Incorporating Multi-Team Effort and Cost Estimation in System Design

Jones Albuquerque
Mobile S/A
Recife, PE - BRAZIL
+55 81 3271.6200
jones.albuquerque@mobile.com.br

Claudionor Coelho Jr.
Diógenes Cecílio Jr.
Antônio Otávio Fernandes
Departamento de Ciência da Computação - UFMG
Belo Horizonte, MG - BRAZIL
+55 31 499.5860
{coelho,dgns,otavio}@dcc.ufmg.br

ABSTRACT

System cost estimation is a key factor in the success of a product in a competitive marketplace. Several algorithmic models have been proposed to estimate costs and other management parameters. However, the probabilistic nature of the parameters and team aspects of the design are often neglected. In this paper, we propose a model for system design effort and cost estimation that incorporates probabilistic parameters and team aspects.

0.1 Keywords

Multi-team design, Risk analysis, Stochastic Linear Programming

1 INTRODUCTION

System specifications and design become critical as size increases, real-time constraints surface, and multiple programmers work on one job. Such designs are so complex that we just do not understand all the implications about the decision we make, even the simplest ones.

Several algorithmic models have been proposed to estimate costs and other management parameters [20, 24], but they are focused on software development and do not treat the uncertainties present on real projects, which are composed by hardware and software components. System design cost models can be found [7, 3, 5], but they do not treat team aspects. Some other approaches [21, 11] consider probabilistic parameters in its cost evaluations, but they are focused on stand-alone development and they do not treat team aspects.

In this work, we propose a model to multi-team design and cost estimation based on a partitioning algorithm [1]. In partitioning phase, the system is partitioned in components, teams, and technologies, in order to minimize costs. We assume that system design is accomplished by several teams, each one with its speciality. We also assume that teams make estimates and their estimates are based on their experience. This experience can be obtained by tracking the team's past tasks [10]. Then, each team can determine appropriate probability distributions for items such as team productivity and load, execution time for tasks, and development time and cost estimated for a task. We use these estimates in our mathematical model and solve the resulting partitioning problem on several scenarios. This makes possible to explore several design options and choose a better task and team allocation that minimizes the costs.

This paper is organized as follows. Section 2 presents how to capture probabilistic parameters from team uncertainties. Section 3 presents the design approach. Section 4 presents the guidelines to use the mathematical model. And the case studies are in Section 5.

2 INCORPORATING PROBABILISTIC PARAMETERS

A key element in system design is to make early design exploration, i.e. make trade-offs between different system solutions. These trade-offs are inherently based on incomplete information. This part of design is very tool scarce [25]. We call the design at this level as *conceptual design*, since at this level, the design has been only outlined and its main functionalities determined.

Since the system has not been implemented yet at conceptual level, designers can only give estimates on any metric for the system. We treat the estimates in a probabilistic way. A team estimate for a task is associated with a Normal Distribution curve [27], in which mean μ and variance σ are determined by team's experience and historical data. We use the strategy that each team estimate is captured by a 3-tuple (m, M, c), where m is the minimum value for a given metric, M is the maximum value for the given metric and c is the confidence degree of the estimates minimum and maximum.

These confidence degrees are based on team historical data. If a team has a precise historical data of its tasks then its confidence degree is maximum. If a team has not historical data, its degree is minimum. The confidence degree degrades in absence of historical data or if the team historically has imprecise estimation methods. Thus, the Normal Distribution curve is defined by the estimation 3-tuple (m, M, c), where $\mu = \frac{m+M}{2}$ and $\sigma = \frac{M-m}{2c}$, where c is defined per project (vide Section 2.2). The possible values for c can be customized. In this way, we can capture the uncertainties of the team estimates and its development capabilities.

Similar approaches have been proposed in the past by considering inaccuracy of estimates given by teams in a probabilistic fashion, but they do not treat complex systems (hardware and software components, several technologies, team management). Examples of these approaches include PERT/CPM techiniques or Monte Carlo simulations [22]. Our work is in the intersection of the three circles illustrated in Figure 1: partitioning algorithms, mathematical modelling and design process management with risks.

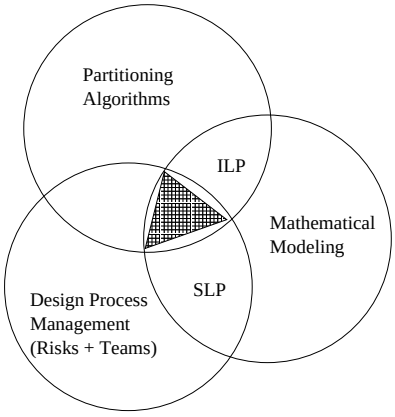


Figure 1: System design areas. Our approach context is shaded, where ILP means Integer Linear Programming and SLP means Stochastic Linear Programming

This intersection illustrates that we are working in order to provide a model to incorporate characteristics of these three areas. Partitioning algorithms from hardware systems, design process from software systems, and probabilistic tools from mathematical modeling.

2.1 Obtaining Team's Historical Data

In software engineering, the process history has been used to predict and manage software projects [9, 29]. Many system quality models use only product metrics such as lines of code or McCabe cyclomatic complexity. This product focus assumes that all modules have a similar process history. Although this is not valid for all system, it has been used with success in software engineering [16].

Some models provide frameworks that help developers to plan and to track their work. PSP [14], Personal Software Process, is a well known model. It is focused on software development by single person, but team aspects have been added to the model [17]. Futhermore, system aspects (metrics) can be also added to the model.

The PSP measures is defined based on the Goal-Question-Metric paradigm [2]. These are the time the engineer spends in each process phase, the defects introduced and found in each phase, and the developed product sizes. These data, gathered in each process phase and summarized at project completion, provide to the engineers a family of process quality measures: size and time estimating error; cost-performance index;

defects injected and removed per hour; process yield; appraisal and failure cost of quality; and the appraisal to failure ratio.

In this way, a complete profile of each engineer and also each team can be obtained using PSP measures [15]. Obviously, any other approach to capture historical data can be used in our approach.

2.2 Obtaining Team's Confidence Degree

The confidence degrees can be obtained from team's reports and historical data collected as described early. We can estimate and plan future tasks by a PSP method called Probe [14]. Probe is a proxy-based estimating method that lets engineers use their personal data to judge a new project cost and required development time. Size proxies help engineers to visualize the probable size of new project modules. Other proxies - function points, screens, or reports - are also possible.

Probe is a linear regression-based size-estimating method. Linear regression estimates the (presumed) relationship between one variable and another by expressing one in terms of a linear (or more complex) function of the other. This function can then be used to predict the value of one variable based on the value of the other. Thus, the linear regression method does produce the statiscally best fit, called the maximum likehood fit, to the historical data collected. Formally, given a set of historical data for variables x and y, to determine a likely value y_k based on a known or estimated new value x_k we use

$$y_k = \beta_0 + x_k \beta_1. \tag{1}$$

The estimating parameters β_0 and β_1 are calculated from the historical data using the following equations

$$\beta_1 = \frac{\sum_{i=1}^{n} x_i y_i - n x_{avg} y_{avg}}{\sum_{i=1}^{n} x_i^2 - n x_{avg}^2},$$

$$\beta_0 = y_{avg} - \beta_1 x_{avg},$$

where, n is the number of terms in the historical dataset, x_{avg} is the average of all the x_i terms and y_{avg} is the average of all the y_i terms.

The prediction interval represents the quality of the estimate and its formula is

$$Range = t(\alpha/2, n-2)\sigma\sqrt{1 + \frac{1}{n} + \frac{(x_k - x_{avg})^2}{\sum_{i=1}^{n}(x_i - x_{avg})^2}},$$

where, t is a value for the double-sided t distribution for the probability $\alpha/2$ and $n-2$ degrees of freedom [14]. Also, σ is the standard deviation of the regression function (calculated below), n is the number of terms in the historical dataset, x_k is the new estimate term, and x_i is the value of each term in dataset. The variance of the data is taken from the regression line instead of from the average value of the data:

$$Variance = \sigma^2 = \left(\frac{1}{n-2}\right) \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2.$$

Probe provides a method to estimate values and their respective prediction intervals for each engineer/team for each task. These parameters are straightforwardly mapped on our 3-tuple (m, M, c), where m and M are obtained from the estimate value in (1) minos the $Range$ ($m = y_k - Range$) and plus the $Range$ ($M = y_k + Range$), respectively. And c is customized per project. For example, Table 1 presents confidence degrees for integer values of c, i.e. $c = 3$ ("Very High"), $c = 2$ ("High") and $c = 1$ ("Low").

In this way, each team estimate is characterized by our 3-tuple (m, M, c), incorporating maturity issues by c parameters.

3 DESIGNING WITH EFFORT AND COST ESTIMATION

Object-oriented techniques seem to provide an efficient solution in mastering development costs by facilitating reuse at all development stages, from the problem analysis to the software implementation. UML *Unified Modeling Technique* [23] is today the leading object-oriented method in software development.

On the other hand, as it is now, UML is not easily applicable to hardware systems. As a consequence, UML must be complemented by formalism purely dedicated to hardware systems: SDL *Specification and Description Language* [18], and MSC *Message Sequence Chart* [26].

We assume that a designer will specify a system using three different views: a hierarchical view, a sequencing view and a development view.

Confidence Degree	Standard Deviation (σ)	Statistical Semantic
Very High (VH)	$3\sigma = M - \mu$	99.7% of the values are in the estimated range.
High (H)	$2\sigma = M - \mu$	95.5% of the values are in the estimated range.
Low(L)	$\sigma = M - \mu$	68.3% of the values are in the estimated range.

Table 1: Definition of the σ parameter

3.1 System Views

The *hierarchical view* allows the system to be modeled as a set of objects which may be composed by other objects, forming a dependency relation [28, 23]. Each object also contains access methods that perform the different functionalities envisioned for that object. In this sense, the hierarchical view is an object-oriented model where dependency relations are obtained through hierarchy.

The hierarchical view presents a static view of the system which is used during task assignment and task allocation. Being hierarchical, any object in the system can be further subdivided during refinement or during the implementation, which may require the assignment of new objects to the teams. If an object performs only one functionality, as in digital signal processing blocks, we will usually omit the access methods for simplicity. As an example, we will use a network controller of Figure 2-a, which is initially subdivided into the objects BUS UNIT, RECEIVE UNIT, TRANSMIT UNIT and EXECUTION UNIT. The BUS UNIT object for example, has as access methods BUS_DMA_WRITE, BUS_DMA, BUS_WRITE and BUS_READ. As mentioned, each object can be further subdivided into additional objects, as shown in Fugure 2-b.

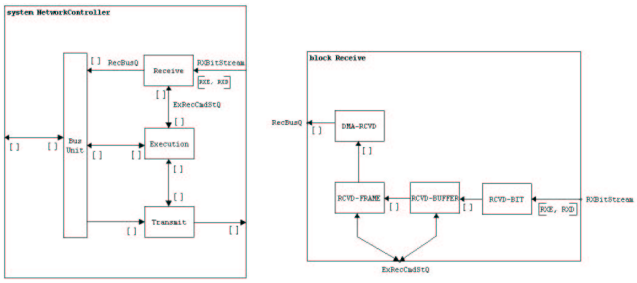


Figure 2: Network Controller, hierarchical view

The *sequencing view* allows the designer to specify the dynamic aspects of a system, usually consisting of sequential paths executed by the access methods of the objects. We use UML and MSC transition diagrams to represent the sequencing view of a system as in Figure 3.

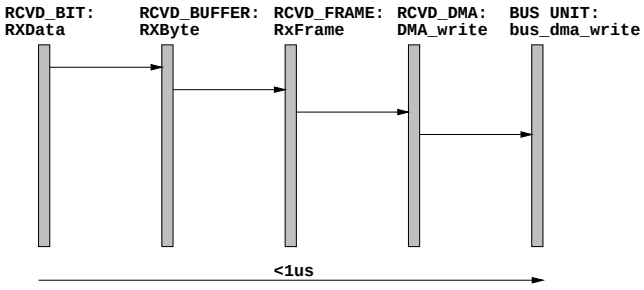


Figure 3: Sequencing View for the Startup Sequence of Reception Unit

Formally, a dynamic view of a system can be described as a sequential graph SG consisting of a set of nodes N and edges E_s. The dynamic aspects takes the form of a labeled direct acyclic graph (DAG). Each of the nodes $n_k \in N$ corresponds to an method of the system. An edge $e_{ij} \in E$ within SG represents an ordered pair (n_i, n_j), where n_j "is a subsequent methot of" n_i and the label is the dynamic constraint, such as execution time, execution rate or power consumption. In this way, each path in this graph represents the limitations of the system. For example, the summation of each execution time of the methods of a path represents a critical constraint and the greater path, when summing the execution times. Which can be interpreted as the execution time limitation of the system.

- $SG = (N, E_s)$
- $N = \{n_1, n_2, \ldots, n_z\}$, where z is the number of objects defined in the hierarchical view
- $E \subseteq N \times N$, $(n_i, n_j) \leftrightarrow n_j$ is a subsequent methot of n_i

The *development view* allows the designer to represent development constraints such the order of development specific objects, which are represented in a project graph PG in which nodes represent objects and edges represent development dependencies. The PG graph is defined similarly as SG graph. Notice that at the very early stages of the design, the objects have not been assigned to teams yet, so the order of development represents constraints that span over different teams.

The development view may contain more objects than those defined in the hierarchical view to represent tasks that are performed by the teams (such as testing) that are not directly linked to the system's functional specification. Specifically, we define initially two dummy objects in the development view that model the initial and the ending times of the project. As an example, a possible project graph for the network controller of Figure 2 is presented in Figure 4.

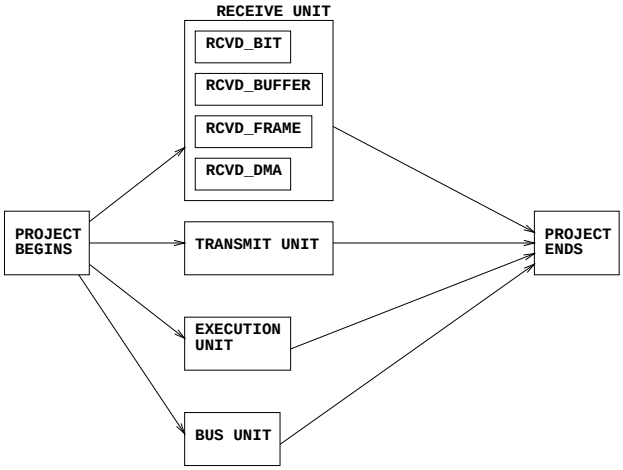


Figure 4: Project Graph for Network Controller System

3.2 Assumptions

The basic assumption behind this work lies in the fact that for large systems no single designer has the complete knowledge of the system. As a result, team assignment and partitioning must be performed before the implementation actually begins and at the same time, since a bad partitioning may increase the possibility of project failure due to team effects. Likewise, if team assignment is performed before partitioning, implementations that do not respect system's constraints may be obtained. The goal is to add as much information to the system as possible in order for an automated tool to make design trade-offs and to help designers to obtain optimal decisions.

We assume that system design is accomplished by several teams, each one with its speciality (technology). We also assume that teams make estimates and their estimates are based on their experience. This experience can be obtained by tracking the team's past tasks [24, 9]. Then, each team can determines appropriate probability distributions for items such as team productivity and load, execution time for tasks, and development time and cost estimated for tasks. We use these estimates to annotate the graphs PG and SG with design parameters estimated by each team, then we use the graphs filled in a mathematical model and solve the resulting partitioning problem on several scenarios. This makes possible to explore several design options and choose a better task and team allocation that minimizes the costs.

Although we presented a design style similar to SDL descriptions with message charts, we could have used any other design specification language, including UML, as long as there is an object-oriented specification with clear separation between static and dynamic models. An important aspect of our work is the decisions that will be taken from a given design specification with uncertain estimates, and not the specification language.

3.3 The Design Process: Integrating the Views

The parameters of PG and SG filled by the development teams with their estimates for each object provide several design scenarios for the system. Thus, we need to decide which allocation (*teams* $\times$ *objects* $\times$ *technology*) can better implement the system, satisfying the constraints. This choice is accomplished by an mathematical optimization model, in which the graphs provide the critical paths of the system and the design constraints are modeled by probabilistic inequations.

3.3.1 Design Steps

We describe below the main design steps used during a system design using our approach.

1. The system is described as a collection of interconnected objects (hierarchical view), and its sequential and development views;
2. The system's main design constraints are determined and annotated in their respective views;
3. The system model is given to the development teams;
4. Each development team estimates values for each object and method considering the technology options and team's ability, and annotates them on the graphs PG and SG;
5. For each design scenario:
   The solver will choose the best implementation considering the objective function that must be optimized and the risk of success for satisfying the design constraints.

In this way, we obtain a multi-team allocation that minimizes the cost and satisfies the system constraints (e.g. team load, execution delay, development time). However, the partitioning process does not begin in the most abstract level of modeling and does not finish when an automatic tool decides if a object will be implemented by one or another team. Partitioning is an interactive process that is improved at each step until the best scenario for the design has been found.

4 THE COST MODEL

As presented before, we model the uncertainties of the team estimates as probabilistic curves. Thus, each curve represents the team experience and historical data about a specific parameter in previous system design. These curves are used in a stochastic linear approach to improve the partitioning problem solutions. In accord with the taxonomy presented in [12], we are working on *Stochastic Integer Linear Programming* (SILP) and *Chance-constrained problems*, which one or more probabilistic constraints have the form

$$\mathbf{min}\{\sum_{i,j} c_j x_j\}$$

s.t.

$$Prob\{\sum_{i,j} a_{ij} x_{ij} \leq b_j\} \geq 1 - \alpha_i;$$

where, $x_{ij} \in \{0,1\}$ - binary variable that assumes value 1 if object i is implemented by team j; the coeffients a_{ij} and b_j are *random variables* - estimates from teams and system constraints, respectively; $i = 1...I$, $j = 1...J$, $0 < \alpha_i < 1$. α represents the probability of uncertainty, consequently $1 - \alpha$ represents the probability of certainty [6].

4.1 Notation

The general symbols are presented in Table 2, the probabilistic parameters in Table 3, and decision variables in Table 4.

Symbol	Description
$Objects$	set of the objects
$Methods_i$	set of methods pertaining to object i
$Teams$	set of development teams available
$Weeks$	ordered set of weeks (or any other time metric: days, months, ...)
$PathsD$	set of ordered development dependencies (objects) defined in the development view models PG
$PathsE$	set of ordered execution dependencies (methods) defined in the sequential view models SG
C	cost desired for the project
DI_p	execution time desired for sequencing view model p
DD_d	develpment time desired for development view model d
Λ_j	maximum load of team j (manpower)

Table 2: Symbols Used in the SILP Formulation

Parameter	Descripton
c_{ij}	estimated cost for the task i by team j (US$, area, KLOC, ...)
d_{mj}	estimated execution time of mathod m when implemented by team j
t_{ij}	estimated (development) time taken by team j to implement object i
λ_{ij}	estimated load of team j to implement object i

Table 3: Probabilistic Parameters Used in the SILP Formulation.

4.2 Constraint Formulation

We present below the SILP formulation for the constraints in the presence of uncertainty.

1. The *probabilistic parameters of each node* x_{ij} are a 4 - tuple (c_{ij}, d_{mj}, t_{ij}, λ_{ijk}) with non-negative numbers for each possible technology option. Each parameter is represented as the 3 - tuple (m, M, c) (minimum, Maximum, confidence degree).

2. Every object i must be implemented by only one team j

   $\forall i \in Objects \;\; \sum_{j \in Teams} x_{ij} = 1$

3. There is only one beginning and ending week for each object i implemented by team j

   $\forall i \in Objects, j \in Teams \;\; \sum_{k \in Weeks} \gamma_{ijk} = x_{ij} \;\; \forall i \in Objects, j \in Teams \;\; \sum_{k \in Weeks} \Gamma_{ijk} = x_{ij}$

4. Path execution time, $PathsE \in SG$

   $\forall P \in PathsE, j \in Teams$
   $Prob\{\sum_{(i,p,m) \in P} d_{mj} x_{ij} \leq DI_p\} \geq 1 - \alpha_{path}$

5. Objects developed sequentially are implemented in that order

   $\sum_{j,k} k\gamma_{i_2 jk} - \sum_{j,k} k\Gamma_{i_1 jk} \geq 1$

Variable	Descripton
x_{ij}	binary variable that assumes value 1 if object i is implemented by team j
γ_{ijk}	binary variable meaning that at week k, team j begins the implemention of object i
Γ_{ijk}	binary variable meaning that at week $k-1$, team j ends the implemention of object i
a_{ijk}	binary variable meaning that at week k, team j is implementing object i

Table 4: Decision Variables Used in the SILP Formulation.

for each edge $(p, i_1, i_2) \in PathsD$, i.e. the edge defines a dependency in the development view model p, $j \in Teams$ and $k \in Weeks$.

6. Development time, $PathsD \in PG$

   $\forall P \in PathsD, j \in Teams$
   $Prob\{\sum_{(p,i_1,i_2)\in P} t_{i_1 j} x_{i_1 j} \leq DD_p\} \geq 1 - \alpha_{develop}$

7. Maximum load for team j (k_1 is the first week)

   $\forall i \in Objects, j \in Teams$
   $a_{ijk_1} = \gamma_{ijk_1} - \Gamma_{ijk_1}$

   $\forall i \in Objects, j \in Teams, k \in Weeks$
   $a_{ijk} = a_{ij(k-1)} + \gamma_{ijk} - \Gamma_{ijk}$

   $\forall j \in Teams, k \in Weeks$
   $Prob\{\sum_{i\in Objects} \lambda_{ij} a_{ijk} \leq \Lambda_j\} \geq 1 - \alpha_{load}$

8. Object assignment

   $\forall i \in Objects, j \in Teams$
   $Prob\{\sum_{k\in Weeks} k\Gamma_{ijk} = t_{ij} x_{ij} \quad + \sum_{k\in Weeks} k\gamma_{ijk}\} \geq 1 - \alpha_{assignment}$

9. Cost of project

   $Prob\{\sum_{i\in Objects, j\in Teams} c_{ij} x_{ij} \leq C\} \geq 1 - \alpha_{cost}$

The objective function can be easily defined by converting any constraint from the set presented above.

Solve this problem means to find a partitioning solution that best sastifies these constraints, minimizing the cost function. Computationally, a more quantifiable approach is to solve the original Integer Linear Programming where all the probabilistic data have been replaced with their *expected values*. In this way, the problem can be written as a large deterministic problem (Expected Value Formulation [4]). The resulting *deterministic equivalent* problem can be solved using any general purpose optimization package. This is the approximation used to solve our model for multi-team design when the variables are randomly distributed.

5 CASE STUDIES

This approach has been tested through the simulation of design of a number of small case studies (SAR-*Synthetic Aperture Radar* [5], Ethernet microcontroller [19], MRF-*Markov Random Fields* [13], MPEG-2 decoder) that were used to examine the capabilities of the approach.

We simulated hypothetical team profiles with different risk constraints and solve the deterministic equivalent SILP using the commercial package AMPL which uses C_PLEX solver [8]. We generated random values for the constraint parameters that were not found in literature and assume the same risk $1 - \alpha$ for each class of equations in each simulation.

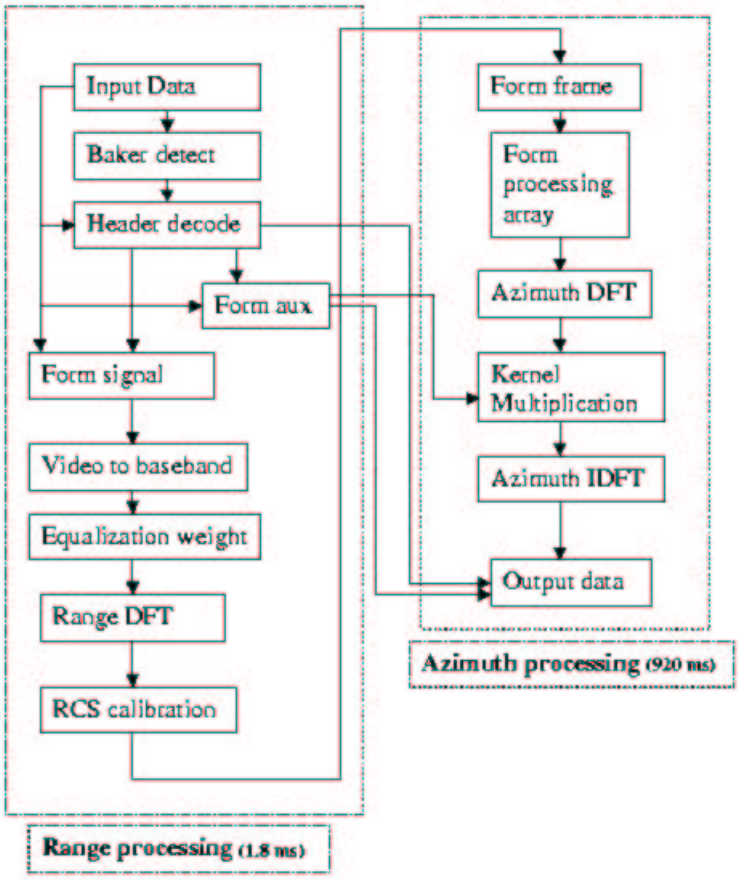


Figure 5: SAR Imaging Processing: functional blocks [5].

5.1 SAR

We modeled the Synthetic Aperture Radar (SAR) imaging processing presented in [5]. We generated some random values for constraint parameters that was not specified in the original paper. Its specification generates fifteen tasks (nodes in task graph). The SAR imaging processing is illustrated in Figure 5.

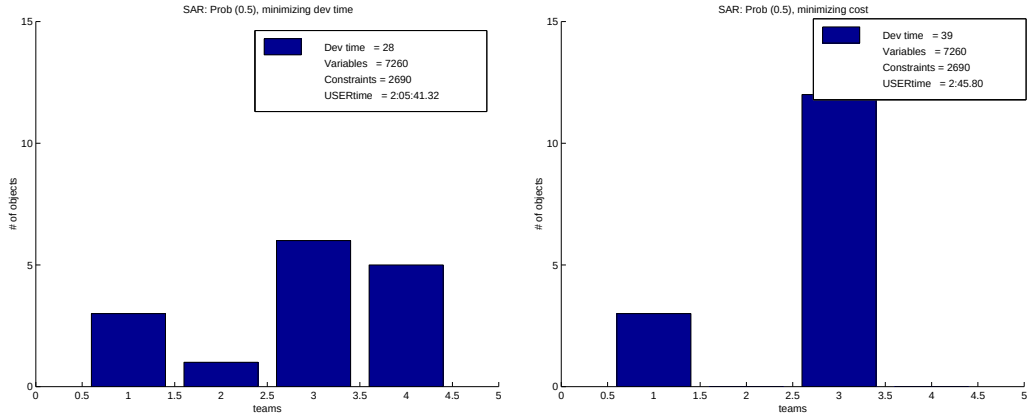
We partitioned the system considering the same risks and constraints (cost, delay time, development time and team load) valus for all objects, and four hypothetical development teams: $1 = hw\_c1$, $2 = hw\_c3$, $3 = sw\_c1$ and $4 = sw\_c3$, where hw represents speciality in hardware, sw represents speciality in software, $c1$ represents no historical data (low confidence degrees on parameter estimates), $c3$ represents a mature team with precise historical data (very high confidence degrees on parameter estimates). We evaluated the partitioning system for these parameters, observing the following facts (illustrated in Figure 6):

- The partitioning algorithm chose teams with tighter constraints, priorizing the best estimates;
- When minimizing delay, the algorithm chose hardware implementations, priorizing the hw_c3 team;
- When minimizing cost, the algorithm chose software implementations, priorizing the sw_c3 team;
- When minimizing team load, the algorithm did not provide a solution in the time limit.

5.2 MRF

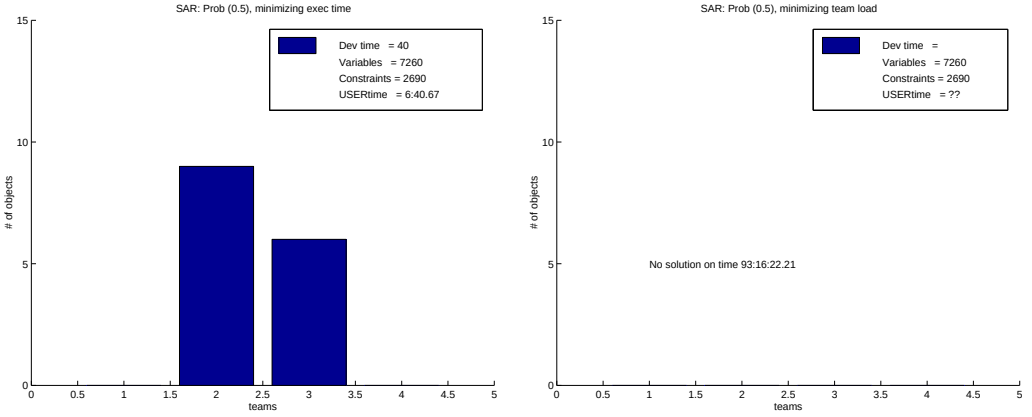
We modeled an example of the application of Markov Random Fields (MRF) to image restoration. The restoration method is based on [13], its class diagram is illustrated in Figure 7.

In this case study we generated random values for the constraint parameters, simulating the design estimates given by the teams. We partitioned the system specified in Figure 7 considering the following design constraints in different risk scenarios: development time for the entire project, team load, execution time for sequencing view models, and cost of project. We assumed that six hypothetical development teams could implement any object of the system: $t1\_c1$, $t1\_c2$, $t1\_c3$, $t2\_c1$, $t2\_c2$, and $t2\_c3$, where $t1\_c1$, $t1\_c2$ and $t1\_c3$ represent teams detaining knowledge in software technologies; $t2\_c1$, $t2\_c2$ and $t2\_c3$ represent teams detaining knowledge in hardware technologies; the suffix c meaning the maturity of the team, e.g. $c3$ means a maturity team which has better estimates than $c2$, and so on.



(a) Minimizing Development Time

(b) Minimizing System Cost



(c) Minimizing Execution Time

(d) Minimizing Team Load

Figure 6: Partitioning under different scenarios with the same risks ($Prob\{1-\alpha\} = 0.5$), when minimizing Development Time, System Cost, Execution Time, and Team Load.

The overall results can be summarized as follows:

- The partitioning algorithm chose teams with tighter constrains, priorizing the best estimates;
- When minimizing for execution time, the algorithm chose hardware implementations, priorizing the teams with best estimates as in Figure 8: when we limited the budget ($cost = 150$) we had software implementations priorized, when we had higher budgets ($cost = 200..300$), hardware implementations were priorized (four objects for $t2\_c3$ team when $cost = 300$);
- When minimizing cost, the algorithm chose teams with worst estimates and software implementations.

6 CONCLUSIONS

In essence, this article described an approach for managing multi-team design in presence of uncertainty. Our approach provides a set of early scenarios to guide the design manager to choose teams/technologies that better satisfy the risks and system constraints.

We also presented how to obtain the paremeters (historical data and confidence degrees from teams) used in the SILP formulation from a commercial model (PSP). Although SILP problems are not yet solved by commercial solvers, some approximations other than expected value can be used to solve them.

As future work, we want to automatically obtain estimates to objects based on the team's design history. We are also investigating other approximation methods presented in the literature to solve the SILP problem.

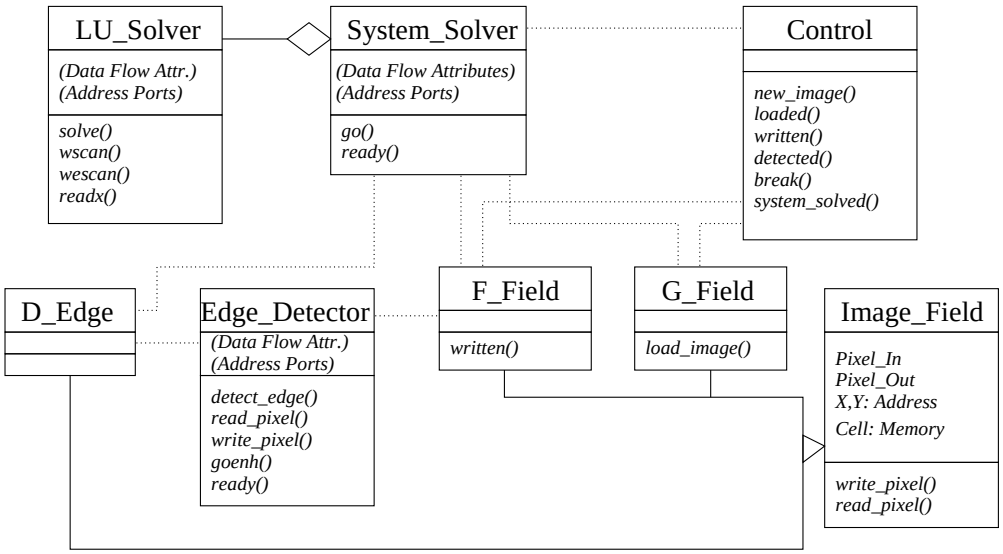


Figure 7: MRF System's class diagram

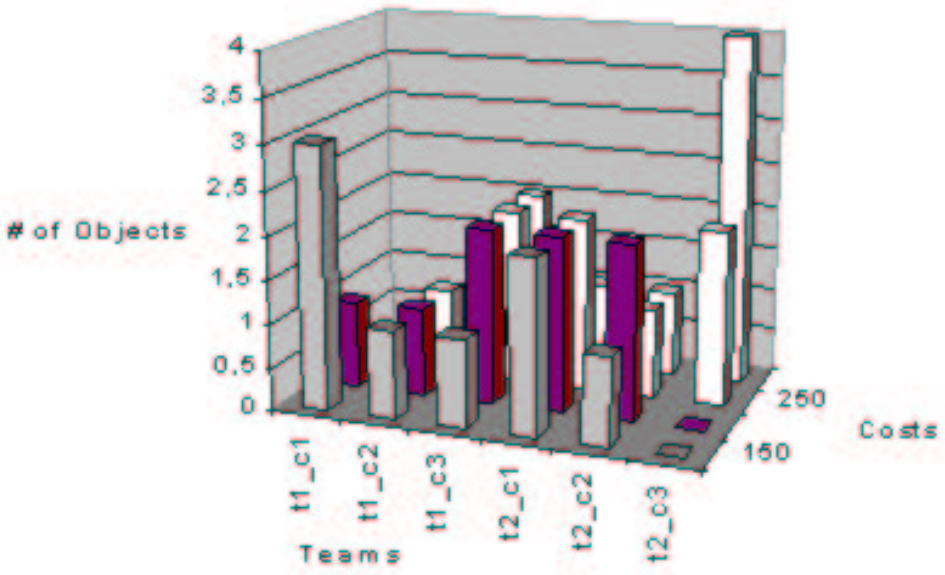


Figure 8: Partitioning results: minimizing Execution Time

ACKNOWLEDGEMENTS

This work has been support by grants from the agencies FAPEMIG, CNPq and PRONEX Finep/CNPq/MCT 76/ 97/1016/00.

References

[1] J. K. Adams and D. E. Thomas. The design of mixed hardware/software systems. In *ACM* 33^{th} *DAC*, Las Vegas, 1996.

[2] V. Basili and D. M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, pages 728–738, Nov 1984.

[3] D. Herrmann et al. An approach to the adaptation of estimated cost parameters in the cosyma system. In *2nd International ACM/IEEE Workshop on Hardware/Software Codesign - CODES'94*, 1994.

[4] G.L. Nemhauser et al., editor. *Handbooks in Operations Research and Management Science: Optimization*, volume 1. North-Holland, 1989. Chapter 8.

[5] J. A. Debardelaben et al. Incorporating cost modeling in embedded-system design. *IEEE Design & Test of Computers*, pages 24–35, Jul-Sep 1997.

[6] M. P. Biswal et al. Probabilistic linear programming problems with exponential random variables: A technical note. *European Journal of Operational Research*, 111:589–597, 1998.

[7] M. Scheffler et al. Modeling and optimizing the costs of electronic systems. *IEEE design & Test of Computers*, pages 20–26, Jul-Sep 1998.

[8] R. Fourer et al. *AMPL: A Modeling Language for Mathematical Programming.* Boyd & Fraser Publishing Company, 1993.

[9] T. M. Khoshgoftaar et al. Using process history to predict software quality. *IEEE Computer*, pages 66–72, Apr 1998.

[10] T. M. Khoshgoftaar et al. Using process history to predict sofware quality. *IEEE Computer*, 1998.

[11] V. Catania et al. Applying fuzzy logic to codesign partitioning. *IEEE Micro*, pages 62–70, May/Jun 1997.

[12] H. I. Gassmann and A. M. Ireland. On the formulation of stochastic linear programs using algebraic modelling languages. *Annals of Operations Research*, 64:83–112, 1996.

[13] D. Geman and G. Reynolds. Constrained restoration and the recovery of discontinuities. *IEEE-PAMI*, 14(3):367–383, 1992.

[14] W. S. Humphrey. *A Discipline For Software Engineering.* Adison Wesley, 1995.

[15] W. S. Humphrey. Using a defined and measured personal software process. *IEEE Software*, pages 77–88, 1996.

[16] W. S. Humphrey. Results of applying the personal software process. *IEEE Computer*, pages 24–31, 1997.

[17] W. S. Humphrey. *Introduction to the Team Software Process.* Adison Wesley, 2000. To appear.

[18] Dieter Hogrefe Jan Ellsberger and Amardeo Sarma. *SDL - Formal Object-Oriented Language for Communicating Systems.* Prentice Hall, 1997.

[19] Claudionor José Nunes Coelho Jr., Diógenes Cecílio da Silva Jr., and Antônio Otávio Fernandes. Hardware-software codesign of embedded systems. In *XI Brazilian Symposium on Integrated Circuit Design*, pages 2–8, Búzios - RJ, sep 1998. IEEE Computer Society.

[20] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, pages 67–74, Sep-Oct 1997.

[21] C. A. Leonhard and J. S. Davis. Job-shop development model: A case study. *IEEE Software*, pages 86–92, Mar 1995.

[22] L. J. Moore and E. R. Clayton, editors. *Gert Modelling and Simulation: Fundamentals and Applications.* Petrocelli / Charter, 1976.

[23] Bernd Oestereich. *Developing Software with UML: Object-Oriented Analysis and Design in Practice.* Addison-Wesley, 1999.

[24] K. Pillai and V. S. S. Nair. A model for software development effort and cost estimation. *IEEE Transactions on Software Engineering*, 23(8):485–497, Aug 1997.

[25] J. Plantin and E. Stoy. Aspects on system-level design. In *7th International ACM/IEEE Workshop on Hardware/Software Codesign - CODES'99*, pages 209–210, may 1999.

[26] E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.

[27] K. S. Trivedi. *Probability & Statistics with Reliability, Queuing, and Computer Science Applications.* Prentice-Hall, 1982.

[28] K. Verschaeve, B. Wydaeghe, V. Jonckers, and L. Cuypers. Translating omt* to sdl, coupling object-oriented analysis and design with formal description techniques. In *Methods Engineering - IFIP WG 8.1/8.2*, 1996.

[29] E. F. Weller. Using metrics to manage software projects. *IEEE Computer*, pages 27–33, Sep 1994.