UML-based Performance Analysis Techniques

Applied to Software Multiprojects Management

F.. Basanieri, A. Bertolino, E. Marchetti[1]
Istituto di Scienza e Tecnologie dell'Informazione
"Alessandro Faedo" , CNR

R. Mirandola[2]
Dip. Informatica, Sistemi e Produzione
Università di Roma TorVergata

Abstract

Performance engineering techniques have been traditionally applied to computer devices and networks and more recently also to software systems. In this paper we propose to use them in an unusual context, i.e., in multiproject software development environment to support manager's decisions. The basic idea is that project teams correspond to the processing elements of a performance model, and project intermediate phases to the tasks to be performed within established time intervals.

The workflows and organization structures are modelled by annotated UML diagrams, so that managers do not need be expert in performance engineering modelling notations. In fact, a tool transforms such diagrams into queueing network models, solving which the predicted completion times for the modelled processes can automatically be obtained. As we use performance analysis techniques, our method can naturally take into account people multitasking on several contemporaneous projects, as well as delays and inefficiencies due to meetings, communications, and personnel overutilization. An example is used to illustrate the approach.

Keywords

Queueing Networks, Project Management, Software Performance Engineering, UML.

1. Introduction

Notwithstanding the emerging new software technologies (e.g., Internet, Mobile code, UML) and paradigms (e.g., COTS, Component-based, Product Families), the classical old problems of personnel management and project planning remain two very critical pieces of the software process puzzle. In this respect, managers have to face today even more difficult problems than in the past, because modern projects are much larger and more complex, while the time-to-market continuously shrinks down to almost unfeasible limits, and system development is increasingly often distributed across remote factories, as well as between people teams with high turnovers.

One consequence of today's software market competitiveness is that project teams are multitasked between a high number of processes. As lively reported by Lister [1], this can well be one of the main reasons why projects are late. When project planning is made, and even more when an unexpected maintenance intervention is required, it is hardly the case that the people to be involved are sitting down waiting for that task: most of them are probably already facing hard-to-meet deadlines in other projects. To make a realistic planning, the manager needs to take into account the current workloads of human resources and take the most appropriate decisions for meeting the project deadlines.

This latter situation is somehow similar to what is routinely done in computer performance engineering. We are taking the metaphor that project teams correspond to the processing elements in performance models, and project intermediate phases are the tasks to be performed within established time intervals. Following this metaphor, the idea we propose in this work is that well known techniques from the software performance analysis field can be usefully adapted to the purpose of handling personnel multitasking and of optimising busy workloads in software project management. This idea actually is not completely new; we provide a survey of related work in the next section.

However, as pointed out in [2], performance modelling techniques used to be "perceived as difficult and time consuming" by software engineers. To overcome this problem, we provided the method with a UML interface. UML [3], [4] is in fact rapidly becoming the standard notation for analysis, design and implementation of Object Oriented systems.

The method to derive queueing networks from UML diagrams was originally proposed by one of the authors in [5]. The method in its original conception uses a subset of the standard UML diagrams, with simple additional annotations: the Use Case Diagram, to derive the user profile and the software scenarios (i.e., the use cases); the Sequence Diagram,

[1] Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo", CNR, Pisa, 56124, Italy, f.basanieri@iei.pi.cnr.it, bertolino@iei.pi.cnr.it, e.marchetti@iei.pi.cnr.it

[2] Dipartimento di Informatica, Sistemi e Produzione Università di Roma TorVergata, Roma, 00179, Italy mirandola@info.uniroma2.it

to derive the software model, and the Deployment Diagram, to derive an hardware platform model and to identify the hardware/software relationships. The method then extracts from these diagrams the main factors affecting system performance and combines them to generate a performance model (see an example of its application to distributed systems in [6]).

Here, we apply that method to solve the several organization and planning problems a manager deals with day-by-day. In particular, the functionalities described in the Use Case Diagram represent the activities planned for product development or maintenance. Each Sequence Diagram represents one of the possible scenarios for a related Use Cases; in particular, the objects in a Sequence Diagram represent the different process steps involved. Finally, in the Deployment Diagram, a node is associated to a team, and the components inside a node represent the tasks assigned to that team.

Indeed, UML is now widely used by many industries in the design phase, to describe all the aspects and functionalities of a software system under development. We believe that the UML expressing power can be applied also to more abstract situations such as the management scenarios. Surely our method requires a slight mind-shift from managers already acquainted with UML, but not the learning of new languages or specialized notations. We are confident that the approach we propose is easily applicable with few additional effort or cost for the software realities already accustomed to UML use.

In the next section, we provide a brief review of related work. In Section 3, we give some basic concepts used in our method, and in Section 4 outline the case study on which the method is experimented. Then, in Section 5, we describe the method, step by step. In Section 6, we illustrate some results. We finally outline conclusions and future work in Section 7.

2. A Brief Literature Survey

A voluminous literature about project management and development exists, but little of it treats the problem of multiprojects planning and of people multitasking on several parallel projects. We report in Section 2.1 a brief survey of previous related studies (we refer to [7] for a more complete review of the literature) and in 2.2 of the most widespread decisional tools.

2.1. Related Studies

Two crucial aspects of project management during development are resources distribution and activity planning. These issues belong to a more general research field that is Concurrent Engineering (CE) [8]. This discipline became popular with the studies of Imai et al. [9] and Takeuchi and Nonaka [10] and has greatly influenced both the academic and the industrial approaches to production. However, these works focus in organizing the tasks within a single project, taking into account the decomposition of a complex product design into smaller activities and their subsequent coordination.

Considering the distribution of resources in a multiprojects environment, which is our study context, PERT (Project Evaluation and Review Technique) and CPM (Critical Path Methods) [11] are probably the first proposed methods. Both refer to an idealized flow of project activities, in which no new project is introduced over time and activity durations are treated as deterministic. Markov chain models [12], [13], which assume an activity time exponentially distributed and use matrix methods for deciding the task time order in development [14], were the natural subsequent evolutions.

The work presented here is close to Adler et al.'s [15]. These authors in fact study the problem of personnel organization and resources distribution among several projects developed in parallel, and like us use queueing networks and stochastic processing network models to represent product development and identify the bottlenecks in task scheduling. The modelling technique proposed is however different and somehow ad hoc. The authors focus on five basic process elements: jobs, tasks, procedure constraints, resources, and flow management control. In particular, a single process may need to handle a variety of job types, which in turn are divided in tasks (i.e., activities or operations). Tasks are connected by precedence relations. The resources are engineers and technicians, who are the units that execute the tasks. The flow management control represents how the resources executed a job's constituent tasks. With reference to [15] Lock [16] identifies a sixth element consisting of the assessment of individual contributions.

More recently queuing theory has been applied to model software maintenance requests [17] and to management planning [18]. Specifically, in the latter case, a queueing based approach for staffing process management and for evaluating service levels is presented. The nodes of a multi-stage, multi-center queueing model are associated to the different maintenance phases. Each stage is considered in series and each entering request goes through a sequence of activities before leaving the system.

2.2. Decisional Tools

Decisional support managers can use generally is of two kinds. One consists in techniques or methods that visualize resources and personnel and distribute them among the phases of project development. Examples are represented by the traditional Control Charts or Gantt

Charts [14], or the more innovative Design Structure Matrix (DSM) [19] which can display the interactions between different teams and process activities.

These methods are extremely intuitive and often supported by tools, but generally the validity of the plans relies strongly on the subjective skill of the managers. Besides, the use of these techniques in a multiprojects context could be rather difficult.

The second kind of decisional support consists of specialized management tools. Microsoft Project [20] or the Kerzner Project Management Maturity Online Assessment tool [21] represent two examples of specific tools: they provide a valid help for maintaining an updated database of the available people and resources, and for producing and visualizing a project plan.

Recently the idea of readapting existing tools for management purposes is acquiring wider spread, also for economic reasons, and some proposals can be found in literature. An example is the work of Dickinson et al. [22], which shows how to use the Dependency Matrix in combination with the existing Portfolio tools to support the decisional process, by analysing the interdependences between projects and combining them together. Another solution is presented in [23], in which the authors propose a tool for production management optimisation that uses Gantt Charts and PERT diagrams for visualizing the obtained results.

However, most existing tools consider only a specific aspect of management, focusing for example either on the completion time or on the personnel distribution and, more importantly, they cannot explicitly manage several contemporaneous projects. Finally, the majority of available tools apply ad hoc algorithms for simulating project evolution, based on some parameter values introduced by the user. Some of those tools generate approximated predictions without any guarantee of statistical significance.

3. Background

In this section we shortly provide some background required to understand the proposed method.

3.1. Performance Concepts Used

To make the paper self-contained, in this section we very briefly introduce some basic performance concepts used in the following. Again, the manager does not need be knowledgeable of these concepts to use the proposed approach, and we only introduce them here for explaining the internal mechanisms of the approach. In particular, we use here the queueing networks models.

Queuing networks are the largest widespread method in the performance field. Anyway the results presented in this paper could be obtained via the application of other approaches like Petri nets [24], LQN or process algebras [25], simply by applying appropriate transformation rules from the UML diagrams to these notations.

Our method is based on the Software Performance Engineering (SPE) approach [26]. The SPE basic concept is the separation of the software model (SM) from its execution environment model (i.e., hardware platform model or machinery model, MM).

The SM captures the essential aspects of software behaviour; we represent it by means of Execution Graphs (EGs). An EG is a graph whose nodes represent software workload components and whose edges represent transfer of control. A software workload component can be a single instruction or a whole procedure, depending on the granularity adopted for the model [26]; this feature makes EGs suitable for modelling software at different levels of detail.

EGs include several types of nodes (or blocks), such as basic, cycle, conditional, fork and join nodes. Each node is weighted by use of a demand vector that represents the resource usage of the node (i.e., the demand for each resource).

The MM models the hardware platform and is based on the Extended Queueing Network Model (EQNM) [27]. To specify an EQNM, we need to define: the components (i.e., service centers), the topology (i.e., the connections among centers) and some relevant parameters (such as job classes, job routing among centers, scheduling discipline at service centers, service demand at service centers). Component and topology specification is performed according to the system description, while parameters specification is obtained from information derived by EGs and from knowledge of resource capabilities.

Once the EQNM is completely specified, it can be analysed by use of classical solution techniques (simulation, analytical technique, hybrid simulation [27]) to obtain performance indices such as the mean network response time or the utilization index (see Section 5).

3.2. UML Diagrams Used

The UML modelling language is rapidly becoming the standard notation for analysis, design and implementation of Object Oriented systems. In the following we recall the main characteristics of only those UML diagrams involved in the applied methodology (for more information see ([3], [4]).

A Use Case Diagram (UCD) (see Fig. 1) provides a functional description of a system, its major scenarios (i.e., use cases) and its external users called actors (an

actor may be a system or a person). It also provides a graphic description of how external users can expect to use the system.

Sequence Diagrams (SDs) (see Fig. 2) show a number of objects and the messages that are passed between them to realize the functionalities described in a specific Use Case. SDs provide specific information about the order in which events occur and can thus provide information about the time required for each activity (this feature is exploited in our method). The behaviour of one Use Case may be given by the combination of a set of SDs.

A Deployment Diagram (DD) (see Fig. 4) shows the configuration of run-time processing elements and the software components, processes and objects that live on them. It is a graph of nodes connected by communication associations. Nodes may contain component instances (indicating that the component lives or runs on the node), and component instances, in turn, may contain objects (indicating that the object is part of the component). The DD can therefore show the mapping of components to processing nodes.

The applied methodology uses the above cited UML diagrams: Use Case, Sequence and Deployment Diagrams. That is to say, a performance model can only be obtained for those systems that are modelled by means of at least these diagrams. Even if this may appear as a limitation of the approach, the additional effort we require in UML formalization will bring beneficial side effects, as we will discuss in the following.

4. Case Study

The case study that we consider to describe the method is an example reflecting a hypothetical (yet realistic, we referred to [28], [29], for its conception) software development environment and the roles of involved people.

The software process life cycle used is a sort of waterfall life cycle organized in three main phases: analysis stage, functional design and main build.

The analysis stage begins with the description of customer needs. A contract is then drafted and iteratively updated following the changes suggested by the customer and taking into account the constraints imposed by project plans, costs, schedule and risks analysis. At this stage, possible alternatives to internal code development, such as the use of COTS or Outsourcing are considered. This part ends with a decision between contract acceptance or rejection.

In the case of contract acceptance, analysts and software architects start a project plan, by which they examine and complete requirements and specifications and start defining test plans.

After this, the functional design phase starts. Here, system and software architectural designs are prepared. Therefore, a top level system architecture, identifying hardware/software components and operator's tasks, is established, as well as its quality characteristics, like performance, environmental conditions, interfaces and security requirements.

At the end of this phase, the software detailed design is transferred to the main build phase. The developers, following the indications of this document, produce the related code and establish test procedures and test cases. In the main build phase, several critical activities may induce delays in project completion or additional costs. For example, problems relative to components integration, or unexpected faults discovered in testing. Their resolution generally requires the manager's intervention.

Each phase described above will be carried on with the support of different teams. In the case study we assumed six teams, representing different working groups that may be involved in the development of different projects going on contemporaneously. These teams are identified as: managers, who lead the organization and control activities; quality assurance team, who is responsible of contract analysis and acceptance as well as of risk analysis and quality evaluation; the analyst and software architect team, who specifies system requirements, the high level design and test plans; the design team, who realizes software and system detailed designs; the development team, who generates code, realizes and integrates system and software components; and finally testers team, who specifies and executes test cases.

The people belonging to a certain team can vary depending on the project exigencies and the number of projects developed at the same time. The latter factor is very important also for predicting the completion time of a project. We will discuss further on this in Section 6.

5. Description of the Methodology

A method to derive a performance model from annotated UML diagrams has been proposed by one of the authors in [5]. The contribution of this paper is the proposal to use that method in a different context. This required us to revise the meaning of all the objects involved in the UML diagrams. However, once this was done, the method itself could be applied virtually unmodified. Indeed, it was really a nice surprise even for ourselves to see how well the method fitted to its usage in this new context, and how well our metaphor of *project steps=computing tasks*, and *processing elements=project teams* worked out.

Let us illustrate the basic steps of the method, tailored to this new applicative context [5], [6]:

1) Functionalities Description

We describe in the UCDs, through the Use Cases, all the functionalities corresponding to project choices or possible real situations occurring during the software product development.

In our case study, the UCD actor (a possible user that interacts with the system through information interchange) is represented by one manager. He supervises all the activities relative to software development or maintenance, and takes the crucial decisions regarding the personnel and the undertaking of new projects (see Fig. 1).

2) Deduce from the UCDs the User Profile

We annotate the arcs in the UCD with the respective frequencies with which the specific activities that descend from each node are expected to occur.

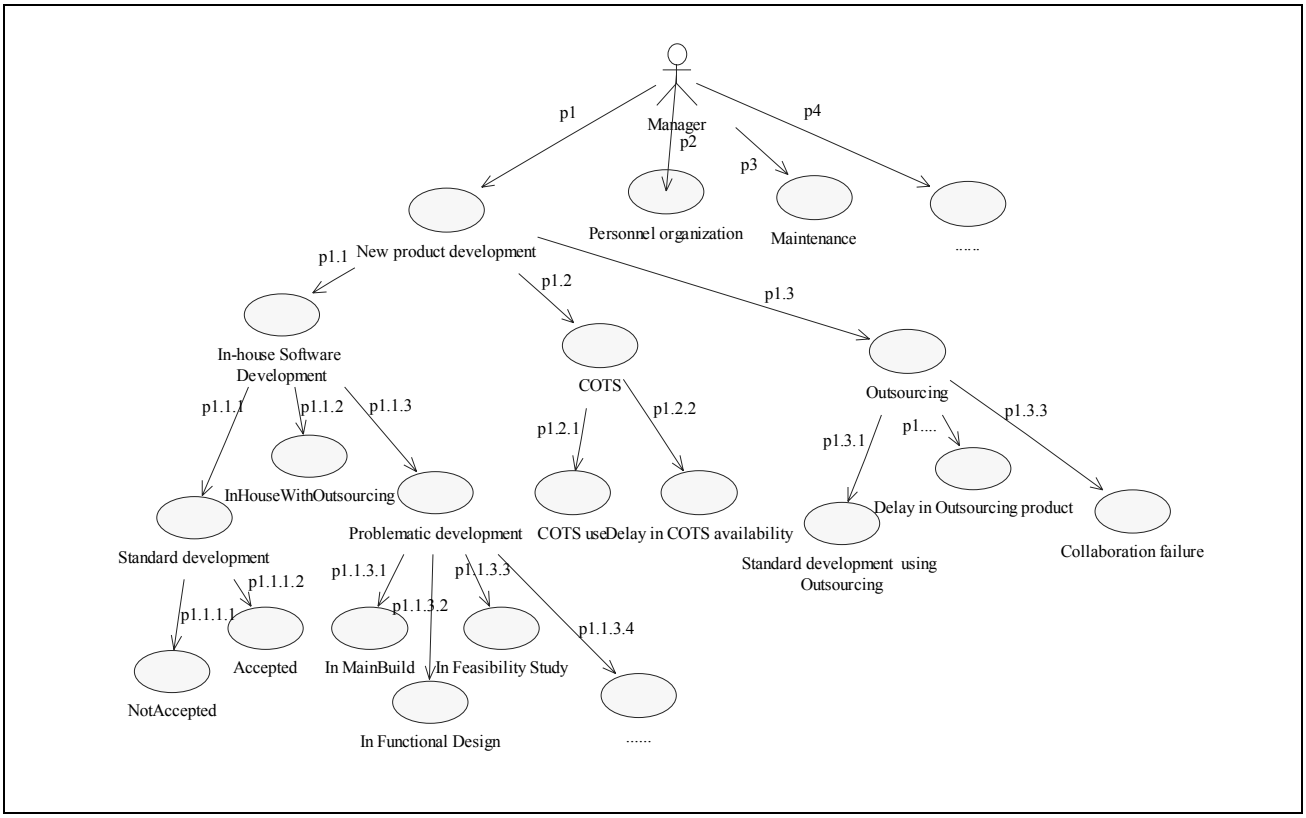

Figure 1: Use Case Diagram

Figure 1 gives an example in which every Use Case at the highest level represents the main manager's activities. The manager deals with several problems with different frequencies. The parametric values, p1, ..., p4, associated with the edges outgoing from the actor "manager" represent the respective portion of working time that the manager dedicates to each of the activities modelled. Each of these high level activities can then be detailed into other, more refined, sub-Use Cases. Also in this case, we specify the frequency of every single scenario. Note that the sum of frequencies of the Use Cases associated with a single actor or node must be equal to 1.

The level of detail down to which the UCD description should be carried on depends on the specific situation and is up to the manager (her) him/self.

3) *For each Use Case in the UCD, generate the corresponding scenarios, and for each scenario the corresponding SDs*

At the end of Use Case modelling, the SDs are derived, each of them representing one of several possible scenarios for the related Use Case. The occurrence frequencies of a SD scenario are given by the product of all the values associated to the edges along the path from the actor to this SD.

We require the manager to annotate each SD with foreseen times. As we show in Figure 2, the sequence of time annotations in each SD simply represents the planned effort for each step in man/months.

Each interaction in the SDs can be identified by the tuple *(l,A1,A2,t),* where *l* is the label of the SD interaction arrow, *A1* is the name of the SD axis from which the arrow starts, and *A2* is the name of the SD axis where the arrow ends and *t* the interaction occurrence time (these labels are used in step 5 below).

4) For each SD, group sets of interactions into higher granularity items

Whatever level of detail the manager adopts, often the SDs derived in step 3) are too complex to allow for the generation of a software model (EG). It is then convenient, whenever possible, to group parts of the diagrams into items of higher granularity.

Even simple grouping criteria could be useful: for example, we could aggregate a set of operations that is repeated in several SDs, or that belongs to a same process phase. By referring to Figure 2, we grouped together the set of interactions from *"Start feasibility study"* to *"Acceptance and completion"*; similarly, the interactions from *"System definition"* to *"Test Planning"*, and so on. At the end of this step, aggregated SDs are obtained.

5) Process the set of Sequence Diagrams (SDs) to obtain the meta-EG.

On the aggregated SDs, the algorithm presented in [5] can now be applied, that translates all the SDs into a high level EG (called meta-EG). Each node in the EG identifies an interaction, and corresponds to the set of operations performed in relation to that interaction. Every node in the meta-EG is labelled with the tuple *(l,A1,A2,t)* that characterizes the translated interaction. Figure 3 illustrates an example of EG generation with proper labels for a very simple SD. The algorithm generates a single EG that models the set of scenarios represented by the SDs and labels the edges with the frequencies introduced in the UML diagrams.

6*) Tailor the meta-EG to the DD, to derive an EG-instance*

The information contained in the DD is used to better specify the obtained software model, the meta-EG, so that it also takes into account its execution environment. The EG thus obtained is called an EG- instance.

As we can see in Figure 4, in our approach a node in the DD is not a hardware resource, but a team. The components inside a node represent the tasks that the people belonging to a team have to perform (obviously, a team can be composed by one or more people). Project phases can be executed with the collaboration of components living inside different nodes of the DD.

Specifically, we substitute the names of the interacting components within the meta-EG block labels with the names of the specific team that accomplishes the operation and the required time (see the next section for an example). Furthermore, when in the label the names of the interacting components are different, an overhead delay due to communications among project teams (e.g., team meeting) is added into the performance model.

In such a way the node label in the EG-instance corresponds to the *demand vector*, that specifies for each team the work-demand relative to the modelled operation, in terms of required man/months.

Figure 5 shows the obtained EG-instance, including the demand vector specification, for the SDs illustrating the In-house scenarios standard development: "*accepted*" and "*not accepted*" (the others SDs are sketched with dotted lines).

7) Use the DD to obtain the Extended Queueing Network Model (EQNM) of the project teams

The EQNM topology can be derived in a straightforward way from the DD. As already stated, in our case the service centers model the project teams involved in the software processes, so the number of service centers in the network correspond to the number of teams. The connections between different service centers are derived from the communications represented in the DD.

In the EQNM corresponding to the DD of Figure 4 is shown. The center "Projects" represent the number of projects that are carried on simultaneously.

The remaining service centers are labelled with the name of the corresponding teams, with a pair of numbers representing the minimum and maximum values of people in the team. Each center is modelled by a multiple server, where the number of active servers corresponds to the actual number of people in the team. The communication delay among teams (e.g., meetings, exchange of documents,...) has been modelled by a service center called Meeting.

8) Combine the EG-instance and the EQNM to derive a complete performance model, and solve the model

Now, by use of well known techniques [26], the EG-instance obtained in Step 6 can be combined with the EQNM defined in Step 7 to achieve the complete definition of the queueing model, precisely as it is done in the classical SPE approach.

The obtained model can now be solved by use of classical solution technique and tools [27] to obtain the

performance indices of interest. We show some examples in the following section.

6. An Example and Preliminary Results

We experimented the proposed methodology on the case study outlined in Section 4. We illustrate here as an example the results relative to applying and solving the model to the Use Case "New product development". With reference to Figure 1, this corresponds to putting in the UCD p1=1, and all the remaining occurrences p2, p3, ... = 0. We consider that in this case the manager must decide among: developing the product using only the internal resources (In-house software development); integrating COTS software for some functionalities; delegating part of product development to another developer (Outsourcing). Each of the three cases, in turn, has been further refined into more sub-cases (Figure 1), requiring finer manager's decisions.

We have developed a set of SDs elaborating the scenarios corresponding to the identified sub-Use Cases. We show as an example in Figure 2 one of the developed SDs: it describes the scenario that big problems are encountered in Main Build phase relatively to the Use Case labelled "In-house software development".

Finally, we constructed the applicable Deployment Diagram, as shown in Figure 4.

On the developed UML diagrams, we applied the method described in the previous section, and we now show the type of results that could be obtained. In general, in order to choose the most convenient solution, managers need to make a trade-off between the foreseen time of completion (TC) and the amount of resources (NR) used. For example, a short completion time, but obtained with a too high employment (or, waste) of resources is certainly not a good choice. Another important factor to consider for the cases of COTS or Outsourcing could also be the cost of software acquisition (CS). Therefore, the manager's decision will be actually a function of these three factors, i.e., f(TC, NR, CS). In our method we make predictions for the factor TC under various situations.

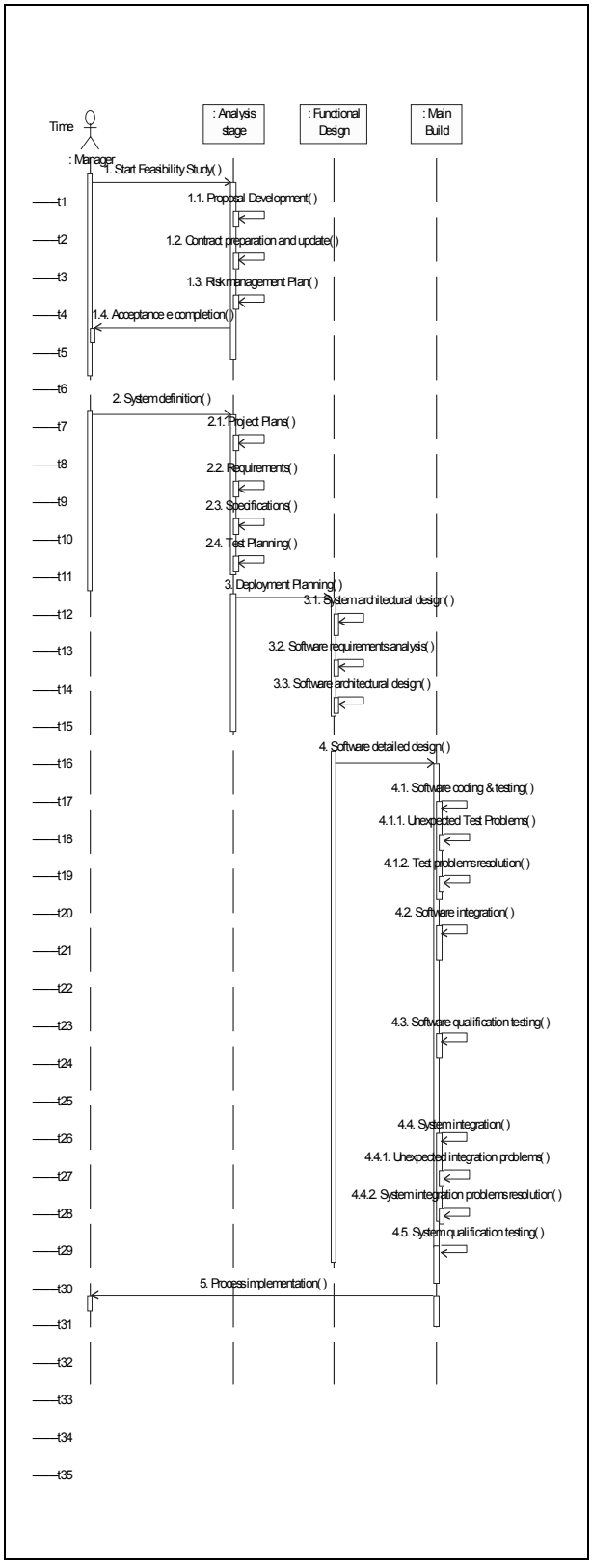

Figure 2: Software Development SD with Problem in Main Build Phase

We took the following assumptions:

A1. all the projects of the same type (i.e., In-house, or COTS, or Outsourcing) take the same number of man/months

A2. the frequencies of problematic scenarios is null, i.e., in practice we reduce here the analysis only to the three main scenarios of successful In-house, COTS and Outsourcing projects

Clearly, such assumptions might see too restrictive, and indeed they are. But they are not required at all to apply the model, we only introduced them to make the analysis simpler.

With reference to assumption A1, in general we consider that using previous experience and acquired knowledge the manager can predict, for every scenario (i.e., in the SDs) the effort required for a project in terms of man/months. In particular, in the example we developed, 43, 36, and 39 man/months are foreseen, respectively, for In-house, COTS and Outsourcing projects.

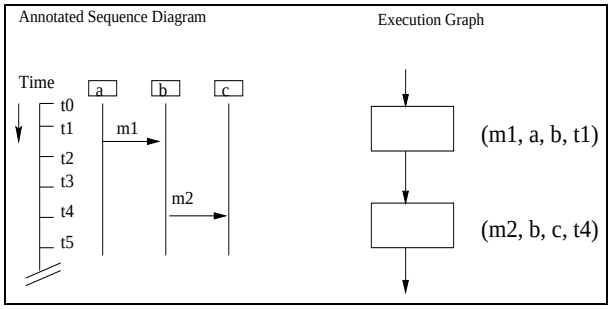

Figure 3 Labelled EG Generation

It is important to notice that such values are the cumulative estimated effort needed to complete all the scheduled activities.

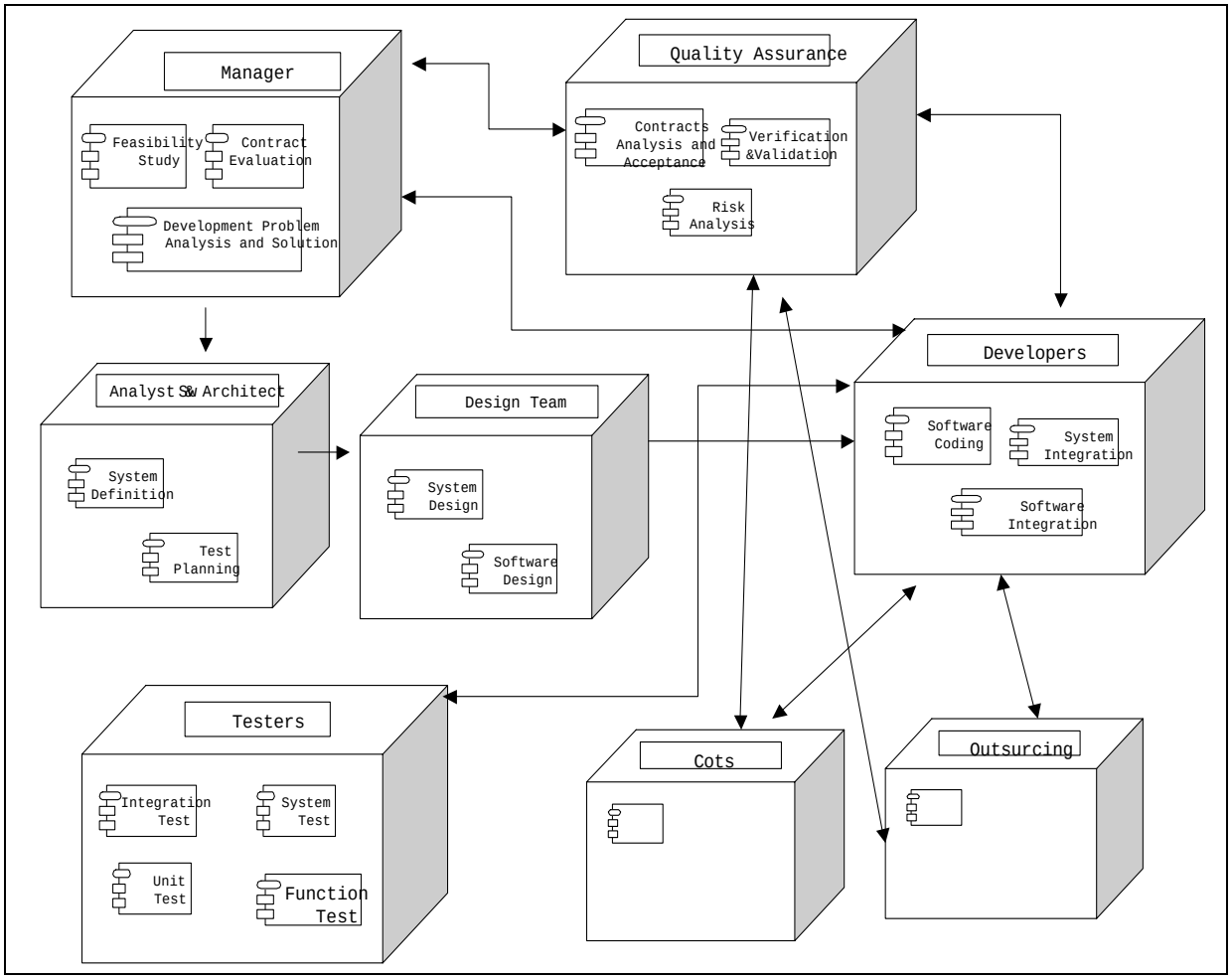

Figure 4: Deployment Diagram

However delays due to meetings, communications, personnel absences, etc, are not yet considered (they will be introduced when translating the SDs in the EQNM model).

In Table 1 some numerical results obtained applying the proposed methodology are shown. In the experiment we varied the following parameters:

1. the number of people belonging to the various teams in the DD. We defined the minimum and the maximum number of people involved in each team.

2. the number of projects that are ongoing at the same time; in Table 1 we show the results relative to assuming 1, 3, 9, or 12 projects contemporaneously under development (as indicated by the first column).

3. the respective frequencies of In-house, COTS or outsourcing (identified by pi, pc, and po, respectively). For the cases of 3, 9, or 12 contemporaneous projects, in Table 1 we show the results obtained for TC considering that all the projects belong to a same typology (i.e., pi=1, or pc=1, or po=1), or that projects typologies are uniformly distributed (pi=pc=po=1/3). This is indicated by the second column.

In the third, fourth, and fifth column, the mean completion time TC obtained respectively for In-house, COTS and Outsourcing is reported: note that it is expressed in real calendar months, because the specific people organization and possible delays are now taken into account. In particular, we report the obtained TC considering the two extreme situations that all the teams have the minimum configurations (white rows), or all have the maximum configurations (grey rows); in practice, mixed configurations could occur.

As we can observe in the table, as the number of projects to be carried on in parallel augments, the average completion time for each project increases considerably, and this is true independently from the kind of development (In-house, COTS, Outsourcing) undertaken. For example, passing from 3 to 9 projects the completion time is more or less doubled in every situation considered.

This effect is due to the creation of queues and delays between the teams: in the configurations that we have hypothesized, some resources, such as the Testers and the Developers, remain lightly utilized, while others, especially the Quality Assurance component, are not sufficient. Such result, i.e., how resources (the teams) are utilized, could also be obtained by solving the model, and is very useful for management. The *utilization index* is measured by the ratio between the frequency at which requests arrive, and the frequency at which the processing element (in our case a team) can deliver services. It varies between 0 and 1, where 1 means that the resource is saturated, and can represent a bottleneck. The results, relative to the typology (pi=1,pc=0,po=0), i.e., In-house, are shown in Table 2. Again, we report the results obtained for the two cases of all teams of minimum size (white rows), or all teams maximum (grey rows).

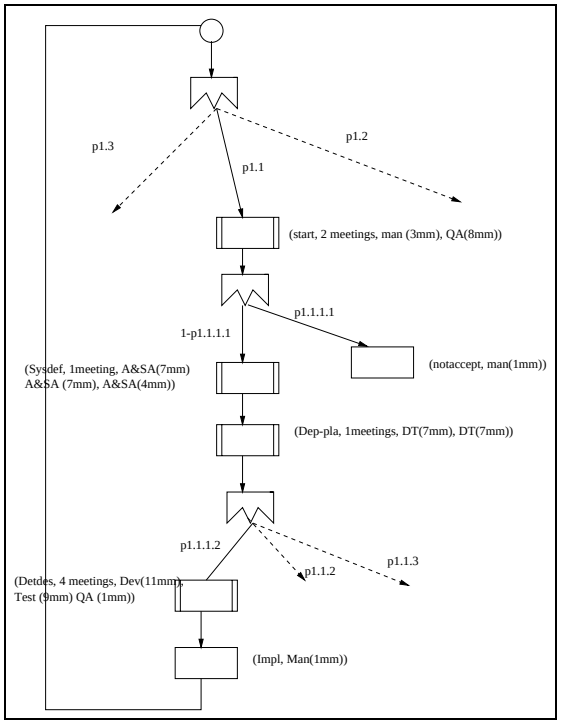

Figure 5: EG-Instance

A conclusion that can be derived in our experiment is that the typology of project undertaken does not make a big difference (at least, under the hypotheses we modelled in the UML diagrams), while the sizing of the involved teams is very important. Using our method, we can automatically predict how the sizing of teams affects the schedule: this can help to balance the assignment of people to tasks as more projects are started, or maintenance interventions are required.

7. Conclusions and Future Work

The contribution of this paper is to show how standard methods from performance analysis literature can be usefully employed for personnel management, whereby the phases of a project are assimilated to the tasks to be performed, and the teams to the processing elements. We illustrated a small case study, and showed how the delays that can accumulate if people are assigned to too many projects can bring to unacceptable completion times. Above all, we showed how queueing networks can be usefully employed for deriving such predictions.

We have used here as the input modelling interface a set of annotated UML diagrams, thus obtaining a method that is easily usable by software managers, in view of the large diffusion of the UML notation.

It is important to notice that the relatively small effort we require in modelling and formalization produces as a beneficial side effect a complete graphical documentation of the software processes in the enterprise, which can be useful both to the manager and the teams involved. The UML diagrams describe, in fact, the tasks and roles of each team and also, in the form of additional annotations, the times foreseen to complete their tasks.

This makes more visible and documented also the updates or the reviews of the planned software process strategies, by which it is possible to identify where, when and how to operate the possible adjustments. Finally, it also helps to keep track of project choices, so that it is possible to evaluate, at any time, whether the adopted methodology, chosen scenarios or decisions were adequate.

At present, the used methodology had to be applied manually, but a tool that performs the automatic derivation of performance models from UML diagrams (according to the method described in Section 5) is currently under development. In particular this tool is based on an extension of the proposed methodology, that we have presented in [30], which uses as the manager's interface a subset of Real-Time UML [31], the recently adopted OMG standard specialized profile for addressing schedulability, performance and timeliness.

The application of the RT-UML profile as the input modelling notation provides in fact the proposed approach with a standard interface, thus reducing further the effort required by the managers for learning the methodology. Therefore, our future vision is that, once a manager has defined and annotated the RT-UML diagrams, the tool will automatically carry out the predictions of interest for the management plan.

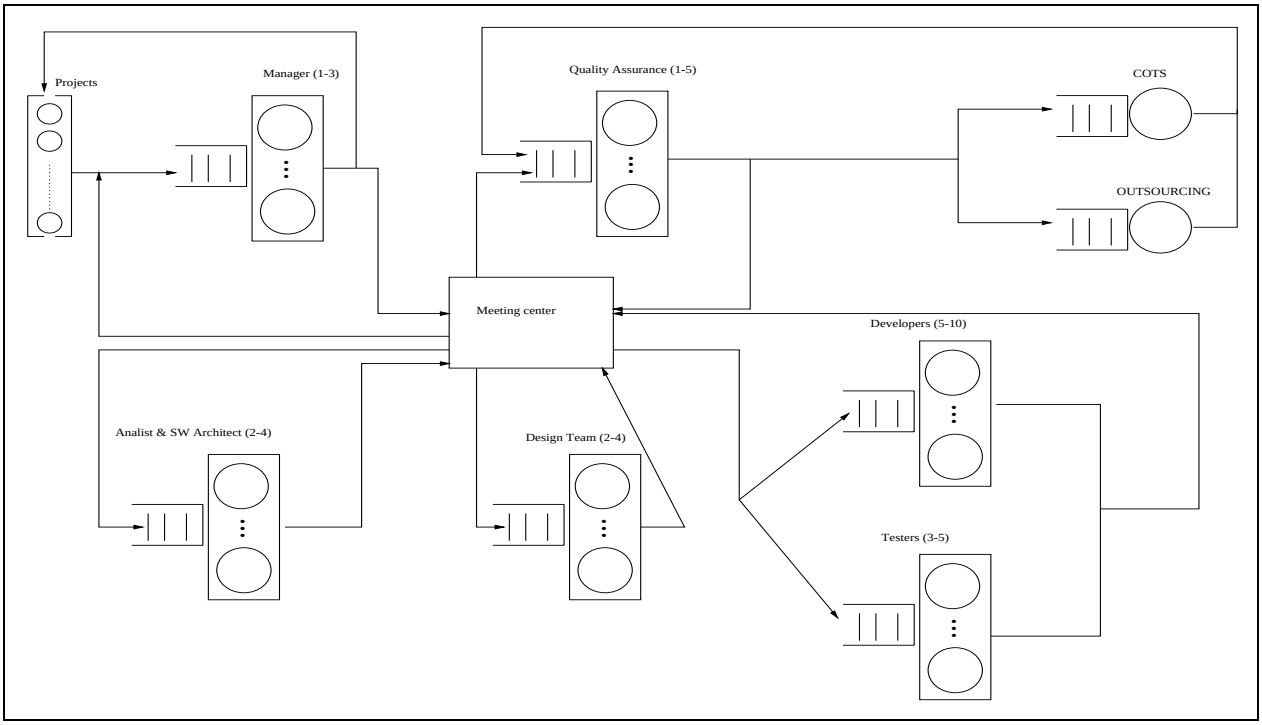

Figure 6: EQNM Corresponding to DD

We are also working towards modelling diverse development processes such as the Rational Unified Process (RUP) [32]. In particular in 0 we present an application of the methodology aimed at augment RUP with the capability to produce reliable schedule and resource utilization estimates of use to decision makers

Future work will also include extending the considered model to encompass more complex situations: for example projects with different dimensions and different priorities, resources with different capabilities and different specializations within the same team. We have also planned the validation of the model on real world industrial case studies.

Projects		(pi,pc,po)	TC-ih	TC-C	TC-O
Min	1	(1,0,0)	42.3		
Max			12.5		
		(0,1,0)		30.7	
				10.4	
		(0,0,1)			34.2
					10.9
	3	(1,0,0)	59.9		
			17.7	0	
		(0,1,0)		53.7	
				15.5	
		(0,0,1)			58.7
					16.9
		(1/3,1/3,1/3)	58.1	53.7	55.3
			17.4	15.6	15.6
	9	(1,0,0)	103.5		
			32.5		
		(0,1,0)		95.4	
				27.4	
		(0,0,1)			106.2
					31.6
		(1/3,1/3,1/3)	95.4	103.5	102.6
			30.9	29.2	28.1
	12	(1,0,0)	133.2		
			35.7		
		(0,1,0)		122.4	
				28.9	
		(0,0,1)			131.4
					34.9
		(1/3,1/3,1/3)	115.9	127.5	127.5
			32.8	29.1	31

Table 1

Projects		Manager	QA	A&SA	DT	Developer	Tester
Min	1	0.09	0.17	0.17	0.10	0.05	0.1
	3	0.23	0.46	0.42	0.26	0.13	0.25
	9	0.41	0.81	0.77	0.47	0.23	0.47
	12	0.43	0.89	0.81	0.48	0.24	0.49

Table 2

References

[1] T. Lister, "Hallucinations at 37,000 Feet", *IEEE Software*, May/June 1998, pp. 105-107.

[2] R. Pooley, "*Software Engineering and Performance: A Roadmap*. *The Future of Software Engineering"*, 22nd ICSE: Finkelstein A. Ed.

[3] J. Rumbaugh, I. Jacobson, and J. Booch, "*The unified Modeling Language Reference Manual",* Addison Wesley, 1999.

[4] UML 1.3 Documentation Web Site. On-line at http://www.rational.com/uml/resources/documentation/index.jsps/

[5] V. Cortellessa, and R. Mirandola, "Deriving a Queueing Network based Performance Model from UML Diagrams", in *Proc. WOSP2000*, Ottawa, Canada, September 2000, pp. 58-70.

[6] R. Mirandola, and V. Cortellessa, "UML based Performance Modeling of Distributed Systems", in *Proc. UML2000,* York, UK, October 2000, LNCS 1939, Springer Verlag, 2000.

[7] V. Krishnan, and K.T. Ulrich, "Product Development Decisions: A Review of the Literature", *Management Science,* Vol. 47, pp. 1-21, 2001.

[8] R. Smith, "The Historical Roots of Concurrent Engineering Fundamentals" *IEEE Transaction on Engineering Management,* Vol. 43, pp. 67-78, 1997.

[9] K. Imai, L. Nonaka and H. Takeuchi, "Managing the New Product Development Process: How the Japanese Companies Learn an Unlearn", in K. B. Clark, R. H. Hayes, and C. Lorenz, (eds.) *The uneasy Alliance*, Boston: Harvard Business School Press, 1985.

[10] H. Takeuchi and L.I. Nonaka, "The New Product Development Game", *Harvard Business Review,* Vol. 64, pp. 137-146, 1986.

[11] B.V. Dean, "*Project Management: Methods and Studies",* Amsterdam: North-Holland, 1985.

[12] V.G. Kulkarni, and V.G. Adlakha, "Markov and Morkov-Regenerative PERT Networks" *Oper. Res.* Vol. 34, pp.769-781, 1986.

[13] G. Weiss, "Stochastic Bounds on Distribution of Optimal Value Function with Application to PERT", *Network Flows and Reliability Oper. Res*., Vol. 36, pp. 595-605, 1986.

[14] T. A. Black, C. H Fine, and E. M. Sachs, "A Method for Systems Design Using Precedence Relationships: An Application to Automotive Brake Systems", M.I.T. Sloan School of Management, Cambridge, MA, Working Paper no. 3208, 1990.

[15] P. S. Adler, A. Mandelbaum, V. Nguyen, and E. Schwerer, "From Project to Process Management: An Empirically Based Framework for Analyzing Product Development Time", *Management Science,* Vol. 42, pp. 458-484, 1995.

[16] C.H. Loch, "Operations Management and Reengineering", *European Management Journal*,Vol.16, pp. 306 – 317, 1998.

[17] R Ramaswamy, "How to staff business critical maintenance projects", *IEEE Software* Vol. 7, pp. 90-95, 2000.

[18] G. Antoniol., G. Casazza., G.A. Di Lucca, M. Di Penta, and F.A. Rago "Queue Theory-Based Approach to Staff Software Maintenance Centers", in *proc. of IEEE International Conference on Software Maintenance, ICSM 2001,* Firenze, Italy, 6-10 November 2001.

[19] T. Browning "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review

and New Direction", *IEEE Transaction on Engineering Management,* Vol. 48, pp. 292-306, 2001.

[20] Microsoft Project tool . On line at http://www.microsoft.com/office/project/

[21] Kerzner Project Management Maturity Online Assessment tool. On line at http://www.iil.com/brochures/kerzner.htm

[22] M. Dickinson, A.C. Thornton, and S. Graves "Technology Portfolio Management: Optimizing Interdependent Projects over Multiple Time Periods", *IEEE Transaction on Engineering Management,* Vol. 48, pp. 518-527, 2001.

[23] S .Bori, J. Lores , R. Pascual, and E. Roures "PROMAN, Planning Production Management and Control System with Intelligent Interface and Advanced Forecast", In *Proc. of ETFA 2001, 8-th IEEE International Conference on Emerging technologies and Factory Automation*, Antibes, France, 15-18 October 2001.

[24] C. Lindemann, "*Performance Modelling with Deterministic and Stochastic Petri Nets"*, John Wiley & Sons, 1998.

[25] U. Herzog, and J. Rolia, "Performance Validation Tools for Software/hardware Systems", *Performance Evaluation*, July 2001.

[26] C.U. Smith, "*Performance Engineering of Software Systems",* Reading, MA: Addison-Wesley, 1990.

[27] S.S. Lavenberg "*Computer Performance Modeling Handbook",* New York: Academic Press,1983.

[28] ISO/IEC 12207: Information Technology – Software Life Cycle Process, 1995.

[29] L.H. Putnam, and W. Mayers "*Measures for Excellence: Reliable Software on Time within Budget",* Englewood Cliffs, New Jersey; Yourdon Press Computing Series, 1992.

[30] A. Bertolino, E. Marchetti, and R. Mirandola "Real-Time UML-based Performance Engineering to Aid Manager's Decisions in Multi-project Planning", in *Proc. Third International Workshop on Software and Performance WOSP 2002,* Rome, Italy, July 24-26, 2002.

[31] B. Selic, "Response to the OMG RFP for Schedulability, Performance and Time", OMG document Ad/2001-06-14.

[32] Rational Unified Process. On line at http://www.rational.com/products/rup/index.jsp

[33] A. Bertolino, G. Lombardi, E. Marchetti, and R. Mirandola "Software Performance Measures to Assist Decision Makers within the Rational Unified Process", in *Proc. 12th International Workshop on Software Measurement IWSM 2002,* Magdeburg, Germany, October 7-9, 2002.

Francesca Basanieri graduated in Computer Science at the University of Pisa. Since 2000, she has been a Grant owner with the Information of Science and Technology Institute of the Italian National Research Council (CNR), in Pisa. She works in Pisatel Laboratory where she is involved in the development of CowSuite tool, to provide an integrated and practical approach to strategic generation and planning of UML-based test suites. Her research interests are in software testing, object-oriented analysis and design. She works with the most spread UML tools (Rational Rose, Rose RT, Argo UML). She is interested in the usage of Queueing Networks theory for automatic management of teams and tasks in software multiprojects. She has (co)authored 10 papers in international conferences.

Antonia Bertolino is a researcher with the Italian National Research Council (CNR) in Pisa, Italy, where she leads the SE group and the Pisatel Laboratory. Her research interests are in software testing and dependability and she investigates cost-effective approaches for architecture and UML-based testing. She is an Associate Editor of the Journal of Systems and Software and IEEE Trans. on Software Engineering. She has been the General Chair of ACM ISSTA 2002 (Rome) and a member of the Program Committees of ISSTA, ESEC-FSE, ICSE, SEKE, Safecomp, and Quality Week. She has served as the Knowledge Area Specialist for Software Testing in the ACM/IEEE Guide to the SWEBOK project. She has (co)authored more than 50 papers in international journals and conferences.

Eda Marchetti graduated cum laude in Computer Science at the University of Pisa in December 1997. She is a PhD Student in Computer Science at the University of Pisa, Italy. Since 1998, she works with Information of Science and Technology Institute "A. Faedo" (ISTI) of the Italian National Research Council (CNR), in Pisa. Her research interests are in software engineering and, in particular software testing, reliability evaluation, application and development of statistical methods and cost weighed strategies to the software test process, use of UML and Queueing Networks for automatic management of teams and tasks in software multiprojects. She has (co)authored more than 15 papers in international journals and conferences.

Raffaela Mirandola received the Laurea degree in Computer Science from the University of Pisa, Italy, in 1989 and the Ph.D. degree in Computer Science from the University of Rome TorVergata, Italy, in 1994. Currently, she holds an Assistant Professor position at University of Rome TorVergata, Italy. Her research interests include software performance engineering, performance and reliability model generation, model analysis techniques, software reliability analysis through statistical techniques, development of methods and tools that can be used to manage the performance and/or the reliability of software throughout the lifecycle. She has(co)authored more than 30 papers in international journals and conferences.