# Assessing Staffing Needs for a Software Maintenance Project through Queueing Simulation

Giuliano Antoniol, Aniello Cimitile,
Giuseppe A. Di Lucca, Massimiliano Di Penta
antoniol@ieee.org, cimitile@unisannio.it, dilucca@unisannio.it,
dipenta@unisannio.it
RCOST - Research Centre on Software Technology
University of Sannio, Department of Engineering - Piazza Roma, I-82100
Benevento, Italy

## Abstract

We present an approach based on queueing theory and stochastic simulation to help planning, managing and controlling the project staffing and the resulting service level in distributed multi-phase maintenance processes.

Data from a Y2K massive maintenance intervention on a large COBOL/JCL financial software system were used to simulate and study different service center configurations for a geographically distributed software maintenance project. In particular, a monolithic configuration corresponding to the customer's point-of-view and more fine-grained configurations, accounting for different process phases as well as for rework, were studied.

The queueing theory and stochastic simulation provided a means to assess staffing, evaluate service level, and assess the likelihood to meet the project deadline while executing the project. It turned out to be an effective staffing tool for managers, provided that it is complemented with other project-management tools, in order to prioritize activities, avoid conflicts and check the availability of resources.

## Index Terms

software maintenance staffing, software process management, discrete-event simulation, process modeling, process simulation, queueing theory, schedule estimation

## I. Introduction

Cost and effort estimations are important aspects of the management of software maintenance projects. Software development and maintenance are labor-intensive activities; the project cost is strictly tied to the required effort, i.e., to the project staffi ng. In today's competitive market, a compromise has to be found between the service levels experienced by the customers and the project costs; sometimes, increasing the project staffi ng may not be convenient. Cost/effort estimation is not a one-time activity at project initiation. Initial estimates should be refi ned throughout a project [1], verifying each time the likelihood to meet the established deadline. Thus, it is essential to track the effort spent, and re-estimate staffi ng level throughout the entire project lifespan.

Traditional tools such as PERT, CPM, Gantt diagrams and earned value analysis help to plan and track the project activities. However, they play a little role to assess the likelihood of meeting the project deadline, to staff maintenance projects, to help predicting and tracking the service level as perceived by customers.

We propose an approach based on queueing theory and stochastic simulation to deal with the staffing, the process management and the service level evaluation of cooperative distributed maintenance projects. The method has been validated on data from a massive[1] Y2K maintenance intervention on a large COBOL/JCL financial software system of a European firm. Up to 80 people were assigned to the project, which spanned across four maintenance centers; maintainers were organized into teams; maintenance teams operated both at the customer site and home site. All maintenance centers cooperated under the coordination of a control board. The main responsibility of the control board was to avoid conflicts, assigning to each maintenance center/team a suitable workload.

Queueing theory is a well-consolidated field. Early studies on queueing theory date back to 1900 [3]. The theory has been successfully applied to a large variety of problems: telephone switching/network design, part repair and maintenance center staffing, merchandise distribution and, more generally, service center management. Traditional queueing theory can be adopted to help project staffing, restaffing and service level evaluation at different dates. Incoming maintenance requests may be thought of as customers queueing up at the supermarket checkout counters. The time spent in the queue is related to the customer arrival rate and to the type of service obtained. The perceived service levels, measured as queueing system waiting times, are obviously related to the number of counters, to the distribution of the service times (i.e., the number of *items* each customer acquires) and to the configuration of the service centers. Increasing the number of counters, also called servers, decreases the time spent to obtain a service. However, increasing the number of servers may not be economically convenient: a compromise between costs and customer satisfaction has to be pursued.

Queueing network models representing, at different levels of granularity, the maintenance processes were considered. A monolithic configuration corresponds to the customer's coarse-grained point-of-view, while fine-grained models explicitly account for different maintenance process phases, project distribution across different maintenance centers and rework activities.

In the proposed approach, stochastic simulations of queueing networks permit a fine-grained evaluation of staffing levels at the project startup and close-down when statistical equilibrium

[1]The term "massive" is referred to those maintenance interventions, such as Y2K remediation, Euro conversion or phone numbering change, involving a large number of applications simultaneously [2].

(See equation (1) in Section III-A) is not achieved. The approach also provides a means to assess the likelihood of meeting the project deadline and balancing the workload between maintenance centers while executing the project. Different service center confi gurations may exhibit different performance levels and may increase (without changing the number of servers) the average perceived service level, as well as increase the likelihood of meeting the project deadline. The features described above combined with the possibility to view the maintenance process from a queueing network perspective (thus determining the optimal number of servants for each maintenance phase, as well as considering the effect of phenomena such as abandon or rework), make the proposed approach a valid complement to the previously mentioned mainstream techniques (PERT, CPM, etc.).

This paper extends and complements previous works [4], [5] with a new approach to deal with queueing networks, representing different process phases or maintenance team allocations. Furthermore, by applying stochastic simulation, statistical equilibrium is no longer required, thus allowing an evaluation of staffi ng levels at the project startup and close-down, and perform dynamic restaffi ng. Given the assumptions made (detailed in Section V-A), the proposed approach is immediately applicable to massive maintenance interventions, such as currency or phone numbering change, and easily extensible to more sophisticated maintenance processes.

The remainder of the paper is organized as follows. After an overview on the related work (Section II), basic notions on queueing theory and stochastic simulation are summarized in Section III for the sake of completeness. Then, Section IV introduces the proposed queueing system models. Section V presents the case study description, the research questions to address, the case study methodology and the adopted tools. Case study results are presented in Section VI. The last sections are dedicated to a summary of lessons learned, paper contributions and outline of future works.

## II. Related Work

In the past, several studies attempted to draw relations between product/process metrics/analysis and maintenance process improvement (e.g., [6], [7]) or product intrinsic characteristics (e.g., [8], [9], [10]), aiming at enhancing customer satisfaction and reducing costs. These approaches were more focused on process or product improvement than defi ning/assessing the staffi ng, forecasting maintenance resources and managing costs.

In 1981 and then in 2000, B. W. Boehm [11], [12] proposed deriving the testing effort and the annual maintenance cost from the development effort. The COnstructive COst MOdel (COCOMO) is founded on empirical formulae for software cost estimation; the model used in forward engineering is integrated with the Annual Change Traffic (ACT) parameter to estimate, at the macro level, maintenance costs. Wiener et al. [13] presented a formal model, based on the Rayleigh model, to estimate the needed manpower for projects in the Bank Trust Company domain.

An important issue when facing maintenance requests is that of classifying them to improve planning when/how they will be served to reduce total effort and costs. Briand and Basili [14] proposed a modeling approach, based on statistical techniques, to support the classification task.

Literature reports several works applying *Process Simulation Modeling* to plan, manage and control software life-cycle activities. Kellner et al. [15] describe motivations and purposes of software development process modeling and simulation. They highlight the factors characterizing simulation and classify the main techniques of simulation and modeling. As will be described in Section III, software process simulation is usually carried out by using system dynamics and discrete modeling paradigms. In [16] Martin and Raffo, to overcome some of the disadvantages of the two paradigms, propose a combined model that represents the software development process as a series of discrete process steps executed in a continuously varying project environment.

Issues connected to empirical studies of software process simulation modeling are discussed in [17], with particular reference to the estimate of simulation parameters from real-world data, and to compare actual results to the model's results. The paper discusses several statistical and analytical techniques to choose among various process alternatives, as well as examples of how utility functions, financial measures and data envelopment analysis could be used to evaluate simulation model outputs.

An important contribution to the literature of process simulation has been given by Abdel-Hamid [18], [19], [20]. In [18] Abdel-Hamid published an approach based on system dynamics modeling to simulate the staffing of real world software projects; the model has been used to verify the degree of interchangeability of men and months [21]. Results from the analyzed case study do not fully support Brooks' law. A hybrid estimation model is proposed in [20]: the model exploits the system dynamics simulator of software development presented in [19], coupled with a variety of algorithmic estimators. This approach, according to the author, can be used to allow continuous-time modeling/estimation. In particular, as the project progresses, the model variables can be adjusted in real time to reflect improved knowledge of the management, while at project completion, the model can be used for post-mortem analysis.

In [22] software maintenance management strategies implemented by some organizations are described and compared on the attributes of performance, economics, efficiency and end-product quality. Subsequently, some measurements are defined for each attribute. It is then the responsibility of the manager to choose/define the most suitable subset of attributes that support his/her own strategy.

An important issue about any prediction model is the accuracy of its estimates. Jorgensen [23] reports an experience from the development and use of eleven software maintenance effort prediction models. In this study, the most accurate predictions were obtained by applying models based on multiple regression analysis and on pattern recognition.

Host [24], published a study that shows how discrete-event simulation can be used to explore overload conditions. In particular, a software requirements management process for packaged software systems was analyzed. The simulation model was created according to a previous study of the process and the simulation parameters were estimated from interviews with process experts.

The idea of applying queueing theory to software maintenance was proposed by Papapanagiotakis and Breuer in [25]. They proposed a model for the management of software maintenance based on queueing networks. The model accounted for technical aspects of the software maintenance process, for metrics obtained on the application and, finally, for the organization's resources and requirements. Although they developed a tool to simulate the behavior of a queueing-based process and to support the management, no data about application to real projects was reported.

Queueing theory was recently applied by Ramaswamy [26] to model software maintenance requests. A similar approach, to model a web-centric maintenance center, was described in [5]. By adopting a *fast lane* approach, authors were able to improve maintenance center performances, as experienced by the majority of customers. Commonalities can be found with these two works. However, the research questions addressed here are different. Simulations of a software maintenance process were performed by Podnar and Mikac in [27] with the purpose of evaluating different process strategies rather than staffing the system.

Most of the work described above deals with the problem of cost and effort estimation; the problem of meeting the project deadline is seldom considered. The latter may represent a critical issue for massive maintenance interventions, often addressed over-staffi ng the project (and thus increasing the costs). These issues, with some analyses reported in the present paper (related to the simplest models) were introduced in [4].

## III. Background Notions

In this section background notions about queueing theory, queueing networks and statistical simulations will be summarized. Further details can be found in [3].

### *A. Queueing theory notions*

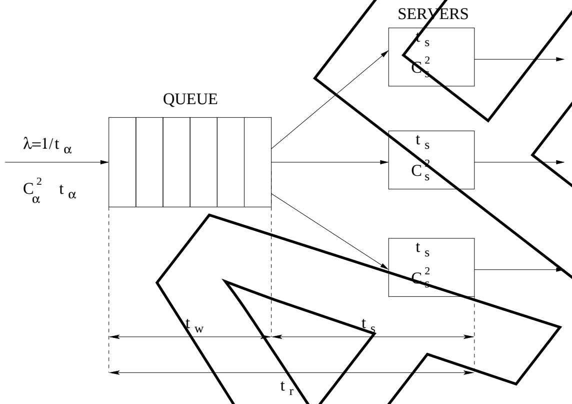

Fig. 1. Queueing model parameters

A queueing system can be described as customers arriving for service, waiting for service if it is not immediate, and leaving the system after being served (by servers). The term *customer* is used in a general sense and does not necessarily imply a human customer (e.g., maintenance requests can be thought of as customers). A queueing system models the steady state of the process, while transient situations are not taken into account. The observable queueing system parameters are:

1) *Arrival traffic rate*: the average rate of customers entering the queueing system, $\lambda$ (and/or the interarrival time $t_\alpha = 1/\lambda$), and, if available, its statistical distribution;
2) *Service time*: the time a server needs to process a request, $t_s$, and, if available, its statistical distribution;

3) *Queue capacity:* finite or infinite; and
4) *Queue discipline:* FIFO, LIFO, random, with priority, etc.

With reference to Fig. 1, a queue bookkeeping may register the distribution and the average values of waiting times ($t_w$), the times spent by customers in the system, (also called response times - $t_r$) and, finally, the average number and distribution of customers in the system and in the queue.

As a shorthand for describing queueing phenomena, a notation has been defined; a queueing system is described by a series of symbols and slashes $A/B/m/a_1/a_2$, where $A$ is the inter-arrival time distribution, $B$ is the service time distribution, $m$ is the number of servers, $a_1$ is the capacity of queue (infinite if omitted), and $a_2$ is the queue discipline (FIFO if omitted).

$A$ and $B$ may be Markovian distributions ($M$), deterministic distributions ($D$), Erlang distributions ($E_k$), or general distributions ($G$). The Erlang distribution is a Gamma distribution with positive integer parameter $k$, and it becomes an exponential distribution if $k = 1$ [3], where $k$ is the number of stages composing the Erlang process.

Often, *a-priori* statistical knowledge of the process is limited; the selection of inter-arrival and service time distribution families may be guided by the square of the coefficient of variation $C$, defined as the ratio of the standard deviation to the mean. Available literature [3] suggests an assumption of deterministic distribution when $C^2 < 0.3$, an Erlang distribution when $0.3 < C^2 < 0.7$, and an exponential distribution when $0.7 < C^2 < 1.3$. Values of $C^2 > 1.3$ require a general distribution.

Note that, if $C^2 < 1$, an exponential distribution may be applied, giving rise to an overly conservative estimate. The case where $C^2 \gg 1$ models cluster arrivals, such as train arrivals. Goodness-of-fit tests are required to support and validate the assumptions about the chosen distribution family.

To measure queueing system performance, several parameters may be considered; for example, the average time spent by a customer waiting for a server (i.e., $t_w$), the average queue length, the average number of busy servers, the server use coefficient (i.e., $\rho$, see below), or the probability $SB$ that all servers are busy. The server use coefficient is related to the system *statistical equilibrium*:

$$\rho = \frac{\lambda t_s}{m} < 1. \tag{1}$$

This means that the chosen number of servers $m$ avoids infinite queue growth. The average time spent by a customer waiting for a server can be evaluated for an $M/M/m$ model using the following equation:

$$t_{w(exp)} = \frac{SB}{m}\frac{t_s}{1-\rho} \tag{2}$$

Equation (2) represents the steady state solution for the $M/M/m$ model, in other words service time distribution is modeled by an exponential distribution ($C^2 = 1$). When $C^2$ departs from the exponential distribution value, more complex and less tractable models known as $M/G/m$ have to be adopted. Pollaczek-Khintchine equation [3] expresses the average time spent by a customer waiting for a server as:

$$t_w = t_{w(exp)}\frac{1+C_s^2}{2}; \tag{3}$$

where $t_{w(exp)}$ is the waiting time as predicted by the $M/M/m$ model and $C_s^2$ is the square of the coefficient of variation of service times.

## *B. Queueing Networks*

Fig. 1 represents a queueing model at a very high level of abstraction: incoming requests are enqueued and processed by one or more servers. More complex models, based on queueing networks [3], are required to obtain detailed information on different process phases.

Queueing network analysis is complex and, in most cases (especially when service times are not exponentially distributed and the model is not trivial), it requires a simulation. However, a preliminary conservative analysis assuming exponential distributions (for all the nodes of the model) is often feasible.

Queueing network parameters are determined by applying *Jackson's theorem* [3]. The theorem states that in a network, composed of $w$ nodes, each one characterized by parameters $N_k$ (number of servers), $t_{sk}$ (average service time, exponentially distributed), and $\lambda_k$ (incoming arrival rate,

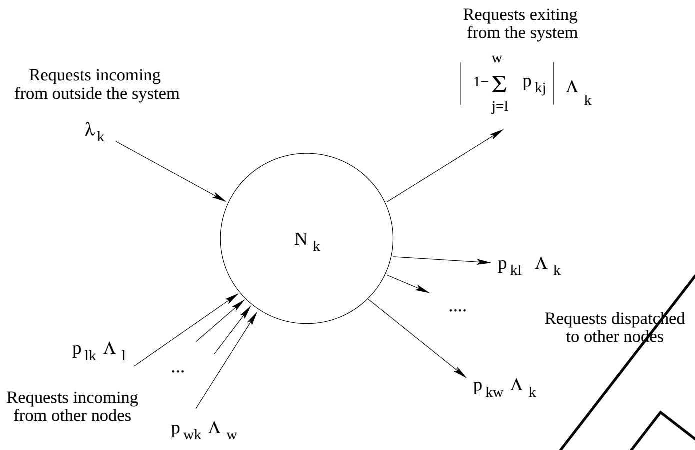
Requests exiting from the system
$\left| 1-\sum_{j=1}^{w} p_{kj} \right| \Lambda_k$
Requests incoming from outside the system
$\lambda_k$
$N_k$
$p_{kl}\ \Lambda_k$
....
Requests dispatched to other nodes
$p_{lk}\ \Lambda_l$
...
Requests incoming from other nodes
$p_{wk}\ \Lambda_w$
$p_{kw}\ \Lambda_k$

Fig. 2. Queueing network node

see Fig. 2), a request may flow to node $j$ ($j = 1 \ldots w$) with probability $p_{kj}$, or may exit from the system with probability:

$$1 - \sum_{j=1}^{w} p_{kj}. \tag{4}$$

This means that, for the $k-st$ node, the *total arrival rate* is

$$\Lambda_k = \lambda_k + \sum_{j=1}^{w} p_{kj}\lambda_j. \tag{5}$$

Therefore, once assigned $\lambda_k$ (traffi c incoming from outside the system), and transition probabilities $p_{kj}$, $\Lambda_k$ can be determined by solving a system of linear equations. It is worth noting that the *Jackson's theorem* applies only if the system is at *statistical equilibrium*: even if, due to the task performed or to different teams' expertise, different servers may exhibit different performances, the number of servers is chosen to ensure *flow balance*: the exit rate of a node is equal to its incoming rate.

## *C. Simulation*

As previously stated, a queueing system only models the steady state. Therefore, to analyze transient and dynamic behavior, a simulation is required. Simulation models non-stationary processes (or processes not following a known distribution), as well as complex dynamics and rules (e.g., reworks, routing problems, etc.).

Simulation attempts to build a model that will mimic a real system for most of its relevant aspects. A simulation may be *deterministic* or *stochastic*. In the first case, the behavior is "point-wise" defined by the output of the model (a point estimate is made for each of the variables of interest) and results are repeatable. Stochastic simulation involves randomness: multiple runs may generate different values. Moreover, a simulation may be *static* or *dynamic*, depending upon whether or not it involves time.

Finally, simulation may be classified as *system dynamics simulation*, *discrete-event simulation* [15], [16] and *state-based simulation* [28]. System dynamics simulation is well-suited to describe the interaction between several project factors, to represent situations such as feedback loops, etc. Discrete-event simulation describes process steps, and is well suited to model queueing systems such as those described in this paper. Stochastic state-based simulation models explicitly capture process-level details, including complex interdependencies among process components.

Discrete-event simulation may be approached in three different ways [29]:

- *Event scheduling:* the idea is to move along the time scale until an event occurs and then, depending on the nature of the event (arrival of a request, finishing of a process phase, etc.), modify the system state, and possibly schedule new events;
- *Activity scanning:* as time passes, the activities are continuously monitored, for the satisfaction of a particular activity, named a *conditional activity* [29]. Such an activity occurs when there are requests in queue and at least one server is idle; or
- *Process-oriented:* the queueing simulator provides three essential components: a source that generates customer arrivals, a queue to hold waiting customers, and a facility to serve customers. Once process-oriented components are available as building blocks, the modeler need only provide process parameters.

This paper addresses the analysis of the dynamic behavior of queueing networks via stochastic, dynamic, discrete-event simulations. The simulator has been implemented (see Section V-E) adopting a process-oriented approach. The choice of this approach was motivated by design considerations, i.e., we decided to create a library of classes representing the basic *building blocks* of the queueing system.

## IV. The Model

To model a maintenance process using queues, different levels of abstraction, corresponding to different granularity and approximation levels can be considered:

- In the simplest model (Fig. 1), maintenance activities and centers are not distinguished; a single queue (i.e., node) models the system;
- At a finer level of detail, different maintenance phases are modeled by different nodes, assigning resources (i.e., servers) to each modeled activity (Fig. 3-a);
- The possibility that a request may leave the system after a certain phase is taken into account (Fig. 3-b); and
- Finally, the possibility of rework among different maintenance phases is considered (Fig. 3-c).

The single-queue model may be regarded as the customer view: requests enter the system and, eventually, a maintenance activity is billed; such a coarse-grained view may be extremely useful to easily assess the overall probability of success with a tractable mathematical model. However, for more complex topologies, stochastic simulation is needed.

Fig. 3-a shows a high level view of a multi-center, multi-stage maintenance process, modeled by a cascade of queues. Each stage may involve several maintenance centers with different numbers of assigned programmers. Maintenance requests enter the system with a constant rate of $\lambda$ requests per day, and are processed sequentially (i.e., FIFO) by one or more nodes[2]. Each node, composed of a queue and one or more servers, represents a phase of the whole maintenance process. There is no limitation on the geographical localization of the servers in that the model abstracts unnecessary details. Communication and coordination activities are accounted for by the estimated model parameters. This corresponds to the software maintenance company internal view: requests enter the system, undergo a sequence of activities, and finally leave the system.

Fig. 3-b shows a model where a request may exit after a certain node of the "chain". Consider, for example, a complex system: there may exist components that do not need any maintenance interventions. Thus, after the components have been analyzed, they leave the system without modification.

[2]If the arrival rate $\lambda$ is not constant, simulation is necessary to study system behavior.

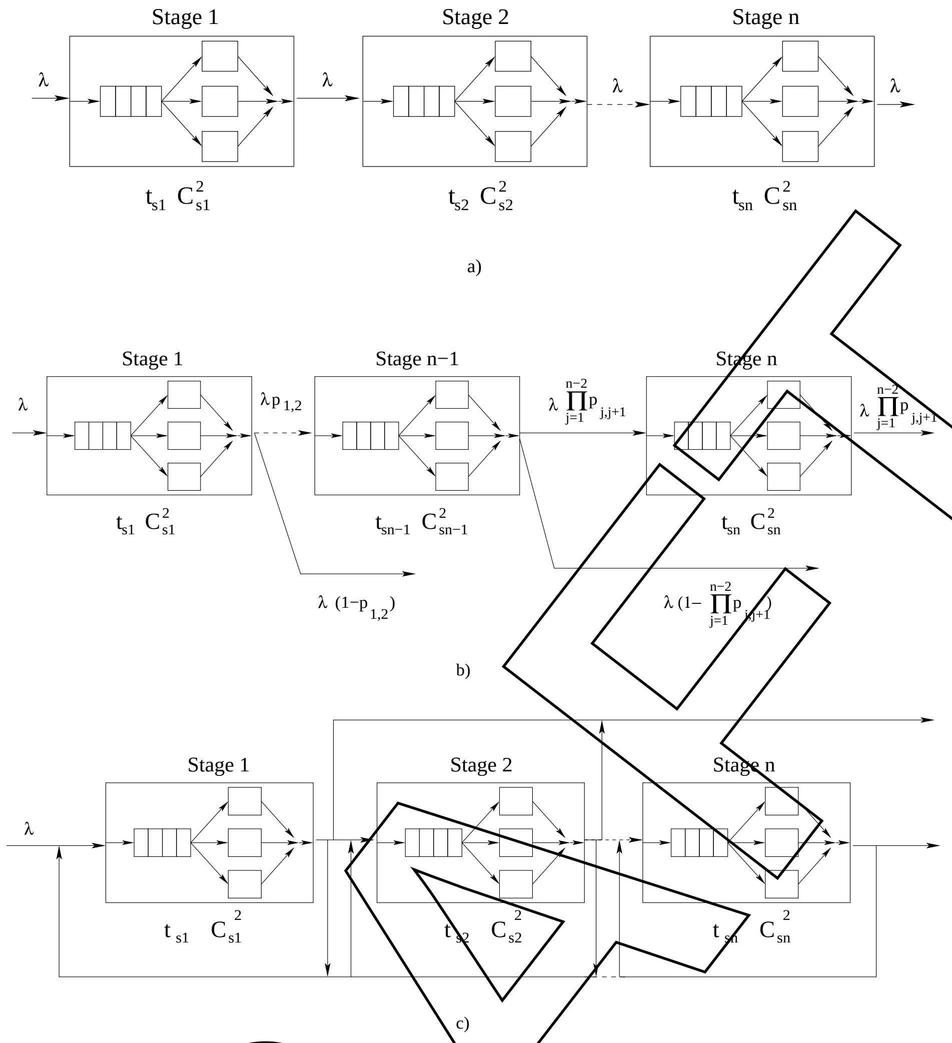

Fig. 3. a) Queueing model of a multi stage, multi center maintenance process; b) Model with exit from the system; c) Model with rework

Finally, Fig. 3-c shows an example of a system in which there is the possibility of rework: some phases may be performed more than once. In a maintenance process, for example, after unit testing, one may realize that it is necessary to re-modify the code. In this case, requests that enter into each node may arrive from the previous node (or from outside the system for the first node) or from one of the subsequent nodes. Arrival rates, as stated in Section III-B and also shown in Fig. 2, have been computed using the Jackson's theorem. For sake of simplicity, formulae have been omitted in Fig. 3-c.

Maintenance centers may be described by a combination of $M/M/m$ models. However, while in the case study reported here, request inter-arrival times may be described by exponential probability distribution, service times sometimes grossly deviate from $M/M/m$ theoretical assumptions. In other words, when $C_s^2 > 1$, the $M/M/m$ model was replaced by an $M/G/m$. In $M/G/m$, the general distribution $G$ accounts for the increase in waiting times due to higher service time variability.

## V. Case Study

The case study reported here is a massive maintenance project, related to fixing the Y2K problem in a large financial software system of a European business firm. The project started on January 2, 1999 and finished (formally) on January 14, 2000. The contract required that all maintenance activities were completed by December 1999.

The project, managed and coordinated by a Production Manager, was organized into three levels:

- *Area*: an aggregation of one or more applications, managed by an Area Manager;
- *Application*: a set of functions related to a particular part of the business, managed by an Application Leader; each application is composed of one or more *Work Packets (WPs)*;
- *(WPs):* loosely coupled, elementary units (from one to nine for each application) subject to maintenance activities; each arriving WP was managed by a WP leader and assigned to a maintenance team (the average team size was about four programmers).

The system was composed of 84 WPs, each one composed, on average, of 300 COBOL and JCL files. These WPs were produced by the *Inventory* phase (see below) from January to the end of July, and all maintenance directly related activities for those WPs ended in October. About 80 people (i.e., maintainers, managers, technicians and secretaries) were involved (not full-time) in the maintenance of these WPs. As detailed in Section V-C, one of the case objectives is to determine the number of people that, working full-time, would allow successful completion of the project within the established deadline.

The project followed a phased maintenance process (similar to what defined in [30]), defined inside the maintenance organization, encompassing five macro-phases:

1) *Inventory*: deals with the decomposition of the application portfolio into independent applications, and successive decomposition of each application into WPs. The activity was

performed in conjunction with domain and application experts. Consistency checks ensured that all the required components were included in each WP. For such a large project, the identification of WPs proceeded incrementally, while processing already identified WPs: this created, as stated in Section VI-A, a Poisson process of incoming maintenance requests. Finally, during the Inventory phase, managers carried out early effort estimation and preliminary project staffing. These activities were performed by applying analogy-based effort estimation [31], using information gathered during the assessment phase on about 50% of the (previously arrived) WPs, and on data available from similar Y2K and Euro projects.

2) *Assessment*: identifies, for each WP, candidate impacted items, using automatic tools. Typically, a starting impact set is identified (i.e., program points directly impacted), and then it is augmented by the set of points impacted by *ripple effects* [32];
3) *Technical Analysis (TA)*: deals with the analysis of impacted items and identifies a candidate solution among a set of pre-defined solution patterns. The identified solution was documented by adding a standard-format comment to the item;
4) *Enactment (Enact)*: an automatic tool was used to apply patches to the problems identified and analyzed in the previous phases. The solution was often based on windowing [33];
5) *Unit Testing (UT)* was performed on each impacted WP; tools were used for automatic generation of test cases. Testing was limited, in the absence of ripple effects, to the code surrounding the impacted point (extracted using program slicing): this technique, called *isolation testing*, reduces testing costs, in that drivers can be constructed from a pre-defined template.

The first two phases were performed on the customer sites by a dedicated team of senior programmers. The team responsibilities included inventory, assessment, workload dispatching and conflict resolution; all the remaining activities were carried out by maintenance teams assigned to four different sites. The phases described above were followed by integration and delivery related activities, which were not subject of our case study. In order to integrate and deliver the system within the contractual deadlines, it was required that all WPs underwent maintenance by the end of October. Conflict resolution and scheduling were under control of the team manager, while the Application Leader handled dependencies among files. Because the inventory was preliminary performed off-line with respect to the project, that activity was not included in the modeled maintenance process.

## *A. Assumptions*

This section summarizes all the assumptions made to model the software maintenance project described above using the queueing models detailed in Section IV.

As stated in Section V-B, this paper assumes requests generated by a Poisson stochastic process (assumption supported by tests detailed in Section V-D.1), an unlimited queue size and a FIFO queue discipline. However, different software maintenance projects may or may not follow the above assumptions: there may be batch arrivals, or different maintenance requests may have different priorities. Details on how queueing models address these factors are beyond the scope of this paper, and can be found in [3]. Limited-capacity queues were not considered because, in the organization subject of our case study, there were no cases in which a request was not accepted (and sent back to the customer or to the previous node) because the queue was full.

As previously highlighted, the maintenance interventions were carried out by applying a standardized procedure; in most cases problem identifi cation, patch application and unit testing were supported by automatic tools. The following assumption were made:

- No particular priority was requested for any WP. The project (except the integration and system testing phases, not considered in our case study) dealt with the application od patches to COBOL and JCL sources affected by the Y2K problem: integration was performed only at the end of the project;
- Because the intervention was highly standardized, different programmer skills and experiences played little (or no) role. The maintenance process was considered as a pool of incoming requests enqueued and dispatched to the fi rst available maintenance team, thereby justifying the FIFO queueing discipline;
- For the same reason, and following in [18], Brooks' law will be assumed not apply in the case study. It was, in fact, assumed that interchangeability between men and months is possible, hence:

$$t_s \simeq \frac{effort}{team\ size} \tag{6}$$

It is well known [21] that this could be an overly optimistic assumption. However, given the team sizes (fewer than eight people) and the maintenance task (see Section V), this approximation was considered reasonable. Furthermore, the model can be generalized to cases for which the Brooks' law does apply, e.g., introducing a non-linearity factor in (6).

## B. Instantiated Models

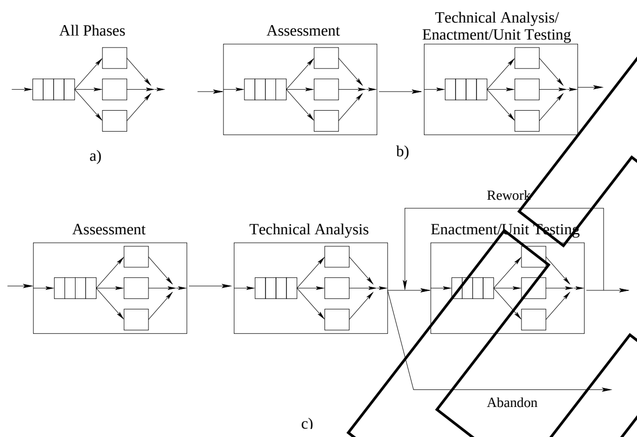

Fig. 4. Queueing network models instantiated in the case study

Models described in Fig. 3 were instantiated according to the adopted maintenance process. In particular, three different models were applied to the process:

- A single-queue system (Fig. 4-a);
- A system composed of a tandem of two queues (Fig. 4-b), one for the Assessment phase and another for all other phases (this model takes into account the geographical distribution of the project); and
- A system composed of three nodes (Fig. 4-c), one for Assessment, one for Technical Analysis and another for Enactment and Unit Testing. It is worth noting that not all the requests go beyond the Technical Analysis phase, and there may be a percentage of rework on Enactment and Unit Testing.

Though project data revealed a negligible level of rework, investigation on the effects of different rework levels is presented in Section VI-C.

## C. Research Questions

Queueing system basic parameters can be either estimated by analogy on past projects or derived from the log of an ongoing activity. However, as time passes, available information increases, and refined estimates can be obtained. In other words, the research questions addressed in this paper are to some extent *time dependent*, and different milestones were established to track project evolution. The following research questions were investigated:

1) How can the collected information on ongoing activities be used to readjust project staffing? (Sections V-D.1 and V-D.2, see results in Table IV)
2) For a given staffing level, how accurate are the $t_r$ estimates obtained by different models, e.g., single node model compared to a system explicitly modeling the different phases? (Section V-D.2, results are shown in Fig. 6-b, 7 and 8)
3) How does the likelihood of meeting the project deadline change as the project progresses? (Section V-D.3, results are plotted in Fig. 9)
4) How do rework and abandonment affect the results? (Rework effects are shown in Section VI-C, while abandonment effects are shown, for the three-queue models, in Table IV and discussed in Section VI-B) And, finally
5) May a periodic restaffing reduce personnel costs? (Sections V-D.5, results are discussed in Section VI-E)

### *D. The Method*

An approach inspired by the leave-one-out cross-validation procedure [34] and by time series analysis [35] was used to measure model performances. Collected data points were considered as a time series with a past, a present and a future. The first $p$ (past) points, the *training set*, were used to train the model while the remaining $n - p$ (future) points, *the test set*, were used to assess model accuracy.

As sketched in Fig. 5, each check was articulated in the following steps:

1) Data distribution analysis and model parameters estimation;
2) Project staffing;
3) Likelihood of meeting the project deadline estimation; and
4) *Test set* model assessment.

In addition, simulation was performed (see Section V-D.5) to analyze the behavior of the model in case of periodical re-staffing.

*1) Data Distribution Analysis and Estimation of Model Parameters:* In order to verify that case study data (interarrival times and service times) follow a statistical pattern, two tasks were carried

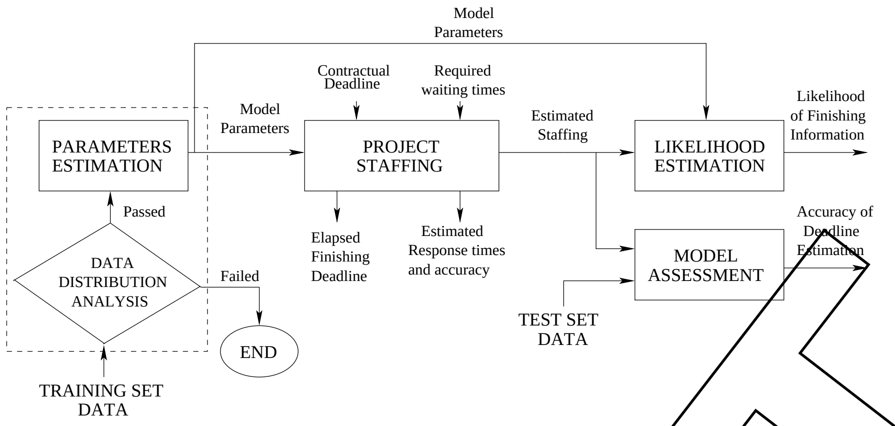

Fig. 5. Information fbw among the method's steps

out. First, histograms of the data were plotted and their shapes analyzed to select a candidate distribution. Then, the Kolmogorov-Smirnov (KS) goodness-of-fi t test [36] and the Chi-square goodness-of-fi t test [37] were performed, to assess the similarity of the data distribution with the candidate distribution.

The KS test computes the largest distance $D_n$ between a candidate distribution function $\hat{F}(x)$ and the distribution function $F_n(x)$ computed from the data. The test (i.e., the null hypothesis that the data fi t the expected distribution) is rejected if $D_n > d_{n,1-\alpha}$ ($d_{n,1-\alpha}$ values, said *critical KS test values*, are tabulated [36]).

The Chi-squared test measures the error between a candidate distribution's density function and the data histogram. The test (i.e., the same null hypothesis as the KS test) is rejected if $\chi^2 > \chi^2_{k-1,\alpha}$, where $\chi^2$ is the Chi-squared statistic and $\chi^2_{k-1,\alpha}$ is the Chi-squared percent-point function with $k-1$ degrees of freedom and signifi cance level $\alpha$.

To select the family of distributions, tests on the interarrival time distribution and on the service time distributions were performed, for the different nodes of the queueing system. It is worth noting that the $M/G/m$ model does not require any hypothesis on service time distribution.

Once it has been verifi ed that the incoming data follow a well-known statistical pattern, the interarrival rate $\lambda$ was estimated using the Maximum Likelihood Estimator (MLE). It can be shown that, for exponential distributions, the MLE is equal to the arithmetic mean. Service time descriptive statistics (mean and standard deviation) were computed over a *training set*: according to [3], no particular estimator was needed in this case.

*2) Project staffing :* Once parameters were estimated, stochastic simulation was used to compute team sizes (for the different nodes of each model) under the constraint to complete maintenance activities, on average, within a given date. Several simulations were carried out with increasing team size for each node of the model, until all the expected WPs were processed by the chosen deadline. The number of servers (maintenance teams[3] for each node), was chosen sufficiently *high* to ensure that adding further resources did not substantially modify the simulated project life-span.

Project staffing levels are then refined to reach a compromise between personnel cost and waiting time. If the number of teams *per* node (servers) is too small then this may produce system instability (the request arrival rate is greater than request dispatch rate), while increasing the number of teams increases the project cost.

Response times evaluated from *training set* parameters were compared with those evaluated from the same model, with $\lambda$ and service times of the *test set*, to measure the error occurred estimating the response times.

*3) Likelihood to meet the Project Deadline Estimation:* The project staffing level cannot be considered without taking into account the risks related to an excessive slack time and the related contractual penalties. The massive maintenance project was devoted to modify mission/business critical software: remediation costs were not considered the principal issue.

The likelihood of meeting the project deadline was estimated by numerical stochastic simulation based on the refined staffing level (in that it guarantees small response times and the statistical equilibrium of the system). For the expected workload and any given possible project time-to-finish, in a range of interest around the required project deadline, iterative simulations (the number of simulations was doubled each time, until the results for two successive iterations matched) were performed to compute the likelihood as the ratio:

$$\frac{\#\, of\ simulations\ finished\ within\ the\ deadline}{Total\ \#\ of\ simulations\ performed}$$

*4) Test Set Model Assessment:* All of the above activities were carried out on the model built by means of a simulation on *training set* parameters. Model assessment was performed on the

[3]From this point the terms servers and teams will be used indifferently.

actual arrival dates and service times (test set). In other words, a deterministic simulation was executed with WPs belonging to the *test set*.

*5) Analysis of Transient and Adaptive Restaffing:* The main drawback of the process described above is that the staffing level is constant across the entire project, therefore:

- Project initial and final transients are not handled: although at the beginning and at the end of the project, the number of WPs to process concurrently is considerably lower, the staffing level considered is the same of the steady state; and
- The possibility of an "adaptive staffing" is not considered. It may be useful to restaff the project whenever the rate of arrivals and the average service times exceed certain thresholds.

To study the effect of periodical restaffing, new simulations were performed as follows:

1) Once a small number of WPs (say 10), necessary for a preliminary estimate of model parameters arrived, the project was staffed to meet the deadline. In particular, given a deadline $d$, the average finishing date of the simulation $\hat{d}$, and an acceptable delay (positive or negative) $t$, a simulation is considered successful if

$$|\hat{d} - d| \leq t \tag{7}$$

2) Periodically (say every 10 days), parameters are re-estimated on data collected from arrived WPs, simulation was executed, and project staffing accordingly modified (if needed).

In the case study reported here, the delay $t$ was chosen equal to 10 days, corresponding to two weeks.

### *E. Tool Support*

To train queueing models (i.e., to estimate model parameters), to simulate system evolutions and compute the likelihood presented in Section VI, two different tools were used:

1) A queueing system simulator; and
2) A tool for computing queueing theory parameters.

The *queueing system simulator* was developed to support the determination of the team size required to complete the project within a given deadline. In the meantime, the likelihood of meeting the deadline is obtained. The software allows the execution of both *deterministic* and *stochastic* simulations. In the latter case, interarrival times and service times are randomly generated, and the simulation is iterated for a sufficiently high number of times so that results obtained do not significantly vary between experiments. For *deterministic* simulations, interarrival times and service times are read from files.

The *queueing system simulator* consists of a library of C++ classes, implementing objects of a queueing system:

- **Sources of service requests:** An ASCII data file (used for *deterministic* simulations: requests are read from a file, containing the arrival dates of the requests) or random sources (used for *stochastic* simulations: requests are produced with an exponential distribution);
- **Queueing nodes:** composed of a queue and one or more servers, that serve requests enqueued from a source. Service times may be both read from a file or randomly produced with an exponential distribution (if the distribution of service times is general, then the Pollaczek-Khintchine formula is used to approximate response times); and
- **Dispatchers:** used for queueing networks, this system randomly decides (given a transition probability matrix) to which nodes a request goes.

The *queueing-theory parameters computing tool* is a Perl script that, given a set of parameters $\lambda$, $t_s$, $C_s^2$ and number of servers $m$, computes the queueing model relevant parameters introduced in Section III (e.g., waiting time, server use coefficient, probability that all servers are busy, etc.). Moreover, it also handles queueing networks, given the matrix of $p_{ij}$, that indicates the probability a request has to move from node $i$ to node $j$.

## VI. Case Study Results

The method described in Section V-D has been applied to the empirical data obtained by monitoring the project described in Section V. Data were initially scrutinized with a senior manager to assess data quality and to avoid duplication. The project plan, assigned by the customer, was assumed as time reference for any further activity. According to the project milestones, a target finishing date was fixed at the end of September. This allowed to minimize the risks of finishing after the established deadline, i.e., the end of October.

The experiments described in this section were performed as follows:

1) The entire data set (composed of arrival date and effort for all 84 WPs) was split, at different dates, into a *training set* (WPs arrived before the considered date) and a *test set*

| Date Set | Distribution | KS Test | | | Chi-Square Test | | |
|---|---|---|---|---|---|---|---|
| | | $D_n$ | $d_{n,1-\alpha}$ | Passed | $\chi^2$ | $\chi^2_{k-1,1-\alpha}$ | Passed |
| Training set | $t_a$ | 0.19 | 0.21 | Yes | 10.11 | 26.51 | Yes |
| | $t_s$ (overall) | 0.16 | 0.20 | Yes | 14.97 | 27.33 | Yes |
| | $t_s$ (Assessment) | 0.69 | 0.20 | No | 44.78 | 27.33 | No |
| | $t_s$ (TA) | 0.16 | 0.20 | Yes | 6.83 | 27.33 | Yes |
| | $t_s$ (Enact+UT) | 0.19 | 0.26 | Yes | 9.66 | 15.38 | Yes |
| | $t_s$ (TA+Enact+UT) | 0.15 | 0.20 | Yes | 16.14 | 27.33 | Yes |
| Test set | $t_a$ | 0.20 | 0.21 | Yes | 5.06 | 26.51 | Yes |
| | $t_s$ (overall) | 0.25 | 0.20 | No | 12.14 | 27.33 | Yes |
| | $t_s$ (Assessment) | 0.58 | 0.20 | No | 33.93 | 27.33 | No |
| | $t_s$ (TA) | 0.16 | 0.20 | Yes | 8.37 | 27.33 | Yes |
| | $t_s$ (Enact+UT) | 0.32 | 0.26 | No | 4.86 | 15.38 | Yes |
| | $t_s$ (TA+Enact+UT) | 0.26 | 0.20 | No | 14.41 | 27.33 | Yes |

TABLE I

KS-TEST AND CHI-SQUARE TEST RESULTS

(WPs arrived after that date).

2) Then, the model was built using the parameters estimated from the *training set*, and assessed using the *test set data*. In particular, the three project milestones (end of March, end of April and end of May) were considered as reference points for splitting the data set: i.e., three pairs of *training set* and *test set* were considered. On the available data set, fixing a restaffing milestone at the end of March did not allow efficient project restaffing; it was too early and data available represented only the initial transient, rather then the project itself, they represented the initial transient. Similarly, restaffing at the end of May produces overstaffing because of the final transient. As reported in [4], a restaffing performed too early or too late also produced a poor response time estimate. A good solution would be fixing a milestone after half of the WPs were received (i.e., at the end of April). This date corresponds to the project mid-term that, according to the company quality guidelines, corresponds to the ideal project milestone to check project-schedule and/or perform restaffing. From this point, the 42 WPs that arrived from January to April will be referred to as the *training set*, and the remaining 42, arrived from May to July, as *test set*.

3) In a different approach, as described at the end of this section, the model parameters were periodically estimated, and a new staffing was determined according to the target date.

| Parameter | Single queue | Tandem queue | | Queue with exit | | |
|---|---|---|---|---|---|---|
| | | Assessment | TA+Enact+UT | Assessment | TA | Enact+UT |
| N. of WPs | 42 | 42 | 42 | 42 | 42 | 27 **(64%)** |
| $\lambda$ | 0.076 | 0.076 | 0.076 | 0.076 | 0.076 | 0.049 |
| $C_a^2$ | 1.38 | 1.38 | 1.38 | 1.38 | 1.38 | 1.38 |
| $effort$ | 400.61 | 10.69 | 389.93 | 10.69 | 263.9 | 196.21 |
| $C_{effort}^2$ | 1.16 | 3.67 | 1.20 | 3.67 | 0.91 | 1.36 |

TABLE II

MAINTENANCE PROCESS PARAMETERS (TIMES ARE EXPRESSED IN HOURS AND EFFORTS IN MEN HOURS)

## *A. Data Distribution Analysis and Parameters Estimation*

Table I reports results from KS-Test and Chi-Squared Test performed on distributions of the *training set* and the *test set* data. As shown, all tests passed successfully for $t_a$ distributions (that was required from the assumption that requests generated by a Poisson process), while failed for some $t_s$ distributions (in particular, for the assessment activity and for the overall $t_s$). However, this did not constitute a problem: $M/G/m$ models make no assumption on $t_s$ distributions (general, not *a-priori* known) and guarantee a conservative staffi ng[4].

Table II shows an excerpt of the maintenance process parameters corresponding to the time interval (*training set*) from January to April. The table summarizes parameters for the three modeled confi gurations.

## *B. Project Staffing*

The steps explained in Section V-D were performed to determine, at the end of April, the staffi ng required to complete the project by the end of September. Firstly, as explained in Section V-D.2, it was necessary to determine team sizes, for the different models, with the estimated average project completion date. Results are shown in Table III.

Subsequently, queueing theory was used to determine, for each phase of each model, the number of servers *per* node, pursuing a compromise between personnel cost and response time. Even if a short response time was less important in order to fulfi ll the established deadline, this

[4]Test results for $t_s$ distributions were reported for sake of completeness, since this constitutes a fundamental step in the choice of the queueing model to be instantiated.

| **Model** | **Average finishing date** | **Team size** | |
|---|---|---|---|
| **Single queue** | 20 Sept | All phases | 3 |
| **Queue series** | 22 Sept | Assessment | 1 |
| | | TA+Enact+UT | 3 |
| **Three queues network** | 22 Sept | Assessment | 1 |
| | | TA | 3 |
| | | Enact+UT | 2 |

TABLE III

PRELIMINARY PROJECT STAFFING

step was necessary to ensure the statistical equilibrium defined by equation (1), and to make all the WPs available as soon as possible for the integration phase.

The first model analyzed was the single-queue multiple-server model. Fig. 6 shows the waiting time as a function of the staffing level, estimated on the *training set*, and the $t_r$ percentage error over the *test set*. As can be seen, a good compromise between performance and cost is near the knee of the $t_w$ curve, that means staffing the system with 14-15 servers. In this case the $t_r$ error is about 10% and, in any case, is always smaller than 30%.

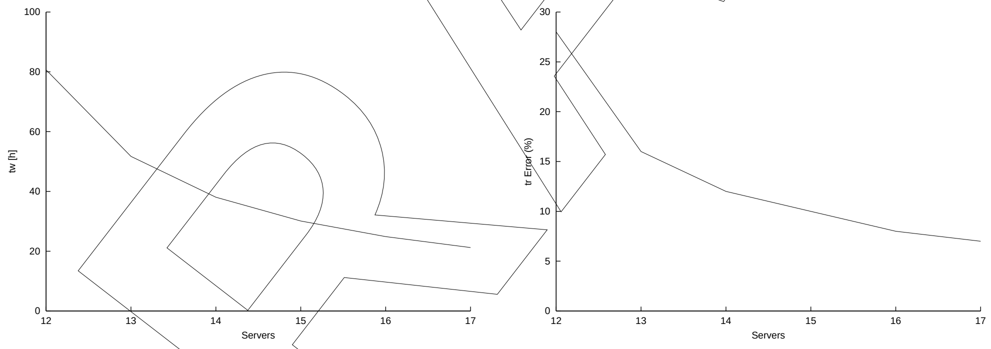

Fig. 6. Single-queue model: estimated $t_w$ and percentage error on $t_r$

A slightly more sophisticated model separates the Assessment phase from the remaining ones (explicitly modeling the geographical distribution of the project): Fig. 7 shows, for this topology and different staffing levels, the estimated waiting times. In particular, two servers for Assessment and 13-14 for TA+Enact+UT ensure the best compromise between personnel cost and low $t_w$

($t_r$ prediction errors below 10%, see [4]).

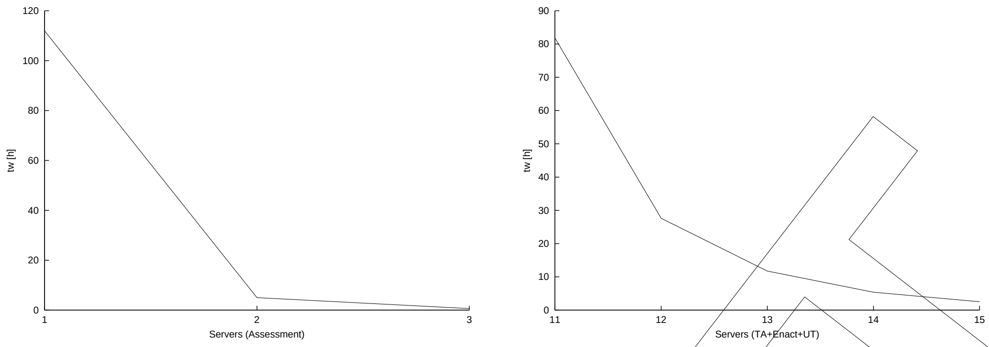

Fig. 7. Queue series: estimated waiting times

Fig. 7, compared with Fig. 6, also highlights that, on the available data, a separate assessment team does not ameliorate the overall $t_w$. In fact, once fixed the number of servers $m_1$ (assessment phase) with a corresponding $t_{w1}$ and the number of servers $m_2$, with $t_{w2}$, for the service center performing Technical Analysis, Enactment and Unit Testing, a single maintenance center (responsible for the entire process) with $m = m_1 + m_2$ maintainers obtains $t_w < t_{w1} + t_{w2}$. The fact is not surprising; assessment teams have a low utilization coefficients, this means that having explicit resources (choice made because of the different expertise needed) devoted to assessment prevents the full resource utilization within the project.

The third model analyzed comprises three different nodes: one for Assessment, one for Technical Analysis, and one for Enactment and Unit Testing. Furthermore, the model accounts for the fact that not all requests needs the Enactment/Unit Testing (in particular, as shown in Table II, the percentage of WPs from the *training set* requiring Enactment/Unit Testing is 64%). After performing (similarly to the previous models) analysis of $t_w$ and $t_r$ error graphs, the compromise configuration was chosen considering the best combination of staffing for the different phases.

Fig. 8 reports the total $t_w$ (considered as the sum of waiting times for the three different nodes, taking into account the fact that only a certain percentage of requests arrive at the final node) and the percentage errors for $t_r$, varying with different server combinations. It is worth noting how, for example, 47 different maintainers may be assigned to different phases, a configuration of 3-10-7 (Assessment-Technical Analysis-Enactment+Unit Testing) performs better than a 2-9-9.

This is not surprising since a percentage of the requests do not require Enactment and Unit Testing. Even in this case a reasonable compromise between cost and performance occurs near the knee of the curves. The figure suggests a configuration of three servers for Assessment, 9 for Technical Analysis and 8 for Enactment and Unit Testing.

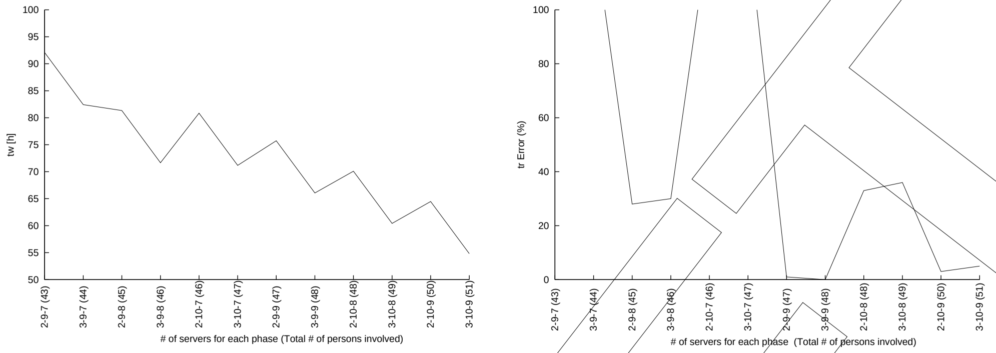

Fig. 8. Three nodes network: overall estimated waiting times and errors on response times for some combinations of servers

| **Model** | **Phase** | **Team size** | **# of servers** | $t_s$ **[h]** | **Avg. finishing date** |
|---|---|---|---|---|---|
| **Single Queue** | All phases | 3 | 15 | 133.52 | |
| | **Total staffing:** | | **45** | | **20 Sept** |
| **Queue series** | Assessment | 1 | 2 | 10.72 | |
| | TA+Enact+UT | 3 | 14 | 130 | |
| | **Total staffing:** | | **44** | | **22 Sept** |
| **Three queues network** | Assessment | 1 | 3 | 10.72 | |
| | TA | 3 | 9 | 87.92 | |
| | Enact+UT | 2 | 8 | 98.08 | |
| | **Total staffing:** | | **46** | | **22 Sept** |

TABLE IV

FINAL PROJECT STAFFING AFTER QUEUEING ANALYSIS

The above considerations lead to the staffing reported in Table IV. It is worth noting that the $t_s$ values reported are functions of the team sizes (i.e., each $t_s$ is computed by dividing the team size the *effort* shown in Table II).

As explained in Section V-D.3, project restaffing needs to be complemented by the evaluation of the likelihood of meeting the established deadline (i.e., the end of October). Simulations to estimate the likelihood of ending the project on a certain date were carried out for the three models, with the total number of programmers involved ranging from 44 to 46. Fig. 9-a shows the results for different dates, ranging from September 30th through to December 15th.

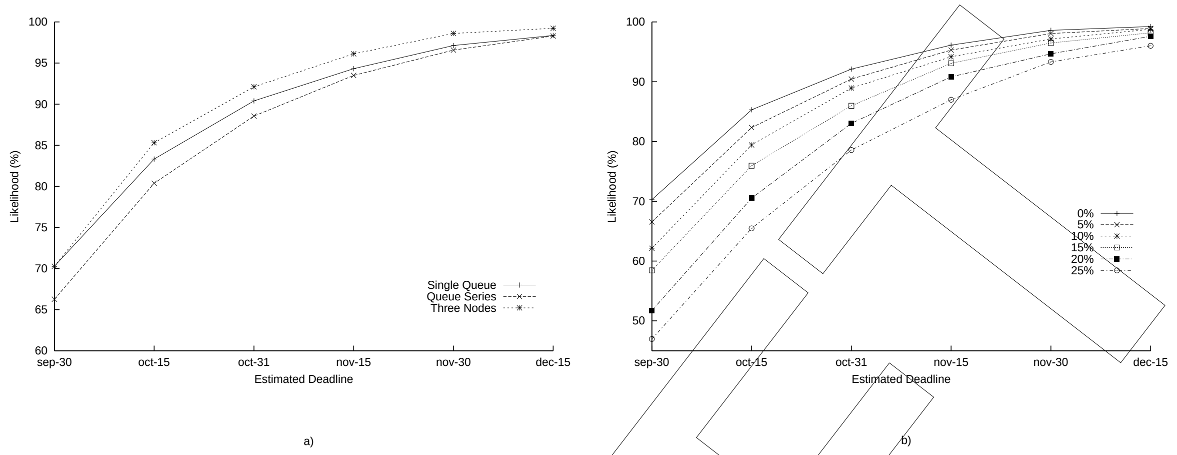

Fig. 9. a) Likelihood of finishing maintenance within a certain date - b) Impact of rework

## C. Analysis of Delays due to Rework

Although data collected from the present case study revealed a negligible level of rework, it is interesting to investigate how rework could affect the project completion date and the likelihood of meeting the deadline. Simulation was therefore performed assuming that a certain percentage of WPs were randomly subject to rework for the Enact-UT phase. It was assumed that the rework effort was independent from the effort previously spent for Enact-UT, even though the distribution was the same. It is worth noting that, if a request was previously subject to rework, it would still have the same probability of being reworked again. Fig. 9-b reports the likelihood of finishing within a certain deadline as function of the percentage of rework.

## D. Model Assessment

As a final step, the test set data were seeded into the model staffed according to Table IV, simulating the behavior on the actual data. Results, reported in Table V, show that for both the single queue model and queue series models, the actual finishing date is very close to the estimated deadline (September 30th). In the three queues network, the error occurring in the estimation was higher. However, as highlighted from Fig. 9-a, as time advances, the likelihood of finishing with a three-queues model is always higher than those of other models.

| Model | Estimated Finishing Date | Actual Finishing Date | Error in the % estimation |
|---|---|---|---|
| **Single queue** | Sep-20 | Sep-28 | 3.74% |
| **Queue series** | Sep-22 | Oct-1 | 6.54% |
| **Three queues network** | Sep-22 | Oct-7 | 10.28% |

TABLE V

FINISHING DATES COMPUTED BY TEST SET SIMULATION

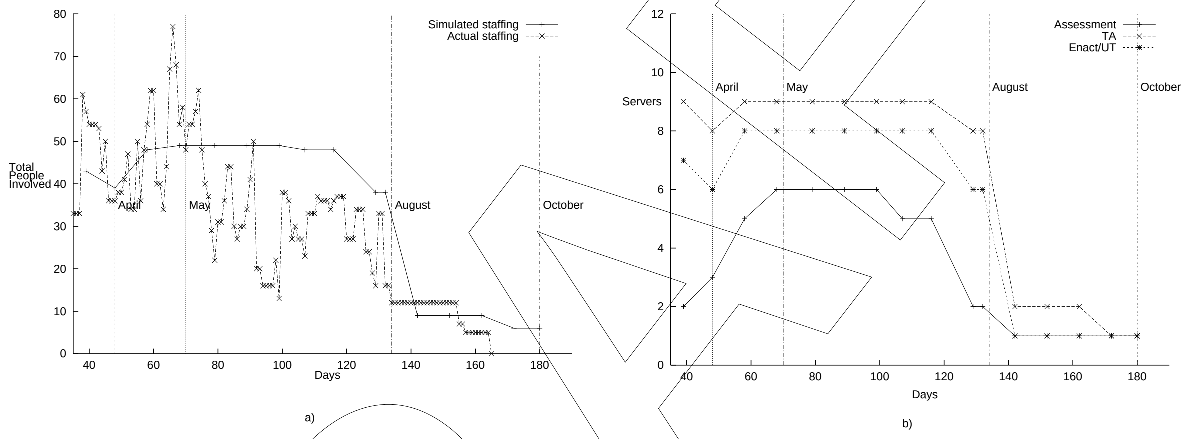

Fig. 10. Periodic restaffing: a) Comparison between actual and simulated staffing - b) Periodic restaffing of different activities (simulated curves)

## *E. Analysis of Transient and Adaptive Restaffing*

Fig. 10 shows how the total staffing and the number of servers for the three different phases change during the project. The average (simulated) staffing consisted of about 36 people, lower than those (46) obtained by considering a constant staffing. The advantages of adaptive restaffing may be summarized as follows:

- At the beginning of the project, the number of requests enqueued into the system did not require a full staffing;

- Similarly, at the end of the project the number of pending requests decreased so that a smaller number of servers sufficed. A few days before the deadline, only one WP remained and, as shown in the figure, a single server for each phase was required; and
- Finally, there are cases for which the arrival rate or the average service times may be subject to relevant variations during the project, so that restaffing to meet the deadline or to avoid wasting resources is necessary.

Fig. 10-a also reports the actual project staffing. These actual staffing levels were obtained by dividing the daily-recorded effort by 8 (the number of working hours per day). This is an optimistic estimate, not accounting for overheads such as training on the job, administrative duties or absences such as vacations. The simulated curve corresponds to an average staffing of 36 people per day, while the actual staffing was, on average, 29 people per day. In other words, the simulation tends to produce an average overstaffing of about 20%. The figure also highlights an initial overstaffing (the leftmost point on the simulated curve), due to the absence of sufficient training material. The Mann-Whitney test (p-value 1.4e-5 for a significance level of 95%) confirmed that the difference between the two curves was significant. Thus, the simulation tends to give us a pessimistic, conservative estimate of the staffing. However, the overall accuracy is higher since the actual staffing shown in Fig. 10-a is the theoretical equivalent actual staffing corresponding to an ideal situation where people work 20 days a month.

## VII. Lessons Learned

Massive maintenance interventions, as well as many other software engineering projects, are firm deadline projects. Thus a key parameter to monitor is the likelihood of meeting the deadline, in that delays may substantially influence the project cost. We experienced that queuing simulation is an effective tool to monitor and assess the likelihood of meeting a given project deadline, while monitoring the project staffing level.

Another advantage of the proposed approach is that rework impact may be easily estimated. Different simulations with different team sizes and rework levels will lead to different estimates, giving relevant insights to the managers.

Model parameters could be adjusted as new data arrives, thus refining early estimates. Queuing simulation and, more generally, queueing theory, has further advantages when applied to software maintenance projects. Indeed, it is essential that the number of model input parameters is kept low and, furthermore, that those input parameters can be easily estimated and understood by managers. Though the model *per-se* can be highly complex, the key observation is that its interface can be kept relatively simple and intuitive. We believe that the parameters required to run a simulation are readily understandable to people working in the software industry. Even with a complex model accounting for multiple maintenance centers and multi-phase maintenance processes, the required input parameters are simply: the maintenance request inter-arrival times, the mean effort required by a maintenance task, and a target deadline (see Fig. 5). Thus queue parameters have intuitive physical meaning and they may be easily obtained from existing project baselines by computing simple descriptive statistics.

When talking to project managers of the Y2K project, we noticed that they had no difficulty in providing estimates for the model required parameters. However, they often based those estimates on a hybrid approach, using analogy and algorithmic methods. Basically, inter-arrival times and effort required in a maintenance intervention were computed as the average value over past project data. Figures were then validated and adjusted according to the manager experience. This empirical study confirmed that queueing simulation requires fine grain recorded data, daily or half-daily based; weekly based recording of effort and inter-arrival times produces models which are too coarse, even worse, more complex models accounting for batch arrivals may be required.

Software projects exhibit characteristics quite different from other phenomena that can be modeled by classic (non simulation-based) queueing theory. In our case study, stochastic simulations revealed the necessity to study the dynamic aspects of the process (i.e., project startup and close-down, average finishing date, likelihood of finishing within a certain date). In this case study, project life span was about one year, project startup and close-down covered about 30% of the entire project: in other words, the transient part was a substantial portion of the entire project and non-simulation based approaches should only be considered as a first approximation.

Queueing-based models tend to produce a staffing level that overestimates the actual staffing. This is especially true for the transient phases. Thus, for a firm deadline project, in the worst-case scenario, a queue-based model will lead to a minimization of the project manager risk of excessive delays. In the case study reported here, it was also observed that the total effort spent on the project and the area under the simulated curve shown in Fig. 10 do not appear to be close. A deeper analysis revealed that the accuracy of the method is higher than the 20%

average error shown in the figure. Discussion with company managers revealed that, according to a company guideline, the average number of working days in a month was estimated to be between 17 and 17.5 instead of 20. This figure accounts for the different source of overheads and absences. Thus the actual staffing of Fig. 10 must be increased by about 13-15%. In other words, the average staffing level any manager of the company would have accepted was not 29 but 33/34 maintainers with a corresponding error of about 6-9%.

In the experience of the authors, the proposed approach has to be complemented and integrated with a detailed work-breakdown structure and a planning process, to dispatch maintenance requests. Given the current state of the art in software engineering, we perceive human intervention to be essential in order to minimize conflicting maintenance interventions, to evaluate the impact of communication overhead inside and among maintenance teams, etc.

A final issue is the applicability and generality of the approach. As pointed out in the related work Section, simulation based approaches were previously proposed in the literature; in particular in [18] the author observed that there are projects where Brooks' law seems not to be completely applicable. This is also central to our approach; we rely on the hypothesis that maintenance interventions may be standardized. In other words, increasing the number of people, decreases the project duration. Other massive maintenance projects, such as the Euro conversion for the new European Union members, or changing the phone numbering system (to be performed in USA by 2010), are likely to exhibit characteristics very similar to a Y2K massive maintenance intervention [2].

## VIII. Conclusions

Data from a massive, corrective maintenance project have been presented together with an approach based on queueing theory and stochastic simulation to deal with the staffing, management and assessment of maintenance projects.

Queueing theory and stochastic simulations are highly effective at evaluating staffing levels, as well as supporting and assessing restaffing decisions. The paper shows how simulations can be carried out to model project startup, close-down, and to evaluate the likelihood of meeting the project deadline. The latter fact, evaluated for different network topologies and rework levels, gives to the management a broader view of the comparison between the actual project estimated status and the project plan. The likelihood of success can be used to establish a trade-off between staffing/restaffing, accepted risks, project delays and customer expectations.

The main advantages of the proposed approach can be summarized as follows:

- It is highly effective at evaluating the likelihood of meeting the project deadline;
- Rework, as well as abandonment, can be explicitly modeled, and their effects on project staffing and project milestones assessed; and
- Project start-up and close-down can be explicitly modeled.

As a final remark, it has been shown how different project management approaches impact on resource utilization, thereby allowing effective comparison of different topologies with respect to the likelihood of meeting the project deadline.

## IX. Acknowledgement

This research has been supported by the project ”Virtual Software Factory”, funded by Ministero della Ricerca Scientifica e Tecnologica (MURST), and jointly carried out by EDS Italia Software, University of Sannio, University of Naples ”Federico II” and University of Bari.

## References

[1] T. DeMarco, *Controlling Software Projects*, Yourdon Press, 1982.

[2] C. Jones, “Mass-updates and software project management in is organizations- *http://www.artemis.it/artemis/artemis/lang_en/libreria/mass-updates.htm*,” 1999, Accessed on Aug, 25 2003.

[3] D. Gross and C. Harris, *Foundamentals of Queueing Theory*, John Wiley & Sons, New York, NY 10158-0012, 1998.

[4] G. Antoniol, G. Casazza, G. A. Di Lucca, M. Di Penta, and F. Rago, “A queue theory-based approach to staff software maintenance centers,” in *Proceedings of IEEE International Conference on Software Maintenance*, Florence Italy, November 2001, pp. 510–519, IEEE Society Press.

[5] M. Di Penta, G. Casazza, G. Antoniol, and E. Merlo, “Modeling web maintenance centers through queue models,” in *Conference on Software Maintenance and Reengineering*, Lisbon, Portugal, March 2001, pp. 131–138, IEEE Society Press.

[6] T. M. Pigoski and L. E. Nelson, “Software maintenance metrics: a case study,” in *Proceedings of IEEE International Conference on Software Maintenance*, Victoria Canada, 1994, pp. 392–401, IEEE CS Press.

[7] G. E. Stark, “Measurements for managing software maintenance,” in *Proceedings of IEEE International Conference on Software Maintenance*, Monterey CA, 1996, pp. 152–161, IEEE Press.

[8] J. Daly, A. Brooks, J. Miller, M. Roper, and M. Wood, “The effect of inheritance on the maintainability of object-oriented software: An empirical study,” in *Proceedings of IEEE International Conference on Software Maintenance*, Opio-Nice France, Oct 1995, pp. 20–29.

[9] T. Khoshgoftaar, B. E. Allen, R. Halstead, and G. P. Trio, “Detection of fault-prone software modules during a spiral life cycle,” in *Proceedings of IEEE International Conference on Software Maintenance*, Monterey, 1996, pp. 69–76, IEEE CS Press.

[10] T. Pearse and P. Oman, “Maintainability measurements on industrial source code maintenance activities,” in *Proceedings of IEEE International Conference on Software Maintenance*, Opio-Nice France, Oct 1995, pp. 295–303.

[11] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[12] Boehm B.W., Horowitz E., Madachy R., Reifer D., Clark B.K., Steece B., Winsor Brown A., Chulani S., and C. Abts, *Software Cost Estimation with Cocomo II*, Prentice-Hall, Englewood Cliffs, NJ, 2000.

[13] W.K. Wiener-Ehrlich, J.R. Hamrick, and V.F. Rupolo, “Modeling software behaviour in terms of a formal life cycle curve: Implications for software maintenance,” *IEEE Transactions on Software Engineering*, vol. 10, no. 4, pp. 376–383, 1984.

[14] L.C. Briand and V.R. Basili, “A classification procedure for an effective management of changes during the software maintenance process,” in *Proceedings of IEEE International Conference on Software Maintenance*, Orlando, FL, 1992.

[15] M. I. Kellner, R. J. Madachy, and D. M. Raffo, “Software process simulation modelling: Why? what? how?,” *Journal of Systems and Software*, vol. 46, no. 2/3, pp. 91–106, 1999.

[16] R. H. Martin and D. M. Raffo, “A model of the software development process using both continuous and discrete models,” *International Journal of Software Process Improvement and Practice*, vol. 5, no. 2/3, pp. 147–157, 2000.

[17] D. M. Raffo and M.I. Kellner, “Empirical analysis in software process simulation modeling,” *Journal of Systems and Software*, vol. 47, no. 9, 2000.

[18] T.K. Abdel-Hamid, “The dynamics of software project staffing: a system dynamics based simulation approach,” *IEEE Transactions on Software Engineering*, vol. 15, no. 2, pp. 109–119, 1989.

[19] T.K. Abdel-Hamid and S.E. Madnick, *Software Project Dynamics: An Integrated Approach*, Prentice-Hall, Englewood Cliffs, N.J., 1991.

[20] T.K. Abdel-Hamid, “Adapting, correcting, and perfecting software estimates: a maintenance metaphor,” *IEEE Computer*, vol. 26, no. 3, pp. 20–29, 1993.

[21] F. Brooks, *The Mythical Man-Month, 20th anniversary edition*, Addison-Wesley Publishing Company, Reading, MA, 1995.

[22] G. E. Stark and P. W. Oman, “Software maintenance management strategies; observations from the field,” *Journal of Software Maintenance - Research and Practice*, vol. 9, pp. 365–378, December 1997.

[23] M. Jorgensen, “Experience with the accuracy of software maintenance task effort prediction models,” *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 674–681, 1995.

[24] M. Höst, B. Regnell, J. Natt och Dag, J. Nedstam, and C. Nyberg, “Exploring bottlenecks in market-driven requirements management processes with discrete event simulation,” in *Prosim*, London, UK, July 1999.

[25] G. Papapanagiotakis and P. Breuer, “A software maintenance management model based on queueing networks,” *Journal of Software Maintenance - Research and Practice*, vol. 6, no. 1, pp. 73–97, 1994.

[26] R. Ramaswamy, “How to staff business critical maintenance projects,” *IEEE Software*, vol. 7, no. 7, pp. 90–95, May 2000.

[27] I. Podnar and B. Mikac, “Software maintenance process analysis using discrete-event simulation,” in *Conference on Software Maintenance and Reengineering*, Lisbon, Portugal, March 2001, pp. 192–195, IEEE Society Press.

[28] D. M. Raffo, W. Harrison, and J. Vandeville, “Coordinating models and metrics to manage software projects,” *International Journal of Software Process Improvement and Practice*, vol. 5, no. 2/3, pp. 159–168, 2000.

[29] Jerry Banks, John Carson, and Barry L. Nelson, *Discrete-Event System Simulation*, Prentice Hall, 1998.

[30] *IEEE std 1219: Standard for Software maintenance*, 1998.

[31] M.J. Shepperd and C. Schofield, “Estimating software project effort using analogies,” *IEEE Transactions on Software Engineering*, vol. 23, no. 11, pp. 736–743, 1997.

[32] R. Arnold, "Software reengineering," 1993.

[33] E.C. Lynd, "Living with the 2-digit year, year 2000 maintenance using a procedural solution," in *Proceedings of IEEE International Conference on Software Maintenance*, Bari, Italy, 1997, pp. 206–212.

[34] M. Stone, "Cross-validatory choice and assessment of statistical predictions (with discussion)," *Journal of the Royal Statistical Society B*, vol. 36, pp. 111–147, 1974.

[35] G. Box and M. Jenkins, *Time Series Forecasting Analysis and Control*, Holden Day, San Francisco (USA), 1970.

[36] Myles Hollander and Douglas A. Wolfe, *Nonparametric Statistical Methods, 2nd Edition*, Wiley-Interscience, 1999.

[37] Jay L. Devore, *Probability and Statistics for Engineering and the Science*, Duxbury Press, 1999.

PLACE
PHOTO
HERE

**Giuliano Antoniol** received his doctoral degree in Electronic Engineering from the University of Padua in 1982. He worked at Irst for ten years were he led the the Irst Program Understanding and Reverse Engineering (PURE) Project team. Giuliano Antoniol published more than 60 papers in journals and international conferences. He served as a member of the Program Committee of international conferences and workshops such as the International Conference on Software Maintenance, the International Workshop on Program Comprehension, the International Symposium on Software Metrics. He is presently member of the Editorial Board of the Journal Software Testing Verifi cation & Reliability, the Journal Information and Software Technology, the Empirical Software Engineering and the Journal of Software Quality. He is currently Associate Professor the University of Sannio, Faculty of Engineering, where he works in the area of software metrics, process modeling, software evolution and maintenance.

PLACE
PHOTO
HERE

**Aniello Cimitile** received the Laurea degree in Electronic Engineering from the University of Naples, Italy, in 1973. He is currently the Chancellor of the University of Sannio in Benevento, Italy, where he is a full Professor of Computer Science. Previously, he was with the Department of 'Informatica e Sistemistica' at the University of Naples Federico II. Since 1973 he has been a researcher in the fi eld of software engineering and his list of publications contains more than 100 papers published in journals and conference proceedings. He serves in the program and organizing committees of several international conferences and in the editorial and reviewer committees of several international scientifi c journals in the fi elds of software engineering and software maintenance. He is a co-editor in chief of the Journal of Software Maintenance and Evolution: Research and Practice. He has been responsible for many international and national applied research projects. His research interests include software maintenance and testing, software quality, reverse engineering, and reuse reengineering.

PLACE PHOTO HERE

**Giuseppe A. Di Lucca** received the Laurea degree in Electronic Engineering from the University of Naples ”Federico II”, Italy, in 1987 and the Ph.D. degree in Electronic Engineering and Computer Science from the same University in 1992. He is currently an associate professor of Computer Science at the Department of ”Ingegneria” of the University of Sannio. Previously, he was with the Department of ’Informatica e Sistemistica’ at the University of Naples ’Federico II’. Since 1987 he has been a researcher in the field of software engineering and his list of publications contains more than 50 papers published in journals and conference proceedings. He serves in the program and organizing committees of conferences in the field of software maintenance and program comprehension. His research interests include software engineering, software maintenance, reverse engineering, software reuse, software reengineering, web engineering, and software migration.

PLACE PHOTO HERE

**Massimiliano Di Penta** received his laurea degree in Computer Engineering in 1999 and his Ph.D. in Computer Science Engineering in 2003 at the University of Sannio in Benevento, Italy. Currently he is with RCOST Research Centre On Software Technology in the same University. His main research interests include software maintenance, reverse engineering, program comprehension and search-based software engineering. He is author of about 25 papers appeared in international journals, conferences and workshops. He serves the organizing committees of workshops and conferences in the software maintenance field.