CHAPTER 2

SOFTWARE PROCESS DYNAMICS: MODELING, SIMULATION AND IMPROVEMENT

Mercedes Ruiz[†], Isabel Ramos[‡], Miguel Toro[‡]

Department of Computer Languages and Systems
[†]Escuela Superior de Ingeniería
C/ Chile, 1. 11003 – Cádiz (Spain)
[‡]Escuela Técnica Superior de Ingeniería Informática.
Avda. Reina Mercedes, s/n. 41013 – Seville (Spain)
E-mail: mercedes.ruiz@uca.es
{isabel.ramos, miguel.toro}@lsi.us.es

The aim of this chapter is to introduce the reader to the dynamics of the software process, the ways to represent and formalize it, and how it can be integrated with other techniques to facilitate, among other things, process improvement. In order to achieve this goal, different approaches of software process modeling and simulation will be introduced, analyzing their pros and cons. Then, continuous modeling will be used as the modeling approach to build software process models that work in the qualitative and quantitative fields, assessing the decision-making process and the software process improvement arena. The integration of this approach with current process assessment models (such as CMM), static and algorithmic models (such as traditional models used in the estimation process) and the design of a metrics collection program which is triggered by the actual process of model building will also be described in the chapter.

1. Introduction

Worldwide, the demand for highly complex software has significantly increased in such a way that software has replaced hardware as having the principal responsibility for much of the functionality provided by

current systems. The rapid pace at which this software is required, the problems related to cost and schedule overruns and customer perception of low product quality have changed the focus of attention towards the maturity of software development practices. Over the last few decades, the software industry has received significant help from CASE tools, new programming languages and approaches, and more advanced and complex machines.

However, it is widely accepted that the potential benefits of better technology cannot be translated into more successful projects if the processes are not well defined, established, and executed. Proper processes are essential for an organization to consistently deliver high quality products with high productivity.

Dynamic modeling and simulation have been intensively used as process improvement tools in the manufacturing area. Currently, interest in software process modeling and simulation as an approach for analyzing complex businesses and solving policy questions is increasing among researchers and practitioners. However, simulation is only effective if both the model and the data used to drive it accurately reflect the real world. As a consequence, it can be said that the construction of a dynamic model for the actual software process provides clear guidelines on what to collect.

Many frameworks are now available for software processes, the Capability Maturity Model (CMM)[1] and ISO 9001[2] being among the most influential and widely used. Although ISO 9001 is a standard, and has been interpreted for a software organization in ISO 9000-3[3], it has been written from the customer and external auditor's perspective. On the other hand, CMM is not a binary certification process, but a framework that categorizes the software process at five levels of maturity and provides roadmaps to evaluate the software process of an organization, as well as planning software process improvements. One of the common features that all these frameworks possess is that they strongly recommend the application of statistical control and measure guides to define, implement and evaluate the effects of different process improvements. Within these frameworks, the availability of data is considered of special importance for building the knowledge required to define and improve the software process.

The aim of this paper is to present a combination of traditional techniques with software process modeling and simulation to build a framework for supporting a qualitative and quantitative assessment for software process improvement and decision making. The purpose of this dynamic framework is to help organizations to achieve a higher software development process capability according to CMM. The dynamic models built within this framework provide the capability of gaining insight over the whole life cycle at different levels of abstraction.

The level of abstraction used in a particular organization will depend on its maturity level. For instance, in a level 1 organization the simulator can establish a baseline according to traditional estimation models from an initial estimate of the size of the project. With this baseline, the software manager can analyze the results obtained by simulating different process improvements and study the outcomes of an over- or underestimate of cost or schedule. During the simulation metric data is saved. This data conforms to the SEI core measures[4] recommendation and is mainly related to cost, schedule and quality.

The structure of the chapter is as follows. Section 2 describes in detail the software process modeling and simulation approach. It includes the benefits derived from this application, the formalisms used to build software process models and a process model building methodology. In section 3, a combination of hierarchical dynamic modeling and some traditional techniques of the software engineering is proposed. The conceptual ideas underlying this combination with the aim of building an integrated dynamic framework for software process improvement are presented. Sections 4, 5 and 6 describe the details concerning the structure of the framework, the modular architecture and some aspects of the implementation. An example of usage is presented in section 7. Finally, section 8 summarizes the chapter and describes the most recent applications of the software process dynamic modeling and simulation approach.

2. Software process simulation

Simulation can be applied in many critical areas in support of software engineering. It enables one to address issues before these issues become

problems. Simulation is more than just a technique, as it forces one to think in global terms about system behavior and about the fact that systems are more than the sum of their components[5]. A simulation model is a computational model that represents an abstraction or a simplified representation of a complex dynamic system. The main benefit of simulation models is the possibility of experimenting with different management decisions. Thus, it becomes possible to analyze the effect of those decisions on systems where the cost or risks of experimentation make it unfeasible.

Another important factor is that simulation provides insights into complex process behavior that cannot be analyzed by means of stochastic models. Like many processes, software processes can contain multiple feedback loops, such as those associated with the correction of defects. Delays resulting from these defects may range from minutes to years. The resulting complexity makes it almost impossible for mental analysis to predict the consequences. According to Kellner, Madachy and Raffo[6], the most frequent sources of complexity in real software processes are:

- Uncertainty. Some real processes are characterized by a high degree of uncertainty. Simulation models make it possible to deal with this uncertainty as they can represent it flexibly by means of parameters and functions.
- Dynamic behavior. Some processes may have a time-dependent behavior. There is no doubt that the behavior of some software process variables varies as the time cycle progresses. With a simulation model it is possible to represent and formalize the structures and causal relationships that dictate the dynamic behavior of the system.
- Feedback. In some systems, the result of a decision made at a given time can affect their behavior. In software projects, for example, the decision to reduce the effort assigned to quality assurance activities has different effects on the progress of these projects.

Thus, the common objectives of simulation models are to supply mechanisms to experiment, predict, learn and answer questions, such as, “What if …?”

A software process simulation model can be focused on certain aspects of the software process or the organization. It is important to bear in mind that a simulation model constitutes an abstraction of the real system, and so it represents only the parts of the system that were intended to be modeled. Furthermore, currently available modeling tools, such as ithink[7], POWER-SIM[8], and Vensim[9], help to represent the software development process as a system of differential equations. This is a remarkable characteristic as it makes it possible to formalize and develop a scientific basis for software process modeling and improvement.

During the last decade, software process simulation has been used to address a wide variety of management problems. Some of these problems are related to strategic management, technology adoption, understanding, training and learning, and risk management, among others. Noticeable applications of this approach to software process modeling can be found in Kellner, Madachy and Raffo[6], Prosim 2004[10] and Prosim 2005[11].

2.1. Software process modeling for simulation

There are different approaches for building simulation models of the software process. In practice, the modeling approach inevitably has some influence on what it should be modeling. Hence, there is no preferred approach for modeling the software process in every situation, but the best approach is always the one that is considered to be the most suitable for a particular case.

There are two broad types of simulation modeling: continuous simulation and discrete-event simulation. The distinction is based on whether the state can change continuously or at discrete points in time. However, even though events are discrete, time and state domains may be continuous. There are three main paradigms that can be used for discrete-event simulation modeling: event-scheduling, activity-scanning and process-interaction. Although state-transition diagrams (e.g., finite-state automata or Markov chains) can be used for software process simulation modeling, they are less common because the state spaces involved are typically very large. Examples of discrete-event simulation

applied to model and simulate the software process can be found in Raffo[12], Kellner[13] and Hansen[14].

A continuous simulation model represents the interactions between key process factors, as a set of differential equations, where time is increased step by step. Frequently, the metaphor of a system of interconnected tanks filled with fluid is used to exemplify the ideas underlying this kind of modeling approach.

On the other hand, discrete modeling is based on the metaphor of a queuing network where time advances when a discrete event occurs. When this happens, an associated action takes place, which, mostly, implies placing a new event in the queue. Time is always advanced to the next event, so it can be difficult to integrate continually changing variables.

Since the purpose of this study is to model and visualize process mechanisms, continuous modeling has been used. This technique also allows systems thinking and it is considered to be better than the discrete-event model at showing qualitative relationships[15]. Examples of continuous simulation applied to model and simulate the software process can be found in Abdel-Hamid[16], Pfhal and Lebsant[17], Burke[18], and Wernick and Hall[19].

2.2. *Continuous modeling and simulation of the software process*

System dynamics is a methodology for studying and analyzing complex feedback systems such as software organizations. Feedback is the key differentiating factor of dynamic systems. It refers to the situation in which A affects B and B affects A, through a chain of causes and effects. It is not possible to study the link between A and B and, independently, the link between B and A to predict the behavior of the system. There are a significant number of software process features that follow this feedback pattern. For instance, known patterns, such as Brook's Law[20] ("Adding manpower to a late project makes it later") or Parkinson's Law[21] ("Work expands to fit the time available"), can be described by continuous modeling.

System dynamics links structure (feedback loops) to behavior over time and helps to explain *why what* is happening *is* happening. The field was initially developed from the work of Jay W. Forrester[22].

To better understand and represent the system structures that cause the patterns of behavior observed in the software process, two kinds of diagrams are used: causal-loop diagrams and stock-and-flow diagrams.

2.2.1. *Causal-Loop Diagrams*

Causal-loop diagrams present relationships that are difficult to describe verbally because natural language presents interrelations in linear cause-and-effect chains, whereas a diagram shows that there are *circular* chains of cause-and-effect in the actual system[23]. Figure 1 shows an example of a causal-loop diagram for a very simplified model of software process dynamics. In this diagram, the short descriptive phrases represent the elements that make up the system described, and the arrows represent the causal influences between these elements. This diagram includes elements and arrows or links that help to connect these elements, but also includes a sign (either + or -) on each link. These signs have the following meaning[23]:

- A causal link from one element A to another element B is *positive* if either (a) A adds to B or (b) a change in A produces a change in B in the *same* direction.
- A causal link from one element A to another element B is *negative* if either (a) A subtracts from B or (b) a change in A produces a change in B in the *opposite* direction.

In addition to the signs of each link, a complete loop is also given a sign. The sign of a particular loop is determined by counting the number of minus signs on all the links that make up the loop. Specifically,

- A feedback loop is called *positive*, indicated by (+), if it contains an even number of negative causal links.
- A feedback is called *negative*, indicated by (-), if it contains an odd number of negative causal links.

Thus, the sign of a loop is the algebraic product of the signs of its links. The diagram shown in Figure 1 is composed of four feedback loops: two positive and two negative. A brief description of the pattern modeled follows.

First feedback loop. Estimations of cost and time for the project can be derived from the initial estimations. With these estimations the required manpower is acquired by performing hiring activities. As the project runs, information about the real progress is obtained. Comparisons of the values obtained with those originally estimated may lead to a change in some of the estimations and, possibly, a modification of the hiring policy.

Second feedback loop. This loop illustrates the effects caused by the schedule pressure on the quality of the software product. If the perceived completion time is greater than the planned time to complete, the project has schedule pressure. To combat this, the project manager may decide either to hire more personnel or have overtime worked. However, permanent overtime may further exhaust personnel, contributing to an increase in the number of errors in the project. This rise in the number of committed errors requires a bigger effort in terms of error detection and rework activities, which holds back progress.

Third feedback loop. The growth in the level of human resources appears to contribute to a growth of productivity. However, it is also important to note that the productivity of the new personnel is significantly less than that of the expert personnel. Hence, some effort of the expert personnel is commonly invested in the training of the newly-hired personnel. These training activities, together with the communication overheads derived from the Book's Law, contribute to a decrease in the net productivity of the working team.

Fourth feedback loop. This loop illustrates the effect of creative pressure. When the personnel know that the project is behind schedule, they tend to be more efficient. This is normally reflected in a reduction of idle time.

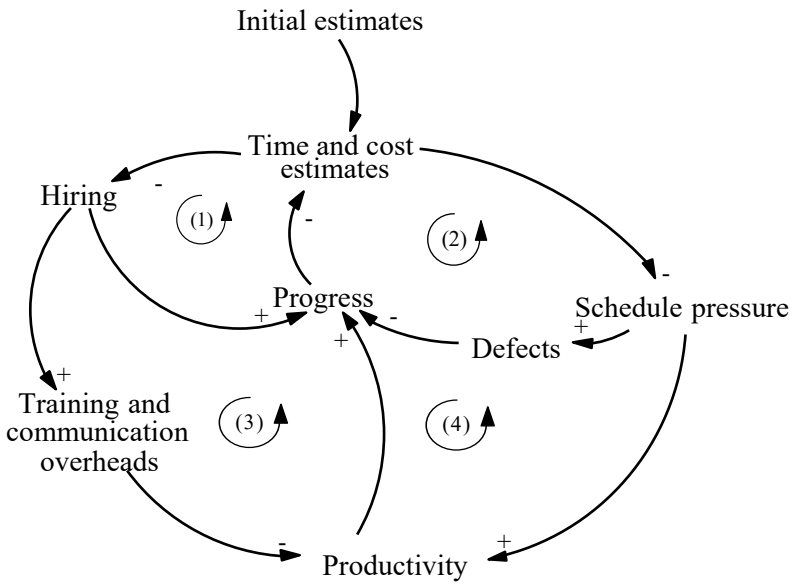


Fig. 1. Simple causal-loop diagram of the software process dynamics.

2.2.2. *Stock-and-Flow Diagrams*

Figure 2 illustrates the main components of stock-and-flow diagrams. This notation consists of three different types of elements: stock, flows and information. These three elements provide a general way of graphically representing *any* process. Furthermore, this graphical notation can be used as a basis for developing a quantitative model that can be used to study the characteristics of the process. As with a causal-loop diagram, the stock-and-flow diagram shows relationships among *variables* that have the potential to change over time. To understand and build stock-and-flow diagrams, it is necessary to understand the difference between stocks and flows. Distinguishing between stocks and flows is sometimes difficult. As a starting point, stocks can be thought of as physical entities that can accumulate and move around. The term *stock* also has an identical meaning to the term *state variable* from the systems engineering analysis. The term *flow* refers to the movement of something from one stock to another.

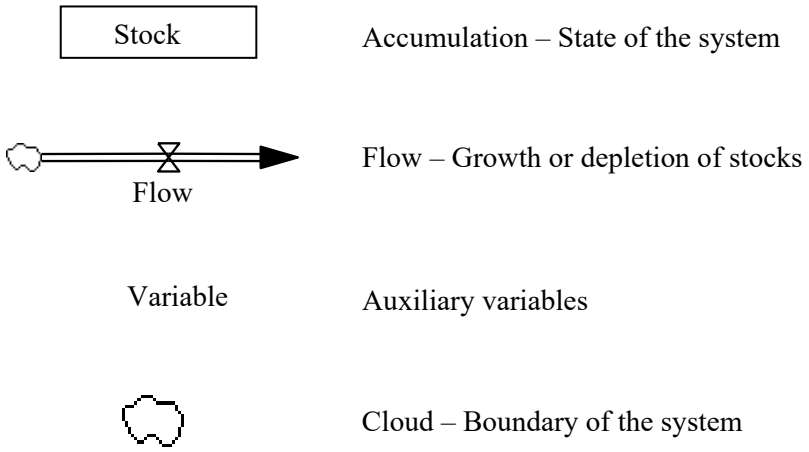


Fig. 2. Main elements of stock-and-flow diagrams.

Figure 3 shows a stock-and-flow diagram for the first feedback loop of the causal diagram shown in Figure 1. The variables are *Pending tasks*, *Accomplished tasks*, *Personnel*, *hiring rate* and *development rate*. The first three are stock or level variables, whereas the last two are flow variables. The number of tasks to be developed is determined from an initial estimate of the size of the project. These pending tasks become accomplished tasks depending on the development rate that is determined by the productivity of the personnel allocated to the development of the tasks under simulation.

The stock-and-flow diagram has a precise mathematical meaning. Stocks accumulate (integrate) their inflows less their outflows. The rate of change of a stock is the total inflow minus the total outflow. Thus a stock and flow map corresponds to a system of integral or differential equations that formalize the model. Mathematical description of a system requires only the stocks and their rates of change. However, it is often helpful to define intermediate or auxiliary variables. Auxiliaries consist of functions of stocks and constants. The set of equations must then be solved applying mechanisms for solving differential equations or by simulation. Simulation packages are often used to solve these sets of equations, since it soon becomes unfeasible to solve such equations by hand as the number of stocks and flows or the complexity of the equations increases.

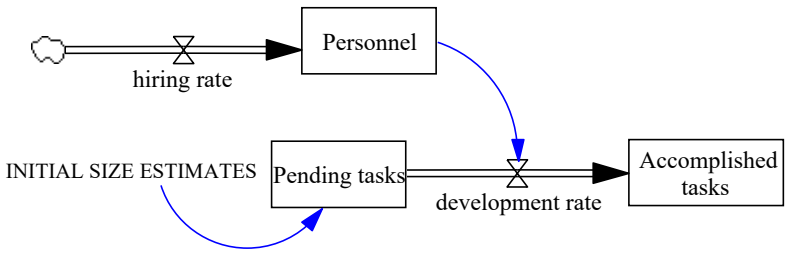


Fig. 3. Simple stock-and-flow diagram.

The equations derived from the stock-and-flow diagram follow:

$$\text{Pending tasks}(t) = \text{INITIAL SIZE ESTIMATES} - \int_0^t \text{development rate}(t)\, dt \quad (1)$$

$$\text{Accomplished tasks}(t) = \int_o^t \text{development rate}(t)\, dt \quad (2)$$

$$\text{Personnel}(t) = \int_0^t \text{hiring rate}(t)\, dt \quad (3)$$

$$\text{development rate}(t) = \begin{cases} \text{Personnel}(t) * \text{Productivity}(t), & \text{if Accomplished tasks}(t) < \text{INITIAL SIZE ESTIMATES} \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

$$\text{hiring rate}(t) = \big(\text{required personnel}(t) - \text{Personnel}(t)\big) / \text{HIRING DELAY} \quad (5)$$

Figure 4 shows the time evolution of the main variables of this illustrative model after solving the equations by simulation.

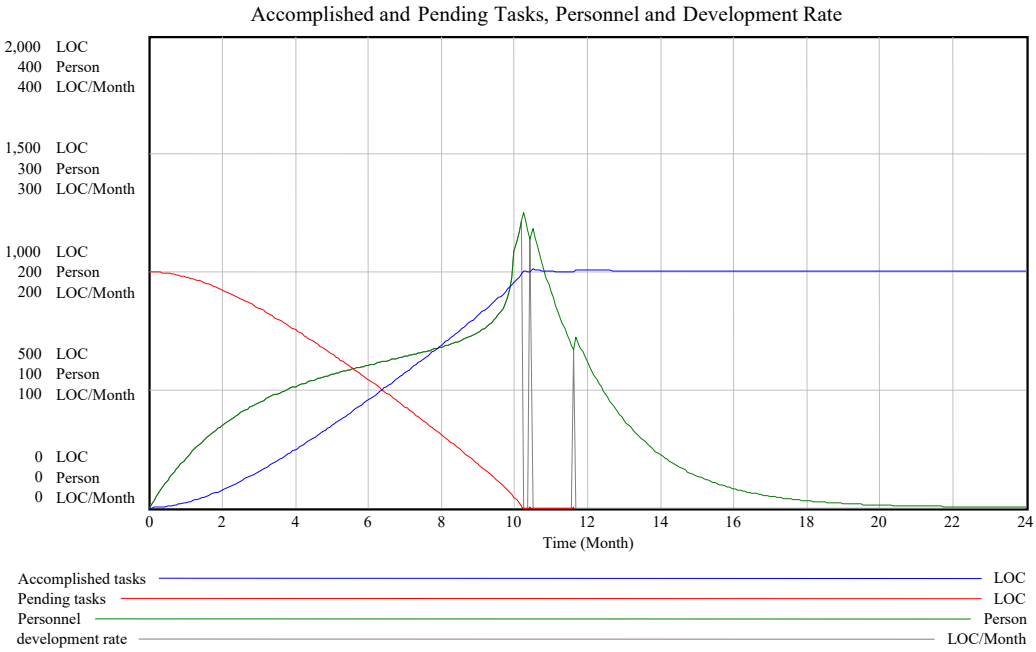


Fig. 4. Time evolution of the main variables of the stock-and-flow diagram.

Nevertheless, as Sweeny and Sterman[24] stated, building a model is not about spending considerable time on the basics of stocks and flows, time delays, and feedback, but developing intuition rather than mathematics.

2.3. *Process model building methodology*

According to Martinez and Richardson[25], the system dynamics model building process involves seven key activities, as shown in Figure 5. The most important ones are: (1) problem identification and definition, (2) system conceptualization, (3) model formulation, (4) model testing and evaluation, and (5) understandings of the model.

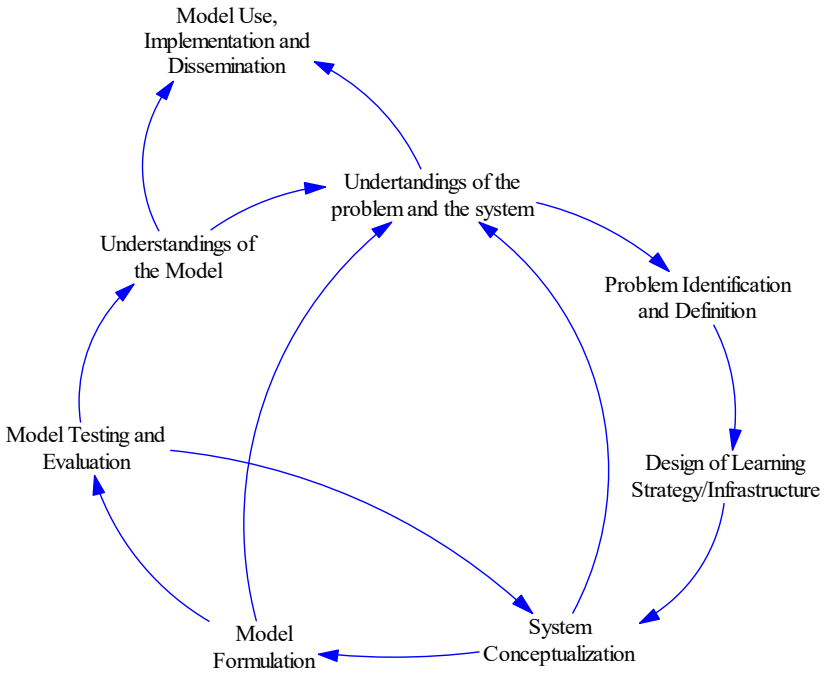


Fig. 5. Steps of process model building methodology.

In *problem identification and definition*, there is a group of practitioners who consistently prefer to model the case at hand, as opposed to another group who thinks that the best way is to model the class to which the system belongs.

In *system conceptualization*, the best practice is considered to be to start with major stock variables. Practitioners can choose to iterate using a causal-loop diagram approach or a stock-and-flow approach to conceptualization.

In *model formulation*, there are two major approaches. The first relates to the issue of starting small and continuously simulating, preferably always having a running model. The second refers to

formulating in big chunks and is not concerned about continuously having running prototypes.

Model testing and evaluation consists of three main activities that determine the correctness of the model[26]. These activities are divided into two categories: activities focused on verifying the model structure and activities that verify the model behavior. Table 1[27] summarizes these activities.

Finally, *understandings of the model* is centered on the knowledge that can be gained from use of the model.

3. Dynamic Integrated Framework for Software Process Improvement: Conceptual approach

Using simulation for process improvement in conjunction with CMM is not a new idea. As a matter of fact, Christie[5] suggests that CMM is an excellent incremental framework to gain experience through process simulation. Nevertheless, there are no dynamic frameworks capable of helping to achieve higher process maturity. One of the main features of the Dynamic Integrated Framework for Software Process Improvement (DIFSPI) is that this help is provided throughout the development of the whole dynamic framework and not only by using the associated final tool. The reason for this is that the benefits that can be gained from the utilization of dynamic models within an organization are directly related to the knowledge and the empirical information that the organization has about its processes. Figure 6 illustrates this idea. It shows the existing causal relationships between the maturity level of the organization, the utilization of dynamic models and the benefits obtained.

The positive feedback loop comes to illustrate the causal relationship that reinforces the collection of metrics within the organization. The metrics collected will be used to calibrate and initialize the dynamic models.

Lower maturity organizations are characterized by the absence of metrics programs and historical databases. In this case, it is necessary to begin by identifying the general processes and what information has to be collected about them. The questions of what to collect, how often and

how accurately have to be answered at this time. The design process of dynamic models helps to find an answer to these questions.

Table 1. Main model testing and evaluation activities[27].

Verification	Structure	Dimensional consistency
		Behavior with extreme values
		Problem adequacy
	Behavior	Parameter sensitivity
		Structure sensitivity
Validation	Structure	Reality check
		Parameter correctness
	Behavior	Scenario replication
		Extreme condition simulations
		Non-conventional input simulations
		Statistical tests
Evaluation	Structure	Size
		Complexity
		Granularity
	Behavior	Intuitive behavior generation
		Knowledge generation

When developing a dynamic model, one needs to know: a) what it is intended to model, b) the scope of the model, and c) what behaviors need to be analyzed.

Once the model has been developed, it needs to be initialized with a set of initial conditions in order to execute the runs and obtain the

simulated behaviors. These initial conditions customize the model to the project and to the organization to be simulated and they are effectively implemented by a set of initial parameters.

It is precisely these parameters that govern the evolution of the model runs that answer the above question of what data to collect: the data required to initialize and validate the model will be the main components of the metrics collection program. Once the components of the metrics collection program have been derived, it can be implemented within the organization. This process will lead to the formation of a historical database. The data gathered can then be used to simulate and empirically validate the dynamic model. When the dynamic model has been validated, the results of its runs can be used to generate a database. This database can be used to perform process improvement analyses. An increase in the complexity of the actions for analysis will lead directly to an increase in the complexity of the dynamic model required and, therefore, to a new metrics collection program for the new simulation modules.

The bottom half of Figure 6 illustrates the effects derived from the utilization of dynamic models in the context of process improvement. Using dynamic models that have been designed and calibrated according to an organization's data has three important benefits. Firstly, the data from the simulation runs can be used to predict the evolution of the project. The graphical representations of these data show the evolution of the project from a set of initial conditions that have been established by the initialization parameters. By analyzing these graphs, organizations with a low level of maturity can obtain useful qualitative knowledge about the evolution of the project. As the maturity level of the organization increases, the knowledge about its processes is also higher and the simulation runs can be used as real quantitative estimates. These estimates help to predict the future evolution of the project with an accuracy that is closely related to the uncertainty of the initial parameters. Secondly, it becomes possible to define and experiment with different process improvements by analyzing the different simulation runs. This capability helps in the decision-making process, as only the improvements that yielded the best results will be implemented. Moreover, it is noteworthy that these experiments are performed at no

cost or risk to the organization, as they use the simulation of scenarios. Thirdly, the simulation model can also be used to predict the cost of the project; this cost can be referred to the overall cost, or to a hierarchical decomposition of the total cost, like, for instance, the cost of quality or rework activities. These three benefits are the main factors that lead to the achievement of a higher maturity level within an organization according to CMM.

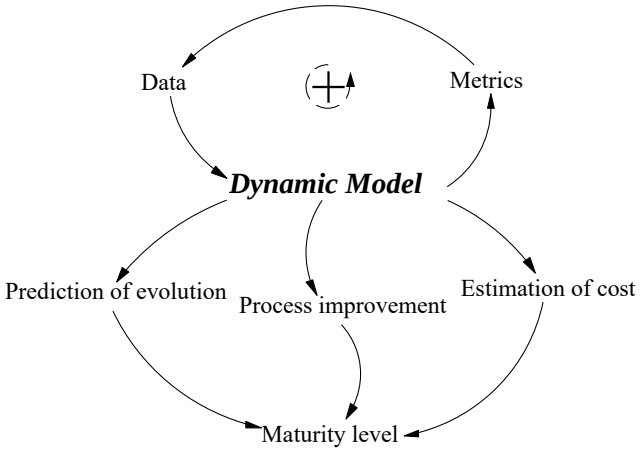


Fig. 6. Causal relationships concerning the utilization of dynamic models.

4. Framework Structure and Module Architecture

Project management is composed of activities that are closely interrelated in the sense that any action taken in one particular area will possibly affect other areas. For instance, a time delay will always affect the cost of the project, but it may or may not affect the morale of the development team or the quality of the product. The interactions among the different areas of project management are so strong that sometimes the throughput of one of them can only be achieved by reducing the throughput of another. A clear example of this behavior can be found in the common practice of reducing the quality, or the number of

requirements to be implemented in a version of the product with the aim of meeting the cost estimates or time deadlines.

Dynamic models are an aid for understanding the integrated nature of project management, as they describe it by means of different processes, structures, and key interrelationships.

In the framework proposed here, project management is considered as a set of dynamic interrelated processes. Projects are composed of processes. Each process is composed of a series of activities designed to achieve an objective[1]. From a general point of view, it could be said that projects are composed of processes that fall into one of the following categories:

- Management process. This category includes all processes related to the description, organization, and control of the project.
- Engineering process. All processes related to the software product specification and development activities fall into this category.

Engineering processes begin to be executed from an initial plan performed by the project management processes. Using the information gathered about the progress of this second group of processes, project management processes determine the modifications that need to be made to the plan in order to achieve the project objectives. The proposed DIFSPI follows this same classification and is structured to account for project management and engineering processes. At both levels, the utilization of dynamic models to simulate real processes and to define and develop a historical database will be the main feature.

4.1. *Engineering processes in the DIFSPI*

At this level the dynamic models simulate the life cycle of the software product. In low maturity organizations, the amount of information required to begin running simulations is relatively small and mainly focused on the initial estimations, that is, the estimated size of the project and the initial size of the working team. The best dynamic model is simulated depending on the paradigm followed to develop the software product and the maturity level of the organization. The main paradigms

that can be currently simulated within the framework are the traditional waterfall and COTS paradigms. Depending on the chosen paradigm, different dynamic modules will be joined in order to create a final and fully operational dynamic model. Once the simulation has been run, it provides data that are saved in a database. This initial data contains the results of the simulation together with a set of initial estimations resulting from the computation of the static models. These initial estimations establish the baseline for the project, and the simulated data obtained represent the dynamic evolution of the project variables throughout the whole life cycle. Apart from storing the initial baseline and the simulated data, the database contains a third component. This third component contains the results of applying some other techniques during the simulation of the project, which are oriented towards gaining insight into the process under simulation. These techniques, which have been integrated with the dynamic modules, are described in section 5.

As mentioned before, the process of modeling the software process requires a good knowledge of the software process itself, and triggers a metrics collection program that can then be used to initialize the parameters of the model and increment the level of visibility the model has of the process. All that has been simulated so far must be put into practice.

After determining the initial estimates and running the simulations to establish the initial baseline, it is possible to run different scenarios in order to find out what effects different initial values have on the project estimates. This reflects, of course, the level of uncertainty that low maturity organizations have at the initial stages of a project. When the real project begins, the metrics collection program may be applied to gather real information about the process. This real data is also saved in the database, enabling the development of a historical database. As this data becomes available, it is possible to perform analysis and calibrate the functions and parameters of the dynamic modules so that their accuracy can be improved. Improving the accuracy of the dynamic modules may require an improvement in the knowledge we have of the software process and, this way, the loop is closed.

The dynamic models of this level of DIFSPI should follow the levels of visibility and knowledge of the engineering processes that

organizations have at each maturity level. Obviously the dynamic model used in level 1 organizations will not be as complex as the models capable of simulating the engineering processes of, for instance, level 4 organizations.

4.2. *Management processes in the DIFSPI*

The control modules model and simulate all the activities that determine the progress of the project, and make the corrective decisions that are required to meet the project objectives. These modules are highly important in the design of the process improvements.

Within the framework, management processes are divided into two main categories:

- Planning. It groups the processes devoted to the design of the initial plan and the required modifications when the progress reports indicate the appearance of problems. The models of this group integrate traditional together with dynamic estimation and planning techniques.
- Control. This group includes all the models designed for monitoring and tracking activities. These models will also have the responsibility of determining the corrective actions to the project plan. Therefore, the simulation of process improvements will be of enormous importance.

Figure 7 shows the utilization of DIFSPI at this level. As mentioned earlier, the initial baseline for the project is established using the static models built within the framework. The dynamic modules that model the planning activities performed in the organization not only have differential equations to model these activities, but also the equations of the traditional static estimation models. To gain useful information from these static models, the very same knowledge about the software process is needed at this point as is required to use these models.

4.3. *Module Architecture*

The approach followed to construct the dynamic models is based on two fundamental principles:

The principle of *extensibility* of dynamic models. According to this principle, different dynamic modules are joined to an initial and basic dynamic model. This initial model models the fundamental behavior of a software project. Each one of the dynamic modules models each one of the key process areas that conforms the step to the next level of maturity. These modules can be either "enabled" or "disabled" according to the objectives of the project manager or the members of the Software Engineering Improvement Group (SEIG).

The principle of *aggregation/decomposition* of tasks according to the level of abstraction required for the model. Two levels of aggregation/decomposition are used:

- Horizontal aggregation/decomposition according to which different sequential tasks are aggregated into a unique task with a unique schedule.
- Vertical aggregation/decomposition according to which different and individual, but interrelated and parallel tasks are considered as a unique task with a unique schedule too.

The definition of the right level of aggregation and/or decomposition for the tasks mainly affects the modeling of the engineering activities and principally depends on the maturity level of the process to be simulated.

To define the initial dynamic model, the common feedback loops among the software projects must be taken into account. The objective of this approach is to achieve a generic model avoiding the modeling of specific behaviors of concrete organizations, which could limit the flexibility of DIFSPI. Data from historical databases described in the available literature can be used to initialize the functions and parameters of the initial model[28]. Figure 7 shows the main structure of the initial model. Four dynamic modules are joined together to develop an operational model that provides the set of final differential equations to generically simulate the software process in low maturity organizations.

By replicating some of the equations of the initial model it is possible to model the progress to higher maturity levels. The initial model can be used to simulate software projects developed in organizations progressing to level 2.

Generally speaking, the software product development process can be considered as follows. The number of tasks to be developed is determined from an initial estimate of the size of the project. These pending tasks become accomplished tasks according to the development rate. During this process, errors can be committed. Thus, in accordance with the desired quality objective for the project, the quality rate and the rework rate are determined. These two rates govern the number of tasks that are revised.

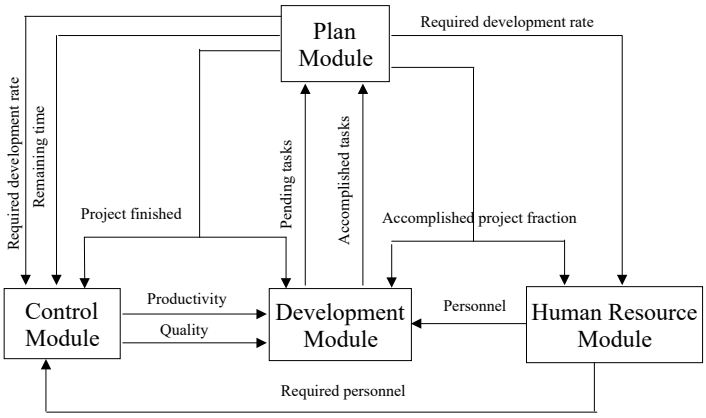


Fig. 7. Submodules architecture of the initial model.

To model the progress to level 3, the model will make use of a horizontal decomposition, creating as many substructures as phases or activities are present in the task breakdown structure of the project (analysis, design, code and test, in the waterfall paradigm). According to this approach, each time a complete model or some part of it is replicated, it will be necessary to define the new fixing mechanisms (dynamic modules) for the new structures. These mechanisms effectively implement the above-mentioned principle of aggregation/decomposition. The replication of

structures also provides the possibility of replicating the modules related to the project management processes. This replication is especially useful for high maturity level organizations, which will be able to establish process improvement practices for each particular activity of the life cycle.

Having described the approach to the elaboration of the dynamic models, this section gives a description of the hierarchical structure of the framework presented in this paper.

Figure 8 illustrates this hierarchy. The dynamic model for level 1 organizations progressing to level 2 is composed of four main dynamic modules, each of them devoted to modeling and simulating each of the four main subsystems of the software process: planning, human resource management, control, and development activities. These four subsystems form an initial dynamic model. This initial model is intended to be used in level 1 organizations progressing to level 2.

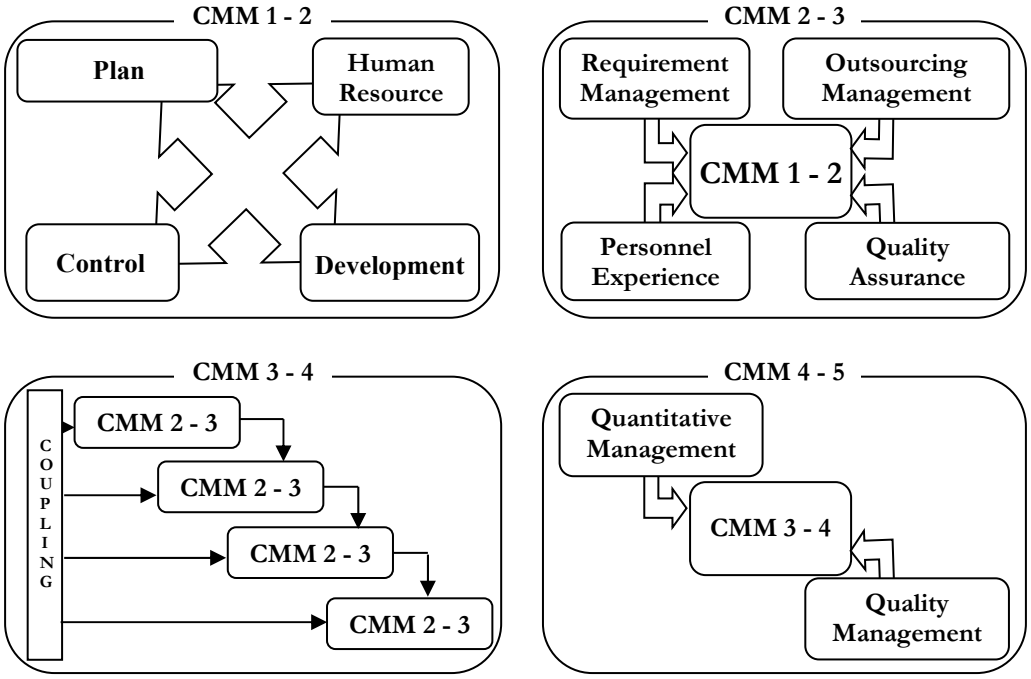


Fig. 8. Hierarchical structure of the dynamic integrated framework.

To get a dynamic model to model and simulate the software process of level 2 organizations, new dynamic modules are added to the initial model.

Outsourcing management. With this module, it is possible to analyze the influence of outsourcing over the life cycle of the project.

Personnel experience. Although this is not a key process area of CMM level 2, the human resource management module of the initial model has been enhanced so that it can reflect the influence of the experience factor on the progress and the cost of the project.

Quality assurance. The necessary structures to model and analyze the cost and state of the quality assurance activities are implemented in this module.

Requirement management. This module helps to determine the impact of requirements variability on software development projects.

The next step towards the following level of maturity does imply an important structural change. This change is determined by the special emphasis on the engineering activities that the CMM suggests as of level 3. While the CMM recommends the development of good planning and management practices in the initial levels of maturity, the engineering process acquires key importance at level 3. The principle of model replication is used to reflect this idea. Thus, to model level 3 organizations progressing to level 4, the model developed for the previous level is replicated as many times as the number of generic phases there are in the work breakdown structure of the project. For the purpose of this study, the four main characteristic phases of a traditional life cycle were considered (analysis, design, code and test). To simulate each phase, a complete dynamic model is used. Each of these dynamic models can be used, separately, to simulate the whole project in organizations with the previous level of maturity. To get all these models working together to simulate a higher maturity organization, coupling structures need to be defined. These coupling structures must allow inter-module communication as well as serving as support structures for the sharing of information.

The last model of the hierarchy is made from the model developed for the previous level, plus the modules required to model and simulate the new key process areas. In this case, the new modules are focused on the specific aspects of the key processes of quantitative management and software quality management.

5. Integrated Techniques

As mentioned before, our aim was to develop a working environment where the simulation of different scenarios can be used to generate the simulated database where managers can experiment with different process improvements and activities focused on the implementation of metrics programs and value analysis. The following techniques and methods are currently successfully implemented in DIFSPI:

Traditional estimation techniques. Traditional algorithmic estimation models have been implemented within this framework with the aim of providing an initial baseline for software projects carried out in low maturity level organizations [29, 30].

SEI Core Measures. Recent studies and experiences highlight the benefits of the application of these four core measures to the software life cycle. The main aspects of the product and process (quality, time, size and cost) are monitored and tracked to facilitate project success and higher maturity achievement. Within this framework these four measures constitute the basics for both the dynamic models and the graphical representation of process performance[4].

Metrel Rules. Given the dynamical nature of the proposed DIFSPI, we consider it could be useful to integrate a taxonomy of software metrics derived from the needs of users, developers, and management. Of all the potential advantages of using this system of metrics, we would like to point out the dynamic performance of these metrics, that is, how their accuracy, precision, and utility changes throughout a project, the life of a product or the strategic plan of an organization. In DIFSPI Metrel rules have been used as an efficient method for depicting on one graph the information needed for management, staff, and customers to view or predict process performance results. We consider that Metrel rules are particularly important in the field of software process modeling as their application provides a formal procedure for the expansion and transformation of models. By employing simple mechanisms like derivatives or integration (over time, phases or even projects), a mathematical model for one level can be transformed into another for another level, providing a simple but powerful extension for the analysis processes[31].

CoSQ. The basis for the Cost of Software Quality (CoSQ) is the accounting of two kinds of costs: costs that are due to a lack of quality and costs that are due to the achievement of quality. We think that CoSQ can help not only to justify quality initiatives, but also have a number of other benefits. Of these benefits, we would like to point out that CoSQ

provides the basics for measuring and comparing the cost effectiveness of the quality improvements undertaken by an organization[32, 33].

Earned value analysis. Earned value analysis has been chosen as the method for performance measurement as it integrates scope, cost, and schedule measures to help managers assess process performance. The three main values and the derived efficiency indexes are used in combination to provide measures of whether or not work is being accomplished as planned. Furthermore, the earned value analysis is used to evaluate the performance of different software process improvements within DIFSPI[34] .

Statistical process control. Current software process models (CMM, SPICE, etc.) strongly recommend the application of statistical control. In the framework, Statistical Process Control (SPC) is used to obtain run charts and control charts with the aim of helping software managers to find an answer to questions such as "How do I know if my software development process is under control?" SPC is also used to test the capability of the process. For this purpose, SPC and earned value techniques can be merged as Lipke and Jennin[35] suggest.

Data mining. Data mining processes can be used to get useful information from the volume of data generated by model simulation. Genetic algorithms are fed with the databases resulting from simulations, and then executed to obtain management rules to guide the process of maturity improvement[36]. Machine learning algorithms based on decision trees such as C4.5[37], decision lists such as COGITO[38], and association rules[39] have been used in combination with other algorithms that transform the simulation outputs into a labeled database. In this labeled database, each record stores information about one simulated scenario (parameters and outputs) and a label that helps to classify the success of the simulated project in terms of time, cost, and quality. After running the machine learning algorithms, a set of management rules is obtained. These rules can be expressed graphically or using natural language. The information they offer is what the best range for the parameters that the algorithm has determined to be the most influential on the success of the project should be to meet the objectives of the project. These objectives, regarding the three key factors of time, cost, and quality together with the labeled databases, constitute the input of the algorithm.

6. Implementation of the Framework

The conceptual ideas presented above were firstly implemented using VemSim® which was used to develop and analyze the different dynamic models. However, there are some drawbacks to using this tool. This simulation environment provides a crude way of modularization, there is no easy way to both overlay objects for abstraction and generate a generic sub-model so that it can be instantiated multiple times without duplicating effort, and hence there is no scoping mechanism, all the elements are global to each other. Like traditional programming languages, a mechanism to allow data encapsulation and modularity is essential for handling complexity in large and complex models. Therefore, the complete framework has been re-engineered using UML and Java™ technology. The purpose of this process was to develop a library of classes, each of which represents a simple dynamic module. When using this tool, a suitable dynamic model is built from the required objects. This way, the abstraction aspect and standardization of the interface of these defined modules may be taken to the point that project managers could transparently "plug-in" the modules regarding the software process improvement they would like to analyze. This approach involves putting special effort into the interfacing mechanism of these different modules when they are plugged together.

7. Example of Usage

This section contains an example of how the use of this framework can help organizations in the field of software process improvement. More precisely, the following example studies one of the key process areas of CMM level 2: influence of the outsourcing activities on software projects. Table 2 shows the initial data for the project.

Table 2: Initial estimates for the project.

Size	20 KLDC
Number of newly hired engineers	3 **engineers**
Number of expert engineers	5 **engineers**

Estimated time	35 months
Number of outsourced tasks	150 tasks
Loss of effort due to outsourcing (%)	15%
Project reduction (%)	5%

Given this initial situation, two different scenarios are simulated. Both of them have the same initial data except for outsourcing activities: one of the projects does not have any outsourcing activities, while the other one does and is driven by the data shown in Table 2. The results obtained from the simulation runs are shown in the following subsections.

7.1.1 . Accomplished tasks

Figure 9 shows the evolution of tasks accomplishment in the project. First of all, it can be observed that the development rates in both projects are of a similar shape. Secondly, the project with outsourcing ends before the project without outsourcing. This may be due to the fact that the organization with outsourcing is carrying out a project that is smaller in size than the project of the organization that is not outsourcing. The vertical dotted line shows when the project with outsourcing is completed.

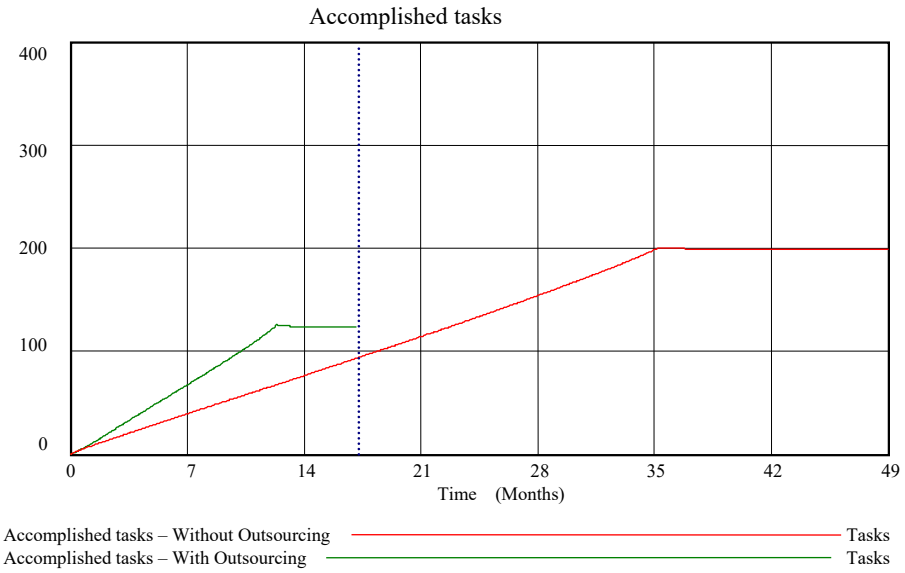


Fig. 9. Evolution of the variable *Accomplished tasks*.

7.1.2 Effort

Figure 10 shows the evolution of the daily effort consumed in the project activities. Notice that the effort values, and therefore cost, for the project with outsourcing are greater than the values for the project without outsourcing.

These higher costs are justified by the effort that needs to be allocated to some activities that are not present in the second project. When a project has outsourcing, some effort has to be allocated to mainly formal communication activities with the members of the outsourced team. This effort allocation leads to the growth of the final costs, a feature that has been illustrated by the simulation outputs.

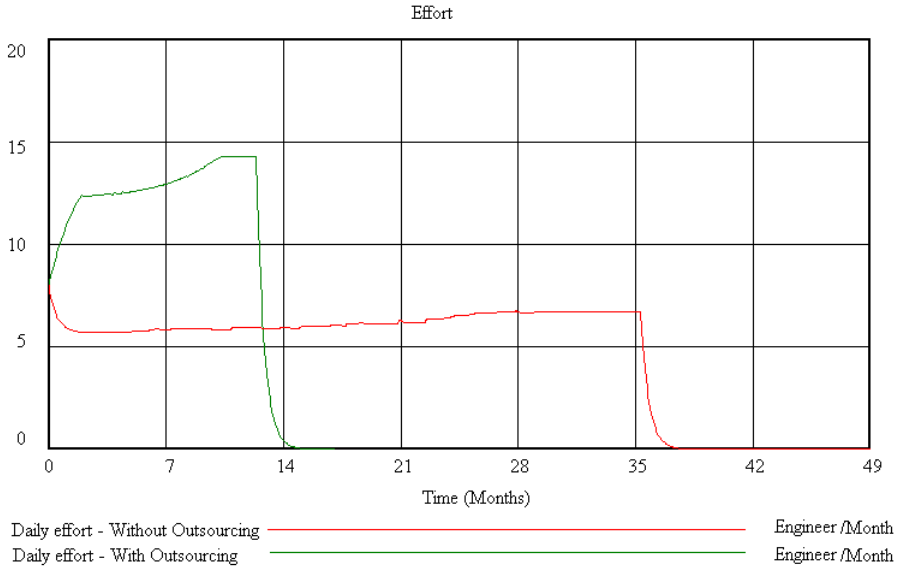


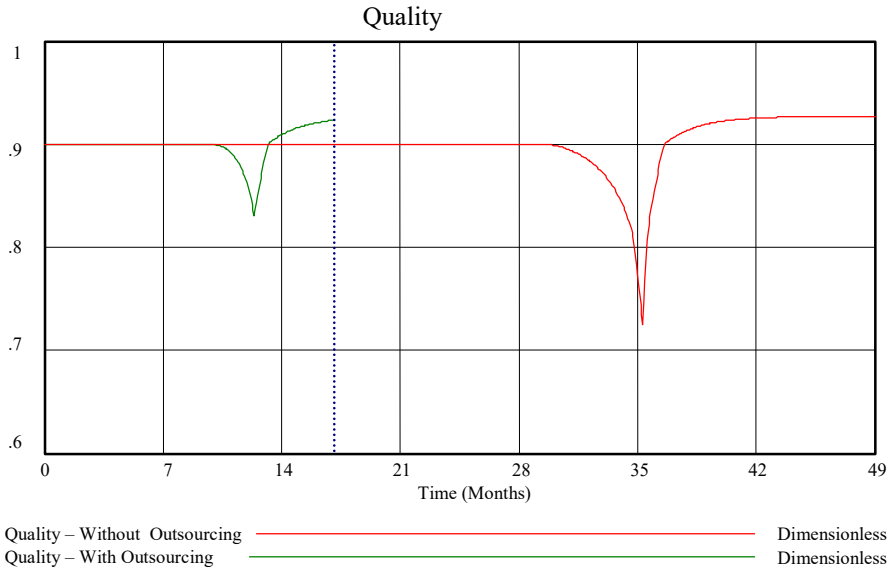
Fig. 10. Evolution of the variable *Daily effort*.

7.1.3. Quality

Finally, Figures 11 and 12 illustrate the aspects concerning the quality of the product under development. The initial quality objective for both projects is set as the number of tasks that need to be demonstrated, tested and corrected. This percentage is 90% for both projects. Figure 11 shows that this percentage is maintained for most of the duration of the lifecycle. However, when the final phase of each project is close, the percentage of tested tasks diminishes considerably. Nevertheless, the project with outsourcing achieves a higher level of final quality. The explanation for this result can be found in Figure 12. Figure 12 shows the evolution of the error detection rate. It can be observed that the project with outsourcing has a much higher error detection rate than the project without outsourcing. This behavior may be due to the fact that in the project with outsourcing, part of the quality assurance activities is performed by the outsourced team. Hence, the volume of

tasks that need to be tested, demonstrated and corrected within the organization is significantly lower, and this makes it possible to achieve higher values in the error detection and correction rates. The increment in these rates translates into a higher quality of the final product.

Fig. 11. Evolution of the variable *Quality*.



8. Conclusions and Outlook

This chapter has focused on software process modeling and simulation together with other traditional techniques to help organizations improve their maturity level according to CMM. There is an important factor that plays a decisive role in the achievement of this improvement. This factor is the knowledge that the organization has of its processes. It is in this field where the modeling and simulation approach can offer important advantages. The first one lies in the actual model building process. A model is a mathematical abstraction of a real system. To effectively build a simulation model, it is necessary to define *what* it is intended to model, define its *scope* and identify the rules that govern its behavior. These

three activities share a common requirement: knowledge about the real system. Without knowledge, there is no information and, therefore, models. According to CMM, without knowledge, it is not possible to define the software process and therefore, to improve the maturity level. Therefore, as far as process maturity level is concerned, knowledge and process improvement go hand in hand.

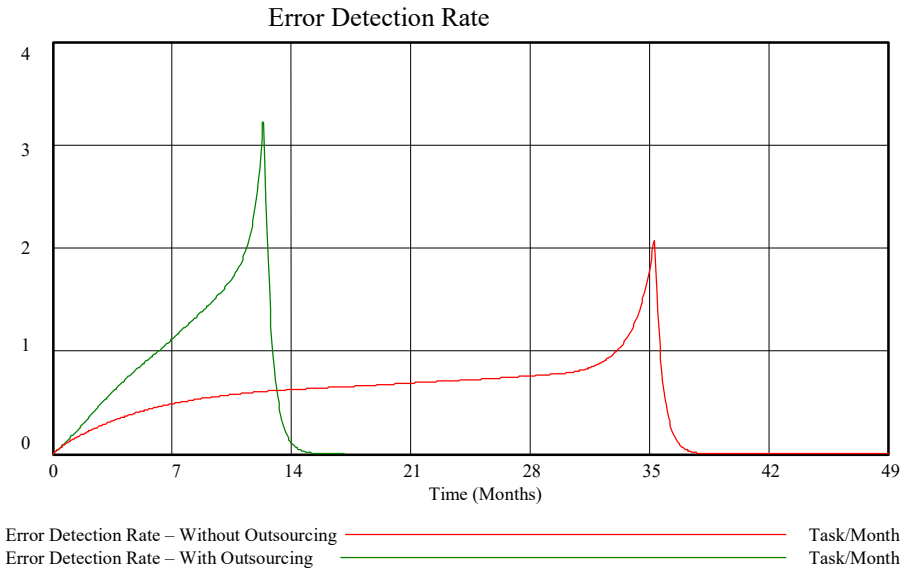


Fig. 12. Evolution of the variable *Error detection rate*.

On the other hand, simulation has always been considered as a powerful tool in the decision-making area. In this chapter, simulation has been proposed not only as a tool to help in the decision-making process, but as a factor that helps to design and evaluate process improvements. It also promotes simulation modeling and modular model building as an approach to automatically trigger the set of metrics that need to be collected, since each new dynamic module developed requires its own set of initial parameters. These initial parameters required to initialize each dynamic module should form part of the metrics collection program carried out within the organization. In addition, this new data is not only

used in the simulation runs, but also to increase the level of knowledge that the organization has of its processes.

As an example of how to integrate traditional software engineering methods with software process simulation modeling, a dynamic integrated framework for software process improvement has been introduced. This framework can build dynamic software process models by means of model abstraction, module construction and reuse. These models can then be used to design and evaluate software process improvements such as analyzing the impact of the size of the technical staff on the main four variables (time, cost, quality, and overall workforce) at a level 1 organization[40] or evaluating the impact of carrying out formal inspection activities in level 3 organizations[41].

Currently, the software process modeling and simulation community is working on the application of this technique to the latest aspects of the software engineering field, such as updating the framework to work according to the CMMi[42]. Some remarkable applications are: web-based open software development[43], open source software evolution[44], extreme programming[45] and COTS-based development[46].

Acknowledgments

This work was supported by the Interministerial Commission of Science and Technology, (CICYT, Spain) through grant TIN 2004-06689-C03-03.

References

1. M.C. Paulk, B. Curtis, M.B. Chrissis, and Charles V. Weber, *Capability Maturity Model for Software, Version 1.1*, Software Engineering Institute, Technical Report CMU/SEI-93-TR-24. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. February 1993.
2. International Standard Organization. *ISO 9001. Quality Systems – Model for Quality Assurance in Design/Development, Production, Installation, and Services*, 1987.
3. International Standard Organization. *ISO 9000-3. Guidelines for the Application of ISO9001 to the Development, Supply, and Maintenance of Software*, 1991.
4. A. Carleton, R.E. Park, W.B. Goethert, W.A. Florac, E.K. Bailey, and Pfleeger S.L. *Software Measurement for DoD Systems: Recommendations for Initial Core*

Measures. Technical Report CMU/SEI-92-TR-19. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 1992.

5. A.M. Christie. S*imulation – An Enabling Technology in Software Engineering*. http://www.sei.cmu.edu/publications/articles/christie-apr1999/christie-apr1999html
6. M.C. Kellner, R.J. Madachy and Raffo, D.M. *Software Process Simulation Modeling. Why? What? How?* Journal of Systems and Software, 46 (1999) 91-105.
7. High Performance Systems, Inc. 45 Lyme Road. Hannover, NH, 03755. http://www.hps-inc.com/edu/stella/stella.htm
8. PowerSim Corporation. 1175 Hendon Parkway, Suite 600, Hendon, VA, 20170. http://www.powersim.com/default_home.asp
9. Ventana Systems, Inc. 60 Jacob Gates Road, Harvard, MA 01451. http://www.vensim.com
10. Proceedings of the 5th International Workshop on Software Process Simulation and Modeling. ProSim 2004. May 24- 25, 2004. Edinburgh, Scotland UK.
11. Proceedings of the 6th International Workshop on Software Process Simulation and Modeling. ProSim 2005. May 14-15, 2005. Saint Louis, MO, USA.
12. D.M. Raffo. *Modeling Software Processes Quantitatively and Assessing the Impact of Potential Process Changes on Process Performance*. Ph.D. Dissertation. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, MA. 1996.
13. M. Kellner. *Software Process Modelling Support for Management Planning and Control*. Proceedings of the First International Conference on the Software Process. Redondo Beach, California. IEEE Computer Society Press, Los Alamitos, CA (1991) 8-28.
14. G.A. Hansen. Simulating Software Development Processes. IEEE Computer, January 1996, 73-77.
15. P.M. Senge. *The Fifth Discipline*. Currency, 1st. Edition, 1994.
16. T. Abdel-Hamid and Madnick, S. *Software Project Dynamics: an Integrated Approach*. Prentice-Hall, 1991.
17. D. Pfhal and Lebsant, K. *Integration of System Dynamics Modelling with Descriptive Process Modelling and Goal-Oriented Measurement.* Journal of Systems and Software, 46 (1999), 135-150.
18. S. Burke. *Radical Improvements Require Radical Actions: Simulating a High Maturity Organization.* Technical Report CMU/SEI-96-TR-025, ESC-TR-96-024. Software Engineering Institute, Carnegie Mellon University Pittsburgh, PA, 1996.
19. P. Wernick, and Hall, T. *Simulating Global Software Evolution Processes by Combining Simple Models: An Initial Study.* Software Process: Improvement and Practice, 7 (2002) 113-126.
20. F.P. Brooks, Jr.. *The Mythical Man-Month*. *Essays on Software Engineering*. 20th. Anniversary Edition. Addison Wesley – Pearson Education, 1995.
21. C. N. Parkinson. *Parkinson's Law: The Pursuit of Progress*, London, John Murray 1958.

22. J.W. Forrester. *Industrial Dynamics*. Walthan, MA: Pegasus Communications, 1961.
23. C.W. Kirkwood. System Dynamics Methods: A Quick Introduction. Technical Report. College of Business, Arizona State University, Tempe, 1998.
24. L.B. Sweeny and J.D. Sterman. *Bathtub Dynamics: Initial Results of a Systems Thinking Inventory*. System Dynamics Review 16 (4): 249-286.
25. I.J. Martinez and Richardson, G.P.. *Best Practices in System Dynamics Modeling*. Proceedings of the 19th International Conference of the System Dynamics Society. Atlanta, GA USA, 2001.
26. J.W. Forrester and Senge, P.M. *Tests for Building Confidence in System Dynamics Models* In Legasto, A.A. Jr., Forrester, J.W. and Lyneis, T.M. (eds.). *System Dynamics*. New York Elsevier North-Holland, 1980, 209- 228.
27. J.D. Tvedt. ***An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time***, Ph.D. Dissertation, Arizona State University, 1996.
28. L.H. Putnam and Meyers, W. *Measures for Excellence: reliable software, on time, within budget*. Prentice Hall, 1991.
29. B. Boehm. *Software Engineering Economics*. Prentice Hall, Inc., 1981.
30. B. Boehm, E. Horowitz, R.J. Madachy, D. Reifer, B.K. Clark, B. Steece, A.W. Brown, S. Chulani and Abts, C. *Software Cost Estimation with COCOMO II*. Prentice Hall, Inc., 2000.
31. T.L. Woodings. *A Taxonomy of Software Metrics.* Software Process Improvement Network (SPIN), 1995.
32. S.T. Knox. *Modeling the Cost of Software Quality* Digital Technical Journal, Vol, 5, No. 4 (fall 1993), 9-16.
33. D. Houston and Keats JB. *Cost of Software Quality: a Means of Promoting Software Process Improvement*. Quality Engineering, 10(3), 563-573, 1998.
34. Q.W. Fleming and Koppleman, J.M. *Earned Value Project Management*, 2nd Edition. Newton Square, Project Management Institute, 1999.
35. W. Lipke and Jennin, M. *Software Project Planning, Statistics and Earned Value.* Crosstalk, December 2000.
36. I. Ramos, J.C Riquelme and Aroba, J. *Improvement in the Decision Making in Software Projects.* Miranda, P., B. Sharp, A. Pakstas, and J. Gilipe (eds.) Proceedings of the 3rd International Conference on Enterprise Information Systems (ICEIS 2001) (on CD-ROM).
37. J.R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kauffman, 1993.
38. J.C. Riquelme, J.S. Aguilar and M. Toro M. *Discovering Hierarchical Decision Rules with Evolutive Algorithms in Supervised Learning.* International Journal of Computer, Systems and Signals 1(1): 73-84, 2000.
39. R. Agrawal, *Mining quantitative association rules in large relational tables*, ACM SIGMOD Record, v.25 n.2, 1-12, June 1996.

40. M. Ruiz, I. Ramos and Toro, M. *A Dynamic Integrated Framework for Software Process Improvement*. Software Quality Journal (10): 181-194, 2002.
41. M. Ruiz, I. Ramos and Toro, M. *Integrating Dynamic Models for CMM-Based Software Process Improvement*. Oivo, M., and S. Komi-Sirviö (eds.) Proceedings of the 4th International Conference PROFES 2002. LNCS 2559. Rovaniemi (Finland), 63-77.
42. M.B. Chrissis, M. Konrad and Shrum, S. CMMi: Guidelines for Integration and Product Improvement. SEI Series in Software Engineering. Addison-Wesley, 2003.
43. C. Jensen and Scacchi, W. *Process Modeling Across the Web Information Infrastructure*. Proceedings of the 5th International Workshop on Software Process Simulation ands Modeling. ProSim 2004. May 24- 25, 2004, 40-49. Edinburgh, Scotland UK.
44. N. Smith, A. Capilupi, and Ramil. J.F. *Qualitative Analysis and Simulation of Open Source Software Evolution*. Proceedings of the 5th International Workshop on Software Process Simulation ands Modeling. ProSim 2004. May 24- 25, 2004, 103-112. Edinburgh, Scotland UK.
45. A. Cau, G. Concas, M. Melis and Turnu, I. *Evaluate XP Effectiveness Using Simulation Modeling*. H. Baumeister, M. Marchesi and M.Holcome (eds.). Proceedings of the Extreme Programming and Agile Processes in Software Engineering. XP 2005. LNCS 3556. June 18-23, 2005, 48-56. Sheffield, UK.
46. M. Ruiz, I. Ramos and Toro, M. *Using Dynamic Modeling and Simulation to Improve the COTS Software Process.* In F. Bomarious and H. Iida (eds.): PROFES 2004, LNCS 3009, 568-581, 2004.