Staffing a software project: A constraint satisfaction and optimization-based approach

Ahilton Barreto\textsuperscript{a,}\textsuperscript{*}, Márcio de O. Barros\textsuperscript{b}, Cláudia M.L. Werner\textsuperscript{a}

\textsuperscript{a}COPPE-UFRJ, Caixa Postal 68511, Rio de Janeiro, RJ, Brazil
\textsuperscript{b}DIA-UNIRIO, Av. Pasteur 458, Urca, Rio de Janeiro, RJ, Brazil

Available online 7 February 2007

Abstract

Software development is a people intensive activity. The abilities possessed by developers are strongly related to process productivity and final product quality. Thus, one of the most important decisions to be made by a software project manager is how to properly staff the project. However, staffing software projects is not a simple task. There are many alternatives to ponder, several developer-to-activity combinations to evaluate, and the manager may have to choose a team from a larger set of available developers, according to the project and organizational needs. Therefore, to perform the staffing activity with \textit{ad hoc} procedures can be very difficult and can lead the manager to choose a team that is not the best for a given situation.

This work presents an optimization-based approach to support staffing a software project. The staffing problem is modeled and solved as a constraint satisfaction problem. Our approach takes into account the characteristics of the project activities, the available human resources, and constraints established by the software development organization. According to these needs, the project manager selects a utility function to be maximized or minimized by the optimizer. We propose several utility functions, each addressing values that can be sought by the development organization. A decision support tool was implemented and used in an experimental study executed to evaluate the relevance of the proposed approach.

© 2007 Elsevier Ltd. All rights reserved.

Keywords: Staffing; Constraint satisfaction problems; Branch and bound

1. Introduction

Software development organizations survive in a competitive market by profiting from the conversion of developers’ effort to useful and successful software products. To build such products, the organization usually follows a process that divides the development effort into several activities. Each of these activities requires specific characteristics (e.g., such as skills, capabilities, and experience). Most of these characteristics are sought in human resources assigned to accomplish the activities. Thus, human resource allocation, or staffing, is an important issue to be analyzed when software development is undertaken as a value-driven business.

The need to properly staff a project raises a fundamental question: Given a group of available developers and a set of project activities, which developer-to-activity assignment yields more value to the organization? Since human resources usually represent the major cost account for a software project and the alignment of their characteristics...
with activities’ requirements is a driving factor towards productivity and quality, optimizing the allocation of available developers in accordance to the constraints imposed to (e.g., such as schedule deadline, project budget, and head-count) or by (maximum allocation, minimum effort to participate, among others) the development organization may determine whether a project will be profitable or not.

However, staffing a software project is not a simple activity. There are several developer-to-activity combinations to evaluate, since the manager is usually required to choose a team from a larger set of available developers. Moreover, team selection is usually constrained by project and organizational needs, such as maximum team monthly cost, estimated development time, and developer’s capability mismatch to activities’ requirements. For instance, consider a situation where there are 10 professionals available and 10 activities to be accomplished. The total number of available combinations is about 10 billions \(10^{10}\). To perform the allocation activity taking all the attributes involved and all possible solutions into account without automated support is a difficult or even impossible task. Due to this complexity, the manager usually is not able to evaluate all the available combinations and has to choose a team based on his experience and intuition. Since some candidate teams are discarded, the manager may choose a team that is not the best fit for the project.

The main goal of our approach is to help the project manager to staff his projects, suggesting teams that satisfy all constraints involved in the problem automatically. We also suggest teams that not only satisfy the constraints, but also optimize some factor of the problem. In our approach, project staffing is addressed as a constraint satisfaction problem [1], based on utility functions that should be maximized or minimized by the selected development team, in order to provide greater value for the organization. Several utility functions are presented, to be selected by the manager according to organizational needs or constraints.

This paper is divided into nine sections. The first one comprises this introduction. The second section presents some project staffing related work. The third section provides a brief review of constraint satisfaction problems. Section 4 provides some assumptions upon which we draw a model for software project staffing. In Section 5, we present the utility functions currently available to optimize the model presented in Section 4. In Section 6, we show our approach to solve the staffing problem. In Section 7, we show an example situation where our approach could be used. In Section 8, we describe the experimental study that was executed to evaluate the relevance of the approach. Finally, in Section 9, we present some conclusions and future work.

2. Related work

Many approaches have been proposed in the literature to support staffing software projects, using distinct techniques, such as simulation, genetic algorithms, fuzzy theory, among others [2–5]. There are also several approaches related to the wider class of search-based solutions for software engineering problems [6,7], which has become a fast growing area of activity lately [8–11]. Search-based software engineering has proved successful in requirements engineering [12], project cost estimation [13,14], testing [15], among others [16]. In this section, we describe some of these approaches, focusing on the ones related to project staffing.

Duggan et al. [4] describe a task allocation optimizer for software construction. The optimizer uses multiobjective evolutionary algorithms and genetic algorithms [17]. It receives project characteristics and developers’ experiences in different areas related to the activities that have to be performed (network, GUI, databases, requirement elicitation, and so on). Based on models developed by the authors, the algorithm selects teams that are closer to the optimal group that could be assigned to accomplish the activities, regarding the multiple objectives that should be emphasized by the project (like schedule, cost, number of defects, etc.). This task allocation approach provides a basis for generating optimal solutions for projects with conflicting objectives. On the other hand, that approach is strictly related to construction activities and the authors have not explored other important activities that compose a software project, such as planning and requirement management. Moreover, it is not possible to determine whether the solutions selected by the algorithm are the best available for the project at hand, since genetic algorithms are not deterministic.

Schnaider [5] describes a tool that supports human resource planning, monitoring and evaluation. The tool is presented in the context of the enterprise-oriented software development environment [18], a software engineering environment that supports the software development process holistically. The tool stores and distributes organizational knowledge about human resources competences and skills. It allows the manager to define the competences required to perform each activity and shows what professionals in the organization are qualified to perform each one of them, although not considering people availability. It is also possible to plan training sessions in order to improve developers’ competences,
enabling them to perform an activity. If no training is possible or viable, it is also possible to request the hiring of a professional to perform activities for which the available developers are not qualified enough. The manager can also reevaluate the competence levels possessed by each professional throughout the project, according to the professional’s performance (i.e., a person can become more qualified or can be less qualified than expected). Although the tool simplifies the search for information needed to staff the project, it does not suggest how a manager should distribute the available developers among activities. The focus is on process guidance, instead of staffing execution itself.

Collofello et al. [2] present a system dynamics [19] based software development process simulator that can be tuned for specific organizations. The simulator allows the manager to evaluate the effects upon schedule, budget, and quality, derived from the application of distinct staffing policies in different scenarios (what-if analysis). The model takes into account factors such as experience level, productivity, schedule pressure, number of defects, and others. This approach does not directly support assigning developers to activities, but helps to predict the impact of different staffing policies. Dynamic models, such as the one described in [2], are able to precisely describe the relationships among the modeled elements and time. In static models, like the ones described in [4, 5], and even in this work, the relationships never vary throughout time. However, the model described in [2] does not take into account the individual details about each developer, but a global view of the project, where developers’ and activities’ characteristics are described in terms of averages of a discrete number of groups. Moreover, the model is not concerned about any kind of optimization.

Antoniol et al. [16] report an evaluation of three different search-based techniques—genetic algorithms, hill climbing and simulated annealing—and two distinct problem representations for planning resource allocation in large-scale maintenance projects. The search-based approach was applied to find an optimal or near optimal order in which to allocate work packages to programming teams, aiming to minimize project duration. According to the model defined by the authors, project duration could vary with the number of people available and team sizes. So, for a given combination of these two variables, it was possible to estimate the time required to accomplish each project. However, the model does not account for the time required to train new professionals that enter project (as for Brooks Law [20]), nor that some work package may require outputs from other work packages, thus limiting their capabilities to be executed concurrently. The authors show that genetic algorithms found better solutions than the other strategies, though the hill climbing approach also performed well. The approach described in [16], on the other hand, does not account for individual capabilities and productivity, nor tasks dependencies and complexity. It assumes that all activities could be performed concurrently, that all tasks require the same effort to be performed, and that each professional performs any activity with the same productivity rate. Such conditions may not hold for every real project.

Antoniol et al. [21] also depict a queuing theory-based approach to staff software maintenance centers. The authors define a model that balances factors like service levels experienced by the clients, project costs, team size and others. It is shown how the queuing theory allows effective evaluation of staffing level (i.e., number of team members along time), as well as to assess restaffing decisions. Simulation was also used to evaluate the probability to meet project deadlines. Although the work can be useful to try to determine team sizes, it does not account for individual capabilities and productivity. Like [16, 21] considers only maintenance projects and not development projects.

Ruskova [22] presents a fuzzy logic based system for human resource selection and evaluation. The model is divided into three parts. The first one describes the evaluation of job positions, where requirements for each job position are defined. The second part deals with candidate appraisal, conveying a description for each candidate (characteristics, competences possessed, among others). The third part of the model describes how the former two parts are combined to suggest a list of candidate professionals that better fulfill the requirements of available job position. A tool that implements the model is also presented. This approach, however, does not address project staffing itself, but the similar problem of selecting people from a limited pool according to the requirements of a job position. So, it does not consider people availability or date constraints that usually exist in project staffing.

3. Constraint satisfaction problems

Many artificial intelligence and engineering problems, such as job shop scheduling [23], can be categorized as constraint satisfaction problems. There are many applications for solutions that address this kind of problem, in many distinct areas, such as graphical computing, database systems, molecular biology, commercial applications, electrical engineering, circuits’ design, resource allocation, and so on.

A constraint satisfaction problem can be formulated as a set of $N$ variables, $X_1, X_2, \ldots, X_n$, that can assume values from predefined domains $D_1, D_2, \ldots, D_n$, respectively. A value $v_i \in D_i$ assigned to a variable $X_i$ is called an
A very efficient heuristic, presented in [28], is often used along with the forward checking algorithm. This heuristic is to search space through backtracking and every time a partial solution is found, its cost is calculated and compared to the available solutions. An algorithm that can solve this problem is branch and bound [30]. This algorithm navigates the solution space as a tree, where each node represents a value assignment to a variable in the context of previous value assignments, which are represented in nodes in the current node’s path to the tree root. For large problems, conveying a large number of variables, each with a domain containing a huge number of values, the total search space is very large. Ultimately, if the algorithm is used to find all available solutions in this search space, it is as costly as a brute-force solution that spans all possible assignments for each variable and checks whether each solution breaks the required constraints. Therefore, some extensions were proposed in the literature to enhance the basic backtracking algorithm.

As a first improvement, it must be noted that backtracking performs some unnecessary steps, like when a value assignment to a variable makes it impossible to assign values to another variable that is further in the assignment line. In this situation, the basic backtracking algorithm will continue to assign values until it reaches the affected variable, thus noticing that a solution based on that previous assignment is not available. Therefore, a better solution would be to detect a wrong search branch as early as possible, minimizing the unnecessary assignments.

A strategy to improve backtracking performance is to use forward checking [27]. This algorithm proposes that, every time a value is assigned to a variable, the values that can no longer be assigned to the other variables are deleted from their respective domains. This is accomplished by testing the constraints that rely upon each variable and value assignment in isolation. If any domain becomes empty, in other words, if no assignment can be done to one or more variables that compose the problem under interest without breaking the constraints, the algorithm backtracks and tries another search branch. Forward checking usually runs faster than basic backtracking and is often easy to implement.

Another important issue is the order in which the variables are selected for assignment by the backtracking algorithm. Experimental studies from several researchers show that this order may have a substantial impact in the search effort. A very efficient heuristic, presented in [28], is often used along with the forward checking algorithm. This heuristic states that variables with less available values in their domain should receive more priority for assignment. Thus, the assignment order is not statically defined when the algorithm starts, but changes as every assignment is made and the forward checking algorithm limits the available values in each domain. This heuristic was called search-rearrangement method or most constrained variable.

We have been considering situations in which at least one solution can be obtained. However, in many important applications the problem may be over constrained, in a way that a complete solution is not possible. In this case, partial solutions can still be useful if a satisfactory number of the most important constraints are not broken. An example of such partial constraint satisfaction problem is the maximal constraint satisfaction problem (MAX-CSP), in which the goal is to find assignments of values to variables that satisfy the maximum number of constraints [29].

Finally, sometimes finding a viable solution is not enough: if there is a need to determine the best solution available, a prioritization among the solutions must be defined. This prioritization is usually described as a function that associates a cost to a solution. To determine the best solution is either to find the solution that presents the lowest cost among the available solutions. An algorithm that can solve this problem is branch and bound [30]. This algorithm navigates the search space through backtracking and every time a partial solution is found, its cost is calculated and compared to the
cost of the best solution found so far. If the partial solution cannot improve the current best one (that is, if its cost is higher than the current best solution’s), the search branch is abandoned and the algorithm backtracks to the previous variable.

4. Modeling staffing as a constraint satisfaction problem

Software project staffing must be done in a way that maximizes the creation of value for a project. The semantics of value depends on project and organizational characteristics and, thus, may vary across several projects. Some projects are schedule driven: to create value for such projects could mean to reduce the time required to develop the project or risks associated to its schedule. Other projects may be driven by budget, resource allocation, and so on. Therefore, the maximization target for the staff allocation problem cannot be a fixed utility function, but several such functions should be available for the manager to choose, according to the driving factors for the project under analysis [31].

In our approach, we assume that the manager is able to identify the characteristics that a team member might have to accomplish each project activity. Moreover, there is also a need to determine the characteristics possessed by each developer in the development organization. A characteristic possessed by a developer denotes a skill, a capability, an experience, some knowledge, a role in the organization, a role in the project, and so on. Each characteristic is associated with a rating scale representing the intensity in which the characteristic can be observed in a developer or required by an activity. Quantifying such information (even in a nominal scale, as proposed) may not be easy for every project or development organization, but may also be an important step towards having more knowledge about its developers and projects. Some research has been done to determine capability-person and capability-role relationships [32], but this issue is out of the scope of this work. Furthermore, we will assume that developer’s characteristics and activities requirements are well known. Based on this information, staffing is performed according to the following rules:

- (α) A developer can only be allocated to an activity if he or she possesses all the characteristics required by the activity, in an intensity greater or equal than required.
- (β) A developer can only be allocated to an activity if he or she is available to perform the activity in the period it needs to be performed (reasons of unavailability could be allocation to other activities, vacation, and so on).

We have investigated the staffing problem as a constraint satisfaction problem. By using the formulation presented in Section 3, that describes a constraint satisfaction problem as a tuple of three finite sets \( S = (V, D, R) \), we define \( V \) as the set of activities that compose the project, \( D \) as the set of professionals that can perform each activity, and \( R \) as the set of constraints (qualification and availability constraints). Expressing the staffing problem in a formal way:

- Let \( A \) be a set of activities from a given project. Each \( A_i \in A \) is described by a name, initial and final schedule dates for the activity, minimum daily effort that should be spent by a developer while working on the activity (expressed in hours), and a set of required characteristics \((CA_i)\) that should be shown by a developer in charge of the activity:
  \[
  A_i = [\text{name}, \text{iniDate}, \text{endDate}, \#\text{hours}, CA_i].
  \]

- Let \( HR \) be a set of professionals available to accomplish project activities. Each \( HR_k \in HR \) is described by a name, cost (per hour), maximum number of daily workable hours, a set of periods in which the developer will not be available to take part in the project \((PU_k)\), and a set of characteristics possessed by the developer \((CHR_k)\)
  \[
  HR_k = [\text{name}, \$\text{hour}, \#\text{hours}, PU_k, CHR_k].
  \]

- Let \( PU_k \) be a set of periods in which a given developer \( HR_k \) will not be available to take part in the project. Each \( PU_{kn} \in PU_k \) is described by initial and final dates, and the number of hours per day that the professional will be unavailable:
  \[
  PU_k = \{PU_{kn}\},
  PU_{kn} = [\text{iniDate}, \text{endDate}, \#\text{hours}].
  \]

- Let \( C_i \) be a characteristic possessed by a professional \( HR_k (CHR_k) \) or required by an activity \( A_i (CA_i) \). Each \( C_i \) is described by a name, a scalar value that denotes a maximum intensity for such characteristic among software
developers or required by activities, and its intensity for a specific developer or required by a given activity:

\[ CHR_k = \{ C_i \} \]
\[ CA_i = \{ C_i \} \]
\[ C_i = [\text{name}, \#\text{maximal}, \#\text{intensity}] \]

Based on these sets, the staffing problem may be described as follows:

\[ S = (V, D, R) \text{ where } V = A, D = HR, \text{ and } R \rightarrow \forall A_i \in A(\exists H R_k \in H R(\alpha \land \beta)). \]

The formulation for \( R \) shows that for each activity belonging to a given project's set of activities, there is some professional for whom both \( \alpha \) and \( \beta \) conditions hold. The \( \alpha \)-condition is described as follows:

\[ \alpha \leftarrow \forall C_m \in CA_i \subset A_i(\exists C_n \in CHR_k \subset HRk(\phi)), \]
\[ \phi \leftarrow \text{Name}(C_m) = \text{Name}(C_n) \land \text{Intensity}(C_m) \leq \text{Intensity}(C_n). \]

The condition above establishes that for each characteristic that belongs to the set of characteristics required by an activity, there must be the same characteristic in the set of characteristics possessed by a professional, so that the professional possesses the characteristic in greater or equal intensity than required by the activity. The \( \beta \)-condition is described as follows:

\[ \beta \leftarrow \neg \exists PU_{kn} \in PU_k \subset HR_k(\text{INTERSECTS}(A_i, PU_{kn}) \land (#\text{Hours}(HR_k) > (#\text{Hours}(PU_{kn}) + #\text{Hours}(A_i))), \]
\[ \text{INTERSECTS}(A_i, PU_{kn}) = \begin{cases} (\text{iniDate}(A_i) \geq \text{iniDate}(PU_{kn})) \land (\text{iniDate}(A_i) \leq \text{endDate}(PU_{kn})) & \lor \\ (\text{endDate}(A_i) \geq \text{iniDate}(PU_{kn})) \land (\text{endDate}(A_i) \leq \text{endDate}(PU_{kn})) & \lor \\ (\text{iniDate}(A_i) \leq \text{iniDate}(PU_{kn})) \land (\text{endDate}(A_i) \geq \text{endDate}(PU_{kn})) \end{cases}. \]

The \( \beta \)-condition establishes that there should be no period over which the developer will be unavailable that intersects with the period in which the activity will be accomplished. It also states that the developer's daily workable hours available to perform the activity should be greater than the minimum daily work hours required by the activity. So, a professional can work on an activity that is in the same period of its unavailability (that could be caused by his allocation to another activity), but only if the number of hours the employee works per day allows him to perform an activity even though he is unavailable for some hours in the same day.

We also assume that an activity is a small task, performed by a single developer in a very short time span. Thus, we do not account for developers that have a break during an activity execution, or activities performed by more than one professional. In such cases, we recommend the manager to divide the activity into smaller tasks, allocating different developers or distinct time windows for their execution.

Productivity is also taken into account in our approach. It can vary from professional to professional and can influence on the amount of time required to complete project activities. We use a productivity modifier to determine each professional's productivity. The greater the modifier, the faster the professional is able to accomplish the activity. In this work, we offer the project manager four ways to deal with the productivity modifier:

- **Ignore productivity**: All professionals will have the same productivity modifier. Therefore, all teams finish the project spending the same time.
- **Productivity based on experience**: The productivity modifier will be calculated based on the qualification required by the activity and the qualification possessed by the professional. Professionals who are more qualified in the characteristics required by the activity will have increased productivity, and will be able to finish the activity earlier.
- **Use a global productivity modifier**: The manager may determine a productivity modifier for each professional, which will be used to determine the productivity of the professional in all activities.
• **Use an activity productivity modifier**: The manager may determine a productivity modifier for each professional in every activity of the project, assuming that professionals produce at different rates depending on the activity they are performing.

Productivity influences the values of the $\beta$-condition. When a productivity model is selected, the time required for each developer to perform an activity is estimated according to the model. Since activities may depend on each other to begin their execution, a change in an activity duration may affect the period in which the following activities are expected to be executed. This downward propagation of a developer’s productivity can be observed in the software industry: the effects of higher productivity are not limited to the current task’s duration, but may affect the whole project schedule, since further activities could be started sooner than expected. Thus, start and conclusion dates for project activities are recalculated and the $\beta$ condition shall be reevaluated to determine whether there are available developers to execute the proposed activities. Thus, the $\beta$ condition indirectly accounts for developers’ productivity.

5. **Staffing utility functions**

To account for the characteristics required by activities and those possessed by developers is not always enough for managers to decide the best human resource allocation for a project. Several other values may be used to distinguish among alternative development teams and could be optimized according to project or organizational characteristics. For instance, a project may require the most qualified and skilled team available, accounting for aspects such as productivity, client satisfaction, product reliability, time-to-market and so on. On the other hand, the best usage of the skills available in an organization may be more important, and these skills should not be wasted. Therefore, a manager may choose the team that minimizes skills’ over usage (in other words, where the distance between the needed and employed skills is minimal). In a different scenario, a manager may need to choose the cheapest team that can execute the project, according to developers’ salaries and overhead costs. Sometimes large teams are a problem, since the number of communication channels maintained among developers may affect the team productivity. Therefore, the manager may want to choose the smallest possible team to the project. A third scenario may be observed when a project has to be finished as earlier as possible. To do so, a manager may choose the professionals with better productivity rates, so that the product can be delivered as soon as possible. Finally, though accounting for the characteristics required by project activities is important, sometimes it is not possible to get a team whose members possess all the characteristics required by such activities. Thus, a manager may choose to minimize the distance between what is needed and what the team possesses, building a team as close as possible to the required characteristics.

Considering such factors, our approach suggests that the manager might select a utility function to be maximized by the constraint satisfaction problem solver. Moreover, acknowledging that the importance of such factors may differ from project to project, we propose several utility functions, from which the manager could select one according to project characteristics. In the following, we present some of these functions:

• **Most qualified team**: Team in which the distance between the characteristics required by project activities and those possessed by the team is maximal. In other words, the optimizer will suggest a team in which developers are more capable than project activities require them to be:
  ○ Let $T$ be the set of possible teams that satisfy all the constraints. Define $Overusage(T_k)$ as a function that returns the distance between the qualification of a given team $T_k$ and the qualification required by the project. The selected team $T_i \in T$ is chosen according to the following equation:

  $$\neg \exists(T_j \in T)(Overusage(T_j) > Overusage(T_i)).$$

• **Cheapest team**: The optimizer selects the team that represents the lowest cost to the project, considering a fixed cost for each team member throughout the project:
  ○ Let $T$ be the set of possible teams that satisfy all the constraints. Define $Cost(T_k)$ as a function that returns the cost (monetary) of a given team $T_k$ to the project. The selected team $T_i \in T$ is chosen according to the following equation:

  $$\neg \exists(T_j \in T)(Cost(T_j) < Cost(T_i)).$$
• **Least qualified team**: Team in which the distance between the characteristics required by project activities and those possessed by the team is minimal. In other words, the optimizer will search for a team in which skill over usage is minimized:

- Let \( T \) be the set of possible teams that satisfy all the constraints. Let \( \text{Overusage}(T_k) \) be a function that returns the distance between the qualification of a given team \( T_k \) and the qualification required by the project. The selected team \( T_i \in T \) is chosen according to the following equation:

\[
\neg \exists (T_j \in T) (\text{Overusage}(T_j) < \text{Overusage}(T_i)).
\]

- **Smallest team**: The optimizer suggests a team composed by the smallest possible number of members:

- Let \( T \) be the set of possible teams that satisfy all the constraints. Let \( \text{Size}(T_k) \) be a function that returns the size (number of different members) of a team \( T_k \). The selected team \( T_i \in T \) is chosen according to the following equation:

\[
\neg \exists (T_j \in T) (\text{Size}(T_j) < \text{Size}(T_i)).
\]

- **Faster team**: The optimizer suggests a team composed by the members that can finish the project earlier. This utility function depends on the productivity model selected by the manager, which dictates the time required for a developer to accomplish an activity. So, the optimizer searches for a solution (that is, a team) that minimizes the time spent to finish the project:

- Let \( T \) the set of possible teams. Let \( \text{Duration}(T_k) \) be a function that returns the number of days a team \( T_k \) takes to finish the project. The selected team \( T_j \in T \) is chosen according to the following equation:

\[
\neg \exists (T_j \in T) (\text{Duration}(T_j) < \text{Duration}(T_i)).
\]

- **Best partial solution team**: The optimizer indicates a team that is the best partial solution to the problem. This solution is usually applied when the available developers’ characteristics do not satisfy all requirements of the project activities. So, the optimizer searches for a solution (that is, a team) that minimizes the number of broken constraints:

- Let \( T \) be the set of possible teams. Let \( \text{Broken}(T_k) \) be a function that returns the number of broken constraints of a team \( T_k \). The selected team \( T_i \in T \) is chosen according to the following equation:

\[
\neg \exists (T_j \in T) (\text{Broken}(T_j) < \text{Broken}(T_i)).
\]

Other utility functions could also be defined and used, according to project and organizational specific needs. The ones described in this section are examples of how a manager can optimize different factors when staffing a software project.

### 6. Solving the project staffing problem

The staffing problem’s complexity is influenced by the number of professionals (\( p \)) and the number of activities (\( a \)). Considering these factors, the problem complexity is \( O(p^a) \). However, when constraints are added to the problem formulation, the effort needed to find the solutions tend to decrease enormously, due to the number of paths in the search space that do not attend the constraints or have less utility (if optimizing) than the best solution so far. If the algorithm can detect, as soon as possible, that it is searching in a path that will not lead to a better solution, the time needed to find a compatible solution tends to be much smaller. To allow for such improvement, we used the techniques presented in Section 3.

To offer the manager the decision support to staff his projects, we have implemented an algorithm that receives characterization data from professionals and activities, along with the (optionally) chosen utility function for optimizing the staff allocation. The algorithm returns a set of teams that satisfy all the imposed constraints and, if a utility function is selected, optimize the selected utility function. Fig. 1 shows an overview of the project-staffing algorithm.

The implementation of the algorithm uses backtracking, forward checking, and most constrained variable. We also use branch and bound to optimize the factors involved in the problem, when a utility function is selected by the manager.

The first step to solve the problem is to determine the initial domains for the variables, that is, to identify what professionals (values) are able to accomplish each activity (variables). Another important step is the selection of the
next activity to which a professional might be assigned. In this step, we apply the most constrained variable heuristic [28], which indicates that the algorithm should select the activity that can be accomplished by the smallest number of professionals as the next variable for assignment.

The main part of the proposed solution consists in determining who will accomplish a certain activity. Once the initial domains are set and the first activity for assignment is chosen, it is possible to start choosing candidate developers who can accomplish the activity. Our implementation uses a recursive algorithm that runs until all activities have an assigned professional or it is not possible to find a compatible solution.

To support the application of the proposed algorithm, we have developed a tool that is able to present the manager with different staff possibilities:

- **Any N solutions**: The tool identifies N teams (or all) that satisfy all the constraints.
- **Optimum problem**: The tool identifies all teams that satisfy all the constraints and selects the best team, according to a given utility function.
- **Optimum partial problem**: The tool identifies all partial solutions that break the smallest number of constraints possible.

**Fig. 2** shows an output screen from the tool. The screen presents the teams that satisfy the requirements of a set of project activities and maximize the cheapest team utility function. The tool allows the manager to maintain information describing the professionals (along with their competences) and the activities (along with their requirements). It also lets the manager to choose the productivity model for each project and to setup its parameters. Finally, it is possible to
import entry information and export results calculated by the tool in a XML based file format, thus allowing the staffing algorithm to be integrated to other tools and environments. In the current version, utility functions can only be added or changed through internal implementation. However, the tool was designed to be particularly susceptible to changes that add new utility functions. The source code is public, and can be obtained in the authors’ homepage. The code can be freely used and modified.

7. Exemplified usage of the approach

To illustrate the use of the constraint satisfaction staffing algorithm, we present a fictitious project composed by seven activities and a set of professionals available to perform these activities. Table 1 presents the characteristics possessed by professionals or required by activities and theirs domains.

In Fig. 3, we show an activity network that depicts the dependences among project activities. Each activity is depicted as an ellipsis, being linked to a rectangular note that presents the characteristics that should be possessed by

Table 1
Characteristics used in the example and their domains

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Domains</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric value</td>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>Relationship with people</td>
<td>1</td>
<td>Was trained on the subject</td>
</tr>
<tr>
<td>Negotiation</td>
<td>2</td>
<td>Has ability</td>
</tr>
<tr>
<td>Team work</td>
<td>3</td>
<td>Has great ability</td>
</tr>
<tr>
<td>Requirements elicitation techniques</td>
<td>1</td>
<td>Knows and can perform under supervision</td>
</tr>
<tr>
<td>Object oriented analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Databases</td>
<td>2</td>
<td>Knows and can perform without supervision</td>
</tr>
<tr>
<td>Object oriented design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td>3</td>
<td>Is an expert</td>
</tr>
<tr>
<td>Tests techniques</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experience in telecommunications</td>
<td>1</td>
<td>Between 2 and 6 months</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Between 6 months and 1 year</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Between 1 and 3 years</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>More than 3 years</td>
</tr>
</tbody>
</table>

Fig. 3. Activities network of the fictitious project.
a professional in order to accomplish the activity. Such characteristics are required to a given level, shown in parenthesis beside the characteristic’s name. The activities have to be executed from left to right, following the arrows. We consider an activity cannot start until all its predecessors’ activities are completed. As shown in the figure, only construction activities can be performed concurrently.

In Fig. 4, we present the available professionals that may join the development team. Each potential team member is presented with its name, its hourly working cost, and its possessed characteristics. As for the characteristics describing the needs of an activity, the intensity with which a characteristic is possessed by a professional is presented within parenthesis. We can see that developers’ cost per work hour tends to increase as their qualifications increase, as it happens in real world projects.

In Fig. 5, we present the number of professionals that can perform each of the activities, according to their characteristics and those required by the activities.

According to the activities and professionals shown in Figs. 2 and 3, it is possible to generate 2160 teams that satisfy all the constraints. If determining a team that does not break any constraint is a reasonably difficult task without any automated support, it should be much more difficult to identify and analyze all possible solutions, accounting for factors like cost, size, and time, among others. So, even in such a simple situation (seven tasks and seven developers), we can observe that performing the staffing activity without support can be very difficult.
Table 2
Variation of factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Min. value</th>
<th>Max. value</th>
<th>Δ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>39 days</td>
<td>46 days</td>
<td>17.9</td>
</tr>
<tr>
<td>Cost</td>
<td>$10,360</td>
<td>$26,360</td>
<td>154.4</td>
</tr>
<tr>
<td>Suballocation index</td>
<td>91</td>
<td>223</td>
<td>145.0</td>
</tr>
<tr>
<td>Size</td>
<td>4 people</td>
<td>7 people</td>
<td>42.9</td>
</tr>
</tbody>
</table>

Table 2 summarizes how the factors involved in the problem vary in the example. It is possible to notice how far from the best solution, considering one of these factors, a manager could get. For example, when trying to find a team that satisfies all the constraints and where the cost is minimal, a manager could, in the worst situation, choose a team $16,000 (or 154.4%) more expensive than the best one. And even if the manager is not trying to find the cheapest team, to know what the smallest amount of money he could spend is can be useful for him to determine a range into which the project cost should be.

8. Empirical study

Common wisdom, intuition, speculation, and proofs of concepts are not always reliable sources of credible knowledge. On the contrary, progress in any discipline involves building models that can be tested, through empirical study, to check whether the current understanding of the field is correct. Progress comes when what is actually true can be separated from what is only believed to be true [33]. So, experimentation plays a very important role in software engineering.

Thus, in order to get some indication about the relevance of the approach proposed in this work, we have planned and executed an empirical study. The goal of the study was:

*To analyze the execution of the staffing activity in software projects with the purpose of evaluating the relevance of using the proposed constraint satisfaction solution with respect to the gains obtained by its use from the point of view of the researcher in the context of Software Engineering post-graduate students.*

Subjects were asked to act as a project manager executing the staffing activity. Their objective was to select the cheapest team to perform a fictitious project defined in laboratory (*in vitro* study) by using *ad hoc* procedures or whatever technique they decided (except for the constraint satisfaction technique proposed in this paper). The project described in Section 7 was presented to all subjects and an extra constraint was added: the project should be concluded on a given deadline. After their selection, the teams chosen by the subjects were compared with the teams chosen by the tool described in Section 5.

Our null hypothesis stated that the use of the proposed approach and its supporting tool would bring no benefit to staffing a software project, i.e., there would be no significant difference in project schedule or cost whether the chosen teams were selected by using *ad hoc* procedures or the proposed approach. The alternative hypothesis, on the other hand, stated that subjects using *ad hoc* procedures would get suboptimal results, that is, teams that would either conclude the project later than the best available team or would consume more financial resources to develop the project.

The study was executed in four sessions, with varying number of participants per session, depending on their availability. Each session took about an hour. The total number of participants was 16 (sixteen). The participants were all M.Sc. or D.Sc. students, with software development experience in both academic and industrial projects. Some of them were also experienced in project management and staffing activities.

The following procedure was used in every study execution session. First, the participants were asked to fulfill a characterization questionnaire, to provide information on their formation and experience. This characterization was used to group participants when analyzing results of the study. Next, the researcher made a description of the proposed project and all subjects received a written briefing explaining that they should play the manager’s role while staffing the project. The briefing also presented the list of activities that composed the project and available professionals that could be selected to form the development team. Then, subjects were introduced to a software tool that allowed them to indicate which developer should accomplish each activity composing the proposed project. Given a list of developer to activity mappings, the tool calculated the resulting project schedule and cost. This tool was built specially to serve
as an instrument for the experimental study and was also useful to:

(i) Log all the teams chosen by each developer throughout the process, providing information for future investigations on how people apply *ad hoc* procedures to staff a software project. The tool allowed the subjects to try as many distinct teams as required, before selecting a definitive team as their best allocation for the project.

(ii) Avoid the need for manual calculations of project schedule and cost that should be made by the subjects to determine if changing some developer to activity mappings would yield a better result (according to time and cost) for the project. Such manual calculations tend to be error-prone, thus imposing a construction threat for the experimental study. Building the tool to automatically calculate such information provided us protection against this kind of error.

The teams found by the subjects are represented as triangle in Fig. 6. They were compared with all possible teams that could be assembled from the available developers and that attend to the deadline constraint, including the best solution, which was found by using the proposed approach and its supporting tool. Such alternative solutions are presented as dots in Fig. 6.

The results of the study were analyzed through a qualitative analysis, once the goal of the study was not to prove in an irrefutable way its hypothesis, but to get an indication if the use of the proposed approach might be useful for a project manager who needs to staff a software project.

The first analyzed aspect was the process used by the participants to find a suitable solution to the staffing problem, i.e., which intermediary solutions were found, in which sequence and in how much time was required to solve the problem. It was observed that it was hard for the subjects to account for two factors (time and cost) simultaneously, though this is a common decision to be made by managers in real projects. Once many subjects were able to find a team that could complete the project in the required time (42 days), they soon stopped trying to find the best cost. We came to that conclusion by analyzing the log generated by the tool that subjects used during the study: we considered the number of attempts before and after each subject found a team that completed the project in 42 days. So, the proposed approach might be useful to managers that are required to staff software projects according to more than a single factor at a time (for instance, schedule and cost).

We also grouped the results in different ways, to have an indication of other factors that could influence the results of the study. We analyzed each group with respect to its average distance from the best solution, considering cost (the difference between the cost of the selected team and the cost of the cheapest teams). We focused on cost because only one subject found a solution with project duration different from the expected.

One of the factors that we have investigated was subject experience. We grouped the results by subject formation (separating D.Sc. Students from M.Sc. Students) and by experience in project staffing in the industry. The separated
classes and their averages are represented in Figs. 7 and 8. Fig. 7 shows that, in average, D.Sc. students chose teams that were closer to the best team than M.Sc. students. It can be considered an indication that academic experience has some influence on the results. We have evaluated this hypothesis statistically by using an ANOVA test, but due to the limited sample size it can be verified only with a $p$-value of about 22% (that is, 78% certainty).

Fig. 8 shows that the participants with some practical experience in project staffing chose (in average) teams with lower $\Delta$Cost, i.e., with a smaller deviation from the best solution. It can be an indication that experience in performing the task in real projects influences the results. So, subject experience (both academic and in performing the staffing activity) had an influence on the results. More experienced subjects presented better results than less experienced ones. Thus, there is an indication that the support might be more useful to people with less experience. Again, we have evaluated this hypothesis by using an ANOVA test and it can be verified with a $p$-value of about 20% (that is, 80% certainty).

We also wanted to investigate if the effort to execute the activity had influence on the results, so we grouped the results by the time spent to perform the study by the subjects, as shown in Fig. 9. We can see that as the time spent increases, $\Delta$Cost decreases, i.e., subjects that spent more time in the execution of the activity chose better teams (in average) than the others. It can be considered an indication that effort to execute the activity influences the results. So,
the effort to execute the activity also had an influence on the results. Thus, there is an indication that the support might be more useful in situations where a decision has to be made quickly. Using an ANOVA test, we have verified that the average $\Delta$Cost for the group that invested less than 15 min to select the project team is higher than the average $\Delta$Cost for the group that invested more than 25 min to the same task. Such conclusion can be drawn with a $\rho$-value of about 12% (that is, 88% certainty).

Finally, only 25% of the subjects were able to find the best team according to the proposed project constraints (cheapest team that is able to conclude the project in 42 days). Therefore, we observe that even in such a simple project, with few activities and available professionals, it was not easy for the subjects to find the best solution. Even the ones who found the best solution took a long time to find it. So, we consider that the alternative hypothesis of the study was satisfactorily confirmed, that is, there was a considerable difference between the results obtained by the tool and the ones obtained by the subjects.

A fundamental question concerning results from an experiment is how valid such results are. In the following paragraphs we address distinct types of threats that could invalidate the experimental results, along with treatments that were applied to each threat.

- **Internal validity**, i.e., the assurance that a treatment causes an outcome [34], is dependent on the number of subjects taking part on the study. We consider that the number of subjects in the study was adequate for our needs, but more subjects would allow a better level of internal validity. Since the study was executed in several sessions, another threat to internal validity could be the exchange of information among subjects that had already executed the study and subjects that would yet participate. To avoid it, we asked the former to avoid talking about the study. We also tried to run the study in a small number of sessions, but were limited by subjects’ agendas. Internal validity could also be threatened by the way the variables involved in the problem (e.g., productivity, capability level possessed by a professional, capabilities required to perform an activity) are measured, since these variables are usually difficult to precisely measure and often involve a certain level of subjectivity. However, in this study, all the subjects used the same pre-assigned values for these variables and assigning values to them was not part of the problem. Therefore, we believe this factor did not have an impact on the results.

- **External validity** is concerned with generalization [34]. The major threat to external validity in our study would be the lack of interest of the subjects in the study. Some of the participants could execute the study in an uncompromised way, without a real interest in finding a good team, in contrast to what would probably happen in the industry. We gave complete knowledge to our participants that experimental results would not imply any sanctions or personal evaluations for them. Moreover, we kept the results private. However, since the study was run in an academic environment, it cannot be generalized to industrial settings at present. Indeed, it was planned as a first evaluation of the proposed techniques, aiming to encourage the planning and execution of further studies in industry that could be more conclusive.

- **Construct validity** is concerned with the relation between theory and observation, i.e., to evaluate whether the treatment well reflects the construct of the cause and whether the outcome well reflects the construct of the
effect [34]. In this study, the same project was used in both treatments, assuring that the treatment reflects the cause.

• Conclusion validity is concerned with finding a relationship between the treatment and the outcome [34]. In our study, the same project was used by the researcher and by the subjects, allowing us to believe that the differences found in project cost and schedule reflect the use of each treatment.

9. Conclusion

Software development involves time, talent, and money. In a competitive market, a software development organization’s major goal is to maximize value creation for a given investment [35]. Therefore, a proper usage of every available resource in a software project is very important.

Software designers, engineers, and managers must understand and reason effectively about the connections between technical decisions and enterprise-level value maximization while executing their work [35]. In our approach, we address these connections by accounting for technical needs (characteristics possessed by people and required by project activities) in the light of business constraints (cost, schedule, time size, among others). By offering a variety of value (utility) functions to drive the technical-oriented staffing optimization process, we allow the manager to balance between project technical needs and organizational constraints.

To support the operational usage of our approach, we developed a tool in which the manager informs the characteristics required by project activities, the characteristics of the available developers, and selects one of the proposed utility functions. The tool runs an optimization algorithm (based in constraints satisfaction and branch-and-bound techniques) and generates a project staffing suggestion according to the given parameters.

We also described an empirical study, which was executed to evaluate the relevance of the proposed decision support system. The experimental results provide indications that the proposed approach can effectively support a project manager in staffing activities, especially if the manager is not experienced in such tasks or is time-pressed to select the project team.

The proposed approach is expected to generate more accurate results when the available estimates are good. Since activity dates, professional’s availability and productivity strongly influence on the suggested teams, the precision of such estimates may have a relevant impact on the results. However, a manager typically staffs projects based on the available estimates, even if they are not so accurate. So, we believe our approach would be useful to support the staffing activity, even with imprecise estimates.

Our approach considers that an activity is a task small enough to be performed by a single developer. We acknowledge that in large projects managers may not plan to that level of detail, but we consider that managers or team leaders will refine the tasks presented in the macroplan to the execution level, that is, such tasks will be decomposed until they can be assigned to a developer or small team. This fine-grained definition for an activity was chosen due to the availability of single developer productivity models [19]; as for today, we are not familiar with models that describe development productivity when heterogeneous groups work together to perform an activity.

In the present research, we chose to use a simplified software project model to focus our efforts on the search and optimization strategies. However, in the near future, particularly after having stronger feedback on the usage of our approach, we intend to refine this project model, possibly addressing dynamic aspects, as in system dynamics project models [3]. This kind of model is able to precisely describe the relationships among the modeled elements and time. In static models, like the one described in this work, the relationships cannot change throughout time. The usage of dynamic models will also allow us to account for aspects that cannot be covered by static models, such as developer motivation, developer exhaustion, learning curve, error propagation, and so on. We will also be able to consider what-if scenarios, to predict the impact of some events (a team member leaves the organization or becomes unavailable, productivity rates are very different from estimates, a team member is added to the team, among others) upon project relevant variables (such as project schedule and cost).

In the present research, we chose to use a simplified software project model to focus our efforts on the search and optimization strategies. However, in the near future, particularly after having stronger feedback on the usage of our approach, we intend to refine this project model, possibly addressing dynamic aspects, as in system dynamics project models [3]. This kind of model is able to precisely describe the relationships among the modeled elements and time. In static models, like the one described in this work, the relationships cannot change throughout time. The usage of dynamic models will also allow us to account for aspects that cannot be covered by static models, such as developer motivation, developer exhaustion, learning curve, error propagation, and so on. We will also be able to consider what-if scenarios, to predict the impact of some events (a team member leaves the organization or becomes unavailable, productivity rates are very different from estimates, a team member is added to the team, among others) upon project relevant variables (such as project schedule and cost).

We also intend to offer some support to determine the characteristics required to perform an activity and the characterization of available professionals. To do so, we intend to provide knowledge to the manager on such tasks, using data from the literature [32] and from further researches. Another possible evolution could be the application of multidimensional optimization, as proposed in [4], so that different factors (like time and cost) can be combined into a single utility function.
References


