Using Dynamic Modeling and Simulation to Improve the COTS Software Process

Mercedes Ruiz[1], Isabel Ramos[2], and Miguel Toro[2]

[1] Department of Computer Languages and Systems
Escuela Superior de Ingeniería. University of Cádiz
C/ Chile, nº1. 11003 – Cádiz (Spain)
mercedes.ruiz@uca.es
[2] Escuela Técnica Superior de Ingeniería Informática. University of Seville
Avda. Reina Mercedes, s/n. 41012 – Seville (Spain)
{isabel.ramos, miguel.toro}@lsi.us.es

Abstract. In the last several years, software industry has undergone a significant transition to the use of existing component products in building systems. Nowadays, more and more solutions are built by integrating Commercial-Off-The-Shelf (COTS) products rather than building from scratch. This new approach for software development has specific features that add new factors that need to be taken into account to successfully face software development. In this paper, we present the first results of developing a dynamic simulation model to model and simulate the COTS-based software development process with the aim of helping to understand the specific features of this kind of software development, and design and evaluate software process improvements. An example of how to use these dynamic simulation models to study how the system integration starting point affects the main project variables is shown.

1 Introduction

In the software industry, demands for new services are outpacing our ability to develop and manage them. Current systems have stringent demands in scalability, reliability, and real-time interaction with other elements. In an attempt to meet the requirements of current systems, new paradigms, methods and processes have been developed by the software engineering community. One of them, the component-based software engineering (CBSE), promotes the building of new systems by incorporating pre-existing software with the aim of lowering overall development and maintenance costs, as well as involving less development time. The practice of 'buy, don't build' was initially introduced by Fred Brooks in 1987 [3]. Since then, the features and trends of new software systems have made nothing but support the benefits of building by integration and not from scratch.

Nowadays, creating, deploying, and offering a new software system often requires complex interactions between several disparate systems, some of which may be legacy ones. The problem is how to achieve this integration in a speedy, costeffective, flexible manner. It is important to minimize costs for building and maintaining integration solutions. Over the last decades, there have been several approaches to solve the problem of integrating systems. Integration techniques can be classified into three generations attending to their evolution over time.

First-Generation integration techniques were the first to appear. This approach coincides with the first attempts of the software industry in software reutilization, and it is based on the concept of point-to-point integration. It becomes evident that this solution to the problem of integration was soon observed impractical because of two factors. First, the number of interfaces required grows exponentially with the number of components to integrate. Second, the impact of minor changes, such as that of adding a new component, is significant, turning maintenance into a nightmare.

Second-Generation integration techniques, try to solve the problem of the increasing number of interfaces by reducing them to a linear increase through the use of middleware. This solution requires interfacing each application to a 'data bus' using, for instance, the Common Object Request Broker Architecture (CORBA) [7] and XML [14] as the data format for the interchange of data through this so-called data bus.

Third-Generation integration techniques encourage the concept of extreme integration. This extreme integration is aimed to obtain a better understanding of the requirements, an easier maintenance, and therefore, an easier accommodation of changes in the system developed. Within this extreme integration approach, the concept of building software using certain components called Commercial-Off-The-Shelf (COTS) products has become a topic of research, development, and production. COTS refers to a particular type of software component which is purchased from and supported by a third party and that is not customized, or is only minimally customized. COTS software has the potential to save both time and money in the software development process and it is a common way of developing software nowadays. Some examples of the systems developed under this approach are: Internet tools and browsers, CASE tools, GUI generators, code generators, database management systems, Geographic Information Systems (GIS), office automation software, and operating systems, among others.

Each COTS software component used means less code that needs to be designed and implemented by the developers. However, the developer is faced with the problem of ensuring that the COTS product does perform the functionality that it claims to perform, that is not intentionally perform functionality to be harmful to the system, that it will not adversely affect the system, and that it can robustly respond to failures and anomalous inputs to prevent errors from propagating through the entire system. Furthermore, the use of COTS products in software development can require a considerable integration effort leading to the research of new models or means to better manage this effort.

Whereas much of the research effort in the area is mainly focussed on the development of methodologies to better document, search and evaluate components, or the improvement of the trading process, it is also necessary to invest some research effort to help in the understanding and improvement of the specific features of the COTS software process. In this paper, we propose the utilization of System Dynamics simulation models to help understand and improve the COTS-based software development process. The structure of the paper is as follows. Section 2 introduces the motivation and problem definition, the concepts of software process modeling and simulation using system dynamics models, and describes in depth the model developed to analyze the phases of glueware development and system integration. An example of how these dynamic models can be used to simulate different scenarios and analyze its results is included in Section 3. Finally, Section 4 summarizes the paper and draws the conclusions and further works.

2 Developing the Simulation Model for COTS Process

2.1 Motivation and Problem Definition

Our main objective is to develop a simulation model to help understand and improve the COTS software process. In the building of this simulation model, several issues regarding modularity, abstraction and reusability have been taken into account. With the aim of obtaining a certain product after this research effort, our goal has been the development of the dynamic simulation modules that can be added to, or literally "plugged" into, an existing framework of dynamic modules that simulate the traditional software process. Following this approach, our intention is to build new dynamic models combining the existing dynamic modules that help in the design of improvement initiatives aimed to achieve higher maturity levels within organizations, with the new ones that have been developed. The dynamic model resulting from the collaboration of the previous modules (for the traditional software development) and the new ones (specific for the COTS software development) will be able to simulate the complete life cycle.

2.2 Simulation Approach

In the simulation domain there are multiple strategies to build models. Among them, there are two main approaches: continuous and discrete modeling. The continuous simulation technique is based on System Dynamics [1]. 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, in most occasions, can imply 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 to include systems thinking and it is considered to be better than the discrete event model at showing qualitative relationships [13].

Traditionally, three important drawbacks have been claimed against the use of dynamic simulation models in the software industry: the level of education or expertise needed to develop and use the models, the effort required to be invested to model the organization processes, and the lack of data available to validate and populate the final models. [8]

Having specific education in order to be able to use simulation models is no more a requirement. Current simulation tools allow to develop user-friendly interfaces capable of hiding all the mathematical details of models, and enabling to carry out simulation games easily.

In an attempt to initiate a research effort aimed to ease the objections previously mentioned, we developed a Dynamic Integrated Framework for Software Process Improvement (DIFSPI) [10, 11, and 12]. This framework had been originally designed with the aim of creating both a conceptual and formal framework, and a working environment to help in the achievement of higher maturity levels according to CMM [6]. The main issues underlying this integrated framework follow. 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.

Although there are some significant applications of System Dynamics to model and simulate the traditional software process, little has been done regarding the modeling and simulation of the COTS-based systems in the sense of this study, except Kim's work [4]. However, concepts such as modularization, abstraction, encapsulation and reutilization that have been widely applied in the field of computer programming, and which have not been so commonly applied in the field of System Dynamics, are indeed strongly used within the framework proposed.

2.3 COTS Process Modeled

Like other software development processes, the process of building a COTS-based system starts with the definition of the system requirements. Once the system requirements have been gathered and reviewed, the processes of COTS identification, evaluation, and selection begin. COTS identification consists of Web searches, product literature surveys and reviews, identification of other reusable system components, and recommendations from external sources. As COTS components are identified, the evaluation and selection processes start. COTS evaluation steps include prototyping, vendor demonstrations, and in-depth review of literature such as manuals and user guides. Vendor training sites and availability are considered. Procurement issues surface such as development fees for added requirements, licensing and maintenance fees, and sustaining engineering support.

It can be said that there are as many COTS software development processes as organizations are applying the principles of integration to develop software. For this study, the suggested process resulting from the survey carried out by the Software Engineering Laboratory (SEL) has been used [5]. This process is targeted to COTS-based projects using several peer COTS or one COTS integrated with a considerable amount of new developed software.

According to the conclusions of this study, the main phases of the COTS software process are: requirements, design, coding and integration. Most phases encompass activities specific to COTS-based development. Figure 1 illustrates the proposed COTS process. It is important to notice that the horizontal line in the figure graphically separates the two tracks existing in COTS-based projects, that is, traditional activities and COTS-specific activities (highlighted in shadowed boxes).

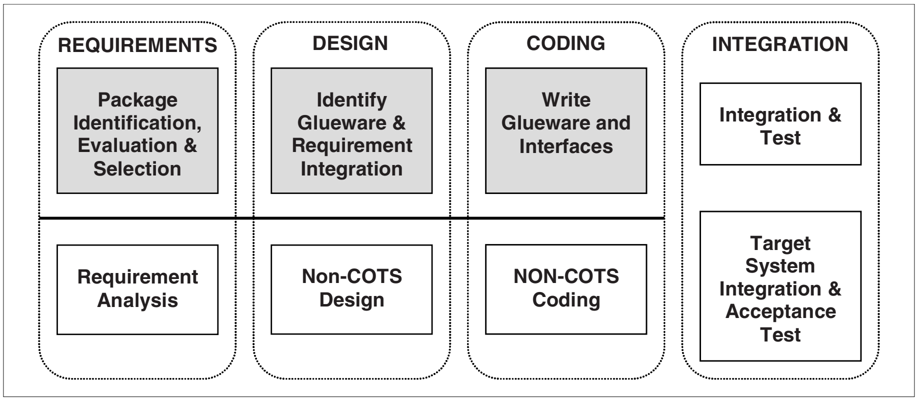
REQUIREMENTS
Package Identification, Evaluation & Selection
Requirement Analysis
DESIGN
Identify Glueware & Requirement Integration
Non-COTS Design
CODING
Write Glueware and Interfaces
NON-COTS Coding
INTEGRATION
Integration & Test
Target System Integration & Acceptance Test

Fig. 1. Proposed COTS Process

It is important to notice that the parallelism between the COTS-specific activities and the traditional activities is strongly emphasized using this process. Hence, a specific effort to support the same parallelism between theses activities in the simulated process has been needed.

2.4 Development of the Simulation Modules

For the purpose of this study, new dynamic modules have been developed and added to the DIFSPI. These dynamic modules are aimed to model the structure, relationships, and behavior of the following processes:

- Glueware Development. Glueware is the new code needed to get a COTS component and integrate it into a larger system that can be the target system or another component that needs to be integrated in a later phase. This special kind of code is considered to be one of the following: 1) any code required to facilitate information or data exchange between a COTS component and the application, 2) any code needed to "hook" the COTS component into the application, even if it may not necessarily facilitate data exchange, and 3) any code needed to provide functionality that was originally intended to be provided by the COTS component, and which must interact with the COTS component [2].

- Application Integration. The integration process varies a great deal from project to project, depending on which and how many COTS products are being used. At system integration and testing the COTS packages are treated as black boxes. The final integrated system is made up from the application system, the COTS packages, and the glueware that has been needed to be developed.

- Another non-process dynamic module has been integrated in the framework to model and facilitate the analysis of cost effects of different improvement initiatives. This new module gathers together all the parameters that have an influence on the COTS-based system development according to the COCOTS model [2].

The following sections describe in depth the components of these modules.

Glueware Development

This dynamic module represents the process of glueware development. The development of glueware can be achieved through the following phases:

- First, the requirements for the glueware to be developed must be elicitated and analyzed.
- Second, these requirements constitute the input to the design phase, and
- Finally, the designed product enters the phase of coding.

Figure 2 illustrates a simplified stock and flow diagram of this module.

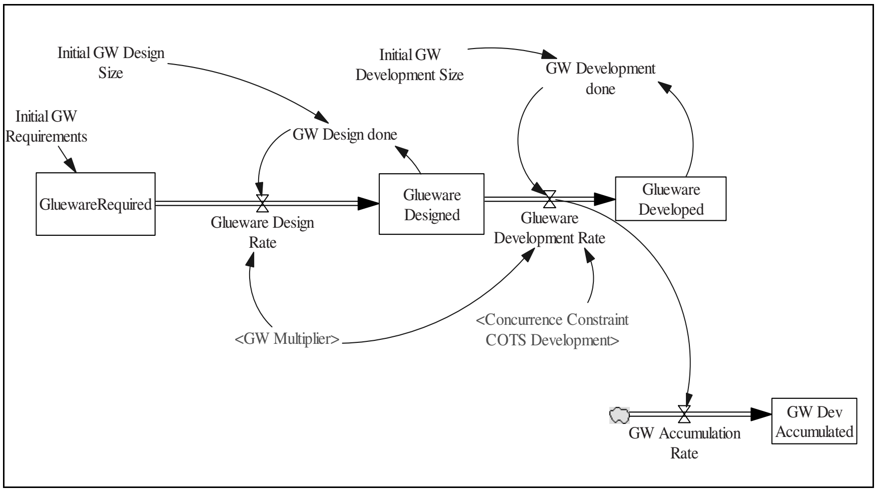

Fig. 2. Simplified stock and flow diagram for Glueware development

As it can be seen, each one of the phases previously mentioned is represented by a level variable. Each level variable is initially populated with the estimated size of the product that needs to be developed during that phase. The meaning of each of these level variables follows:

- `Glueware Required`: Glueware that needs to be developed, measured in number of tasks.
- `Glueware Designed`: Glueware that has been designed, in number of tasks.
- `Glueware Developed`: Glueware that has been coded, in number of tasks.
- `GW Dev Accumulated`: Glueware that has been coded, in number of tasks.

By applying a development rate, work flows from a level to the following. Rates mainly depend on the productivity of the personnel assigned to the development of each phase. They also are influenced by other factors such as the GW Multiplier (*Glueware Multiplier)*, or component drivers (see section COTS component factors).

The variable `Concurrence constraint COTS development` determines when the activities of glueware development can start. Figure 3 shows the diagram for the computation of this variable. `Percentage of GW designed needed to development` keeps the percentage of design tasks that need to be accomplished before the implementation or coding phase can be initiated. Knowing the initial size of this phase and the amount of tasks that have been achieved at each moment (`Initial GW Design Size` and `Glueware Designed`, respectively), it is possible to calculate the fraction of glueware that has been designed at a given time. By comparing this value with the parameter that keeps the percentage needed to begin development, it is possible to model the decision of start the following phase. Hence, `Concurrence constraint COTS development` will act as a flag variable that helps to determine the beginning of the coding phase.

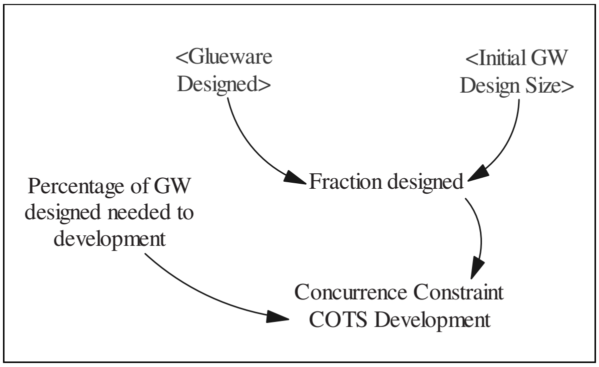
<Glueware Designed>
<Initial GW Design Size>
Percentage of GW designed needed to development
Fraction designed
Concurrence Constraint COTS Development

Fig. 3. Concurrence constraint for COTS development modeling

There is another group of variables that need an explanation now because the ideas underlying their modeling are used in almost every module of this framework. These variables do not have a proper semantic regarding the system under modeling, but are needed to shut down the generation of non-real values in the model. In order to stop activity on task completion, the rates variables are all affected by some other flag variables (`GW design done`, `GW development done`). These flag variables act in a similar way of the variables that model the concurrence constraints in the model. By comparing the estimated size of a task with the current size accomplished, the percentage of achievement is calculated. When this percentage is 100%, these flags are activated to indicate the end of the activity, and hence, the respective development rate falls to a value of zero.

The last level variable GW Dev Accumulated, does not serve to a specific semantic feature of the problem, and it is only maintained as an accumulation variable in order to know, at every moment, the amount of glueware that has been coded. Without this variable, it would not be possible to know how much work has been completed as it flows from a level to the following one (in this case, glueware flows to the system integration phase).

Application Integration

Depending on the type of application, the amount of application development needed to obtain the final product can vary, but it is obvious that first, a process of application development is needed before the integration of the components can start. This module represents the application development and integration processes. Again, like the previous development process described above, it consists of three level variables, one for each phase of the development process, plus an additional level to represent the process of system integration. Figure 4 shows the simplified stock and flow diagram of this module. It can be seen that this module is made of two subsystems: the first one deals with the process of application development itself. It consists of three level variables, as three is the number of phases of the application development process. The second subsystem models the application integration itself, and it consists of one level variable as the process needs only one phase to be executed.

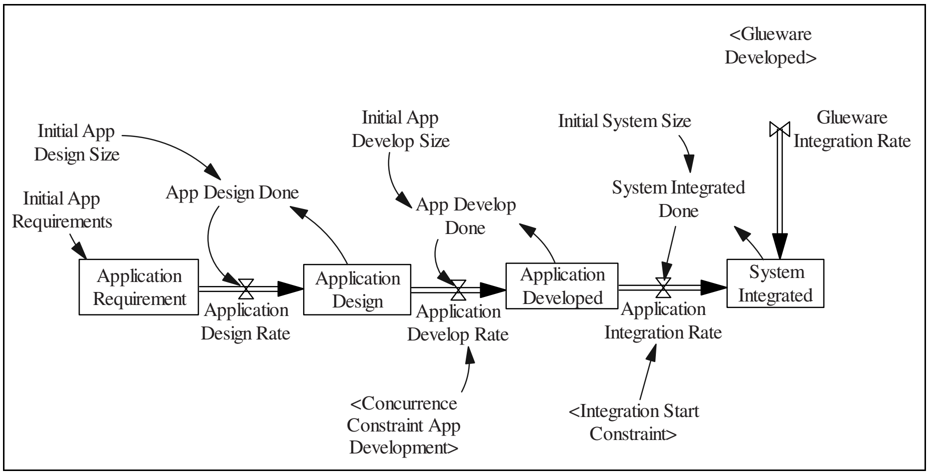

Fig. 4. Simplified stock and flow diagram for Application Integration

The level variables in the model are the following:

- `Application Requirement`: Number of application tasks that represent the application requirements left to be implemented.
- `Application Design`: Number of application tasks that have been designed.
- `Application Developed`: Number of application tasks that have been coded.
- `System Integrated`: Number of application tasks that have been integrated. It consists of application code and glueware.

It is important to notice that the components that control the concurrence constraint for the beginning of the coding phase, and those that control the workflow between whatever two phases are included in this model too. From an abstract point of view, what is done is the instantiation of a generic dynamic meta-constructor that can be used to model a component whenever a task development process is needed to be modeled. This generic meta-constructor does not only help to formalize mathematically the development process, but to effectively implement the principles of modularization and reutilization in System Dynamics. The ultimate representation of this abstraction is the re-engineering of the models using Java™ technology. Using this technology, these constructors have been represented under the concept of interface to define a set of methods and a protocol of behavior that can be then implemented by any class in the class hierarchy.

One important parameter that plays a decisive role in the integration subsystem is `Integration Start Constraint`. This parameter determines the starting point of the integration process based on the glueware that has been developed. For instance, if this parameter is set to 65%, then the integration process will start when the glueware developed has achieved 65% of glueware requirements. Different values in this parameter can have significant effects on the integration process resulting in different quality and costs outcomes. These outcomes can be studied and analyzed using simulation within a non-cost process.

COTS Component Factors

This module represents COTS component factor module. The variable GW multiplier is calculated by multiplying the COTS component drivers suggested by Abts [2]. For the purpose of this study, these factors have been transformed into a set of input parameters of the dynamic model. The parameters take their values from a qualitative domain ranging from very low (point value 0) to very high (point value 5). A brief description of the parameters associated with the COTS component factors follow:

1. COTS Product Documentation: How strongly is effort/productivity affected by the extent to which the COTS product comes with the necessary documentation to install, maintain and use the product? Does the software come with extensive and well written documentation? Or does it come with little documentation?
2. COTS Product Vendor Support: How strongly is effort/productivity affected by the extent to which the vendor offers technical support for the COTS product? Does the vendor provide extensive support for its products? Or no support?

3. COTS Product Ease of Installation: How strongly is effort/productivity affected by the ease or difficulty anticipated to install and integrate the COTS product? Are the interfaces required between the COTS product and the larger system simple or complex?
4. COTS Product Ease of Maintenance or Upgrade: How strongly is effort/productivity affected by the ease or difficulty anticipated to maintain or upgrade the COTS product, particularly after it has been integrated into the larger system? Are upgrades to the COTS product simple to perform, or difficult?
5. COTS Product Ease of Customization: How strongly is effort/productivity affected by the ease or difficulty anticipated to customize or modify the COTS product to make it suitable for use in the larger system if adaptation is necessary? Is customization simple, or difficult?
6. COTS Product Portability: How strongly is effort/productivity affected by the portability of the COTS product across platforms? Is the product easily portable, or difficult to port?
7. COTS Product Ease of Use: How strongly is effort/productivity affected by the ease or difficulty anticipated for the user to operate the COTS product, particularly after it has been integrated into the larger system? Is the product easy, or difficult to use?
8. COTS Product Training: How strongly is effort/productivity affected by the extent of the training the user will require learning to operate the COTS product? Will the user need a lot of training, or little training?
9. COTS Product Dedicated Database: How strongly is effort/productivity affected by the extent to which the COTS product has specialized data needs? Does the product require a specialized database? Or require the population of new elements within an existing database? Or are the product's specialized data needs minimal?

3 First Results of the Simulation

This section shows some of the results obtained with the simulation of the resulting dynamic model that simulates the life cycle illustrated in Figure 1. It is essential to notice here the important lack of real data that organizations have about their own processes. This lack of data often reveals a major problem which is the lack of knowledge and definition for the software process. With the aim of validate and test our modules, we have used data from the literature, mainly those shown in [2].

To show an example of how these models can be used, the effect of different integration starting points on key project variables will be analyzed. The following figures show the sensitivity analysis of the integration start constraint and its effect on the delivery time of the final product (see figure 5), the productivity (see figure 6), and the effort needed to end the project (see figure 7). The results obtained with these simulations can help to determine the most efficient value for this parameter in terms of delivery time, productivity, and cost.

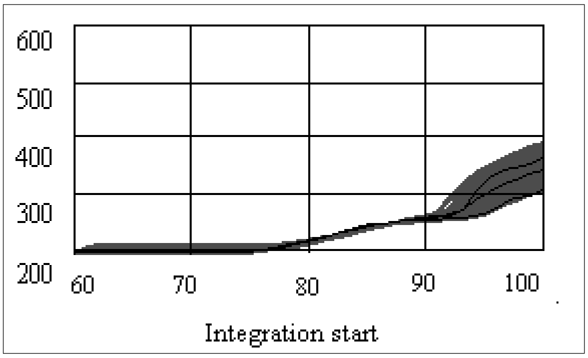

Fig. 5. Sensitivity analysis of the integration start constraint on the delivery time

In order to start the integration phase parts of the glueware code and the application system need to be completed. In the model, the parameter that determines the starting moment of the integration phase is measured as the percentage of the glueware code that needs to be coded before the integration phase can start.

Figure 5 shows different moments to begin the integration phase (expressed in percentage of the glueware coded) and its effect on the delivery time. The simulation outputs suggest that the best delivery time is achieved when the integration phase starts when 60% to 80% of the glueware has been developed.

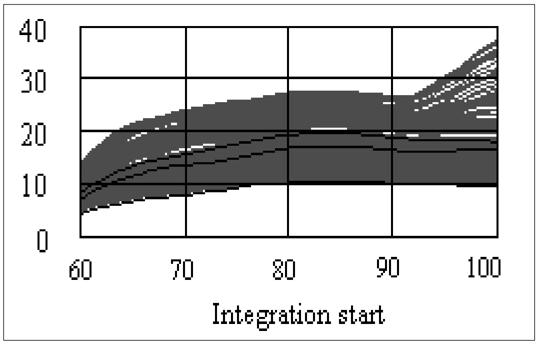

Fig. 6. Sensitivity analysis of the integration start constraint on productivity

Figure 6 shows the influence of the moment at which the integration phase starts on the productivity needed in the project. The simulation outputs suggest that the later the integration process starts, the higher values of productivity need to be achieved to meet the deadline and the objectives of the project.

In Figure 7, the effect of the integration start point on the effort needed to end the project is shown. Best results are again obtained when the integration phase starts when 60% to 80% of the glueware has been developed. For higher values, the effort needed, and therefore the overall costs, grow rapidly.

Sensitivity analysis of the model demonstrates that, from a qualitative point of view, the patterns of behavior shown are consistent with reality and are the ones that one can expect under the scenarios modeled.

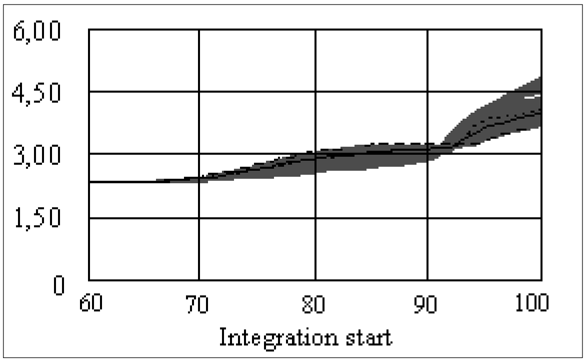

Fig. 7. Sensitivity analysis of the integration start constraint on effort needed

Figure 8 shows the evolution of the global number of tasks developed in each one of the phases of the lifecycle. The number of tasks accomplished shown here is the result of adding the number of tasks accomplished given by each dynamic module that is simulating the specific phase. Note that the figure represents this variable as the percentage of accomplished tasks, showing the temporary evolution of the life cycle in accordance with the sequence and prerequisites restrictions loaded in the inputs of the model.

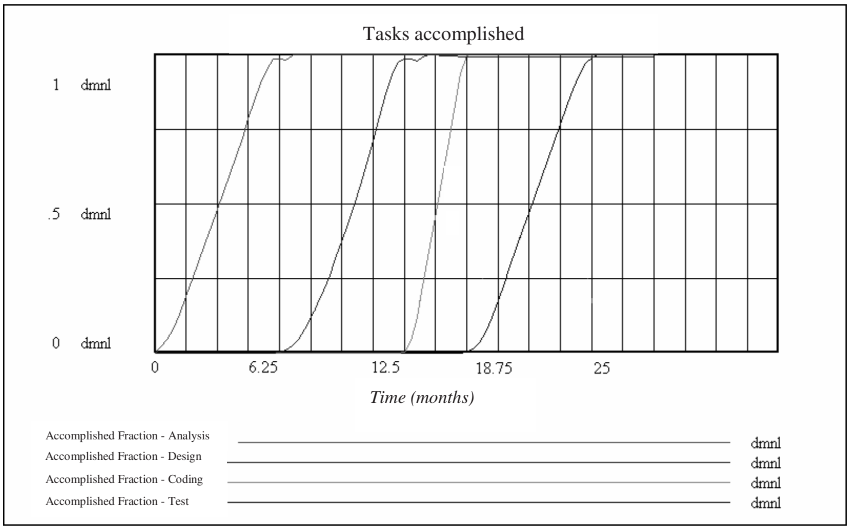

Fig. 8. Tasks accomplished evolution of lifecycle phases

Finally, as it was previously mentioned, all the models within this framework have been re-engineered in order to develop a tool for software process improvement. The new dynamic modules developed for the specific features of COTS have been coded into a family of Java^{TM} classes that inherit and implement the protocol of behavior defined in the framework. As it has been previously said, the tool is intended to asses in the design of software process improvement. The experiment of different improvement initiatives can be done by populating the model with, either, ordinal values for the parameters, or a range of values where the parameters can vary. As a result of running the simulations, a database is fed with data that can be then analyzed to determine the effect of the improvement initiative. One of the techniques that we are currently using to perform this analysis is automatic learning [9].

4 Conclusions and Further Work

In this work, we have presented the first results of a research effort aimed to the development of a set of dynamic modules to model and simulate COTS-based software development process. These modules are then integrated in a dynamic framework that had been previously developed to help organizations design and evaluate process improvement initiatives [12].

The resulting and enhanced framework integrates a set of traditional techniques for software process management, measuring, monitoring and control, with an extensive use of System Dynamics to build models for the software process. It is important to notice that one of the main features of this dynamic framework is that the process of model building triggers itself a metrics collection program [10]. This metrics collection program contributes to a better understanding of the software process carried out in the organization. In addition, the data collected by these programs are useful too to validate and populate the dynamic modules. In the case of COTS development this is very important as the number of parameters or process drivers that have been proposed in literature is high [2].

As it has been previously said, the building approach followed has the features of modularity, abstraction, and reusability. These are features that intend to ease and promote the use of this kind of modeling, which has been proved to be successful in other areas of engineering, in the software industry.

The conceptual ideas have been implemented to develop a tool using VenSim® (design, development, and testing of the dynamic modules) and Java™ technology. The results obtained from the simulation 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 just by changing the values of the parameter(s) required and running new simulations. All the results obtained are saved in a database. This database may then be used to feed some machine learning algorithms in order to automatically obtain management and process improvement rules.

Our future work is mainly concentrated on the full development of dynamic modules to model the formal reviews that take place after each of the phases shown in Figure 1. In addition, although the experiments carried out with the current modules prove that they reproduce the expected behavior from a qualitative point of view, we intend to obtain real data to validate them from a quantitative perspective.

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.

References

1. Abdel-Hamid T. Madnick S., Software Project Dynamics: an Integrated Approach. Prentice-Hall, 1991.
2. Abts CM. Boehm, B., COTS Software Integration Cost Modeling Study. June, 1997 [on-line] (January 2004) <http://sunset.usc.edu/COCOTS/cocots.html>
3. Brooks FP. No silver bullet. Essence and Accident of Software Engineering. IEEE Computer, 20 (4) 10-19, April, 1987.
4. Kim W. Baik, J. Dynamic Model for COTS Glue Code Development and COTS Integration, 1999 [on-line] (January 2004) <http://sunset.usc.edu/classes/cs599_99/projects/COTS.pdf>
5. Morisio M. Sunderhaft N., Commercial-Off-The-Shelf (COTS): A Survey. December, 2002.
6. Paulk M. Garcia S.M. Chrissis M.B. Bush. M., 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)
7. Pope A., The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture. Addison-Wesley, 1998.
8. Raffo D. Spehar G. Nayak U., Generalized Simulation Models: What, Why and How? Proceedings of Software Process Simulation Modeling Workshop, ProSim 2003. Oregon, USA, May 2003.
9. Ramos I. Aguilar J. Riquelme JC. Toro M., A new method for obtaining software project management rules. SQM 2000, Greenwich, UK, June 2000, 149-160.
10. Ruiz M. Ramos I. Toro M., A Dynamic Integrated Framework for Software Process Improvement. Software Quality Journal, Vol. 10, Nº2, 2002, 181-194.
11. Ruiz M. Ramos I. Toro M., An Integrated Framework for Simulation-Based Software Process Improvement. Proceedings of Software Process Simulation Modeling Workshop, ProSim 2003. Oregon, USA, May 2003.
12. Ruiz M. Ramos I. Toro M., Integrating Dynamic Models for CMM-Based Software Process Improvement, in Markku Oivo, Seija Komi-Sirviö (Eds.): Product Focused Software Process Improvement, 4th International Conference, PROFES 2002, Rovaniemi, Finland, December 9-11, 2002, Proceedings. LNCS 2559, Springer-Verlag, 2002,
13. Senge P., The Fifth Discipline. Currency, 1st edition, 1994.
14. Williamson H., XML: The Complete Reference. McGraw-Hill Osborne Media, 1st edition, 2001.