Chapter V
Process Models of SDLCs: Comparison and Evolution

Laura C. Rodríguez
Autonomous University of Aguascalientes, Mexico

Manuel Mora
Autonomous University of Aguascalientes, Mexico

Miguel Vargas Martin
University of Ontario Institute of Technology, Canada

Rory O'Connor
Dublin City University, Ireland

Francisco Alvarez
Autonomous University of Aguascalientes, Mexico

ABSTRACT

The software engineering discipline has developed the concept of software process to guide development teams towards a high-quality end product to be delivered on-time and within the planned budget. Consequently, several software-systems development life-cycles (PM-SDLCs) have been theoretically formulated and empirically tested over the years. In this chapter, a conceptual research methodology is used to review the state of the art on the main PM-SDLCs formulated for software-intensive systems, with the aim to answer the following research questions: (a) What are the main characteristics that describe the PM-SDLCs?, (b) What are the common and unique characteristics of such PM-SDLCs?, and (c) What are the main benefits and limitations of PM-SDLCs from a viewpoint of a conceptual analysis? This research is motivated by a gap in the literature on comprehensive studies that describe and compare the main PM-SDLCs and organizes a view of the large variety of PM-SDLCs.

INTRODUCTION

In order for a product to be developed, a development (formal, semi-formal, or informal) process is required. For the specific case of software artifacts, a software (development) process is a method of producing such artifacts. This process is usually denoted as **the software-systems development life-cycle.** To guide its execution under different design conditions, a set of process models have been also proposed: **process model of systems development life cycles** (PM-SDLCs). In general, the aim of each single process is "to facilitate the engineer doing the job well rather than to prevent them from doing it badly" (Tyrrel, 2000).

In the software engineering discipline, the concept of a software *process* has been developed to guide the development team on constructing a high-quality end product that be delivered on-time and within the planned budget. Consequently, several PM-SDLCs have been theoretically formulated and empirically tested over the years, and in general many have been an evolution of previous models. In some cases, the evolution is originated as a result of a major advance in information and communications technologies (ICT), and in other cases, as a result of more planned changes in the organizations' settings and their business environments.

In this chapter, we use a conceptual research methodology (Glass, Vessey, & Ramesh, 2002; Mora, 2004) to review the state of the art on the main PM-SDLCs formulated for software-intensive systems, with the aim to answer the following research questions: (a) What are the main characteristics that describe the PM-SDLCs?, (b) What are the common and the unique characteristics of such PM-SDLCs?, and (c) What are the main benefits and limitations of PM-SDLCs from a viewpoint of a conceptual analysis?

The conceptual research approach is widely used in the software engineering domain (Glass et al., 2002). According to Cournellis' ideas (2000)—quoted by Mora (2004)—this research method studies concepts, ideas, or constructs on empirical objects. This chapter uses the research methodology process, described in Mora, 2004, that consists of the following phases: (1) formulation of the research problem; (2) analysis of related studies; (3) development of the conceptual artifact; and (4) validation of the conceptual artifact. The first phase and second phases are similar to other well-known research methods. In the third phase, two activities are conducted: the development of a general framework/model and the detailed development of this general framework/model. This third phase is a creativity-intensive process guided by the findings, contributions, and limitations found in the second phase and a set of preliminary proforms that are fixed through an iterative process (Andoh-Baidoo, White, & Kasper, 2004). Finally, in the last phase, the conceptual artifact's validation can be conducted using a panel of experts, a logical argument discourse, or/and a proof of concept developing a prototype or pilot survey. In this study, we used the first procedure with two internal academic experts and an expert practitioner in the development of SwE projects. Satisfactory average scores of 4.6 in a 5-point Likert scale of an instrument conceptual composed of eight items was achieved (Mora, 2004).

This research is motivated by the knowledge gap in the literature on comprehensive studies that describe and compare the main available PM-SDLCs. The research relevance can be considered high because the main objective of software engineering is the development of high-quality, on-time, and within budget software projects, which can only be delivered with the utilization of a systematic development process, as has been proven in other engineering disciplines. Therefore, this study contributes to organize the diverse and partial views of PM-SDLCs.

BACKGROUND

Software engineering, according to the *IEEE Standard Computer Dictionary* (1990) is the: " (1) Application of quantifiable approach, disciplined to the software development, operation and maintenance;

this is, the application of the software engineering and (2) The study of the approaches that refers the point 1 of this definition." In turn, systems engineering can be defined as "an interdisciplinary field and their means for achieving the realization of successful systems" (INCOSE, 2004). Finally, the information systems discipline is "the study of the administration, use and impact of information technologies with the consideration of technical, socio-economic, cultural and organizational aspects" (Lee, 2004). From a software-intensive system's developer perspective, the foundation knowledge upon PM-SDLCs and their related disciplines becomes critical.

The relevance of studying PM-SDLCs has been reported in numerous studies. Fuggetta (2000) suggests, for example, the need for investigating the software process, because this lead to the completion of the most successful software products. For Fuggetta, the concept of life cycle is directly related to the notion of software process. A life cycle defines the different stages of software-product life: "…a software lifecycle defines the skeleton and philosophy according to which the software process has to be carried out." A life cycle also defines the *software process* as "the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain a software product."

In the domain of information systems, Avison and Fitzgerald (2003) observe the need to follow a development methodology: "… perhaps we are in danger of returning to the bad old days of the pre-methodology era and its lack of control, standards, and training." They define a development-systems methodology as the "…recommended collection of phases, procedures, rules, techniques, tools, documentation, management, and training used to develop a system." Further, they remark on the importance of the philosophy of the methodology, which is, all those assumptions that are not stated explicitly by the authors of a methodology, but, that ruled them out of being successful. For example, a methodology may not explicitly consider the size of projects, the environment, the technology or the organizational context. The authors finally suggest that the main reason for using a methodology is to get better products, improve the development process, and gain standards that ensure the quality of products.

Another relevant study (Sorensen, 1995) reports the concept of methodology as a model plus techniques. Under such premise, the author analyzes the models of Waterfall, Incremental, Spiral, and the techniques: Prototype[1], Cleanroom, and Object-oriented. Sorensen also suggests that a possible combination of method and techniques can be used. For example: waterfall method with prototyping and object-oriented techniques. Thus, if for each model there is a total of eight combinations (it could include the case where none technique used) of techniques that may be applied, then there are a total of 24 methodologies for the list of models and techniques reported by Sorensen (1995). With more models and techniques, the number of potential methodologies can be extensive.

The objective of software engineering is to build and maintain software-intensive systems under planned schedules, agreed functionality, and cost restrictions. Within the domain of software engineering, software process improvement (SPI) aims to improve the quality of development products and the productivity of software engineers. However, the existence of a large variety of PM-SDLCs obfuscates its understanding and practical usage. Furthermore, as PM-SDLCs have been posed in various disciplines—software engineering, systems engineering, and information systems—for academics and practitioners, this situation increases both learning curve and application complexity. While we support that an interdisciplinary development is worthy for scientific progress, we consider that a systematic view of the different proposals is required. Then, academics and practitioners could acquire a more shared mental model of the PM-SDLCs.

COMPARISON AND EVOLUTION OF PROCESS MODELS OF SDLCS

Well-Defined Software Process

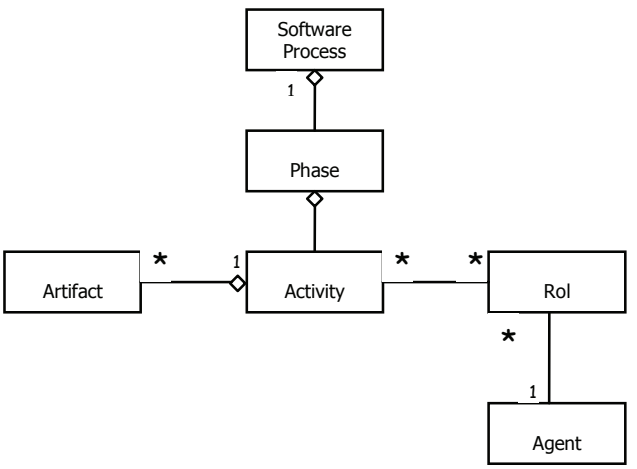
Oktaba and Ibargüengoita (1998) have developed a general meta-model of software process (Figure 1), which provides a 'parsimonious' and 'engineering-based' mode to conceptualize a well-defined software process. In this model, a process is composed of the following elements: *phases*, *activities*, *artifacts*, *roles*, and *agents*; where a *software process* is the main concept that is being modeled; a *phase* is the highest-activity level of a process that is being modeled; and an *activity* is the execution of an useful work to deliver a main *artifact* or artifacts (e.g., pieces of the full software artifact, documents, components, data files, or codes). The concepts of *role* and *agent* complete this semi-formal definition. A *role* is a functional responsibility in the process that is assigned to an *agent*, which can be a human-being, a tool, or a combination of both. The class *software process* is made up of the several instances of the *phase* class. Figure 1 shows the meta-model using a class diagram notation.

An instance of a *process* class is composed of several instances of *phase* class. *Phase* class is related with several instances of *activity* class. Under an operation of specialization, the authors report that the *phase* class can be specialized in *analysis*, *design*, *code and test*, and *installation* phases. Similarly, the *activity* class can be specialized in *production*, *control*, *technology*, and *communication* activities, and each one of these, in other specializations. The *activity* class is also related to at least an input *artifact* and an output one, represented by the instances of the *artifact* class. Specialization of *artifacts* is also feasible in the model. Finally, a many-to-many association between the *role* and *activity* classes and a one-to-many from *agent* and *role* classes are defined in the model. We use this meta-model as base for a conceptual framework to compare different process models. Under the consideration of each life cycle is an instance of the model; the comparative framework provides a theoretical base to develop instances for the generic classes of *phase*, *artifact*, and *role*. *Activity* and *agent* classes are not considered in this chapter.

Phases and Artifacts in the PM-SDLCs

It has been also reported (Fuggetta, 2000) that: *software applications are complex products that*

Figure 1. Well-defined software process model



are difficult to develop and test. Very often, software exhibits unexpected and undesired behaviors that may even cause severe problems and damages. For these reasons, researchers and practitioners have been paying increasing attention to understanding and improving the quality of the software being developed. The underlying assumption is that there is a direct correlation between the quality of the process and the quality of the developed software. The research area that deals with these issues is referred to using the term ***software process****.*

The large diversity of PM-SDLCs suggests, then, that apparently none of the PM-SDLCs is sufficient for covering all needs to guarantee a successful development of software-intensive systems. This study, then develops a comparison of the main PM-SDLCs based on their historical evolution, and in terms of their component structure (e.g., based in the Oktaba & Ibargüengoitia meta-model, 1998) to help organize the available knowledge on these models. For this, the following specialization of *phases* was identified in the same study: *user conditions, business context pre-systematization, component identification, requirements, analysis, design, coding, test, implementation, postmortem analysis,* and *iteration decisions*. These activities constitute the generic life-cycle (proposed in this chapter) that includes all *activities* of the PM-SDLC under study. The phases are proposed by considering all activities that are part of each phase of each PM-SDLC under study.

The 13 PM-SDLCs analyzed are: waterfall (Royce, 1970), SADT (Dickover, McGowa, & Ross, 1977), prototyping (Naumann & Jenkins, 1982), structured cycle (Yourdon, 1993), spiral (Boehm, 1988), win-win spiral (Boehm & Rose, 1994; Egyed & Boehm, 1998), unified process (Rational Software Corporation, 1998), MBASE (Center for Software Engineering, 1997), component-based cycles (Aoyama, 1998; Brown & Wallnau, 1996), XP (Beck, 1999), PSP (Humphrey, 2000), TSP (McAndrews, 2000), and RAD (Cross, 2006). Table 1 shows the comparative *framework*, for the *phase* class of the 13 PM-SDLCs analyzed. A scheme of three macro-phases (definition, development, and deployment) well-known in systems engineering is used to group the *phases* (Sage & Armstrong, 2000).

The symbol ♦ used in Table 1 indicates that the *phase* reported in the related row is part of the PM-SDLC reported in the corresponding column. No similar comparison was found in the literature. *Phases* are grouped by the general macro-phases: definition of the system, development of the system, and deployment of the system, a well-know systems engineering model. Table 1 contributes to organize comprehensively the phases reported of practically all public PM-SDLCs, and suggests from its analysis a set of generic phases.

The most relevant findings from Table 1 can be summarized as follows:

a. The set of common phases includes the **analysis**, **design, codification, testing,** and **implementation** phases (Note: the emergent agile-based systems approaches such as XP also support an engineering view of these phases);
b. The initial business and high-level systems *phases* (as part of the macro-phase definition of the system) are only part of some PM-SDLCs;
c. The iterative approach was disseminated by prototyping SDLC, but this was originally suggested in the Royce's[4] (1970) variant of the waterfall model. Later, was reinforced and extended by the spiral model; and
d. The *postmortem* phase, which appeared previous to year 2000, was indirectly suggested by MBASE and XP, and it is attributed mainly to PSP and TSP models.

It must be noted that the unique features are considered in the period of their formulation, then, some elements that were considered unique at once, later were incorporated to other models. Table 2 shows the comparative framework for the "*artifact*" component versus the several PM-SDLCs studied.

Table 1. Comparative framework for PM-SDLCs

Phases		Waterfall (1970)	SADT (1977)	Prototyping (1982)	Structured Cycle (1988)	Spiral (1988)	RAD (1991)	Win-Win Spiral (1994)	Unified Process (1998)	MBASE (1998)	Component-based cycles (1998)	XP (1999)	PSP (2000)	TSP (2000)
Definition	*User Conditions (Survey, Agreements, Stories)*				♦			♦		♦		♦		
	Business Context Pre-systematization				♦	♦	♦	♦	♦	♦	♦	♦	♦	
	Component Identification						♦				♦			
	Requirements	♦	♦	♦		♦	♦	♦	♦	♦				♦
	Analysis	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦		♦
Development	Design	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦
	Codification	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦
	Testing	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦
Deployment	Implementation	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦	♦
	Postmortem Analysis									♦[2]		♦[3]	♦	♦
	Evolution/ Iteration Decisions			♦		♦		♦		♦		♦		

We propose a comparative specialization of ***artifacts***. In this table, each number means the number of *artifacts* that are generated as equivalent to the *artifact* specialization proposed. Then, "1" implies that an *artifact* of the PM-SDLC (indicated in the column) is equivalent with one artifact of the *artifact* specialization proposed. For the cases of "2," "3," "4," and "5," they indicate that more of one *artifact* of the PM-SDLC (indicated in the column) corresponds to a unique *artifact* of the comparative specialization. A greater number of artifacts implies that the model aggregates more control or detail in the definition of such *artifacts*.

Due to space limitations, the specialization of *roles* is not reported here. However, we can report that in general, the *roles* of *agents-persons* have not suffered much variation, but the number of activities each one executes as well as the number of required agents has been increased. Additionally, the PM-SDLC descriptions usually do not report explicit information about *roles*.

From the previous tables and the conceptual analysis of each PM-SDLC, we identified a set of common, distinct, and unique features. Table 3 summarizes the common and distinct features, whereas the Table 4 summarizes the set of unique ones.

These distinct features remark the historic evolution and allow to establish a time-line evolution (based from Avison and Fitzgerald, 2004) of critical events of the PM-SDLCs that is shown in Table 5.

The time line (Table 5) shows how the several PM-SDLCs have been proposed since 1970s. The

Table 2. PM-SDLCs vs artifacts

Main Artifacts Delivered	Waterfall (1970)	SADT (1977)	Prototyping (1982)	Structured Cycle (1988)	Spiral (1988)	RAD (1991)	Win-Win Spiral (1994)	Unified Process (1998)	MBASE (1998)	Component-based cycles (1998)	XP (1999)	PSP (2000)	TSP (2000)
List of Stakeholders				1			1		1		1		
List of Functions				1									
List of Conditions							1		1				
Objectives, Constrains an Early Risks				4	1	1	1	5	1	1		1	
System Architecture						1			1	1			
Early Prototype					1	1	1		1				
Initial Operational Prototype					1		1						
Operational Concept					1		1	1	1			1	
Components						1				1			
Life Cycle Plan				1	3		3	3	4		1	2	
SRS	1	1	1	1	1		1					1	3
Design Specification	1				1		1						
General View of the System	1	1						2					3
Database Design	1	1		3	1		1		1				3
Subroutine Storage Allocations	1											1	
Subroutine Execution Times Allocations	1												
Operational Procedures	1					1							
Prototype	1	1	1					1					
Risks			2					3	5				
Integrated System				1	1		1		1	1	1	1	2
Prototype Review	1												
Interface Design Specification	1												
Interface Design Document	1												
Test Planning	1												
Review of Critical Software	1												

continued on following page

Table 2. continued

Final Design Document	1												
Testing and Results Document	1	1		1									
Final Acceptance User's Document	1				1		1	1					
Converted System		3		1				1					
Operational User's Manual	1			1				2	2				
Implemented System			1	2	1	1	1	2		1	1	1	3
System's Evolution Analysis Document						1						5	

Table 3. Common and distinct features for PM-SDLCs

Common Features	Distinct Features
1. **Group of activities that are performed to identify stakeholders and the set of user requirements and conditions**. First, these activities were focused to collect and specify requirements and then they were extended to business modeling, system conceptual definition, based-scenarios analysis and design, stories construction, and in some cases the building of prototypes. 2. **Group of activities that are performed to define the scope of system** (negotiation of stakeholders' conditions; objectives, alternatives and restrictions determination; risk analysis). First, these activities were limited to objectives, alternatives and restrictions determination, and risk analysis was then integrated. 3. **Group of activities that are performed to define the system architecture** (system-requirements definition, architecture design and analysis). First, these were limited to system-requirements definition and analysis but system architecture was integrated later as a common feature. 4. **Group of activities that are performed to design, build, test, and implement the software artifact** (design, coding, test, implementation). 5. **Five models consider the development of a prototype (**waterfall[5], SADT, spiral, prototyping, and RAD). 6. **Four models include explicitly the elaboration of the user manual** (waterfall, a modern structured cycle, unified process, and MBASE).	1. **Non-iterative, iterative, and incremental approaches**. Some methodologies carry out several iterations by repetitions of the phases of the same manner in each iteration (iterative approach like **unified process**), while others PM-SDLC carry out several iterations executing the phases with distinct tasks in each iteration (incremental approach like **spiral**). Other ones do not utilize iteration (like **structured cycle**). 2. **Iteration/increment next-entry-condition**. Some models execute the next iterations/increment strictly depending on some condition that indicates the product has reached these objectives. Others execute it in a certain number of iterations/ increments, independently of the total fulfillment of the condition. 3. **Level of detail of the formulation of the PM-SDLC.** There are significant differences between the models regarding to the level of detail used to formulate/describe the tasks and the concept of a well-defined process that the software engineering claims. 4. **Sophistication of techniques and tools**. The most recent models suggest the utilization of techniques and tools for analysis, design, codification, testing, and implementation of more sophisticated models.

historic evolution is since the 1950s with code & fix, to the 2000s with MBASE, unified process, component-based, XP, PSP, and TSP. Hence, a relevant research purpose that emerges is to explore the next evolution stage.

FUTURE TRENDS

We can generalize the evolution in terms of advances in "technology" (that push advances in PM-SDLCs) or advances in "knowledge" (demanded for a better project management control in PM-SDLCs). The distinct elements shown in Table 4 were unique at the moment in which this PM-SDLC was proposed. Lately, such "unique" features were included in subsequent PM-SDLCs. Such repetition of elements from a PM-SDLC to a new PM-SDLC suggests how the models evolve and the need to use PM-SDLC to develop success software systems. The evolution of PM-SDLCs is a response to the advances in technology, and to the interest in reinforcing the project management control weaknesses of the previous PM-SDLCs. This evolution indicates the scientific advance and the need to accumulate the previous knowledge on PM-SDLC. The challenge is to cope with this evolution caused by the appearance of new technologies, but also incorporating the knowledge gained in the formulation and utilization of previous PM-SDLCs.

The evolution map (Figure 2, extended from Rodriguez, Mora, & Alvarez, 2007), summarizes the evolution based in the two aforementioned drivers (technological advances and knowledge

Table 4. Set of unique features of PM-SDLCs

PM-SDLC	Unique Elements
Waterfall	Execution of preview phases to the coding phase for the development of a system (e.g., requirements, analysis, and design phases).
SADT	A graphic-based model of the process that was strongly influenced by theory of systems and hierarchical models from systems engineering.
Prototyping	First to suggest the building of an operational first version of the system, deploying the prototype, and then executing iterations to evolve the prototype till to generate sufficient level-functionality for the users' needs.
Structured Cycle	An explicit control of documentation and users' training activities in the SDLC. Based on a "*top down*" and "*divide and conquer*" design approaches.
Spiral	Explicit consideration of risk analysis as a critical part of the software-system definition.
Win-Win Spiral	Augments the spiral model, with the concept of "***win-conditions***," to strengthen the first phase of the SDLC trying to include all of stakeholders' conditions. It also carries out business reconciliation between stakeholders on the conditions agreed in each iteration of development cycle.
Unified Process	Conceptualization of the SDLC in two time-dimensions (phases execution and tasks executions). The level of effort conducted in each tasks varies according to the phase performed. A most-to-least effort shift and vice versa usually occurs for the first and last tasks respectively.
Component-based Cycles	Explicit reutilization and building of software components as the fundamental design approach.
XP	Explicit utilization of: (a) stories to know the environment and users' needs, (b) the coding by pairs, (c) the need of a multi-role training by agents that participate in the development cycle, and (d) the lean-manufacturing design approach.
PSP	A systematic and explicit quality control of activities that a single person must perform under the concept of software engineering.
TSP	A systematic and explicit quality control of activities that a development team must perform in each development phase (*launch/relaunch*, inspection, and analysis *postmortem*).
RAD	Explicit utilization of: (a) scenarios based-analysis, (b) specification of components to maximize the reuse, (c) rapid development, and (d) frequent tests. It is highly related to the original prototyping model with the utilization of CASE tools to support fast development.

Table 5. PM-SDLCs time line of critical events

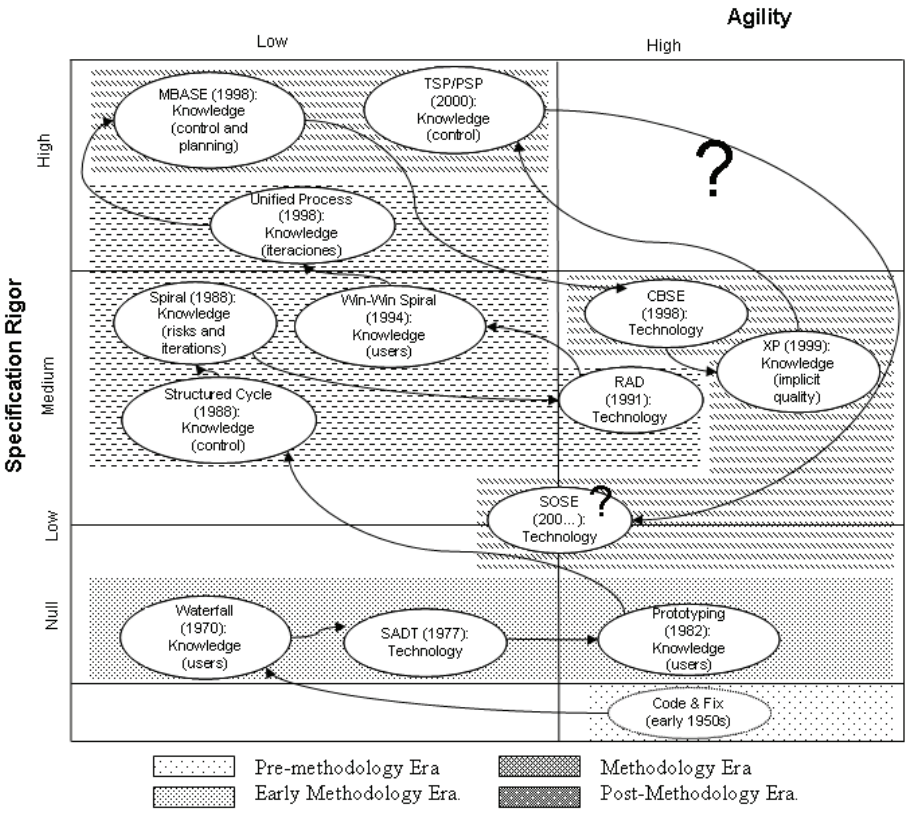
Pre-Methodology Era Early software development was often done using the simple technique of ***Code & Fix*** (Boehm, 1988); prior to 1956, a basic model used in this early days of software development included two steps: (1) write some code and (2) fix the problems within the code.
Early Methodology Era After 1956, the experience gained on the first large software projects led to the recognition of control problems and the development of a stage-wised model. This model stipulated that software products should be developed in successive stages (operational plan, operational, operational specifications, coding specifications, coding parameter testing, assembly testing, shakedown, and system evaluation, Boehm, 1988).
Methodology Era In 1970 Royce proposes the ***waterfall*** model, that includes for first time in its internal structure the risks and the use of prototypes as well as the inclusion of users in the development process. According to Boehm (1998), it provides two primary enhancements to stage-wised: "(1) the recognition of the feedback loops between stages to minimize the expensive rework involved in feed back across many stages and (2) an initial incorporation of preliminary prototype in the phase "do it twice", in parallel with requirements analysis and design;" later in the 1970s, the ***SADT*** improves the ***waterfall*** model a more controlled development and the addition of several specifications and appears ***prototyping*** that develops the initial concept of preliminary design ("***do it twice***") of a program, originally reported in the ***waterfall*** model. Next, evolution state corresponds to the emergence of the ***structured cycle***, ***spiral***, ***RAD*** and ***win-win spiral*** models that mainly integrate the risk analysis to avoid the accumulation of problems and reviews until the final *phases*. All of them, except ***structured cycle***, incorporate the advantages of ***prototyping*** about the early view of a system.
Post-Methodology Era A set of new models are developed: ***MBASE***, ***component-based***, ***XP***, ***PSP***, and ***TSP***, that are based in the re-utilization of components, CASE tools, and CMMI quality maturity frameworks, as well as the emergent agile systems-development paradigms. All of them with the aim to accelerate the development periods without sacrificing the quality dimension. According to the previous evolution period, planning activities are a fundamental part of these models.

gaps) and classifies the PM-SDLCs in terms of two industrial design philosophies: "specification rigor" and "agility."

In the evolution map (Figure 2), the arrows represent the time-line of models (Table 5) and it considers the era perspective for classifying them (Avison & Fitzgerald, 2004). Similar analysis to the evolution map in Figure 2 could be elaborated by using the findings and analysis reported in the previous section of this chapter. According to Rodriguez et Al. (2007): *...under this methodology chronological era perspective, it is interesting to note that every new era starts with a model with high or medium specification rigor level, except in the pre-methodology era. This evidence suggests that software engineering principles and foundations toward the specification rigor are pursued before agility attributes. However, the industrial pressure to reduce manufacturing or realizing time cycles keeping high-quality products, also suggest that a trade-off between specification rigor and agility must be balanced.*

Recent reports suggest that the next steps will lead to the development of service-oriented systems (SOS), and thus a service-oriented software engineering (SOSE) discipline is emerging (Di Nitto et al, 2006; Arsanjani et al, 2006). PM-SDLCs for developing service-oriented systems are only now being formulated, with initial proposals considering the importance of systems requirements; business process models and their way for system specification and construction; tools for fast building and the new languages for specification and executing of business process based commonly in Web services. However, the initial analysis reveals that the evolution is being driven by technological factors (e.g., by the apparition of new development language and tool), and consequently, the risk to return at the initial characteristics of lack of rigor. This risk of return to past problems is caused by rapid changes in technology that have exceeded the methodologies' capabilities for engineering such systems efficiently and effectively. Thus, the new research question emerging from this continuous evolution is: "What elements of the existing PM-SDLCs can be used to define a new service-oriented PM-SDLC?" We claim this research is relevant because the benefits by the using of PM-SDLC to develop software-intensive

Figure 2. Map of PM-SDLC's evolution



systems were evidenced through the self evolution of PM-SDLCs. The evolution map (Figure 2) shows a tentative placement for service-oriented PM-SDLC. Given an initial analysis, this new PM-SDLCs still cannot be classified in a single octant.

CONCLUSION

We can conclude, under the consideration of a well-defined SDLC process (Oktaba & Ibargüengoitia, 1998), that no model includes all the required components defined as part of a process model. Each process model has common and unique elements, and each PM-SDLC has its own benefits and limitations. The formulation of newer PM-SDLCs are all influenced by previous models. Main drivers to create new process models are "technological" (to cope with the challenges of the new technologies) and "knowledge gaps" (to cope with the user's demands and organizational settings and environments for a better project management control). We can also identify that there is not a single unique PM-SDLC that could be applied in all cases. This implies that the application of a specific PM-SDLC will rely on the particular characteristics of the application under development, the organization's size, the technology used, and the developer's experience. The benefit of using a process model for the development of software-intensive systems has been proven and its rejection could lead to non successful products (Fuggetta, 2000).

This conceptual and comparative study of such models is useful to reveal some future trends

and recommendations for the theory and praxis: (a) PM-SLDC formulations have improved from early periods, (b) all of them are still based in the first waterfall model and their iterative and feedback recommendations, (c) the comparative framework and findings suggest a comprehensive way to understand and learn PM-SDLCs reported in the literature, and (d) also, the new PM-SLDC cycles—such as the ones based on agile systems development approach—still requires and uses the phases of analysis and design to avoiding the risk of returning to the pre-early methodology practice (Avison & Fitzgerald, 2003).

In turn, we can suggest the following research recommendations: (a) to study the selection features of PM-SDLCs, (b) to study the combination of well-documented models with the agile-based approaches, (c) to study the customization or adaptation of generic PM-SDLC models, and (d) under the consideration of evolution map knowledge derived in this study, and the emergent technology for service-oriented systems, we can conclude the need and relevance to study or develop service-oriented PM-SDLCs (Di Nitto et al., 2006).

REFERENCES

Aoyama, M. (1998, April 25-26). New age of software development: How component-based software engineering changes the way of foftware development? In *Proceedings of the First International Workshop on Component-based Software Engineering* (pp. 1-5). Kyoto, Japan.

Andoh-Baidoo, F., White, E., & Kasper, G. (2004). Information systems' cumulative research tradition: A review of research activities and outputs using pro forma abstracts. In *Proceedings of the Tenth Americas Conference on Information Systems* (pp. 4195-4202) New York.

Arsanjani, J., Hailpern, B., Martin, J., & Tarr, P. (2006). *Web services: Promises and compromises*. IBM Research Division, Thomas J. Watson Research Center.

Avison, D., & Fitzgerald, G. (2003). Where now for development methodologies? *Communications of the ACM, 46*(1), 78-82.

Beck, K. (1999). Embracing change with extreme programming. *IEEE Computer, October*, 77-70.

Boehm, B. (1988). A spiral model of software development and enhancement. *IEEE Computer, May*, 61-72.

Boehm, B., & Bose, P. (1994). *A collaborative spiral software process model based on theory W* (Technical Report USC-CSE-94-501). University of Southern California.

Brown, A., & Wallnau, K. (1996). Engineering of component-based systems. In *Proceedings of the Second IEEE International Conference on Engineering of Complex Computer Systems* (pp. 414-422).

Center for Software Engineering (1997). *Guidelines for Model-Based (System) Architecting and Software Engineering (MBASE)*. University of Southern California.

Counelis, J. (2000). Generic research design in the study of education: a systemic typology. *Systems Research, 17*, 51-63.

Cross, S. (2006). Toward disciplined rapid application development. *Software Tech News 2*, 1.

Dickover, M., McGowan, C., & Ross, D. (1977, October 16-19). Software design using SADT. In *Proceedings of the 1977 Annual Conference of the Association for Computing Machinery (ACM)* (pp. 125-133). Seattle, Washington.

Di Nitto, E., Hall, R., Han, J., Han, Y., Polini, A., Sandkuhl et al. (2006). Report on the international workshop on service oriented software engineering (IW-SOSE06). *ACM SIGSOFT Software Engineering Notes, 31*(5), 36-38.

Egyed, A., &Boehm, B. (1998, September). Telecooperation experience with the winwin system. In *Proceedings of the 15th IFIP World Computer Congress*. Vienna, Austria.

Fuggetta, A. (2000, June 4-11). Software process: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 25-34). Limerick (Ireland). New York: ACM Press.

Glass, R., Vessey, I., & Ramesh, V. (2002). Research in software engineering: an analysis of the literature. *Information and Software Technology, 44*, 491-506.

Humphrey, W. (2000). *The personal software process (PSP)* (Technical Report CMU/SEI-2000-TR-022). Software Engineering Institute, Carnegie Mellon University.

INCOSE. (2004). *Systems engineering handbook*. CA: INCOSE.

Lee A. (2004). Inaugural editor's comments. *MIS Quarterly, 23*(1), 5-11.

McAndrews, D. (2000). *The team software process (TSP): An overview and preliminary results of using disciplined practices* (Technical Report CMU/SEI-2000-TR-015). Software Engineering Institute, Carnegie Mellon University.

Mora, M. (2004). *The conceptual research approach* (Technical Internal Report). Autonomous University of Aguascalientes, Aguascalientes, Mexico (in Spanish language).

Naumann, J., & Jenkins, A (1982). Prototyping: The new paradigm for systems development. *MIS Quarterly, 6*(3), 29-44.

Oktaba, H., & Ibargüengoitia, G. (1998). Software process modeled with objects: Static view. *Computación y Sistemas CIC-IPN ISSN 1405-5546, 1*(4), 228-238.

Rational Software Corporation. (1998). *Rational unified process: Best practices for software development teams* (White Paper from Rational Software Corporation).

Rodríguez L., Mora, M., & Alvarez F. (2007, September 24-25). A descriptive/comparative Study of the evolution of process models of software development life cycles (PM-SDLCs). *SIS 07: Simposio de Ingeniería de Software, in ENC 2007: Encuentro Internacional de Computación*. Morelia, México.

Royce, W. (1970, August). Managing the development of large software systems. In *Proceedings of the IEEE WESCON* (pp.1-9).

Sage, A., & Armstrong, J. (2000). *Introduction to systems engineering*. New York: Wiley.

Scacchi, W. (2001). Process models in software engineering. In J. J. Marciniak (Ed.), *Encyclopedia of software engineering* (2nd ed.) (pp.) New York: John Wiley and Sons.

Sorensen, R. (1995). A comparison of software development methodologies. *Crosstalk Journal, January*, 1-15.

SWEBOK. (2001). *Guide to the software engineering body of knowledge*. Los Alamos, CA: IEEE Society.

Tyrrel, S. (2000). The many dimensions of the software pProcess. *ACM Crossroads Student Magazine, 6*(4)*,* 1-7.

Yang, H. (2002). *Successful evolution of software systems*. Norwod, MA: Artech House, Incorporated.

Yourdon, E. (1993). *Modern structured analysis* (Spanish version). New York: Prentice-Hall.

KEY TERMS

CASE—Computer-Assisted Software (or systems) Engineering: Software tool for assisting the software development.

Classic PM-SDLCs: (70-90's): Waterfall, SADT—s*tructured analysis and design technique,* prototyping, modern structured cycle, spiral.

CMMI—Capability Maturity Model Integration: A quality model for software process assessment and improvement.

Current PM-SDLCs: (90's-2000) RAD—*rapid application development,* win-win spiral, unified process, PSP—p*ersonal software process,* TSP—t*eam software process.*

Emergent PM-SDLC: (1997-present) MBASE—m*odel-based (system) architecting and software engineering,* XP—e*xtreme programming* (ASP) [6], CBSE—component-based software engineering (ASP).

Process Model of Software Development Life-Cycle = PM-SDLC: A chain of activities, transformations, events, and artifacts to guide the full process of software creation. Such models can be used to develop more precise and formal descriptions of software life-cycle activities (based in Scacchi, 2001).

Software Life-Cycle Model: *The term software life-cycle model is equivalent to term life-cycle framework model. These "frameworks models" are the definition of phases to software development, at high level. These models do not provide detail definitions but show the high-level activities and its interrelationships. The most common models are waterfall model, prototyping model, spiral model"* (based in the SWEBOK, 2000).

ENDNOTES

1 Author considers prototyping a technique that can be used in the first phases of any SDLCs. Other authors could report prototyping as a SDLC.

2 In MBASE, a *postmortem phase* is not explicitly reported but it is implied.

3 In XP-based process, the concept of postmortem analysis is performed as a *retrospective analysis*.

4 Royce (1979) reported that the classic waterfall cycle could be modified to integrate some iterations.

5 **Waterfall** model -according to Royce's ideas (1970)—considers a preliminary design phase that could produce a pilot system or prototype. Often literature confuses **waterfall** with the **stage-wised** model.

6 ASP—agile software process. Also know as *light methodologies*