

Integration of Simulation Based Performance Assessment In a Software Development Process

Michael Barth

Institut für Informatik,
Ludwig-Maximilian-Universität München,
Oettingenstr.67, D-80538 München,
Tel.:+49 89 2180 9135, Fax.: +49 89 2180 9152
EMAIL barth@informatik.uni-muenchen.de

Abstract

From the early design phase through the implementation performance assessment of software has been subject to a great variety of approaches in the past. Performance modeling artifacts have to be created and results have to be evaluated. I chose a simulation technique based on a configurable environment simulator and I will show how the development of software in a process can be influenced and improved by intermediate results. This proposal achieves a simple integration of the artifacts in a classical software development process. When performance requirements have to be met, I want to achieve improvement of response time or to save money by terminating the process early enough, if requirements can not be achieved. As the improvement of performance properties can often only be achieved by reimplementing the entire application, performance aspects have to be considered in time. Expenses in modeling can be returned, if results can be derived early in the process. I introduce methods to resolve dependencies and consider dependencies to performance aspects with priority.

Keywords: Performance Modeling, Performance Assessment, Software Development Process, Simulation, UML

1. Introduction

Performance properties often are crucial aspects of a software development project. In many cases only a few parts of the implementation do have to meet critical performance requirements. If at least only one part can be identified as critical, the development process has to be adapted to provide a special treatment. One can distinguish between two main cases. The first case: dynamical

properties of the software shall be optimized. Second case: the project can not be realized at all, if only one part of the software can not meet the requirements. In the last case one risks the software product not to return the investment. The main goal is to choose a process that grants a minimum risk to fail. In both cases a performance assessment has to be carried out as early as possible. As performance properties can only be observed at an entirely implemented software system. However, assessments can be made, if some additional artifacts describing the missing parts have been developed with priority. Dependencies with not yet existing components can also be resolved this way. If a set of concurrent and equivalent design models is available, one may choose a design that allows the best optimization of its development process.

This paper illustrates the influence of typical artifacts on the development process and conditionally created modeling documents on demand. In the following I explain concepts that allow to adapt a classical process to meet certain requirements and an algorithm to design a work flow that grants maximum risk reduction with a minimum of expenses for additional modeling. In section 3 I give a short introduction in the relevant elements of a classical process and the additional artifacts produced by a performance modeling technique based on a configurable environment simulator. Section 4 explains how to integrate the development of several artifacts in a single threaded process and Section 5 deals with the scheduling of several activities in a process, if concurrent development is possible.

2. Related Work

A lot of proposals have been made about performance modeling in the past. Most of them are based on Petri-Nets, queuing networks, stochastic algebras or simulation

techniques. A good overview is given in [9]. Also several proposals have been made to apply performance modeling to the construction of the software development process itself. In [4, 5, 7, 8] performance engineering techniques are applied to the software development process to achieve optimization and suitable tailoring of the process. An incremental methodology for performance validation using the UML has been introduced in [6]. In this proposals mostly a software and a machinery model is used to generate an extended queuing network models (EQNM). In this proposal appropriate activity/job and data models are assigned to the elements of the software model. The models have to be refined through the incremental process. The environment simulator, introduced in [2], is configurable. So the environment model (machinery model) can be refined and recalibrated during the advancing process.

3. Background

A software development that distinguishes between models and implementation is necessarily separated into several phases. The phases' products control the proceeding work flow. I assume hereby the reader is familiar with the RUP [3] and therefore I only give a short introduction. Second exemplary artifacts of the simulation based modeling technique, which I refer to are explained.

3.1. The Classical Software Development Process

The classical software development process as described in [3] consists of several phases. Each phase is composed of one or more iterations. Each iteration is executed in several steps.

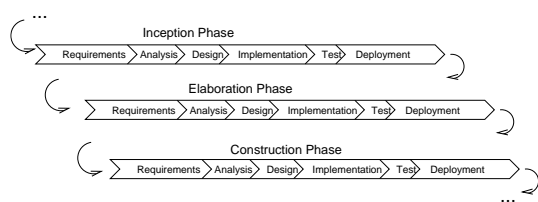


Figure 1. Complete Work Flow

3.1.1. Phases of a Process and an Iteration In Figure 1 some of the phases known from the RUP are depicted schematically. During the inception phase the basic architecture and the structures of the solution are designed. The first basic functionality is implemented during the elaboration phase. This might be treated as a first proof of concept. If the architecture could be validated successfully, the main part of the systems functionality is realized during the

construction phase. Experiences with the software behavior in its final environment lead to minor changes and optimizations of the implementation during the deployment phase. There is a rather complete structural description at the end of the inception phase and we are about to make expenses in implementation after the transition to the elaboration phase.

The iteration steps are depicted schematically as sections of each bar. These steps are executed repeatedly. Typically the analysis and design steps are dominant during the inception phase while the implementation is dominant during the construction.

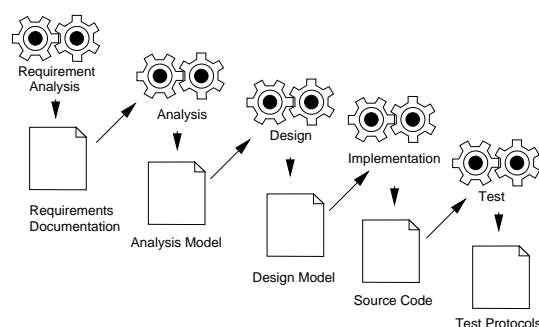


Figure 2. Artifacts

3.1.2. Artifacts Different typical artifacts are produced by each step of an iteration. In Figure 2 the artifacts are assigned to their correspondent iteration step. These artifacts may be created originally during the activity or may be extended as an increment of a preceding document.

The requirement analysis describes the customer's demands in a more formal way. The subsequent analysis model describes an abstract solution, that realizes the demands of the preceding document. This activity is more like a synthesis, but the document only contains a model of *what* a solution should achieve. This type of model may consist of a huge set of UML diagrams. It may be a complete description of the system, but it should not consider *how* the system should be realized. This is the content of the design model. A simulation can only be performed on a model, that describes how a system is finally implemented. Therefore performance assessment requires a design model that does not contain any unresolved dependencies.

3.2. Simulation Based Performance Modeling

Many proposals have been made to derive EQNM, Petri Nets or other models from UML models more or less automatically; see e.g. [11, 14, 15].

The proposal of using a configurable environment simulator allows to introduce some outreaching features. Each

resource can not only be modeled as simple load related delay but as a component with own dynamic properties. The jobs to be executed on a resource are described by the software model annotated with appropriate job classes

These model elements can be calibrated by running probes on corresponding resources. So the accuracy may be improved during the steps of refinement. Also we derive a performance model from the SM using UML activity diagrams. The activity symbols are annotated with appropriate job models describing the properties of the jobs to be executed. These job models can also be refined during the process. The whole methodology is described in [2].

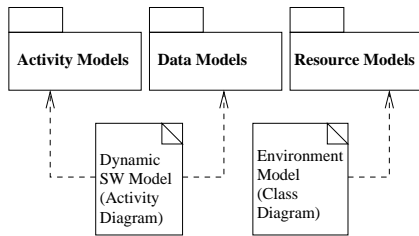


Figure 3. Model elements

3.2.1. Model Elements Dynamical aspects of software functions are modeled with activity diagrams. The control flow is modeled with activity states and transitions. The data flow is modeled also in activity diagrams using Object Flow States. Even though the UML 2 is already published, I adhere to the activity diagrams introduced in the 1.X versions. Most modelers prefer this classical way to use activity diagrams. The merge of control and data flow may cause disadvantages in using algebraic methodologies, but it describes the algorithm as it is and as it has to be transformed to the implementation.

We need activity or job models and data models to create diagrams describing the dynamical behavior. These models are provided in several packages shown in Figure 3. ObjectFlowStates model the data flow in activity diagrams. These states describe data objects, created or consumed by activities using them as parameters. The data package elements can classify ObjectFlowStates. The data models provided in the Data Models package deliberately abstract from the data contents of objects, but rather concentrate on the memory relevant parameters, such as memory consumption and internal structure. Every activity state of our model represents a job that is to be executed by a resource. In general, only activities that consume execution time or space will be annotated with an activity/job model, that can be selected from the Activity Models package. The developer chooses an appropriate model out of this package and defines the name of a resource to execute an instance of this job. The job models

generally contain information that is essential to calculate the processing time.

The modeler wants to describe the environment that will execute the software. The environment is a set of resources, mostly hardware. Type, size, speed and structure of the resources influence the performance properties. The environment description is laid down in an UML class diagram. Each resource is described by an object. The names of these instances correspond with the names used in the dynamic model. The resource model package shown in Figure 3 provides resource model classes. Every object is classified by the appropriate model class.

3.2.2. Scenarios Activity diagrams describe infinite sets of execution sequences and scenarios. A single scenario, an execution path, has to be specified to calculate response time or throughput. A sufficient count of variables used in annotations of the activity graph, like guards or multiplicities e.g., have to be replaced by values. This for example can be achieved using the Tag Value Language (TVL) proposed in [10]. These specifications can be described in a sequence diagram or laid down in several tables bundled to a scenario specification, as proposed in [2].

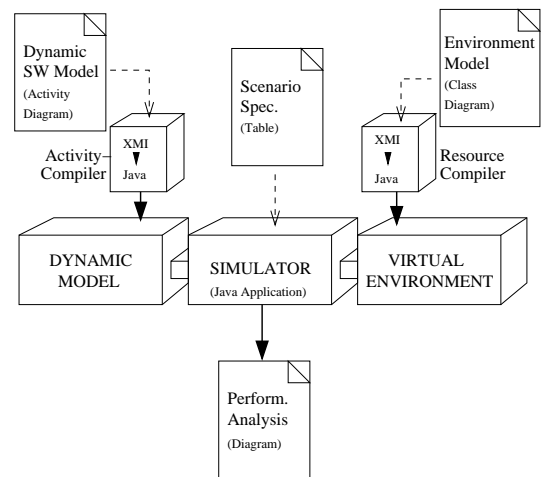


Figure 4. Environment Simulator

3.2.3. Simulation A simulation software as shown in Figure 4 calculates the requested values describing the software's performance properties. Dynamic software models, environment models and scenario specifications are new artifacts in a software development process. The performance analysis is a new type of document that will influence the design and the implementation of the software as well as the process of the software development.

4. Integration of Performance Aspects

In this section I consider how to integrate the creation of the artifacts described in Section 3.2 and how their dependencies are treated during the process.

4.1. Overview

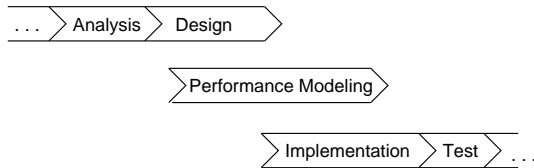


Figure 5. Modified Steps of an Iteration

4.1.1. Integrating PM in an Iteration To improve performance properties or reduce costs or risks the performance modeling should appear in an iteration as early as possible. Performance assessment needs the description *how* a functionality is realized. Therefore, performance modeling can be executed at the earliest subsequent to the completion of the design model as shown in Figure 5.

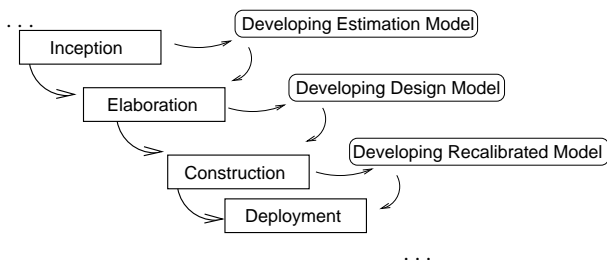


Figure 6. Modified Phases of a Process

4.1.2. Performance Assessment through the Phases If performance is a critical aspect of the software the inspection of this aspect can be inserted at the earliest subsequent to the inception phase, when a complete idea of design is available and no expenses in implementation have already been made.

In Figure 6 this is depicted as *developing an estimation model*. In this early phase only the assignment of simple standard models for the jobs and data as well as for the environment is possible. The analyst has to be experienced to choose appropriate models. Precise design models, real implementations or code samples will be available in the preceding elaboration phase. More accurate models for jobs, data and resources can be selected or developed. However, most of the performance requirements have been investigated during this phase. The performance model can be improved further on. At the end of the construction

phase the complete implementation can be run on the final environment. Accurate performance models can be developed based on performance measurements.

There is a fuzzy border margining the phases. Similar artifacts appear in each phase, containing models with increasing accuracy. But they differ with reference to their sources and their technical realization. These differences are compared in the following.

4.2. Detail

4.2.1. Artifacts of the Design Phases In Figure 7 a part of an iteration is depicted. The iteration starts with the analysis of an informal, mostly textual, description. The performance requirements artifacts have to be derived from this source as well as the first UC analysis. First, a performance requirements document is developed, that contains a description of all requirements that have to be met. Second, an environment analysis is developed, that describes what kind of resources will be available. Third, a scenario and context analysis is developed, that describes all external factors that have influence on the running software and the additional load of the environment. These artifacts depend on the classical use case analysis, because performance requirements are assigned mostly to certain functionality and use cases. The analysis model will be developed as usual. If only the analysis model is available, we know *what* we want to develop. The selection of job, data and resource models are based on the analysis experiences.

A dynamical model first can be derived from the software model, when the complete design is available. Prepared activity and data models have to be assigned. A set of code samples has to be selected or developed to calibrate the environment model later. An environment model can be adjusted according to the estimations of the analyst. Also a context and scenario specification can be developed or derived from the scenario and context requirements independently. If code samples are available the environment model can be calibrated. For each specified context and scenario one test run can be evaluated in the simulator. The results are summarized in a performance analysis document. This document may either help to improve the implementation or terminate the process, if it shows that requirements can not be met.

4.2.2. Artifacts in the Implementation During the Implementation the developed code may replace the code samples of the design phase. In Figure 8 we show how measurements of the entire implementation can be used to improve the performance model's accuracy. The initial calibration of the environment model was executed with a sample of code. The calibration based on the assumption that the implementation will behave analogously to the

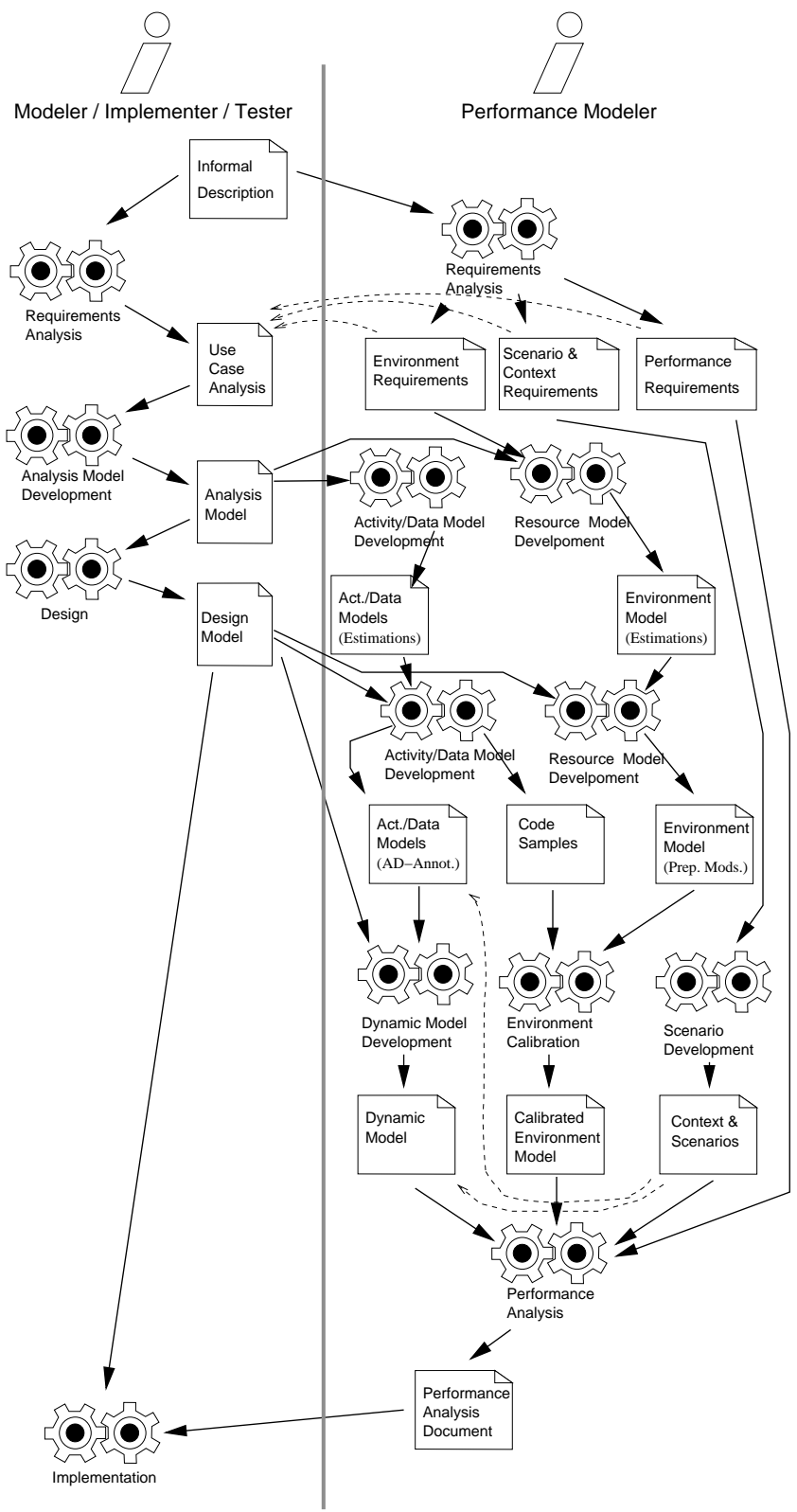


Figure 7. Integrated Process

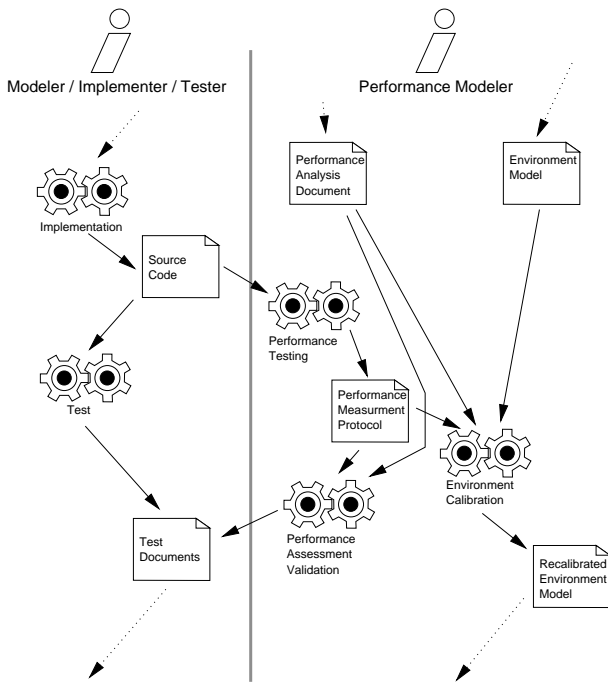


Figure 8. Recalibration During Test

sample of code. Now it can be replaced by a real implementation. This may result in more accurate models. The accuracy can be improved by a repeatedly executed recalibration in each iteration. A final measurement of the entire system will create a model that can be used to optimize the software's function in the deployment phase.

5. Optimization of the Process Design

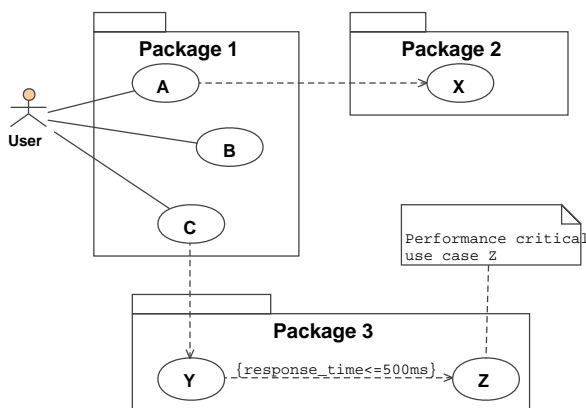


Figure 9. A Design Model Example

In the design model we often find a distribution of functional elements to several packages. In Figure 9 an exem-

plary set of packages is depicted. The systems' use cases are spread over the package structure. Some use cases depend on others. The use case Z in our example is performance critical. The annotated condition signifies this use case is expected to terminate within 500 ms.

5.1. Concurrent Development Process Threads

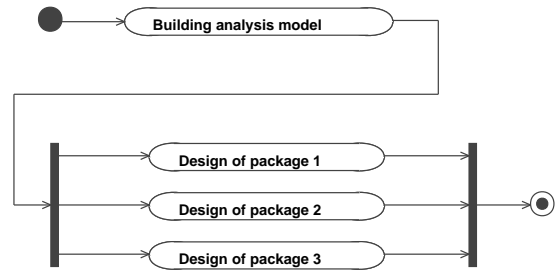


Figure 10. Concurrent Design Threads

Often packages or subsystems can be developed concurrently during a phase. If no performance aspects have to be regarded, the design could be executed concurrently as depicted in Figure 10.

Regarding the performance requirements the use case Z should be considered first. The design can be optimized if no dependencies exist. A subsequent performance assessment may either indicate the clearance for the design of the other packages or may terminate the development process, if the requirements could not be met. Thus wasteful development of models can be avoided.

5.2. Preferred Development Process Threads

In Figure 11 a modified work flow is depicted. Two concurrent sections are shown. In the first section packages with performance constraints are designed. In the subsequent section packages without performance constraints are designed.

Figure 9 shows a dependency between use case Y and Z. Some preconditions of the use case Z may be granted by the use case Y. Perhaps this coherence will cause an additional workload to the executing resource. In this case an additional context model has to be developed. This model contains assumptions of the load that the use case Y represents. As this use case is not modeled yet the load can only be assumed. Once all dependencies are resolved by additional context models the dynamic behavior can be assessed.

In the following section these assumptions have to be regarded. If any assumption is violated the concerning function has to be redesigned to meet the requirements. (If the requirements can not be met at all, the performance

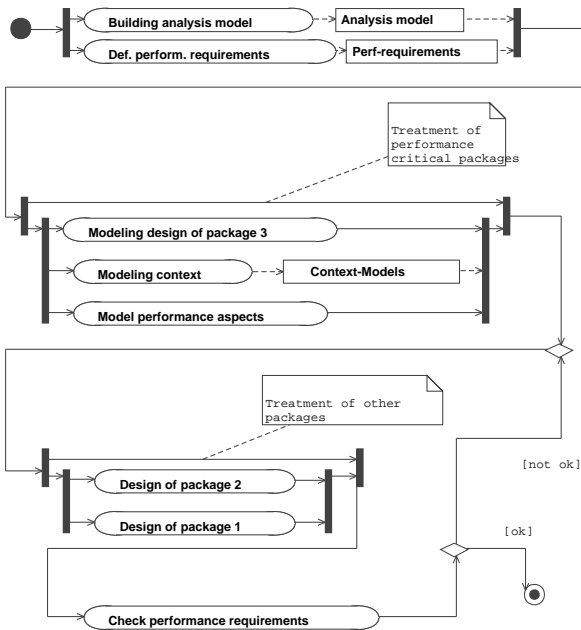


Figure 11. Design Threads with Priority

critical function has to be redesigned. The treatment of this trigger is not depicted here.)

The administration of this process needs a repository to manage all triggers and correspondent model documents. Every model has to be tested if constraints resulting from former assumptions have been violated.

5.3. Process Optimization

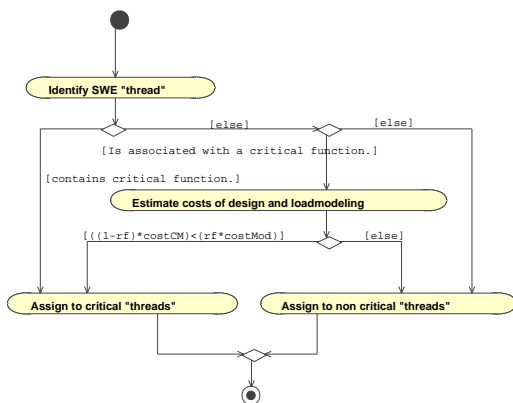


Figure 12. Scheduling Algorithm for Concurrent Process Threads

A concurrent process thread, that should develop a performance critical package or subsystem design can be as-

signed to the corresponding section. However, a concurrent process thread without performance requirements does not necessarily have to be assigned to its correspondent process section. If this package has dependencies to performance critical packages it may cause the development of some additional artifacts (context models), for the performance properties may else not be assessed. These artifacts do not represent a part of the solution. The costs of these artifacts should not exceed the value of a more secure approach. If they exceed the costs the entire design, this package immediately should be developed in this preferred section.

We have to consider the assignment to both process sections. To achieve the right assignment we first need an estimation of the failure probability concerning the performance aspects. Then we have to estimate the costs of designing the whole package and the costs of developing the additional models.

Now we consider the loss in both scenarios. If we meet the requirements, the development of the additional models was in vain. If we do not meet the requirements, we modeled the whole package in vain.

With

- rf : risk of failure
- $costCM$: costs of context development
- $costMod$: costs of the entire design

Additional costs do not exceed the value of a more secure process if:

$$\bullet ((1 - rf) * costCM) < (rf * costMod)$$

This equation can be used as a guard for a scheduling algorithm depicted in Figure 12.

The most problematic parameter is rf , of course. There are a lot of techniques to work out the values of $costCM$ or $costMod$. The correct estimation of the rf parameter depends on an experienced modeler or statistic analysis of the company's experiences in prior projects.

6. Conclusions and Future Work

In this paper I proposed modification of a classical software development process to integrate typical artifacts of a simulation based performance assessment methodology. We achieved a development process design that grants either a secure approach to an entire implementation, that meets all requirements or an early termination of the attempt, if it can not be successfully accomplished.

Equivalent designs and architectures, if they exist, may be selected not only regarding their design quality but also

their capabilities of optimization of the development process.

Some new artifacts have to be created in this process. They either influence the administration of this process or mediate influence the content of the models. Some more experiences have to be made to quantitatively assess expenses of their development. The quantitative reasoning of the process administration has to be improved in the future. However, the minor changes and few additional documents induce a remarkable gain of security, efficiency and quality.

7. References

- [1] OMG, "Unified Modeling Language Specification", September 2001, Version 1.4
- [2] M. Barth, "Performance Assessment of Software Models In a Configurable Environment Simulator", in Proc. of The 2003 International Conference on Software Engineering Research and Practice (2003), CSREA, 2003, ISBN:1-932415-19-X.
- [3] I. Jacobson, G. Booch, J. Rumbaugh, "The Unified Software Development Process", Addison Wesley, 1999
- [4] F. Basanieri, A. Bertolino, E. Marchetti, R. Mirandola, "Automating the Management of Teams and Tasks in Software Multi-projects Using UML and Queuing Networks", In Proc. of the SNPD02, Madrid, Spain, 2002
- [5] A. Bertolino, G. Lombardi, E. Marchetti, R. Mirandola, "Software Performance Measures to Assist Decision Makers within the Rational Unified Process", In Proc. of the IWSM 2002, Magdeburg, Germany, 2002
- [6] V. Cortelessa, R. Mirandola, "PRIMA-UML: a performance validation incremental methodology on early UML diagrams", Science of Computer Programming, Elsevier, Vol.44, 2002
- [7] F. Basanieri, A. Bertolino, E. Marchetti, R. Mirandola, "UML-based Performance Analysis Techniques Applied to Software Multi-projects Management", Int. Journal of Computer and Information Science, IJCS, Vol.4, No.1, March 2003
- [8] A. Bertolino, G. Lombardi, E. Marchetti, R. Mirandola, "Real-Time UML-based Performance Engineering to Aid Manager's Decisions in Multi-Project Planning", In Proc. of the WOSP2002, Rome, Italy, 2002
- [9] S. Balsamo, A. DiMarco, P. Inverardi, M. Simeoni, "Software Performance: state of the art and perspectives", Universita del Aquila/Univ.d.Venezia, Technical Report CS-2002-13, Jan. 2003
- [10] "Response to the OMG RFP for Schedulability, Performance, and Time", OMG, June 2001 docnr. ad/2001-06-14
- [11] S. Balsamo M. Simeoni, "On transforming UML models into performance models", Univ.d. Venezia, WTUML, ETAPS 2001
- [12] L. Williams, C. Smith, "PASA-A Method for the Performance Assessment of Software Architectures", on SW Performance, Rome, 2002
- [13] H. Ammar, V. Cortelessa, A. Ibrahim, "Modeling resources in a UML-based simulative environment", Univ. West Virginia, ACS/IEEE, Beirut, Lebanon, 2001
- [14] S. Balsamo, M. Simeoni, "Deriving Performance Models from Software Architecture Specifications", Univ. d. Venezia Italy, ESM, 2001
- [15] D. Petriu, M. Woodside, "Software Performance Models from System Scenarios in Use Case Maps", Carleton Univ. Canada, Proc. Performance TOOLS 2002, London
- [16] L. Williams, C. Smith, "PASA: An Architectural Approach to Fixing Software Performance Problems", Proc. CMG, Reno, 2002