An Integrated Framework for Simulation-Based Software Process Improvement

Mercedes Ruiz[1], Isabel Ramos[2], Miguel Toro[2]
Department of Computer Languages and Systems
[1]University of Cádiz [2]Univeristy of Seville
Spain
[1]mercedes.ruiz@uca.es
[2]{isabel.ramos, miguel.toro}@lsi.us.es

Abstract

In this paper we present an integrated framework for software process improvement according to CMM. The framework is double-integrated. First, it is based on the systematic integration of dynamic modules to build a dynamic model to model each maturity level proposed in the reference model. As a consequence, a hierarchical set of dynamic models is developed following the same hierarchy of levels suggested in CMM. Second, the dynamic models of the framework are integrated with the use of different static techniques commonly used in planning, control, and process evaluation.

The paper describes the reasons found to follow this approach, the integration process of models and techniques, the implementation of the framework, and shows an example of how it can be used in a software process improvement regarding the cost of software quality.

1. Introduction

Software process modeling and simulation is known to be a useful tool for software process improvement, providing important benefits for organizations no matter their maturity level. Although it is relatively simple to build and simulate a process model, we consider that it could be very useful to accomplish the task of modeling under a systematic approach and a set of regular practices or methodologies.

Having in mind that current frameworks for software process assessment and improvement present a defined and hierarchical structure, our approach consists on using the same structure followed by a process improvement framework to guide the development of another process: the development of a set of dynamic models to help in the software process improvement field.

The Dynamic Integrated Framework for Software Process Improvement (DIFSPI) has been developed with the aim of creating both a conceptual framework and a working environment to help in the achievement of higher maturity levels according to CMM. The conceptual ideas underlying the framework are focused on the integration of dynamic models and, then, the integration of dynamic simulation and static techniques. Using the process of model building, a metrics collection program is triggered in order to get the numerical drivers for the model parameters and functions. Besides, this metrics collection program has an important effect as it contributes to an increase in the knowledge the organization has about its processes. This knowledge increase together with the use of dynamic models have three main effects: they lead to better prediction activities, more accurate cost predictions, and more effective process improvement initiatives. Factors which are known to drive organizations towards higher maturity levels according to CMM.

The structure of the paper is as follows. Section 2 describes the reasons found to develop a dynamic integrated framework. Section 3 and 4 relate, respectively, the approach we have followed to integrate first the dynamic models and then the static techniques inside the proposed framework. The principles underlying DIFSPI implementation are discussed in Section 5. Section 6 shows an example of how the framework can be used to analyze the effect of a software process improvement initiative using both, the dynamic models and their integration with a technique to determine the cost of software quality. Finally, Section 7 summarizes the paper and draws the conclusions.

2. Reasons found to develop a dynamic integrated framework

Dynamic modeling and simulation as process improvement tools have been intensively used in the manufacturing area. Current research has also demonstrated their effectiveness as an approach to analyze complex business and solve policy questions in the software process improvement field.

In previous works [1], we attempted to apply the dynamic modeling and simulation approach to support process improvement in small and low mature local software development organizations. In these attempts, we always found the same problem. These organizations sometimes lacked of measurement practices, so that they did not have historical databases, or when they had them, the information available was poor or inaccurate so that it did not supply the numerical drivers required to initialize the parameters and functions of the dynamic models. Finding that the dynamic models had a significant complexity and required an amount of information which was not available in these organizations, we then decided to apply Eberlein's work [2] about understanding and simplification of models to obtain a reduced dynamic model capable of reproducing the software process dynamics, yet with the necessity of less initial information. Nevertheless, simulation is only effective if both the model and the data used to drive the simulations accurately reflect the real world. So, no matter how simple or complex the model is, it will not be of much usefulness if the organization is not able to provide accurate data to initialize and drive the parameters and functions used in the model.

It becomes apparent that there is a strong relationship between the availability of accurate data and the benefits obtained of using dynamic simulation of the software process. It is also known that the availability of quality data is associated with the maturity of the software process executed inside the organization, as this maturity can be thought of the level of knowledge the organization has about its own processes.

Having these ideas in mind, our approach was to develop a set of dynamic models to help organizations improve their software processes according to the Capability Maturity Model [3]. The initial lack of historical data can be solved helping organizations to define and implement metrics collection programs. Come to this point, it is useful to remember that literature is full of good and bad examples of these types of programs. In our proposal, we decided to use the own development process of the dynamic models to point out what data should be collected, thus helping as a clear guideline on what to collect.

On the other hand, knowing the defined structure of CMM, we thought it could be useful to apply the same ideas to develop a hierarchical set of dynamic models. This set is based on principles of abstraction, reusability, flexibility, and extensibility which have been successfully used in the programming field. According to this idea, the framework is made of a set of dynamic models, each of them corresponding to each level of CMM. Generally speaking, each dynamic model is made of a set of basic dynamic modules. The integration of these basic modules, together with certain coupling structures, that allow this integration, makes it possible to obtain the dynamic model for a certain level. This way, what considered to be a dynamic model for, say, level 1 organizations, can be (and if fact is) considered as a dynamic module to build the dynamic model for the following level of maturity.

This is the first reason to use the word *integrated* with this dynamic framework, because it is mainly made of the integration, at different levels of abstraction, of different dynamic modules. But there is another reason for this: the integration of traditional static techniques with the dynamic simulation approach. Although traditional methods for software process management have revealed their weaknesses during the last decades, there is common agreement that they are still useful. These two types of techniques have different approaches and work under different perspectives. It is this feature, which was pointed out by Rodrigues [4], what makes each of them the complement of each other, and what has made us considered their joint utilization.

3. Integration of dynamic models

Once the approach to the elaboration of the dynamic models has been described, this section offers a description of the hierarchical structure of the framework presented in this work.

Figure 1 illustrates this hierarchy. The dynamic model for organizations in level 1 progressing to level 2 is composed of four main dynamic models. Each of them devoted to the modeling and simulation of each of the four main subsystems of the software process: plan, 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.

In order to obtain a dynamic model to model and simulate the software process of level 2 organizations, new dynamic modules are added to the initial model. These new modules model and simulate the new key process areas of the following level of maturity. The new dynamic modules model the following key process areas:

- Requirement management. This module helps to determine the impact of the requirement variability over software development projects.
- Outsourcing management. With this module it is possible to analyze the influence of outsourcing over the life cycle of the project.
- Quality assurance. In this module, the necessary structures to model and analyze the cost and state of the quality assurance activities are implemented.
- 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 over the progress and cost of the project.

The next step towards the following level of maturity does imply an important structural change. This change is

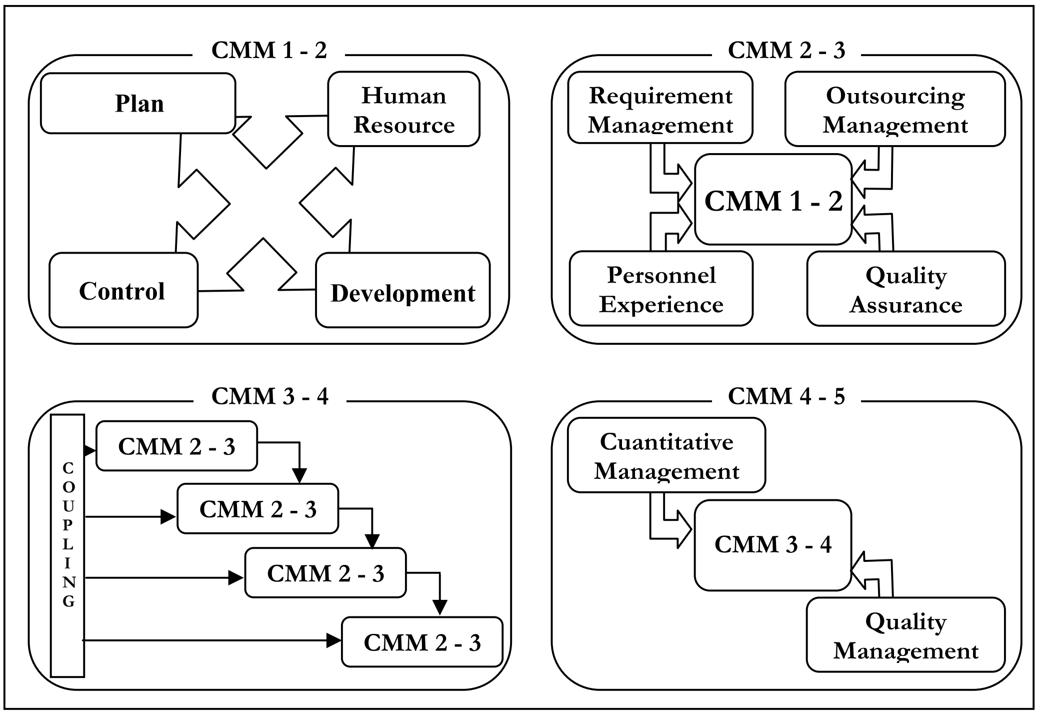

Figure 1. Hierarchical structure of the integrated dynamic framework

determined by the special emphasis on the engineering activities that CMM suggests beginning from level 3. If in the initial levels of maturity CMM recommends the development of good plan and management practices, it is in level 3 when the engineering process acquires a key importance. To reflect this idea, the principle of model replication is used. Thus, to model organizations in level 3 progressing to level 4, the model developed for the previous level is replicated as many times as generic phases compose the work breakdown structure of the project. In our particular case, we considered the four main characteristic phases of a traditional life cycle: analysis, design, code, and test. To simulate each phase, a complete dynamic model is used. Each of these dynamic models can be used, in isolation, to simulate the whole project on organizations with the previous level of maturity. In order to get all these models working together to simulate a higher maturity organization, coupling structures need to be defined. These coupling structures must allow the inter-module communication as well as serving as support structures for the sharing of information.

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

4. Integration of techniques

The following techniques and methods are currently successfully implemented in DIFSPI:

Traditional estimation techniques. Traditional algorithmic estimation models have been implemented inside this framework with the aim of providing an initial baseline for software projects carried out in low maturity level organizations. Algorithmic models such as COCOMO [5], COCOMOII [6] and others (Basily, Doty, and Waslton) are used to drive an initial estimation of effort and time, especially in models for low maturity organizations.

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. Inside this framework, these four measures constitute the basics for both, the dynamic models and the graphical representation of the process performance [7]. Although this set of metrics is available for all the levels of maturity, models developed for the lower levels have it as the main source of reliable information about the software process under simulation.

Metrel Rules. Given the dynamical nature of the DIFSPI proposed, we consider it could be useful to integrate a taxonomy of software metrics which is derived from the needs of users, developers, and management. Among all the advantages that can be obtained with the use of 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 over the duration of a project, the life of a product or the strategic plan of an organization. In DIFPSPI 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 as derivative 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 [8]. Metrel rules can be used to evaluate a specific software process improvement, as well, as quantifying the return on investment of tackling this SPI and its benefits. [9] shows an example of application of the Metrel taxonomy to analyze the evolution of the percentage of detected and corrected errors (which is the process metric in that example) when the parameter "percentage of people assigned to inspection" is varied. This SPI is suitable for level 3 organizations in which CMM gives a special importance to formal inspections. For that reason, a dynamic module to simulate the process of formal inspections was developed and "plugged" in the structure of the dynamic model for level 3 organizations. The results of the simulations of the joint structure were then analyzed under the Metrel taxonomy.

CoSQ. The basis for the Cost of Software Quality (CoSQ) is the accounting of two kinds of costs: those which are due to a lack of quality and those which are due to the achievement of quality. These costs have an inverse relationship that is normally shown as a set of two-dimensional curves that plot costs against a measure of quality [10], [14]. Section 6 of this paper shows an example of the combined use of this technique with dynamic simulations.

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 inside DIFSPI [11].

Statistical process control. Current software process models (CMM, SPICE, etc.) strongly recommend the applications 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 for questions such as "How do I know if my software development process is in control?" SPC is also used to test the capability of the process. In order to do that, SPC and earned value techniques have been merged in the way [12] suggests.

Data mining. Using data mining processes it is possible to obtain 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 in the process of maturity improvement [13].

5. DIFSPI implementation

The former conceptual ideas were initially implemented using VenSim®. The initial use of this tool let us develop, test and analyze each one of the basic dynamic modules. For the first initial models (level 1 and 2), it was necessary to duplicate the effort of coding, as the final model had to join together the different dynamic modules. To obtain the more complex models, the subscripting capability of the mentioned environment was used.

However, this simulation environment offers a crude way of modularization and does not make it easy to overlay objects for abstraction. It does not supply an easy way to generate generic sub-models which can be instantiated multiple times. Other important aspect that it lacks of, is the absence of a scoping mechanism, so that all elements are global to each other.

Like in traditional programming languages, a mechanism to allow data encapsulation and modularity is essential to handle the complexity in large and complex systems. Due to the problems encountered, the complete framework has been re-engineered using Java™ technology with the purpose of developing a library of classes. This way, the abstraction aspect may be taken to the point where it is possible to "plug-in" the dynamic modules that form the main model and those regarding the software process improvement intended to be analyzed.

Using this technology, a tool for designing and analyzing software process improvements to help in the achievement of higher maturity levels is under development. A fully functional initial version has been developed, and the results obtained have been encouraging, so far.

Using this tool, three groups of initial parameters are required to drive a simulation run: parameters related to the organization (delays, average accepted overwork, etc.), parameters related to the project (size, quality objective, initial staff, etc.) and parameters to drive the simulation run. With these initial data, it is possible to run a first simulation to establish a baseline to the project. The results obtained can be graphically displayed in order to merge in a single view the static data offered by the traditional models with the dynamic data provided by the simulation runs. After this, it is possible to experiment different process improvements and alternative plans by changing the values of the parameter(s) required and running the new simulations. All the results obtained are saved in a database of simulations. It is important to notice that the saved data is not only the data resulting of simulating the equations of the model, but the results of applying the metrics program (initially developed for the organization) to the dynamic model itself. This database may then be used to feed some machine learning algorithms in order to automatically obtain management and process improvement rules.

6. Utilization of DIFSPI

In this section we provide an example of the joint utilization of the dynamic models of DIFSPI and one of the integrated techniques. In this example, CoSQ has been used to initially obtain the cost of software quality in a certain project, and then to investigate the effect of different timing management policies over CoSQ. Using simulation to determine the effect of software process improvement initiatives has also been studied by Houston and Keats [14]. In their work, simulation is used to evaluate the effect of developing inspection activities on the overall cost of the software quality.

6.1. Principles of Cost of Software Quality

As it has been previously mentioned, the value for the CoSQ is determined by the sum of the costs due to both the lack of quality and the achievement of quality in a software project.

Table 1 [14] provides the definition of the four CoQ categories in which CoSQ can be divided with typical costs of software quality. The costs of achieving quality and the costs due to the lack of quality have an inverse relationship: as the investment in achieving quality increases, the costs due to lack of quality decrease. While costs of quality assurance and process improvement have been a topic of concern for over 20 years, very limited data has been available in the open literature for discussing the cost of software quality [10], [14] and [15].

Houston and Keats outline the following benefits of using CoSQ:

Table 1. Definition of Cost of Quality Categories

Category	*Definition*	*Typical Costs for Software*
Internal failures	Quality failures detected prior to product shipment	Defect management, rework, retesting
External failures	Quality failures detected after product shipment	Technical support, defect notification
Appraisal	Discovering the condition of the product	Testing and associated activities, product quality audits
Prevention	Efforts to ensure product quality	SQA administration, inspections, process improvement, metrics collection and analysis

- CoSQ can be used to compare process improvements and identify the most effective one. Hence, it results a very appropriate technique to be integrated in a software process improvement framework.
- CoSQ can be used to identify quality improvement candidates. Examination of the CoSQ components often reveals higher quality costs in a particular area. For example, high appraisal costs due to integration testing, may indicate a lack of consideration for interfaces during design, or it may indicate late design changes. Analysis of the causes of these costs can provide the basics for quality initiatives in development processes.
- CoSQ provides a measure to compare the success of projects. Even within a given organization, the software process may vary from one project to another. The many factors, tangible and intangible, that characterize a project make it difficult to compare projects. CoSQ can be measured within and across projects and provides a means of comparison.
- CoSQ can provide a basis to budget the quality operation. Rather than allocating the quality operation a percentage of the total development budget, management can budget the quality operation based on expected returns.

6.2. Specific dynamic modules

In order for the simulations to provide the required data for the CoSQ analysis, two modules gain a special importance. The main features of these modules, devoted to the modeling and simulation of the quality assurance and testing activities, are further explained in the following subsections.

6.2.1. Quality Assurance Subsystem

The Quality Assurance Subsystem models the activities of detection and correction of the defects in the development project. In a software development project there are two sets of factors that affect both, the generation of defects and the capability to their detection. The first set is integrated by some features of the organization (use of structured or object-oriented techniques, personnel experience, etc.) and the second set contains those aspects that are related to the project itself (complexity, languages used, size, etc.).

These two sets of factors can vary from one organization to another and also between projects of the same organization, but they remain stable in one project. The effect of these two sets of factors determines a nominal capacity to the generation of defects inside a given project. By the other hand, the volume of defects that can be generated inside a project is also influenced by others factors of a dynamic nature such as the schedule pressure (as the pressure grows, the number of defects that people commit will be larger) and the experience of the personnel (the lack of experience of a technician not only diminishes his/her productivity, but it increases the number of defects in his/her activities). These last factors together with those of static nature determine the *real* capacity to defect generation in a project. The capability to detect the defects of a project is determined by the effort assigned to the software quality assurance, the experience of personnel and the effort required to correct an error. The effort assignment to the quality activities can be done in two different ways. The first way consists on assigning a fixed value of effort for all the life cycle. This value is a percentage of the available effort for the development activities. The alternative way assigns a variable effort along the whole life cycle that will fundamentally depend on the pressure under which the project is in every moment. The capability to correct the defects of a project is a function of both, the assigned effort to the correction activities and the required effort to correct a defect.

6.2.2. Testing Subsystem

The defects that have not been detected by the quality assurance activities or those owed to bad corrections, must be detected by the testing activities. The propagation and reproduction of the defects and the cost of detecting and correcting them is gathered in the model. The capability for testing tasks is determined by the effort assigned to these activities, the effort required to test a task, and the efficiency of the personnel assigned. The testing activities end when all the developed tasks have been tested. The model considers that in this moment, the project has ended because it does not consider the maintenance phase.

6.3. Determination of CoSQ

For this study we properly calibrated the dynamic model to reproduce the medium size real project proposed in [15]. We used this project because of the large amount of information available of it and also because the model has been completely validated with these data.

Table 2. Costs of a medium size project

PC (t-d)	*ICF (t-d)*	*AC (t-d)*	*CoSQ (t-d)*	*Total Cost (t-d)*
480.26	266.50	258.09	1,004.85	1,950

The metrics used to evaluate the different costs of CoSQ are:

1. Costs due to lack of quality:
- *Internal Costs of Failure* (ICF): Effort employed in correction.
- *External Costs of Failure* (ECF): Effort required to correct defects when the product has been shipped. This value is computed using the number of defects in the product when the testing phase has finished. We have not been able of evaluate this metric with the simulation of the dynamic model because it does not contemplate the maintenance phase as we mentioned before.

2. Cost of achieving quality:
- *Appraisal Costs* (AC): Effort invested in testing activities.
- *Prevention Costs* (PC): Effort invested in inspections of software quality.

With these metrics we define the cost of software quality as:

$$CoSQ = ICF + ECF + AC + PC$$

The costs obtained by the simulation of the model (measured in technician-day) for a medium size project in normal conditions are shown in Table 2 and graphically in Figure 2.

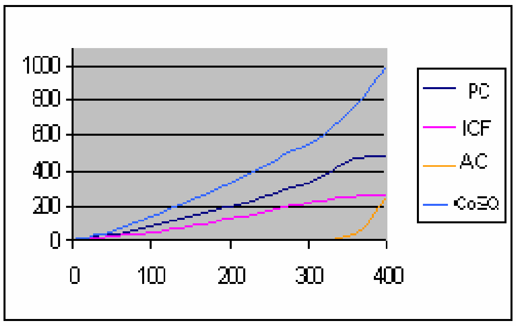

Figure 2. Cost of software quality evolution

6.4. Impact of different management policies over CoSQ

To study the effect of different management policies that affect the severity of the deadline of the project over the CoSQ, we propose using simulation and:

- Determine the influence of policy with a restricted deadline of finalization over the cost of quality.
- Discover the influence of a combination of management policies over the cost of quality. This combination of management policies is determined by a policy with a restricted date of finalization and different policies of effort assignment to the development activities.

The results obtained are:

1. Policy with a restricted date of finalization: We simulated the model twice: the first time to reproduce the situation of working without an important schedule pressure (non fixed date) and the second to contemplate the scenario of the same project with a very important restriction in its final deadline (fixed date). The results obtained are shown in Table 3.

   When the project has a fixed deadline, the cost of quality represents 57.3% of the total cost of the project face to the 51.5% of the case in which the project does not have fixed date. Curiously, the number of defects by KLDC is very similar in both scenarios, but we discover that this is due to the large inversion in prevention activities (PC) and appraisal ones (AC) face to the case in which the project does not have a fixed date.

Table 3: Effects of the restricted date policies over CoSQ

	Fixed Date	Non Fixed Date
Number of defects by KLDC	6.54	6.45
Total Cost (technicians-day)	3,234.00	1,950.00
PC (technicians-day)	678.86	480.26
ICF (technicians-day)	314.24	266.50
AC (technicians-day)	861.12	258.09
CoSQ (technicians-day)	1,854.22	1,004.85

2. In this case, we simulated the model and found out the influence of a combination of a policy of fixed date and different policies of effort assignment to development activities, over the cost of software quality. The effort assignment to the project is done in the following way: First, you must subtract the effort dedicated to train new personnel from the total effort estimated for the project. The resulting effort is distributed between the activities of the project as it follows: a percentage is assigned to the production activities (design, code and quality assurance) and the rest is assigned to the testing activities. The required effort for quality assurance is obtained as a percentage of the effort assigned to production activities. In the simulations performed, the assignment of effort to the software quality assurance activities was done attending to the actual state of the project. The results obtained are shown in Table 4.

Table 4: Results obtained combining different policies

Restrictions in the Date of Finalization	***% of Effort Assigned to Production***	***CoSQ***	***Nº of Defects by KLDC***	***Total Cost***	***% CoSQ over Total Cost***
Non Fixed Date	**Low Assignment**	972.29	5.33	2,098	46.3%
	High Assignment	970.49	5.45	1,903	51%
Fixed Date	**Low Assignment**	1,012.36	5.34	2,189	46.2%
	High Assignment	1,703	5.53	3,020	56.4%

With these results two conclusions can be obtained: First, from the point of view of the cost of software quality, the policy with fixed dates is stronger than the policy of effort assignment to production activities. We know that the policy of fixed date increases the costs of a project [16] as the management generally decides to acquire more technicians to finish the project in the planned date. This increase of personnel causes an increase of the total cost of the project and, therefore, the costs of the quality assurance and testing activities also increase. However, it results curious that although this increase of the CoSQ in projects that have fixed dates is important, the effect over the quality of the final product is not so important. In fact, if you look at the column relative of the number of defects by KLDC, you see that there are no significant differences between the projects with higher CoSQ. The reason for this behavior can be understood if we think about the fact that in projects with fixed date, the personnel commit more defects due to the whole pressure and to the accumulated fatigue. This rise of committed defects requires a bigger amount of activities of detection and correction, which are not able to improve the quality, but guarantee a normal level of quality. As far as the dedication of the personnel regards, the results are not less interesting. In the cases where the dedication of personnel to development activities is low, CoSQ is situated round 46% of the total cost of the project. This percentage is smaller than the value obtained in projects with a higher assigned effort. Among the three components of CoSQ evaluated, the one that more contributes to the final value are the appraisal costs, as in projects with this distribution of effort, testing activities result improved. Nevertheless, if you focus on the nominal values, you discover that the policy of low effort assignment to development activities, always increments the final cost of the project and the quality cost of it, independently of the policy of deadline restriction used.

7. Conclusions

With the main objective of assessing project managers and members of the SEIG to define, evaluate, and implement different process improvements to achieve higher maturity levels, a dynamic integrated framework for software process improvement has been initiated. This framework is double integrated. First, it integrates the use of dynamic modeling and simulation with other static and traditional techniques. Second, the dynamic models built in the framework are obtained by the integration of different dynamic modules previously generated. Dynamic modules that may have been generated themselves by the integration of other previously developed dynamic modules.

The framework has a defined architecture that follows the same hierarchical structure of CMM. The methodology followed to obtain each of the dynamic models of the framework (each for a different maturity level achievement) relies on the principles of aggregation/decomposition, and extensibility of dynamic models.

With the application of DIFSPI some important benefits are expected to be obtained. First, during the process of model building, the project manager may gain much new insight into those aspects of the development process that mostly influence the success of the project (time, cost, and quality). Second, having the possibility of gaming with the DIFSPI, it allows project managers to better understand the underlying dynamics of the software process. As a consequence, several process improvement suggestions may easily be designed and, most importantly, analyzed using simulation of scenarios. Third, templates and guidelines for a metrics collection program may be almost automatically derived from the requirements of the dynamic modules. Fourth, the approach of abstraction and encapsulation followed to develop the dynamic modules makes it possible to easily instantiate a dynamic model using different dynamic modules which can be plugged in the final model. Finally, the combination of the dynamic approach with other techniques allows project managers to perform complete analysis and quantification of the effects and the benefits of different software process improvements. All these features combined in the framework intend to help organizations to design and execute more mature processes and, therefore, to increase their maturity level.

As far as the joint utilization of the dynamic models of the framework with the CoSQ technique is concerned, the following specific conclusions can be obtained:

According to [14], CoSQ is a proven technique in manufacturing and service industries both for communicating the value of quality initiatives and for indicating quality initiatives candidates. CoSQ offers the same promise to software process improvements but it has not been widely used by the software development community. Initial uses of CoSQ show that software quality constitutes a very large percentage of the development costs (60% and higher for organizations which have not yet undertaken process improvement programs).

In our opinion, this technique together with the simulation of dynamic models offers three important benefits:

1. It provides a mechanism to evaluate the effect of an initial quality investment, allowing to experiment with different effort assignments.
2. It offers an objective measure for the comparison of different scenarios of a given project, different projects of a same organization, or even different projects of different organizations.
3. CoSQ and simulation used together allow the obtaining of metrics values *during* the project execution. Therefore, it is not needed to wait until the end of the project to obtain the evolution of the CoSQ, and, what is more important, it is totally possible to simulate any of the phases of the life cycle, for instance the design phase,

and observe the evolution of the product and its quality. Therefore, different alternative policies aimed to the optimization of the results can be *tried*. In conclusion, the combination of the techniques described before, will allow to try different management alternatives and analyze the results obtained.

Our future work is mainly oriented towards the full development and implementation of some key process areas in DIFSPI, the study of its adaptation to other software process models (such as CMMi, and SPICE), and the study of what lessons can be learned from this attempt, in order to develop a methodology to develop dynamic models based on principles of integration, abstraction, encapsulation, reusability, etc. of dynamic modules.

8. References

[1] Ruiz M., I. Ramos, and M. Toro, A simplified model of software project dynamics. Journal of Systems and Software, Vol. 59, No 3 (2001) 299-309.

[2] Eberlein, RL, Simplification and understanding of models. System Dynamics Review, 5, 1989;1:51-68. System Dynamics Society.

[3] Paulk M., SM. Garcia, MB. Chrissis, and M. Bush., Key practices of the capability maturity model. Version 1.1 Technical Report CMU/SEI-93-TR-25. Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA (1993)

[4] Rodrigues A, and B. Bowers, System dynamics in project management: a comparative analysis with traditional methods. System Dynamics Review 12, 2.

[5] Boehm B., Software Engineering Economics. Prentice-Hall Inc. (1981).

[6] Boehm B., E. Horowitz, R. Madachy, D. Reifer, BK. Clark, B. Steece, AW. Brown, S. Chulani, and C. Abts, Software Cost Estimation with COCOMO II. Prentice-Hall Inc. (2000).

[7] Carleton A., RE. Park R.E., WB. Goethert, WA. Florac, EK. Bailey, SL. Pfleeger, Software measurement for DoD systems: recommendations for initial core measures. Technical Report CMU/SEI-92-TR-19. Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA (1992).

[8] Woodings T.L.: A Taxonomy of Software Metrics. Software Process Improvement Network (SPIN) (1995).

[9] Ruiz, M., I. Ramos, M. Toro, Integrating Dynamic Models for CMM-Bases Software Process Improvement. Oivo,M., and S. Komi-Sirviö (eds.) Proceedings of the 4th. International Conference PROFES 2002. LNCS 2559. Rovaniemi (Finland), 2002.

[10] Knox ST., Modeling the Cost of Software Quality. Digital Technical Journal, Vol. 5, No 4 (fall 1993), 9-16.

[11] Fleming QW., JM. Koppelman, Earned Value Project Management, 2nd Edition, Newton Square, Project Management Institute (1999).

[12] Lipke W., M. Jennin, Software Project Planning, Statistics, and Earned Value. Crosstalk, December 2000.

[13] Ramos, I., JC. Riquelme, and J. Aroba: Improvement in the decisión 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), 2001.

[14] Houston, D., and JB Keats, Cost of Software Quality: a Means of Promoting Software Process Improvement. Quality Engineering, 10(3), 563-573, 1998.

[15] Hersch, A., Where's the Return on Process Improvement. IEEE Software, July, 1993.

[16] Abdel-Hamid, T., and E. Madnick, Software Project Dynamics: an Integrated Approach. Prentice-Hall, Inc. 1991.

Acknowledgements

The authors wish to thank the Comisión Interministerial de Ciencia y Tecnología, Spain, (under grant TIC2001-1143-C03-02) for supporting this research effort.