Automating the Management of Teams and Tasks in Software Multiprojects Using UML and Queueing Networks

Francesca Basanieri
Antonia Bertolino
Eda Marchetti
IEI, CNR, Pisa, Italy
f.basanieri@iei.pi.cnr.it
bertolino@iei.pi.cnr.it
e.marchetti@iei.pi.cnr.it

Raffaela Mirandola
Dip. Informatica, Sistemi e Produzione
Università di Roma, TorVergata,Roma, Italy
mirandola@info.uniroma2.it

Abstract

We propose to apply classical performance engineering models for the purposes of personnel management and project scheduling in a multiproject software development environment. The basic idea we draw on 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 tasks and the teams involved are modeled by annotated UML diagrams. A tool transforms such diagrams into queueing network models, solving which the predicted completion times for the modeled processes can be automatically obtained. The methods takes into account people multitasking on several contemporaneous projects, as well as delays and inefficiencies due to meetings, communications, and personnel overutilization.

1. Introduction

Notwithstanding the emergence of new software technologies and paradigms, personnel management and project planning still 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 tend to increase in size and complexity, while the time-to-market continuously shrinks down to almost unfeasible limits.

To keep the development activities under control and make realistic plans, managers need to dynamically consider the workloads of the involved human resources and take the most appropriate decisions for meeting the project deadlines. To support their decisions, in this paper we propose to borrow from the field of computer performance engineering well-known techniques, such as queueing networks models. We follow 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. Accordingly to this metaphor, we adapt performance analysis methods to the purpose of handling personnel multitasking and of optimizing workloads in software project management. The advantage is that performance engineering is a mature field, for which rigorous analysis methods and tools have been developed.

The input requirements to our approach are close to those of existing project management tools, i.e., the manager needs to derive a model of the development process and to make some quantitative estimations, such as the number and the roles of the people involved, the time necessary for completing the different process phases, the resources available and so on.

The peculiarity of our approach is to model the involved activities by using the UML notation, which is becoming the standard notation for analysis, design and implementation of object oriented systems [14], [16]. The approach we propose is easily applicable with few additional effort or cost for the software realities already accustomed to UML use.

In this paper we illustrate the method on a case study. This is an example reflecting a specific software development environment. The referred process is kind-of waterfall life cycle, organized in three main phases: analysis stage, functional design and main build.

Considering the UML notation, we represent in a Use Case Diagram the activities planned for development. By means of the Sequence Diagram we represent one of the possible scenarios for a related Use Case; in particular, the objects in a

Sequence Diagram represent the different process steps involved. Finally, using a Deployment Diagram, we associate a node with a team, and the components inside a node with the tasks assigned to that team.

Using these diagrams and the proposed methodology, we show the type of results that can 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. In the case study considered we introduce also a third factor: 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). We make predictions for the three different factors under various situations. For example we demonstrate how the delays that can accumulate if people are assigned to too many projects can bring to unacceptable completion times.

The paper is structured as follows. 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.

2. A Brief Literature Survey

A voluminous literature about project management can be found in the last years, but little part of it treats the problem of activities planning and people multitasking on several contemporaneous projects. We report here a brief survey of previous related studies (we refer to [8] for a more complete review) and of management decision support tools.

Two crucial aspects of software project management are resources distribution and activity planning during development.

PERT (Project Evaluation and Review Technique) and CPM (Critical Path Methods) [4] are probably the first proposed methods to handle the distribution of resources in a multiproject environment. They describe an idealized flow of project activities, in which no new project is introduced over time and activity times are treated as deterministic. Personnel organization and resources distribution among several projects developed at the same time is instead the problem studied by Adler et al. [1]. These authors use queueing networks and stochastic processing network models to represent product development and to identify which are the bottlenecks in task scheduling.

The decisional support managers can rely on is generally of two kinds. One consists of traditional techniques, like Control Charts or Gannt Charts [2], that visualize resources and personnel and distribute them among the phases of project development. GUI tools oftentimes support these methods, which are extremely intuitive, but generally the validity of the plans depends strictly on the subjective skill of the managers. Besides, the use of these techniques in a multiproject context could be rather difficult.

The second kind of decisional support consists of specialized tools for managers, like Microsoft Project tool [11], or the Kerzner Project Management Maturity Online Assessment tool [7]. These provide a valid help for maintaining an updated database of the available people and resources, and for producing and visualizing a project plan. However, such 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 the project evolution, based on some parameters values introduced by the user. Some of those tools generate approximate predictions without any guarantee of statistical significance.

3. Performance Concepts Used

To make the paper self-contained, in this section we outline some basic performance concepts. In particular, we use here the queueing networks models, which are the largest widespread method in performance field. Anyway the results presented could be obtained via the application of other used approaches, like Petri nets [10], LQN or process algebras [5], simply by applying appropriate transformation rules from the UML diagrams to these notations.

Our method is based on the Software Performance Engineering (SPE) approach [15]. 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 behavior; we represent it by means of Execution Graphs (EG). 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 [15]; this feature makes EGs suitable for modeling 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) [9]. 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 [9]) to obtain performance indices such as the mean network response time or the utilization index (see Section 5).

4. Case Study

The case study that we consider in this paper is an example reflecting a hypothetical (yet realistic, we referred to [6], [13] 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 the 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 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 [2]. The contribution of this paper is the proposal to use that method in a different context, that is for project management. This required us to re-interpret that method in this new application field, for instance by assigning a suitable meaning to 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 fits to its usage in this new context, and how well our metaphor of project steps=computing tasks, and processing elements=project teams works out.

Let us illustrate the basic steps of the method, tailored to this new applicative context [2], [12]:

1) Functionalities description: we describe in the UCDs, through the UseCases, 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 (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 gives an example in which every UseCase 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 modeled. Every 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 UseCases associated with a single actor or node must be equal to 1.

3) For each Use Case in the UCD, generate the corresponding scenarios, and for each scenario the corresponding SDs: at the end of UseCase modelling, the SDs are derived, each of them representing one of several possible scenarios for the related UseCase. The occurrence frequency of a SD scenario is 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 where 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 4 below).

4) Process the set of Sequence Diagrams (SDs) to obtain the meta-EG: on the SDs, the algorithm presented in [2] 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 labeled with the tuple (l,A1,A2,t) that characterizes the translated interaction.

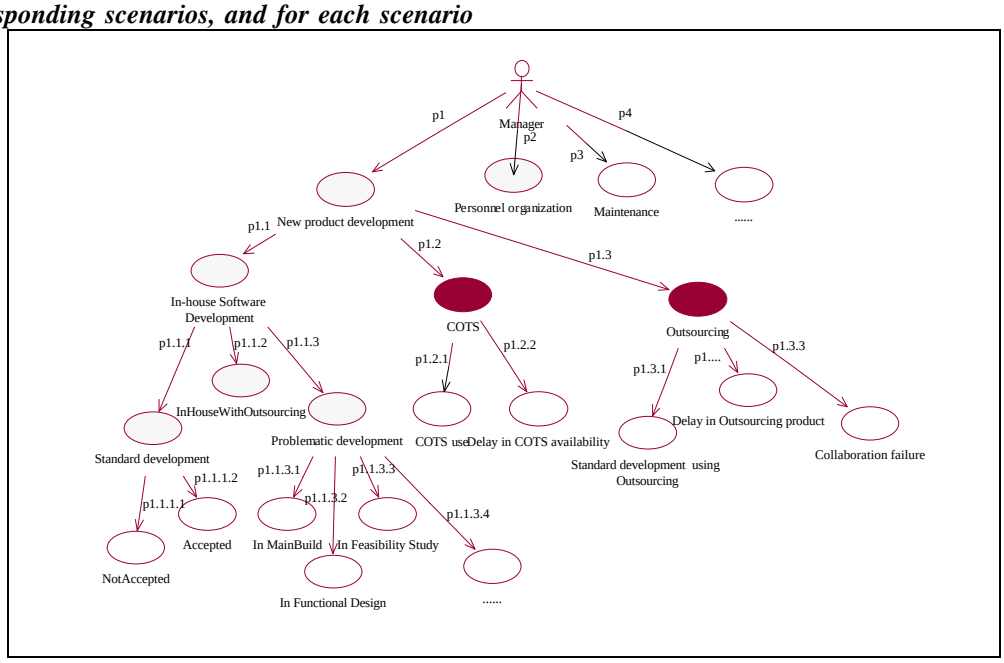

Figure 1: Use Case Diagram

5) Tailor the meta-EG to the DD, thus obtaining 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. A node in the DD here 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.

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

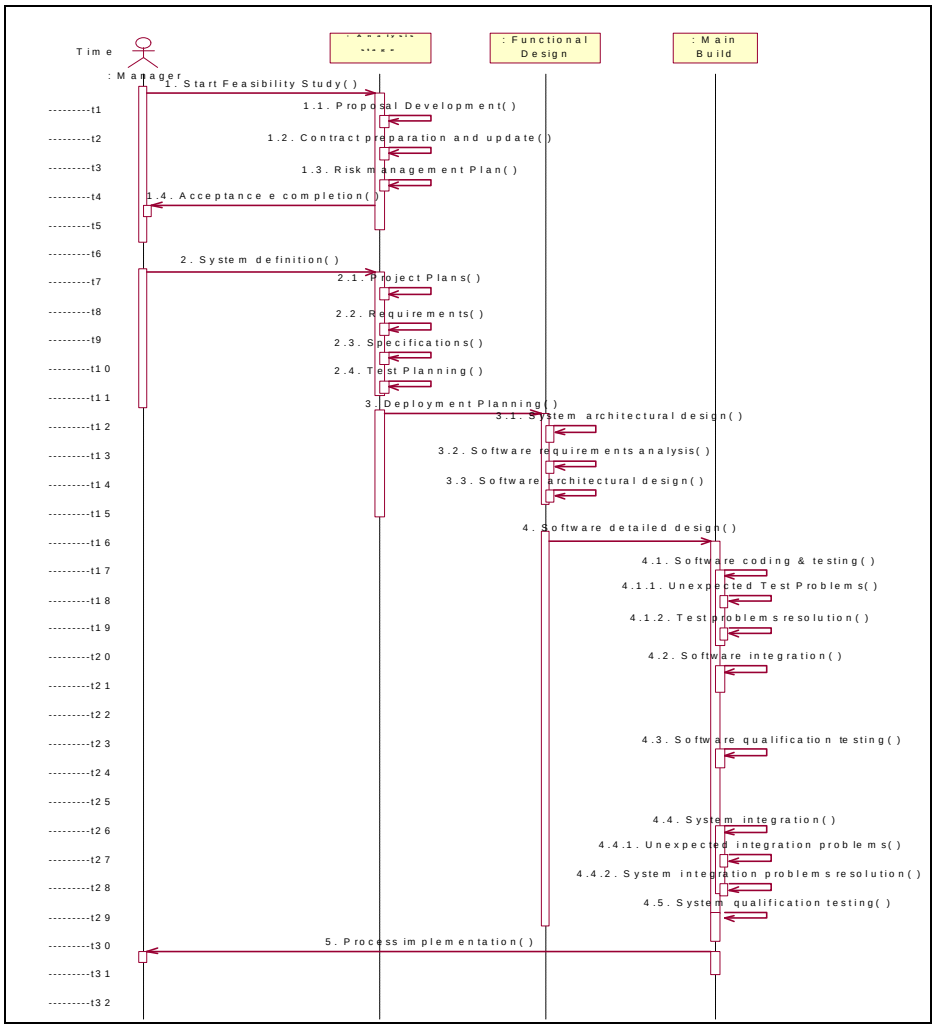

Figure 2: Software development SD with problem in Main Build phase

6) 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 Figure 6 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 labeled 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 modeled 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 modeled by a service center called Meeting.

7) Combine the EG-instance and the EQNM to derive a complete performance model, and solve the model: finally, by use of well known techniques [15], the EG-instance obtained in Step

5 can be combined with the EQNM defined in Step 6 to achieve the complete definition of the queueing model, precisely as in the classical SPE approach. The obtained model can now be solved by use of classical solution technique and tools [9] 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; due to space limitation, in this section we only illustrate 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.

For this case several decisions need to be taken by the manager: 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.

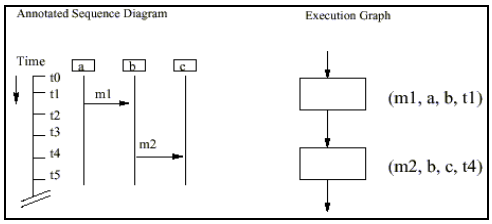

Figure 3 Labeled EG generation

Finally, we constructed the already shown DD (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 (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. 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 are 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.

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

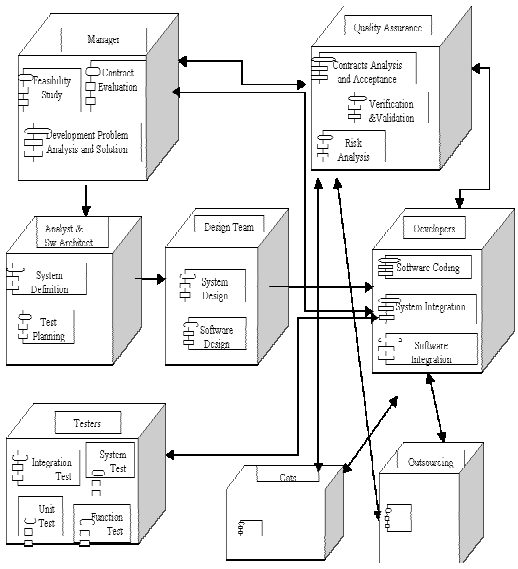

Figure 4: Deployment Diagram

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.

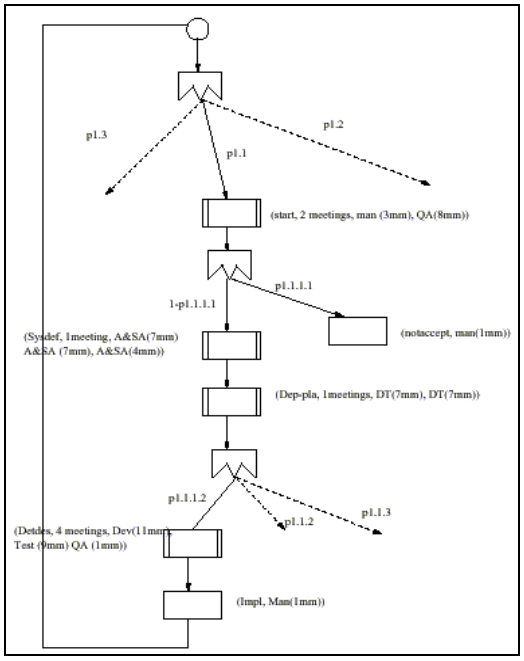

Figure 5: EG-instance

2. the number of projects that are ongoing at the same time; in Table 1 we will 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 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.

A conclusion that can be derived for our experiment is that the typology of project undertaken does not make a big difference (of course under the hypotheses we modeled in the UML diagrams), while the sizing of the involved teams is very important.

Using our method, the consequences on the schedule from the sizing of teams can be predicted: 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, where 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 observing such predictions.

Future work will extend the considered model to include also more general cases, for instance with different values for pi, pc and po. The model will also be further extended to include more complex situations: for example projects with different dimensions and different priorities, resources with different capabilities and with different specialization within the same team. Of course, we have also planned the validation of the model on real world industrial case studies.

It is important to notice that the little effort we require in formalization produces as a beneficial side effect a complete documentation of the software processes in the enterprise, that can be useful both for 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 time foreseen to complete their work.

This makes more visible and documented also the updates or the reviews of the planned software process strategies, because it is possible to identify where, when and how to operate the

modifications. Finally, it also helps to keep track of project choices, so that it is possible to evaluate, at any time, if the adopted methodology, chosen scenarios or decisions were adequate.

At present, the used methodology had to be manually applied, but a tool that performs the automatic derivation of performance model from UML diagrams (according to the method described in Section 5) is currently under development. Therefore, our future vision is that, once a manager has defined the annotated UML diagrams, the tool will automatically carry out the predictions of interest for the management plan.

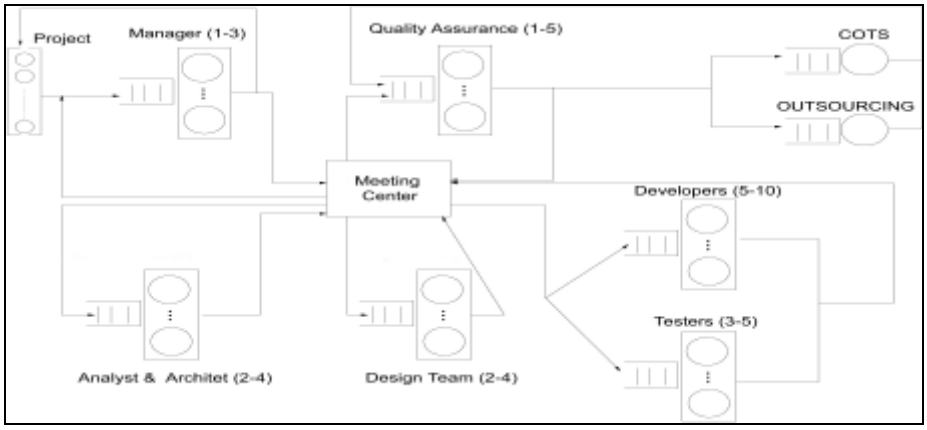

Figure 6: EQNM corresponding to DD

Table 1: Results for the foreseen time to completion

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 2: Results for the utilization of resources

Projects		Manager	QA	A&SA	DT	Developer	Tester
Min	1	0.09	0.17	0.17	0.10	0.05	0.1
Max		0.03	0.03	0.08	0.05	0.14	0.04
	3	0.23	0.46	0.42	0.26	0.13	0.25
		0.09	0.1	0.26	0.15	0.07	0.012
	9	0.41	0.81	0.77	0.47	0.23	0.47
		0.25	0.30	0.70	0.41	0.20	0.34
	12	0.43	0.89	0.81	0.48	0.24	0.49
		0.30	0.36	0.84	0.51	0.25	0.40

References

[1] 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, 1995, pp. 458-484.

[2] A. Burr, and M. Owen, Statistical Method for Software Quality: Using Metrics for Process Improvement. Int. Thomson Computer Press, 1996.

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

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

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

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

[7] http://www.iil.com/brochures/kerzner.htm

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

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

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

[11] http://www.microsoft.com/office/project/

[12] R. Mirandola, and V. Cortellessa, "UML based Performance Modeling of Distributed Systems" *UML2000*, LNCS 1939, Springer Verlag, 2000.

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

[14] J. Rumbaugh, I. Jacobson, J. Booch, *The Unified Modeling Language Reference Manual,* Addison Wesley, 1999.

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

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