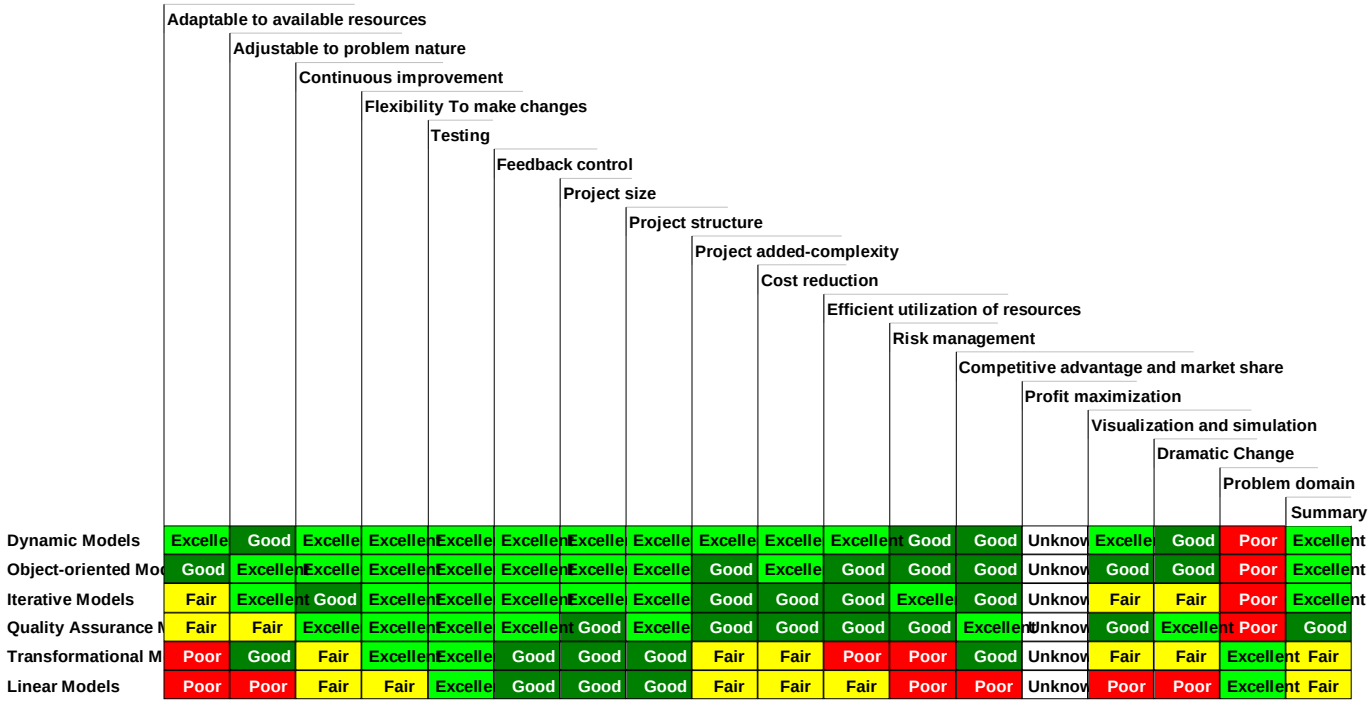
Based on the underlying inputs discussed above and automated capability of the selected CASE tool, a final report was generated as follows:

4. Evaluation of software process models report

Some elements in the decision table, which generated this report, are labeled "Unknown," and it may therefore be premature to draw conclusions from the data. The following sections address the major results and inferences.

1- Introduction

The question of "Evaluation of software process models" was evaluated by means of a decision table.



Decision Table for Evaluation of software process models

Alternative choices considered are listed down the left side of the table. The criteria used to evaluate the various options are listed along the top. Initially entered in no particular order, both the choices and the criteria were then repositioned according to importance of criteria and effectiveness of individual choices in meeting them. As criteria are evaluated and weights assigned according to which factors are considered to be most significant, the factors are sorted from left to right in order of importance (i.e., the factor considered by the decision maker to be most significant in meeting overall needs ends up in the leftmost position).

Similarly, as choices are evaluated according to effectiveness in meeting criteria, the best choices migrate to the top of the list. When the process is complete, the best choice should emerge at the top. As selection alternatives and the criteria to be used in evaluating them are entered into the table, weights are assigned to each of the evaluation factors so that they are ranked in order of their importance in fulfilling the overall task. For the decision "Evaluation of software process models," the criteria used to evaluate the choices, and their weightings, were:

Adaptable to available resources - High

Adjustable to problem nature - High

Continuous improvement - High

Flexibility to make changes - High

Testing - High

Feedback control - High

Project size - High

Project structure - High

Project added-complexity - High

Cost reduction - High

Efficient utilization of resources - High

Risk management - Medium

Competitive advantage and market share - Medium

Profit maximization - Medium

Visualization and simulation - Medium

Dramatic Change - Low

Problem domain - Low

Among the 6 choices considered, 3 were considered to be "top options" A top option is defined as follows: If the choice immediately following the preferred choice is rated in the same rating category as the recommended selection, then all choices in that category are considered top options. If the second ranking choice is in a different category, the top options are considered to be the recommended choice plus all choices in the same category as the second-place option. Thus, the "top options" list will always have at least two choices in it and may include all of the choices considered in the entire table.)

For the decision of "Evaluation of software process models," the top options were:

Dynamic Models

Object-oriented Models

Iterative Models

2- Discussion of Requirements

The criteria used in this decision making process were:

-Adaptable to available resources (overall importance: High)

-Adjustable to problem nature (overall importance: High)

-Continuous improvement (overall importance: High)

-Flexibility to make changes (overall importance: High)

-Testing (overall importance: High)

-Feedback control (overall importance: High)

-Project size (overall importance: High)

-Project structure (overall importance: High)

-Project added-complexity (overall importance: High)

-Cost reduction (overall importance: High)

-Efficient utilization of resources (overall importance: High)

-Risk management (overall importance: Medium)

-Competitive advantage and market share (overall importance: Medium)

-Profit maximization (overall importance: Medium)

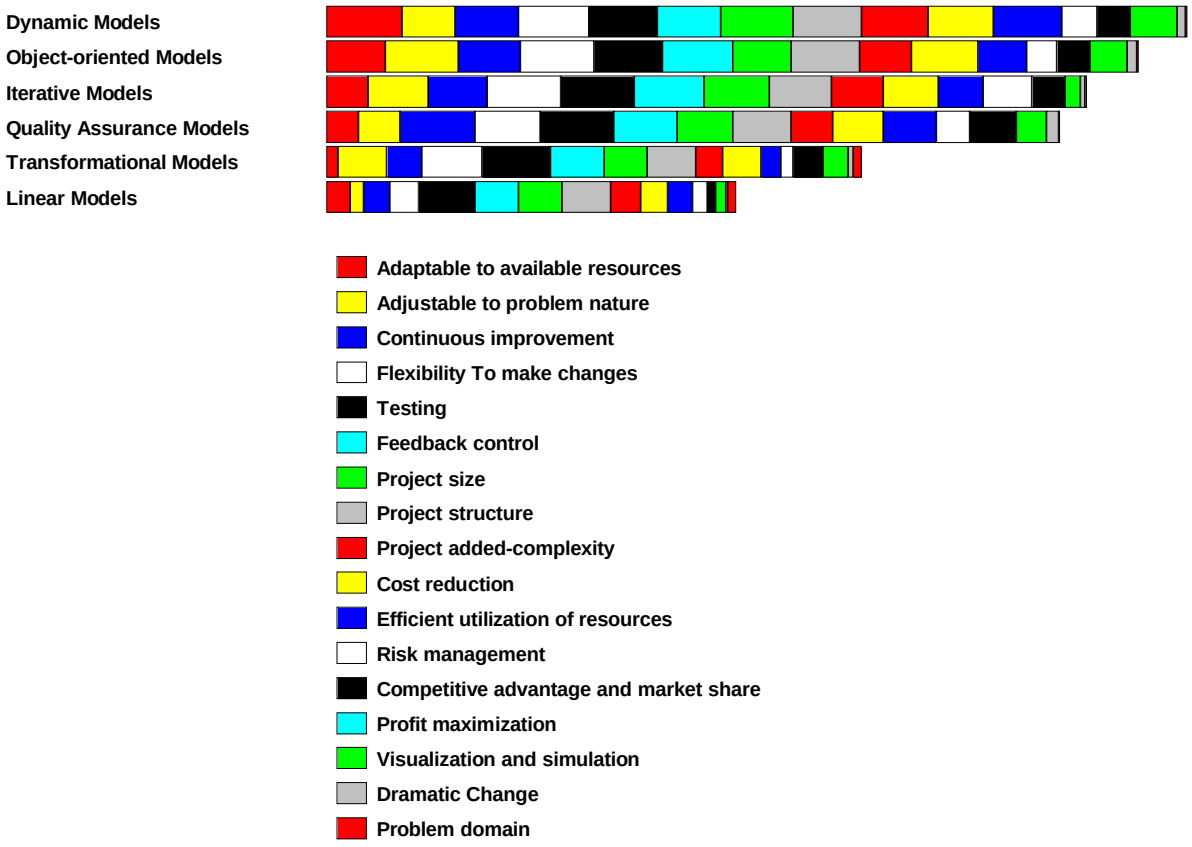
-Visualization and simulation (overall importance: Medium)

-Dramatic Change (overall importance: Low)

-Problem domain (overall importance: Low)

3- Comparisons among Choices

Relative strengths of the various choices in each of the factors is illustrated in the following graph:



Relative Strengths

3.4.1.3– Classical Decision Table for process model tailorability:

As an alternative decision table method based on the values of the projects evaluation dimensions addressed above was also applied .The following classical decision table is constructed for process model class selection.

Conditions	**Rules**															
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Problem Nature	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
Financial goals	Y	Y	Y	N	N	N	N	Y	N	N	N	Y	Y	Y	Y	N
Capability Of control	Y	N	Y	N	Y	N	Y	N	N	Y	N	Y	N	Y	N	Y
Customization	Y	N	N	N	N	Y	Y	Y	N	Y	Y	Y	Y	N	N	N

Actions																
Linear Models									**X**							
Iterative models						**X**										
Object- oriented Models	**X**															
Transformational Models									**X**							
Dynamic models								**X**								
Quality assurance models							**X**									

Rules values Legend:
Y= High score in the evaluation matrix
N= Not a high score in the evaluation matrix

5. Conclusion:

Process diversity is a remarkable observation in software process modeling literature. This diversity has its cons and pros as it represents both what is really needed and what should be overcome. On the one hand, it reflects evolution from outdated approaches of developing software systems to more optimized approaches that project changes in business requirements, technological capabilities, methodologies and developers' experience. It also represents the increasing involvement of more interdisciplinary impacts in modeling software processes. This implies that over the time, process models become more mature in their capabilities to address evolving project requirements. This maturity can be sometimes achieved by shifting from old models to the most updated versions of modern successful models based on specific criteria obtained via requirements engineering. A comparison chart among existing software process models can be extremely helpful in accomplishing this goal. However, maturity can be also achieved by consciously improving the current process model to adapt it to the evolving business process in the real world system. On the

other hand, process diversity might represent the different dimentions of projects needs. In other words, many individual software process models are only capturing part of user requirements. When combined, they can be more effective in responding to requirements. In this case, the tailoring process shouldn't tend to compare between a process model and another but to integrate several process models in some well-engineered combinations. These combinations can offer comprehension in capturing requirements and can be designed to have flexible and adaptive solutions to respond to a variety of software projects' needs.

Based on the previous analysis, one can observe three significant aspects in software process modeling evolution:

- Firstly, in several models a combination of process models was used to construct the final model. For instance, in the components assembly model, a general iterative approach, a spiral model and object-oriented methodology were used to construct the model. If this was combined with some 4-GT techniques, it can have a significant affect on the performance of the model since many relevant software solutions are becoming more available in the market.

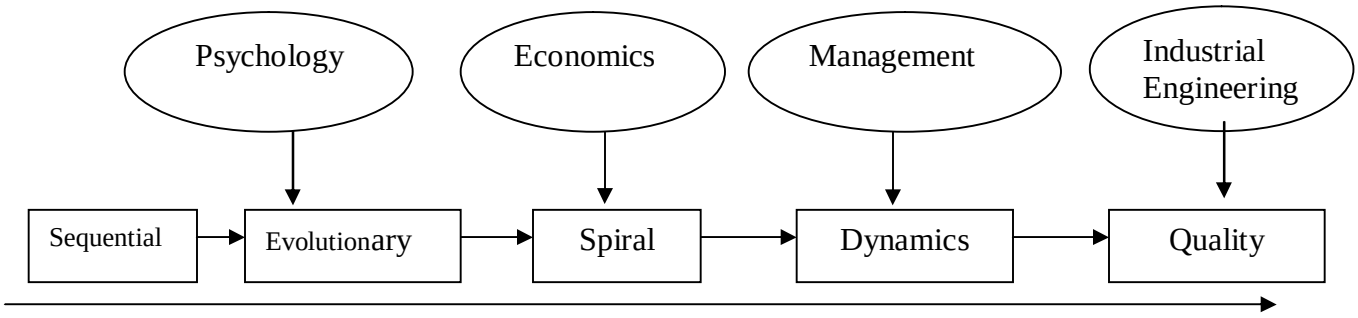


Figure 4. The impact of time and interdisciplinary impacts variables on evolution of process modeling

- The second observation is the combined impact of two independent variables on the evolution of software process modeling. These independent variables are the time

dimension and the interdisciplinary impacts. Figure 4 demonstrates a partial picture of this aspect in a more visualized manner.

- The third observation is the evolution the degree of visualization across process models. While initial models such as the waterfall, the evolutionary, and the spiral models were demonstrating the static view of the software development process, the behavioral models showed explicitly the dynamic picture of real world software development processes. By the arrival of process improvement models accompanied by the state-of-the –art advancements in CASE tool technology are now able to monitor the development process in a three dimensional picture with full simulations of the dynamic behavior of the process. Clearly, this adds to the goal of controlling the software processes more efficiently. Figure 5 shows this evolution in visualization capability.

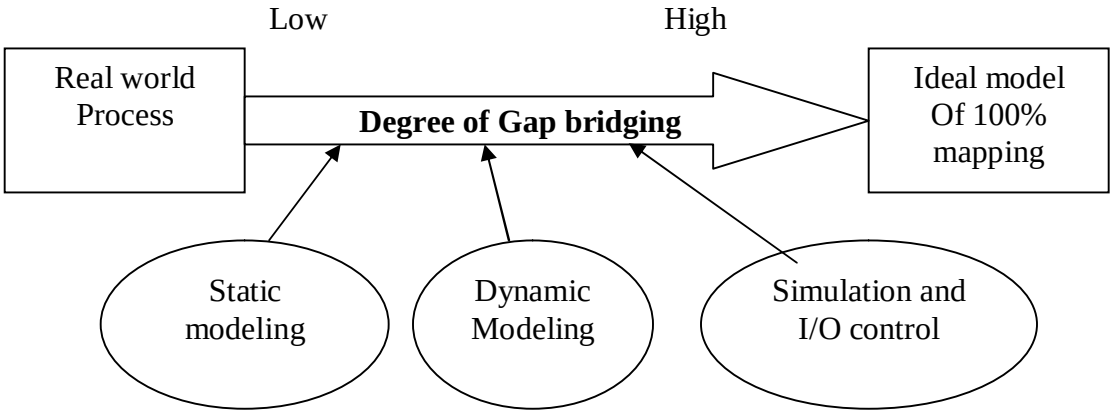


Figure 5. Evolution of software process models capabilities of visualizing real world development

According to the business CASE tool (DECIDE-RIGHT) utilization for the purpose of process modeling evaluation, an automated decision table was developed which had significant results. Based on a careful evaluation of how well each of the 6 possible choices could meet the 17 major criteria considered, Dynamic Models appears to be the best choice. The 6 choices considered were:

1. Dynamic Models
2. Object-oriented Models

3. Iterative Models
4. Quality Assurance Models
5. Transformational Models
6. Linear Models

The criteria used to evaluate the options were (in order of importance):

1. Adaptable to available resources
2. Adjustable to problem nature
3. Continuous improvement
4. Flexibility To make changes
5. Testing
6. Feedback control
7. Project size
8. Project structure
9. Project added-complexity
10. Cost reduction
11. Efficient utilization of resources
12. Risk management
13. Competitive advantage and market share
14. Profit maximization
15. Visualization and simulation
16. Dramatic Change
17. Problem domain

Of all of the choices considered, 3 were considered to be leading candidates. These "top options" were:

Dynamic Models

Object-oriented Models

Iterative Models

Efficient utilization of resources was the most significant factor leading to the choice of Dynamic Models over Object-oriented Models. Adaptable to available resources was the most significant factor leading to the choice of Dynamic Models over Iterative Models.

Comparing the three top options obtained from the utilized CASE tool, we can also draw the following inferences:

1- Dynamic Models versus Object-oriented Models

Dynamic Models was considered to be a better choice than Object-oriented Models in eight of the 17 criteria considered. Of these, the critical factors were:

Efficient utilization of resources

Adaptable to available resources

2- Object-oriented Models versus Iterative Models

Object-oriented Models was considered to be a better choice than Iterative Models in 10 of the 17 criteria considered. Of these, the critical factors were:

Visualization and simulation

Adaptable to available resources

After a careful evaluation of each option, dynamic models appear to be the best choice. From the classical decision table illustrated above, we can conclude that to meet the maximum number of requirements for software projects the object-oriented models class may be used. Object oriented models have the advantage of combining risk management, reusability and iterative feed back control in their structure. Because object components are loosely coupled and highly cohesive, they feature a better quality in process design with high degree of flexibility. This is an extremely desired feature for the purpose of customization and adaptability. Iterative models are more adequate when projects have more focus on problem nature and customization. Dynamic models are more suitable when projects are considering a combination problem nature, financial goals, and customization dimensions. Quality assurance models work better when projects are considering problem nature,

capability of control and customization. However, there are different levels of process models capabilities for each one of the evaluation dimensions as addressed in the previous evaluation matrix.

According to the classical decision table, both linear models class and transformational model class were unsuccessful to meet essential projects requirements whether separately or in combination. However, this doesn't imply that these process model classes should be avoided or ignored. Needs and requirements vary from an organization to another. In fact, linear models were not only incorporated with the other process models but also have proven to be effective with several large-scale projects such as NASA and DOD applications.

Transformational models have been adopted by IBM very successfully and proven to be the most accurate solutions for software process automation. However, due to their relatively high costs and need for trained human resources, their application was considerably limited. Clearly, there is a discrepancy between the results obtained by the automated decision table method and the classical one. This discrepancy can be attributed to the following factors:

1. The automated method used a more numerical accurate input whereas the classical method was based on the L/M/H differences, which have a high margin of error.
2. The classical method did not consider the numerical combined effect of criteria items on final results, which can add to the margin of error.
3. None of these methods considered the variations of scenarios that can result from combining process models themselves and its impact of process model performance against different criteria items.

It is clear from the previous analysis in this paper that combined models can add significant value to process modeling performance. The following examples illustrate this issue:

Spiral model = iterative model + linear model

Components assembly model= OO model + spiral model + 4GT

IBM clean room = transformational model + incremental model + 4GT + Dynamic model

It was also clear that the more iteration exists in a process model, the more efficient will be the model. Iteration implies more customer satisfaction and user involvement. Therefore, it is left for future research efforts to study the impact of customer dominance in today's software markets on software process modeling as the company strategy became more customers focused. This is particularly crucial with the increasing impact of virtual e-business in the Internet age. These future studies should also consider the utilization of data mining, as an intervening variable that might play a significant role in identifying customer needs in a more efficient manner. This is also critical as strategic investment in organizations is more concentrated on data collection and data analysis software tools.

This paper has also discussed the effect of basic dimensions of project requirements on the selection of adequate software process models. It is becoming clear that software project is becoming more and more influenced by the customer involvement and the financial goals of the organization as it is no longer valid to work in a project that doesn't guarantee customer satisfaction or achieve a great deal of competitive advantage for the firm. As software production is getting more competitive in nature, the *killer applications* are becoming of more impact on development teams. This implies that more accurate and comprehensive metrics are needed to assess product quality and orient decision criteria definition. However, there are other project factors that were not explicitly addressed. For instance, project management was tackled from the management across projects angle but the issue of project management within projects was not addressed as many software organizations deal today with a portfolio of multiple projects rather than single projects.

Moreover, the effect of process technology is becoming a significant factor in process modeling. This paper has addressed the issue of 4GT modeling and its association with several process models. In addition, the paper introduced the effect of software technology more explicitly in the process models comparison matrix. CASE tools are advancing very rapidly from documentation-generators to code-generators to concept-to-code approaches that tend to provide

complete automated solutions for today's business problems. However, the paper did not discuss that in details especially with the increasing importance of workflow management software as an alternative to CASE tools.

In sum, business metrics expressed in this paper in terms of financial goals are gaining more attentions today. Furthermore, organizations are becoming more integrated with their software systems, which should be adequately mapped in process modeling in parallel with business targets. The dynamic behavior of real world systems is a crucial issue that should be carefully monitored and controlled. Part of mapping dynamics is the iterative feedback control and the other part is dynamic modeling. However, with the usage of reusable components such as business objects, more success might be achieved in terms of cost reductions and reliability. It might be extremely difficult to develop a one static process model that fits all projects needs. However, with the adoption of system dynamics utilizing simulation techniques, the iterative approach and reusable components, a customizable model can be achieved with great deal of flexibility to work in a variety of situations. Clearly, this adaptive model should consider spiral model features to assess risk .It can also benefit from the transformational approach in creating more automation with the assistance of 4GT and work flow management software. This adaptive model should continuously improve software quality as the organization process matures via CMM and ISO 12207 standards. In addition, it can benefit from the reengineering approaches in re-designing the whole process with more synchronization with the real world business process. In other words, this final model is a dynamic-iterative version of a combination of all successful features of the process model classes addressed in this paper.

References

[1] Benbasat, I., and Taylor, R. N., "Behavioral Aspects of Information Processing for the Design of Management Information Systems", IEEE Transactions on Systems, Man, and Cybernetics, Volume SMC-12 (4), pp. 439-450, July-August 1983.

[2] Tilley, Scott R., "A Reverse-Engineering Environment Framework", Software Engineering Institute, Technical Report Cmu/Sei-98-Tr-005 Esc-Tr-98-005, April 1998.

[3] Victor R. Basili and H. Dieter Rombach., "The TAME Project: Towards Improvement-Oriented Software Environments", IEEE Transactions on Software Engineering, vol. SE-14, no. 6, pp. 752-772, June 1988.

[4] Paulk, M. et al.," Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1993.

[5] Leon J. Osterweil, "Software processes are software too, revisited: an invited talk on the most influential paper of ICSE 9", Proceedings of the 1997 International Conference on Software engineering, ACM, pp. 540 – 548, 1997.

[6] Watts S. Humphrey and Marc I. Kellner, "Software process modeling: principles of entity process models", Proceedings of the 11th international conference on Software engineering, pp. 331 – 342, 1989.

[7] Bandinelli, S. et al., "Modeling and Improving an Industrial Software Process", IEEE Transactions on Software Engineering, vol .21, no. 5, pp. 440-454, May 1995.

[8] W. L. Sutton, "Advanced models of the software process", Proceedings of the 4th international software process workshop on Representing and enacting the software process, pp. 156 – 158, 1988.

[9] Mark E. Nissen, "Valuing IT through virtual process measurement", Proceedings of International Conference of Information Systems, Vancouver, Canada, 1994.

[10] Maria Letizia Jaccheri, Gian Pietro Picco and Patricia Lago, "Eliciting software process models with the E3 language", ACM Transactions on Software Engineering Methodology, vol. 7, no. 4, pp. 368 – 410, Oct. 1998.

[11] Roger Pressman, "Software Engineering: A Practitioner's Approach", 4th Edition, McGraw-Hill, ISBN 0070521824- 1438, 1996.

[12] Tarek Abdel-Hamid and Stuart E. Madnick, "Lessons learned from modeling the dynamics of software development", Communications of the ACM, vol. 32, no. 12, pp .14-26, Dec. 1989.

[13] B. Boehm, "Software Engineering Economics", IEEE Transactions on Software Engineering, vol. 10, no. 1, pp. 4 - 21, January 1984.

[14] Bill Curtis, Marc I. Kellner and Jim Over, "Process modeling", Communications of the ACM vol. 32, no. 9, pp. 75 – 90, Sep. 1992.

[15] Victor R. Basili, "Iterative Enhancement: A Practical Technique for Software Development", IEEE Transactions on Software Engineering, v. ~SE-1, pp. 390-396, Dec 1975.

[16] Jonathan E. Cook and Alexander L. Wolf, "Discovering models of software processes from event-based data", ACM Transactions Software Engineering Methodology, vol. 3, pp. 215 – 249 Jul. 1998.

[17] Agarwal, R., De, P.; Sinha, A.P., "Comprehending object and process models: an empirical study", IEEE Transactions on Software Engineering, vol. 25, no. 4, pp. 541 –556, July/August 1999.

[18] Shaoying Liu, Offutt, A.J., Ho-Stuart, C., Sun, Y., and Ohba, M., “SOFL: a formal engineering methodology for industrial applications“, IEEE Transactions on Software Engineering, Volume: 24 Issue: 1, pp. 24 –45, Jan. 1998.

[19] Frank DeRemer and Hans H. Kron. "Programming-in-the-Large Versus Programming-in-the-Small", IEEE Transactions on Software Engineering, v. ~SE-2, n. ~2, pp. 80-86, June 1976.

[20] Barry Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, vol.21, no. 5, pp. 61-72, May 1988.

[21] Ian Somerville, Software Engineering, Addison-Wesley, ISBN 0-201-17568-1, 1995.

[22] Behforooz, Ali, Software Engineering Fundamentals, Oxford University Press, 1996.

[23] Ivar Jacobson, Grady Booch, James Rambaugh, The Unified Software Development Process, Addison Wesley, 1998.

[24] Carmen J., Trammell, Leon H. Binder and Catherine E. Snyder, “The automated production control documentation system: a case study in cleanroom software engineering”, ACM Transactions on Software Engineering Methodology, vol. 1, no. 1, pp. 81 – 94, Jan. 1992.

[25] John D. Riley, “An object-oriented approach to software process modeling and definition”, Proceedings of the 1994 conference on TRI-Ada '94, pp. 16 – 22,1994.

[26] Ropponen, J., Lyytinen, K., “Components of software development risk: how to address them? A project manager survey”, IEEE Transactions on Software Engineering, vol. 26, no. 2, pp. 98 - 112, February 2000.

[27] Bradac, M., D. Perry, and L. Votta, "Prototyping a Process Monitoring Experiment", IEEE Transactions on Software Engineering, vol. 20, no .10, pp. 774-784, October 1994.

[28] Barry Boehm and Frank Belz, “Experiences with the spiral model as a process model generator”, Proceedings of the 5th international software process workshop on Experience with software process models, pp. 43 – 45, 1990.

[29] W. Morven Gentleman, “Effective use of COTS (commercial-off-the-shelf) software components in long-lived systems”, (tutorial) ACM Proceedings of the 1997 International Conference on Software Engineering, pp. 635 – 636, 1997.

[30] Karen Lantner, “A Software Development Process for COTS-Based Information System Infrastructure: Part 1”, Greg Fox, TRW Systems Integration Group, EDS Steven Marcom, TRW Information Services Division.

[31] Christine L. Braun, “A lifecycle process for the effective reuse of commercial off-the-shelf (COTS) software”, ACM Proceedings of the Fifth Symposium on Software Reusability, pp. 29 – 36, 1999.

[32] Jung, Reinhard and Robert Winter, “Case for WEB SITES Towards an Integration of Traditional Case Concepts and Novel Development Tools,” Institute for Information Management University of St. Gallen, http:\\iwi1.unsg.ch\research\webcase, 1998.

[33] Weske, M., Goesmann, T., Holten, R., Striemer, R., “A Reference Model for Workflow Application Development Processes”, In: Georgakopoulos, D., Prinz, W., Wolf, A. L. (Hrsg.): Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration WACC '99, San Francisco, CA, 22. -25.2.1999, S. 1-10, also published in: Software Engineering Notes, vol. 24, no. 2, 1999.

[34] Jayashree Ramanathan and Soumitra Sarkar, "Providing Customized Assistance for Software Lifecycle Approaches", IEEE Transactions on Software Engineering, v. ~14, n. ~6, pp.749-757, June 1988.

[35] Torli, K.; Matsumoto, K.; Nakakoji, K.; Takada, Y.; Takada, S.; Shims, K, “Ginger2: An environment for computer-aided empirical software engineering, “IEEE Transactions on Software Engineering, vol. 25 no. 4, pp. 474 – 491, July/August 1999.

[36] Carmen J. Trammell, Leon H. Binder and Catharine E. Snyder, “The automated production control documentation system: A case study in cleanroom software engineering,“ ACM Transactions Software Engineering Methodology, vol. 1, no. 1, pp. 81 – 94, Jan. 1992.

[37] Somerville, I.; Sawyer, P.; Viller, S., “Managing process inconsistency using viewpoints,” IEEE Transactions on Software Engineering, vol. 25, no. 6, pp. 784 –799, Nov.-December 1999.

[38] Leveson, N.G., “Intent specifications: An approach to building human-centered specifications”, IEEE Transactions on Software Engineering, vol. 26, no. 1, pp. 15 – 35, January 2000.

[39] J. D. Chase, Robert S. Schulman, H. Rex Hartson and Deborah Hix, "Development and evaluation of a taxonomical model of behavioral representation techniques", ACM Conference Proceedings on Human Factors in Computing Systems: Celebrating Interdependence,” pp. 159 – 165, 1994.

[40] Liu, L. and Horowitz, E.,"A Formal Model For Software Project Management”, IEEE Transactions on Software Engineering, vol. 15, no. 10, pp. 1280 - 1293, October 1989.

[41] W. S. Humphrey, “Software engineering process: Definition and scope”, Proceedings of the 4th International Software Process Workshop on Representing and Enacting the Software Process, pp. 82 – 83, 1988.

[42] Blackburn, J.D., Scudder, G.D., Van Wassenhove, L.N., “Improving speed and productivity of software development: a global survey of software developers”, IEEE Transactions on Software Engineering, vol. 22, no 12, pp.875 – 885, December 1996.

[43] Armitage, James W. and Marc I. Kellner., "A Conceptual Schema for Process Definitions and Models", Proceedings of the 3rd International Conference on the Software Process, pp. 153 - 165, October 1994.

[44] Madhavji, N.H., Hoeltje, D., Hong, W. and Bruckhaus, T., ”Elicit: A Method for Eliciting Process models“, Proceedings 3rd International Conference on Software Process, pp. 111 - 122, 1994.

[45] The process cycle. Software Engineering Journal, IEE and The British Computer Society, September 1991, vol. 6, no. 5, pp. 234 - 242. Reprinted in: Process-centered Software Engineering Environments, (Eds. Garg and Jazayeri, IEEE Computer Society Press, pp. 50 - 58, 1996.

[46] Khalifa, M., Verner, J. Khalifa, M., Verner, J.M. M., “Drivers for software development method usage“, IEEE Transactions on Engineering Management, vol. 47, no 3, pp. 360 – 369, August 2000.

[47] Martin, Robert, Raffo David, “A Comparison of Software Process Modeling Techniques,” pp. 577 - 580, July 1997.

[48] H. Krasner, J. Terrel, A. Linehan, P. Arnold, and W.H. Ett., “Lessons learned from a software process modeling system”, Communications of the ACM, vol. 35, no. 9, pp. 91 - 100, 1992.

[49] Boehm, B, "Anchoring the Software Process", IEEE Software, July 1996.

[50] Madhavji, N.H., Hoeltje, D., Hong, W. and Bruckhaus, T.,“Elicit: A Method for Eliciting Process models, “Proceedings of the 3rd International Conference on Software Process, pp. 111 - 122, 1994.

[51] Bruce I. Blum, “Taxonomy of Software Development Methods”, Communications of the ACM, vol. 37, no. 11) pp. 82 - 94, 1994.

[52] Boehm, B. and Port, D., “Escaping the software tar pit: Model clashes and how to avoid them”, Software Engineering Notes, vol. 24, no. 1, pp. 36 - 48, January 1999.

[53] Kadary, V., Even-Tsur, D. Halperin, N., Koenig, S, “Software life cycle models-industrial implication”, Proceedings of Fourth Israel Conference on Computer Systems and Software Engineering, pp. 98 – 103, 1989.

[54] El-Emam, K., and Birk, A.,“Validating the ISO/IEC 15504 Measure of Software Requirements Analysis Process Capability”, IEEE Transactions on Software Engineering, pp. 541 - 566, 2000.

[55] John H. Baumert, Quality Time, "Process Assessment with a Project Focus", IEEE software, pp. 89 - 91, March 1994.

[56] Yamamichi,N., Ozeki,T., Yokochi, K., Tanaka, T., “The evaluation of new software developing process based on a spiral modeling”, Global Telecommunications Conference: The Key to Global Prosperity, GLOBECOM '96, vol. 3, pp. 2007 - 2012, 1996.

[57] Thomas J. Cheatham and John H. Crenshaw,“Object-oriented vs. waterfall software development”, Proceedings of the 19th annual Conference on Computer Science Conference, pp. 595 – 599, 1991.

[58] Shari Lawrence Pfleeger, Software Engineering: Theory and Practice, 1998.

[59] Alavi, M., "An Assessment of the Prototyping Approach to Information Systems Development", CACM, vol. 27, no. 6, pp. 556 - 563, June 1984.

[60] Lichter, Horst, Matthias Schneider-Hufschmidt, Heinz Zullighoven, "Prototyping in Industrial Software Projects", IEEE Transactions on Software Engineering, vol 20, no. 11, pp. 825 - 832, 1989.

[61] D. Graham, “Incremental Development and Delivery for large Software Systems“, Colloquium on Software Prototyping and Evolutionary Development, IEE, 11, November 1992.

[62] Royce W., "TRW's Ada Process model for Incremental Development of Large Software Systems", TRW Technologies Series, TRW-TS-90-01, January 1990.

[63] Daft, R. L., and Lengel, R. H., "Organizational Information Requirements, Media Richness and Structural Design", Management Science vol. 32, no. 5, pp. 556 - 557, 1986.

[64] Keen, Peter G.W., " Information Systems and Organizational Change", Communications of the ACM, vol. 24, no. 1, pp. 24 - 33, January 1981.

[65] Curtis, Krasner, and Iscoe, "A field study of the software design process for large systems", Communications of the ACM, vol. 31, no. 11, pp. 1268 - 1287, November 1988.

[66] Nicola Gibson, Christopher P. Holland and Ben Light, "Enterprise Resource Planning: A Business Approach to Systems Development", Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences, 1998.

[67] Lynda Applegate et al., "Corporate Information Systems Management: Text and Case", Irwin Publishers, Fifth Edition, 1999.

[68] Mitroff, Ian and Murray Turoff, "Technological Forecasting and Assessment: Science and/or Mythology?," Journal of Technological Forecasting and Social Change, vol. 5, pp. 13 - 134, 1973.

[69] Davenport, T.H. and D. B. Stoddard, "Reengineering: Business Change of Mythic Proportions?" MIS Quarterly, vol. 17, no. 2, pp. 125, July 1994.

[70] Markus, M. Lynne, "Power, Politics, and MIS Implementation", Communications of the ACM, vol. 26, no. 6, pp. 433, 1983.

[71] M.M. Lehman, "Software engineering, the software process and their support", IEEE Software Engineering Journal, vol. 6, no. 5, pp. 243 - 258, 1991.

[72] Mantei and Teorey, "Cost Benefit analysis for incorporating human factors in the software lifecycle", Communications of the ACM, pp. 31, April 1988.

[73] Royce, W., Software Project Management: A Unified Framework. Reading, MA: Addison Wesley Longman, 1998.

[74] Lindvall, M. and Rus, I., "Process diversity in software development", IEEE Software, vol. 17, no. 4, July-August 2000.