Feeding Dynamic Behavior into Optimization: Simulation-based Multi-objective Overtime Planning

Márcio de O. Barros*, Luiz Antonio O. de Araujo Jr†
Federal University of the State of Rio de Janeiro
Av. Pasteur 458, CEP 22.290-240 Rio de Janeiro, Brazil
{marcio.barros*, luiz.araujojr†}@uniriotec.br

***Abstract*—IT professionals are particularly subjected to working overtime due to difficulties in estimation, requirement specification, and measuring progress during a software development life-cycle. However, excessive overtime has detrimental effects on developers and the software they produce. This contrast creates the need for overtime-planning strategies to support project managers in taking the most from as few overtime hours as possible. In this paper we introduce a formulation for the overtime-planning problem which extends the state-of-art by taking into consideration both the positive effects of overtime on productivity and its negative effects on product quality. We use heuristic optimization to explore close to optimal overtime allocations under this formulation. We present an empirical study that compared our approach with current industrial practices and a similar formulation without negative effects. Results show that suppressing the flip-side of the gains brought by overtime may lead to wrong decisions. For instance, excessive allocation of overtime in large projects may lead a manager to underestimate project cost and makespan by 5% and 3%, respectively, due to longer testing activities required to fix the exceeding errors introduced by overtime. Evidence also supports an overtime-management strategy used in the industry that concentrates overtime in the second half of a project schedule to compensate for early delays - about 60% of the overtime allocations produced under this strategy are optimal.**

***Index Terms*—Overtime planning; project management; simulation; schedule optimization.**

I. Introduction

Software engineers are professionals particularly subjected to working overtime, frequently without previous planning and in stressful, fire-fighting situations [1][2][3]. Difficulties associated to measuring the progress of a software project, late and volatile requirements, compressed schedules, and the to shrink the time-to-market of new features [4] are among the factors that contribute to the need of working overtime.

Excessive overtime affects both the life of developers and the software they produce. On regard of the developers, Nishikitani et al. [5] have observed that overtime work in the IT sector presents positive correlation with depression, anger, and hostility indexes. They also observed an increase in equilibrium- and motor-related problems on Japanese workers from the IT sector subjected to long shifts [6]. Regarding the outcome of developers' work, DeMarco [7] and Akula and Cusick [8] relate overtime work with higher error generation rates in software projects. DeMarco and Lister [26] also relate excessive overtime with increased personal attrition, which may reduce knowledge exchange amongst developers [9] and hamper the ability of a company to hold domain expertise and properly deal with developer turnover [10].

Problems related to overtime are not exclusive to the IT sector. Caruso [11] identified negative impacts of long work hours on American workers (fatigue, lack of alertness, weight gain, and health degradation), their families (delayed marriage and childbearing), and their employers (absenteeism due to illness, increased errors, and rework). The author also reminds about the extra cost incurred in order to identify and correct errors committed due to exhaustion (loss in quantity and quality of the goods produced and services provided by workers subjected to excessive overtime). Donald et al. [12] also found negative correlation between stress and productivity. Finally, overtime problems do not seem related to a specific culture, affecting Americans [11][13], Brazilians [14], Japanese [6], [15], and Chinese [3], to cite a few.

Despite the recent growth in works on search-based approaches to software project management [16], there is a single previous study that addresses techniques to handle overtime planning [17]. The approach uses a multi-objective genetic algorithm to identify the activities providing the best leverage in terms of reducing project makespan and risk of schedule overrun for an amount of overtime work. However, the former paper does not take account of the negative impact of overtime and the consequent rise in errors produced by tired developers. Previous studies [7], [8] have presented empirical evidence that this is an important factor: the work required to fix the exceeding errors may subdue the gains in schedule brought by working overtime, as well as increase project cost more than expected by stakeholders. The results presented in this paper confirm that ignoring the loss in work quality may lead project managers to incorrect decisions. Furthermore, the present paper finds empirical support for a practice currently used by the industry for overtime allocation.

This paper extends the state-of-the-art by taking into account the documented impact of overtime on the quality of the services provided by developers working in extended hours. It introduces a new formulation for the overtime planning problem which combines search-based optimization with a scenario-based simulation that takes into account an increase in the number of errors injected into a software product by a team working overtime. We compared our results to current practices on regard of overtime management and provide insights to support project managers on deciding when to use overtime work. By doing this, we found scientific support for an overtime-management practice used in the industry, that is, evidence aligned with project management intuition. Our results show that, on average, 60% of the allocations that might be built according to this strategy are optimal. We also compared our results with a similar model that ignores the dynamics of error generation caused by overtime, showing that wrong decisions may be taken if the increase in error generation rates is not taken into account. For instance, excessive allocation of overtime in large projects may lead a manager to underestimate project cost and makespan, respectively, by 5% and 3% due to longer testing activities required to fix the exceeding errors introduced by overtime. The primary contributions of this paper are:

1) We introduce a new formulation for the multi-objective overtime-planning problem that considers an increase in error generation rates due to fatigue. Our approach takes into account trade-offs among the number of overtime hours worked in a project, makespan, and cost;
2) We present an empirical study based on 6 real-world projects with up to 635 function-points in size. Results show that concentrating overtime hours in the second part of project schedules (a practice used in the industry) is a good strategy that can be improved by optimization;
3) We present evidence that the negative impact on product quality caused by overtime must be taken into account while selecting the overtime allocation for a project. A strategy that may be optimal if such impact is disregarded may not be the best choice if it is considered.

This paper is divided into eight sections, starting with this introduction. Section II presents our formulation for the overtime-planning problem, as well as the simulator used to capture the dynamics of error propagation throughout the software development process. Section III presents the proposed solution for the overtime-planning problem. Section IV presents the design of the experimental analysis carried out to evaluate the proposed solution. Section V presents the analysis and results collected from the experimental study. Section VI addresses validity threats that may limit the generalization of experimental results. Section VII presents related work and Section VIII shows conclusions and future research directions regarding overtime planning.

II. Problem Formulation

The innovation in our overtime-planning problem (OPP) formulation on respect of previous works relates to capturing the dynamics of error generation and propagation throughout the sequence of activities comprising a software project. This allows estimating the effects of an increase in error generation rates due to excessive overtime and fatigue. To this end, we devised a strategy to build a project schedule from a set of work packages and use continuous simulation to emulate the dynamics of error propagation as the schedule is executed.

We depart from a set of work packages, each to be analyzed, designed, coded, and tested. A project is represented as an acyclic directed graph <WP, WPDEP> consisting of a node set WP = $\{wp_1, wp_2, ..., wp_n\}$ of work packages and an edge set WPDEP = $\{(wp_i, wp_j)$: $i\neq j, 1\leq i\leq n, 1\leq j\leq n\}$ of precedence dependencies amongst work packages. Each wp_i is described by the amount of effort required to elicit its requirements, the amount of effort to design it, code it, and test it as if these tasks were performed by a mean developer in a regular work environment. A work package is also described by the number of errors its requirements analysis, design, and coding activities might generate in a similar context. On Section II.B we discuss how these parameters were filled for our instances. Each dependence in WPDEP represents a work package (wp_i) whose development can only start after the requirements for a second work package (wp_j) are known.

From these work packages, we develop a project schedule. Our schedule is a partial order of software development activities on which four types of activities are created for each work package: requirement analysis, design, coding, and testing. A testing activity is always preceded by a coding activity, which is preceded by a design activity, which comes after a requirement analysis activity. These dependencies represent the basic workflow of modeling the problem, designing the solution, implementing its source-code, and testing its features. The schedule also includes dependencies amongst work packages: the requirements analysis activity for work package wp_i cannot start before the requirements analysis activity for wp_j is concluded for all (wp_i, wp_j) pairs in WPDEP.

A project schedule is represented as an acyclic directed graph <ACT, ACDEP> consisting of a node set ACT = $\{a_1, a_2, ..., an\}$ of activities and an edge set ACDEP = $\{(a_i, a_j)$: $i\neq j, 1\leq i\leq m, 1\leq j\leq m\}$ of finish-start (FS) dependencies amongst activities. A FS dependency A$\rightarrow$B indicates that work on activity B can only start when activity A is finished. Each activity a_i is classified as either requirement analysis, design, coding, or testing. Analysis, design, and coding activities are described by the expected effort required to produce its results and the expected number of errors they might generate if performed by a mean developer in a regular environment. Testing activities are described by the average effort required to identify and correct an error, which is collected from information provided in work packages. We consider three objectives for overtime planning: the number of overtime hours (NOH), project makespan (MKS), and overall project cost (CST). OPP is formulated as a three objective decision problem in which makespan, cost, and overtime hours are conflicting objectives to be minimized. A candidate solution for the OPP is a sequence of numbers that represents the amount of daily overtime hours to be spent on each activity comprising the schedule. Our formulation is based on a 40-hours work week, accepts multiples of 30 minutes for overtime work, and limits the maximum amount of daily work to 12-hours turns, according to Brazilian law. Thus, for any given activity we may assign zero overtime, 30 minutes of overtime, one hour, one hour and 30 minutes, and so on up to four hours of daily overtime. Our first objective, NOH, is calculated as the sum of the amount of overtime hours assigned for each activity in the schedule.

The <ACT, ACDEP> graph is comprised of a set of paths Π denoting the precedence-respecting sequences of activities that must be executed in order to complete the project. Based on these paths, the makespan of a project (MKS) is calculated by Equation (1). The duration of a given activity is calculated by a simulator according to the characteristics of the activity and its execution setup (including the number of daily overtime hours assigned to it). The simulator calculates the number of errors produced by each development activity and propagates these errors to succeeding activities. For testing activities, the simulator calculates the effort required to identify and correct all errors received from former activities. The makespan of a project is given by the longest path in Π, which is known as the critical path.

$$MKS = MAX_{p \in \Pi}(\sum_{a_i \in p} Duration(a_j))$$

The last objective, project cost (CST), is the sum of each activity's cost. The cost of an activity depends on the number of regular work hours and overtime hours its developers consume to produce its expected results and is related to the duration calculated by the simulator. Again, we modeled the problem according to Brazilian laws: if a regular working hour costs X, each of the first two overtime hours costs 120%.X and the next two overtime hours cost 150%.X each.

A. Simulation Machanism

Simulation is the imitation of the behavior presented by a real-world process or system over time [18]. It supports predicting and understanding the behavior of complex systems, containing hundreds or thousands of interconnected components. Simulation is useful because the number of components and the complexity of the data and control exchange among them usually hampers our ability to interpret system behavior by intuition or analytical reasoning.

Simulation has been used to support understanding the behavior of large software systems. Continuous [19] and discrete-event [20] simulation are the most used mechanisms to simulate software projects, though other mechanisms, such as state-based [21] and rule-based [22] simulation, have also been tested. Nonetheless, even the most frequently used simulation mechanisms have limitations, such as the inability to separate general from situation-specific behavior [23], difficulties to separate the parts of a model related to a given component, to model simultaneous consumption of limited resources by two or more clients (continuous simulation), and to model state changes unrelated to an event (event-driven).

To overcome some of these problems, we have designed an object-oriented, continuous-time simulator to model the effects of overworking in software projects and predict project behavior (cost and makespan) accordingly. The simulator is based on three constructs: simulation objects, resources, and scenarios. Instead of creating a specific syntax to represent these elements, we relied on the Java programming language to describe and execute them. Designing the simulator in Java also simplified its integration to the heuristic search framework we have used in our experimental study.

Simulation objects are the elements whose behavior we want to simulate. They represent the nodes of a directed and acyclic graph whose edges denote precedence dependencies amongst simulation objects. They are initialized before the simulation starts and thereafter kept dormant until all their dependencies have been processed. Then, they become alive and start participating in the simulation until signaling their conclusion to the simulator. Simulation ends when all objects presented to the simulator are finished.

Time is managed in a continuous fashion. Simulation starts at time zero and is comprised of consecutive simulation steps, each advancing the simulator's clock by an infinitesimal increment. At each step, all alive simulation objects are allowed to update their internal state. The internal state of an object is defined according to the relevant characteristics of the real-world element it represents in the simulation. The behavior of an object is thus represented by the changes observed in its internal state during the simulation.

Simulation objects are Java objects that implement an interface defining a protocol through which they are controlled by the simulator. The internal state of an object is kept in its instance variables. Internal state adjustment operations are coded in methods activated by the simulator: *init()*, to initialize the simulation object at time zero; *start()*, called when all dependencies for an object are finished; *step()*, called at each simulation step; and *finish()*, called when the object signals its conclusion.

During a simulation step, simulation objects may use resources. A resource has a limited amount of materials or capabilities that can be consumed by simulation objects. These materials or capabilities are refilled before each simulation step. An object may request part of the resource's stock and adjust its internal state accordingly to its availability. The simulator controls the usage of limited resources by different simulation objects running in parallel. To model a software project, simulation objects represent an activity network with FS dependencies. The developers available to work on activities are resources whose amount of work is limited to a certain number of hours per day and is renewed in a daily basis. Developers are assigned to activities and if two or more activities executing in parallel share the same developer, only one of them will use the developer's capabilities at a time.

Scenarios are used to represent complementary behavior that may be imposed to certain simulation objects. A scenario is written to influence a certain type of simulation object and only participates in the simulation if activated on one or more objects of that type. Scenarios can query and update the internal state of their associated objects before and after each event on which the objects update their internal state (*start*, *step*, *finish*). Scenarios can be used to represent managerial decisions imposed to a subset of a software project's activities. They are reusable model components that modify the behavior of their related simulation object according to the policy they represent. For the purpose of the OPP, a scenario was developed to represent the effects of overtime while a base model describes the expected behavior for a project, its activities and developers as if uninfluenced by managerial policies and decisions (such as overtime allocation).

B. The Base Model

The base simulation model is comprised of activities and developers. Activities are simulation objects linked to each other by precedence dependencies. An activity is available for simulation as soon as its precedent activities are concluded. Developers are resources who can spend a limited amount of work on each simulation step. They are characterized by their productivity and error generation rates. Activities are separated into development and testing activities. Development activities are described by an expected amount of work to be performed (in developer-hours) and an amount of errors expected to be generated while this work is performed. The duration of a given activity depends on the amount of work to be performed and the productivity of the developer assigned to it. We calibrated the model using data from Jones [24]. Given a work package measured in function-points (FP), Jones [24] presents average distributions of effort required for each development phase. For instance, for projects of about 1,000 function-points, requirement analysis activities consume 7% of project effort, design activities consume about 10%, coding activities consume about 30%, and testing consumes 30%. The remaining effort is related to management, documentation, and other activities. In our model, this effort is proportionally distributed among the analysis, design, coding, and testing activities. Given this distribution, the size of a work package, and an average productivity of 27.8 function-points per developer-month [24], we calculate the expected duration for each activity. Error dynamics is trickier. When an activity starts, it collects all errors produced by its former activities (if any) carrying them toward the end product. Abdel-Hamid and Madnick [25] present a model for error reproduction which states that errors inherited by an activity are not only forwarded to succeeding activities, but they contribute to the generation of new errors. Jones [24] suggests that the average number of defects introduced by each kind of activity is 1 error/FP in requirement analysis, 1.25 errors/FP in design, and 1.75 errors/FP in coding activities. Based on these data, our model assumes that each activity introduces 1 new error/FP, design activities regenerate 25% of errors it receives from analysis, and coding regenerates 33% of the errors received from design. These regeneration rates lead to the mark of 4.0 errors/FP on average after the implementation and are compatible with the former model [25] and source [24]. Testing activities receive the results of development activities and aim at identifying and correcting the errors introduced in the product. These activities are characterized by the average effort required to identify and correct an error. Their duration depends on this parameter along with the assigned developers productivity. Given the amount of work expected to be invested in the testing phase [24] (i.e., 30% of total effort for 1,000 FP projects) and the expected number of errors reaching this activity in a regular situation (4 errors/FP), we calculate the average time required to identify and correct each error and use this value on testing activities.

C. A Scenario for Overwork

The scenario used in our simulation describes the dynamics of working overtime and its influence on developer's productivity and error generation rates. If a developer is expected to work more than 8 hours per day in a given activity, his productivity will be increased proportionally to the extra number of work hours, reaching up to a 50% increase if working to the upper limit of 12 hours per day. However, according to de Marcos law [26], a developer working more than the regular daily hours may produce software below the quality standard. The more hours a developer works, the more tired he becomes. Tired developers become less careful to the work they are performing and, consequently, they tend to introduce more errors in the software. We have used data from [25] to estimate the increase in the number of errors produced according to the number of hours worked by a developer. Such data suggest that software quality tends to deteriorate if developers work more than 9 hours/day and that a developer working 12 hours/day commits 50% more errors than while working on a regular shift.

III. Solution Approach

Our solution uses the NSGAII algorithm for optimization along with the simulator described in Section II. NSGAII [27] is a multi-objective genetic algorithm based on a ranking procedure which classifies candidate solutions according to their dominance. A candidate solution is described by the values calculated for the objectives of interest under that solution. In our case, these objectives are the number of overtime hours, project makespan, and project cost, all to be minimized. A solution A is said to dominate a solution B if the values of all objectives in A are no greater than the respective objectives in B and at least one objective is smaller in A than in B. Non-dominated solutions are assigned a rank of 1; solutions dominated only by non-dominated solutions are assigned a rank of 2; solutions dominated only by the former are assigned a rank of 3, and so on. The algorithm evolves a population over a number of generations, applying crossover, mutation, and selection upon candidate solutions. The selection process prioritizes low-ranking solutions and, when a subset of solutions having the same rank must be selected, a density measure allows selecting candidate solutions covering the search space as uniformly as possible. As any heuristic algorithm, NSGAII has components that must be specified for a given problem. Three components are discussed below, while population size and stop criteria were subject of analysis and are discussed in the next section.

Representation. A candidate solution for OPP is a sequence of m integer numbers, m being the number of activities. Each number in the solution pertains to the [0..8] integer interval and represents a multiple of 30-minutes overtime for a given activity. Thus, if the number 4 is assigned to activity A, A will receive 1.5 hours of overtime work. Each individual solution being evolved by NSGAII is evaluated through simulation. The simulator activates the overtime scenario upon each activity in the schedule according to the number of overtime hours assigned to that activity in the solution. Then, the simulator calculates the number of overtime hours, project makespan, and project cost for that solution.

Crossover. We reused the crossover operator proposed by Ferrucci et al. [17]. This operator was adopted because it was shown to outperform the one-point crossover operator commonly used in NSGAII for their formulation for the overtime planning problem. Instead of exchanging genetic material of two parents P_1 and P_2 after a point of cut C $\in$ [1..m], the resulting offspring O_1 and O_2 produced by this operator are defined by the equations below (given a random variable p with uniform distribution). The operator is executed with 50% probability; otherwise, the parents are copied to the next generation. Parents are selected by binary tournament.

$$O_1(g) = \begin{cases} P_1(g) & 1 \leq g < C \\ max(P_1(g), P_2(g)) & C \leq g \leq m, p < 0,5 \\ min(P_1(g), P_2(g)) & C \leq g \leq m, p \geq 0,5 \end{cases} \tag{1}$$

$$O_2(g) = \begin{cases} P_2(g) & 1 \leq g < C \\ P_1(g) + P_2(g) & C \leq g \leq m \end{cases} \tag{2}$$

Mutation. As in Ferrucci et al. [17], mutations are executed for all crossover offspring. Each gene of an offspring O is changed with 30% probability, being assigned a new value in the [0..8] integer interval.

IV. Exeperimental Design

This section explains the design of our empirical study, presenting research questions, the instances selected to address these questions, and the measurements used to compare solutions found under different configurations and algorithms.

A. Research Questions

In this section we present the research questions addressed by our study. They were designed to provide evidence on the validity, competitiveness, and usefulness of our approach.
RQ1 (Validation): How does NSGAII perform if compared to random search on finding solutions for OPP? This is a "sanity check" to determine if our formulation allows a systematic search to outperform random search. If such cannot be verified, either the formulation is poorly designed, parameters are not adequate, or the problem is simple enough to allow a non-systematic procedure to find good solutions;
RQ2 (Competitiveness): How does NSGAII perform if compared to currently used overtime planning approaches? Besides beating random search, the proposed approach has to outperform current management practices related to overtime planning. In this sense, we compare the results produced by NSGAII with overtime planning practices used in the industry;
RQ3 (Usefulness): Do results of overtime planning change if we consider the dynamics of error generation and the loss of quality caused by overtime? To address this issue, we compare the results produced by NSGAII with and without considering an increase in error generation due to overtime. Different results denote that a manager may pick a distinct overtime planning strategy if error dynamics are not considered.

B. Software Projects Used in the Study

Table I presents information about the selected instances for the experimental analysis. The Transaction Functions and Data Functions columns count, respectively, the number of transaction and data functions for each instance. These values are related to the number of function points of the instance, presented in the last column. The Work Packages column presents the number of groups on which the data and transaction functions distributed to form the project schedule.

TABLE I
Instances used in the experimental studies

Name	Transaction Functions	Data Functions	Work Packages	Function Points
ACAD	39	7	10	185
WMET	48	7	11	225
WAMS	67	8	15	381
PSOA	65	9	18	290
OMET	129	22	21	635
PARM	98	21	27	451

ACAD is an academic administrative system that allows managing classes, students, registrations, and teachers in an university setting. WMET allows users to add meteorological observations to a database and provides weather-related reports. WAMS is a message routing system responsible for receiving and routing air traffic control messages. PSOA is a personnel management project used by a large company for authentication, authorization, and centralization of references from enterprise systems. OMET is a system which manages, stores, and delivers meteorological information, including messages and statistical information regarding the weather. PARM stores configuration and setting values used by other systems, allowing for fast configuration of user profiles and sharing configurations amongst distinct applications. ACAD was developed to support post-graduate programs in a Brazilian university and has been in use for more than 10 years. WMET, WAMS, and OMET were developed and are currently used to support air traffic control in Brazil. PARM and PSOA are used by the IT company in charge for government-managed retirement funds in Brazil.

C. Multi-Object Evaluation Measurements

A comparison between two or more mono-objective heuristic search algorithms can rely on the values found by these algorithms for the fitness function guiding the optimization. However, a similar comparison for multi-objective algorithms cannot rely directly on objective values because each algorithm provides a curve of non-dominated solutions (a Pareto front) as its results. Therefore, comparing different algorithms implies comparing different Pareto fronts.

To compare two Pareto fronts F_A and F_B we need surrogate measures of their quality. These measures are called quality indicators. A quality indicator receives a set of points defined in the objective space and produces a real number. The result for a given front F_A can be compared to the result for a second front F_B, leading to conclusions about the quality of these fronts and, consequently, the algorithms and configurations that produced them. Quality indicators usually compare each front with a 'reference front' F_R built from the non-dominated solutions of all Pareto fronts under comparison. The ideal reference front would be the optimal set of non-dominated solutions for a given instance, but these are rarely available for practical purposes. Thus, the union of all available fronts followed by the elimination of duplicate and dominated solutions is used as a proxy to the ideal reference front.

Given F_R and two fronts to be compared a number of quality indicators can be defined. They usually address two properties of Pareto fronts: convergence and diversity. Convergence indicators measure how close a given front F_A is to the reference front. Diversity indicators measure the amount of the solution space that is covered by a given front F_A. For the purpose of our study, we selected three indicators: Contributions (I_C), Hypervolume (I_{HV}), and Generational Distance (I_{GD}). These indicators are measured in the [0, 1] interval. I_C is a convergence indicator that represents the proportion of solutions in a given front F_A that also pertain to the reference front. Good Pareto fronts have high I_C, thus contributing heavily to the composition of F_R. I_{HV} calculates the volume of the objective space that is covered by solutions of a given Pareto front [28]. It mixes convergence to the reference front and diversity amongst the selected solutions comprising a given front. High-quality Pareto fronts present high hypervolume, thus covering a large extension of the objective space. Finally, is a convergence indicator that measures the distance between a front and the reference front [29]. Good fronts are closer to the reference front and thus present low generational distance.

D. Parameter Setting for the Algorithm

Section III presented the selected configuration for most components of NSGAII in order to fine-tune the algorithm to the OPP. However, two components were not addressed in that section because they required experimental analysis to be defined: the stop criteria, expressed in terms of a maximum number of fitness function evaluations NSGAII is allowed to perform while searching for solutions, and population size, that is, the size of the genetic pool evolved by NSGAII as a multiplier to the number of activities in a given instance.

To determine the most adequate stop criteria, we executed NSGAII for all selected instances with a maximum of 5.000, 10.000, 20.000, and 50.000 fitness evaluations. To properly account for the randomness inherent to heuristic search algorithms, the optimization was executed 50 times for each configuration and instance. Fifty thousand fitness evaluations was considered as an upper limit due to the time required to process each optimization cycle, particularly due to the cost of running the simulation on every fitness function evaluation. Afterwards, we built a reference front for each instance based on non-dominated solutions collected from all 50 cycles of all configurations under which the instance was optimized. Then, we calculated for the front resulting from each optimization cycle based on the reference front. This led to a data frame containing 50 observations of for each instance and configuration. Being bound to the [0..1] interval, quality indicators data cannot be normally-distributed and thus cannot be subjected to parametric statistical inference test. Thus, hereafter we will only use non-parametric tests. A Kruskal-Wallis non-parametric inference test was executed upon this data and found significant differences between configurations at 95% significance level. A post-hoc analysis based on pairwise comparisons through a Wilcoxon test using Bonferroni correction followed to find that there were significant differences among all configurations and the configuration using 50.000 fitness evaluations as stop criteria yielded smaller I_{GD} for all instances. Thus, we used 50.000 fitness evaluations as stop criteria for all following analysis.

A similar procedure was used to determine the most adequate population size for each instance. First, we executed NSGAII for all instances with population sizes of 2m, 4m, and 8m, m being the number of activities for the instance and using 50.000 fitness evaluations as stop criteria. The optimization was executed 50 times for each configuration and instance, a reference front was built for each instance from the results of all 50 cycles, and we calculated for the front resulting from each cycle, ending up with a data frame comprising 50 observations of for each instance and configuration. A Kruskal-Wallis test was executed upon this data on an instance basis and found significant difference between configurations at 95% significance level. The post-hoc analysis revealed significantly different results for all configurations on all instances and the configuration using 2m as population size yielded the best average for all instances. Therefore, we selected 2m as population size for all following analysis.

V. Analysis and Discussion

Below we present results obtained from our experiments to support answering the proposed research questions. **RQ1:** To compare NSGAII to random search (RS), we ran 50 independent optimization cycles for both algorithms using the same stop criteria. NSGAII achieved higher I_{HV}, higher I_C, and lower on all six instances compared to RS. Wilcoxon-Mann-Whitney tests applied on an instance basis found significant differences between results produced by the algorithms at 95% significance level for all quality indicators and all instances. Standardized effect-sizes (non-parametric Vargha-Delaney Â12 test) also indicate that NSGAII produces higher I_{HV} and lower in 100% of the runs, while producing higher I_C for more than 88% of these runs. Thus, there is strong evidence that NSGAII outperforms RS while searching for solutions for the OPP and that our formulation and parameterization passes the ”sanity check” implied in the first research question.

RQ2: To address this question we compared our formulation for the OPP with three overtime management strategies (OMS) presented by Ferrucci et al. [17] as reference practices from the industry: (i) ”margarine” management (MAR) suggests spreading overtime hours evenly over all activities comprising the project schedule; (ii) Critical Path (CPM) suggests loading overtime onto the schedule's critical path to reduce completion time and risk of schedule overrun; and (iii) Second Half (SH) suggests loading overtime hours onto the later part of the project's schedule to compensate for delays introduced in early activities.

The objectives of interest (NOH, MKS, and CST) were calculated for each OMS. Nine configurations were created for CPM, each loading a certain amount of daily overtime, from zero to four hours in 30-minute intervals, onto the critical path. The objectives were calculated for each configuration and a Pareto front was built from non-dominated solutions. A similar approach was taken for MAR and SH. The front produced for each OMS was compared with fronts produced by NSGAII over 50 independent optimization cycles on the basis of the quality indicators presented in Section IV.C.

A summary of values collected for each quality indicator and instance is presented in Table II. The distributions of these values are shown in the box-plots presented in Figure I. Data for NSGAII is shown in the form of mean ± standard deviation over the 50 cycles. Data for OMS is shown as a single value, since each OMS produced a single Pareto front for each instance. I$_{HV}$ and for CPM on instances PARM and PSOA are not calculated because the OMS produced a Pareto front comprised of a single vertex, leading to undefined and I$_{HV}$. Bold face OMS values were found significantly different from NSGAII using Bonferroni-adjusted Wilcoxon-Mann-Whitney inference tests at 95% significance level. Bold face NSGAII values were found significantly different from all OMS using the same inference test.

On regard on I$_C$, SH contributed with more vertices to the reference Pareto front than the average front produced by each cycle of NSGAII in 4 out of 6 instances. For the remaining instances, results slightly favor NSGAII, but without statistical significance in comparison with SH. On average, SH contributed with 5.2 solutions to the reference front, while NSGA-II contributed with 2.4 solutions. Each remaining OMS contributed with a single solution to the front (zero overtime). On regard of I$_{HV}$, NSGAII produced the best results except for the largest instance (PARM), on which it was significantly outperformed by SH. Finally, on regard of NSGAII outperformed all OMS in 4 out of 6 instances, losing to SH on the OMET and PARM instances (the largest instances). For the latter, all solutions produced by SH were part of the reference front. Thus, we ratify the results presented by Ferrucci et al. [17] in the sense that the CPM and margarine OMS are not competitive with optimization results. On the other hand, SH seems competitive and outperforms NSGAII in terms of closeness to the reference front on large instances.

The Â12 standardized effect-size was calculated for pair-wise comparisons between NSGAII and each OMS for all quality indicators. Results are presented in Table III, denoting the chances of NSGAII to produce higher IC and I$_{HV}$ and lower than the related OMS. Â12 is very high and favors NSGAII in almost all instances for I$_{HV}$ and (except PARM for I$_{HV}$, OMET and PARM for I$_{GD}$). It strongly favors NSGAII in terms of IC if compared to MAR and CPM, except on instance PARM (a draw on effect-size among both OMS and NSGAII). On the other hand, Â12 tells a different story for I$_C$ between NSGAII and SH, clearly favoring SH for almost all instances.

Effect-sizes are in agreement with former results indicating that MAR and CPM are not effective overtime management strategies. Moreover, effect-sizes provide additional evidence on the differences between NSGAII and SH: if there is enough

TABLE II
QUALITY INDICATOR VALUES FOR EACH INSTANCE UNDER OUR PROPOSED APPROACH (NSGAII) AND THE THREE SELECTED OMS. PARETO FRONTS PRODUCED UNDER NSGA-II PRESENT MORE DIVERSITY THAN THOSE PRODUCED UNDER THE SELECTED OMS. SOLUTIONS PRODUCED UNDER THE SECOND-HALF OMS ARE CLOSER TO OPTIMAL FOR THE LARGEST INSTANCES.

Instance	Config	I$_C$	I$_{HV}$	I$_{GD}$
ACAD	NSGAII	0.0193 0.0211	**0.4262 0.0063**	**0.0052 0.0013**
	MAR	0.0085	0.1318	0.1649
	SH	**0.0339**	0.3514	0.0153
	CPM	0.0085	0.0071	0.0127
WMET	NSGAII	0.0196 0.0186	**0.4196 0.0051**	**0.0034 0.0005**
	MAR	0.0062	0.1892	0.2612
	SH	0.0188	0.3464	0.0125
	CPM	0.0062	0.1502	0.0930
WAMS	NSGAII	0.0197 0.0158	**0.4123 0.0032**	**0.0028 0.0003**
	MAR	0.0056	0.1900	0.2613
	SH	0.0168	0.3470	0.0118
	CPM	0.0056	0.1357	0.0831
PSOA	NSGAII	0.0194 0.0141	**0.4534 0.0094**	**0.0091 0.0032**
	MAR	0.0094	0.0368	0.0918
	SH	**0.0283**	0.3600	0.0439
	CPM	0.0094	n/a	n/a
OMET	NSGAII	0.0186 0.0125	**0.3550 0.0047**	0.0042 0.0003
	MAR	0.0080	0.1897	0.2557
	SH	**0.0720**	0.3469	**0.0000**
	CPM	0.0080	0.1637	0.1079
PARM	NSGAII	0.0173 0.0162	0.2949 0.0054	0.0080 0.0011
	MAR	0.0147	0.1358	0.1108
	SH	**0.1324**	**0.3503**	**0.0000**
	CPM	0.0147	n/a	n/a

TABLE III
EFFECT-SIZE FOR COMPARISONS BETWEEN NSGA AND EACH OMS IN TERMS OF EACH QUALITY INDICATOR.

QI	OMS	ACAD	WMET	WAMS	PSOA	OMET	PARM
I$_C$	MAR	67%	75%	83%	71%	78%	50%
	SH	17%	41%	53%	29%	0%	0%
	CPM	67%	75%	83%	71%	78%	50%
I$_{HV}$	MAR	100%	100%	100%	100%	100%	100%
	SH	100%	100%	100%	100%	96%	0%
	CPM	100%	100%	100%	n/a	100%	n/a
I$_{GD}$	MAR	100%	100%	100%	100%	100%	100%
	SH	100%	100%	100%	100%	0%	0%
	CPM	100%	100%	100%	n/a	100%	n/a

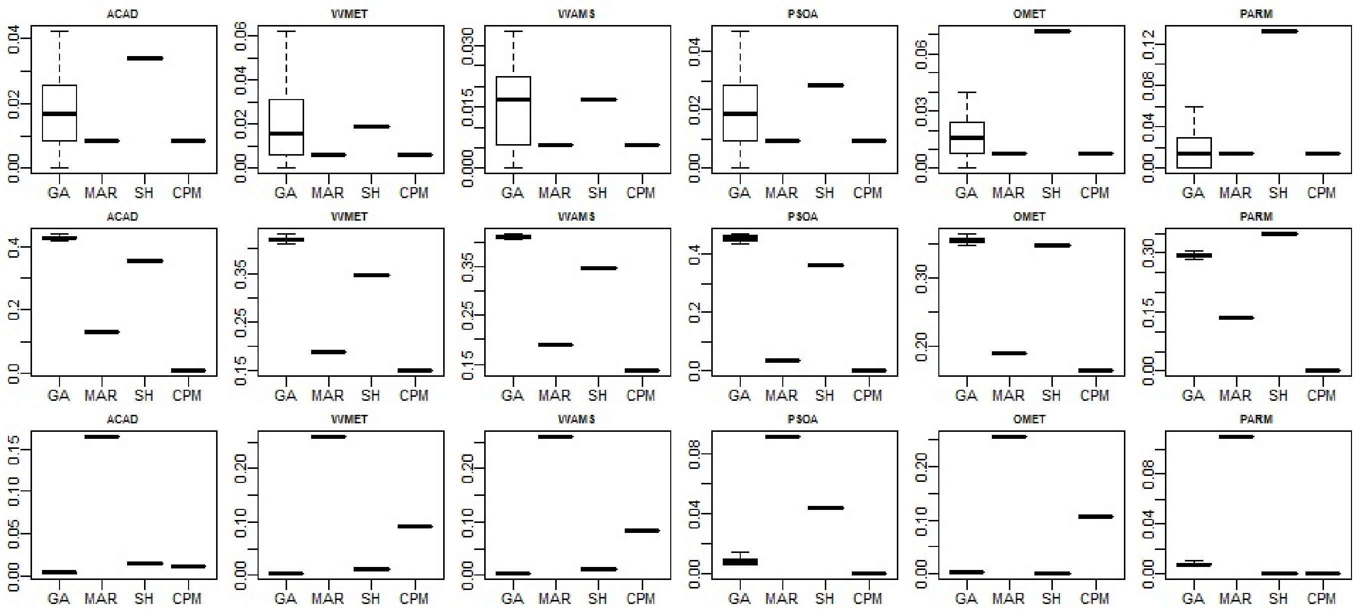

Fig. 1. Boxplots for I_C (top), I_{HV} (middle) and I_{GD} (bottom) quality indicators for nsgaii (shortened to ga) and the oms strategies. The few solutions found under the second-half oms are closer to optimal, though nsga-ii produces a more diverse set of solutions.

time to run the optimization/simulation process, NSGAII can produce Pareto fronts which present more diversity than those resulting from SH, but the latter can quickly provide a good approximation (limited to a few points) for the reference front.

Thus, the industry has good reason for concentrating overtime hours on the second half of project schedules, but it can improve its practices by combining SH with our proposed approach in order to provide more overtime allocation alternatives for the decision maker. **RQ3:** To address this question we compared our approach (NSGAII) with a similar formulation without the increased error generation rates caused by working overtime (NSGA$_{NE}$). We aim to observe if ignoring the increase in error generation rates would change a decision-makers' perceptions on which overtime planning alternative should be selected for a project. Table IV presents summary values for the quality indicators calculated over 50 optimization cycles for each algorithm. All values are presented in the mean $\pm$ standard deviation format. Complete box-plots were suppressed due to space limitations. Bold face values are significantly different compared to the other algorithm, as ascertained by Wilcoxon-Mann-Whitney tests at 95% confidence level.

Â12 effect-sizes for each quality indicator are presented in Table V. It can be observed that NSGAII consistently presents higher I_{HV} (good) and higher (bad) compared to NSGA$_{NE}$. On regard of I_C, the algorithms are competitive, with small effect-sizes favoring NSGA$_{NE}$. These results are explained by the larger number of solutions produced by NSGA$_{NE}$: its Pareto fronts have on average 1.8 times the number of solutions of fronts calculated for NSGAII, reaching up to 3.9 times this number for the largest instance (PARM). The larger number of solutions allows NSGA$_{NE}$ to cover parts of

TABLE IV
QUALITY INDICATORS FOR EACH INSTANCE UNDER OUR APPROACH (NSGAII) AND A MODEL WITHOUT ERROR DYNAMICS (NSGA$_{NE}$). NSGA$_{NE}$ PRODUCES PARETO FRONTS THAT COVER A LARGER PART OF THE SEARCH SPACE, LEADING TO SMALLED . SOLUTIONS PRODUCED UNDER NSGA-II TEND TO CONCENTRATE IN CERTAIN PARTS OF THE SEARCH SPACE

Instance	**Config.**	$\mathbf{I}_C$		$\mathbf{I}_{HV}$		$\mathbf{I}_{GD}$	
ACAD	NSGAII	0.020	0.022	**0.464**	**0.009**	0.0066	0.002
	NSGA$_{NE}$	**0.020**	**0.004**	0.333	0.001	**0.0003**	**0.0**
WMET	NSGAII	0.020	0.019	**0.492**	**0.006**	0.0037	0.0
	NSGA$_{NE}$	0.020	0.003	0.334	0.001	**0.0003**	**0.0**
WAMS	NSGAII	0.0200	0.016	**0.479**	**0.004**	0.0032	0.0
	NSGA$_{NE}$	0.0200	0.003	0.337	0.001	**0.0002**	**0.0**
PSOA	NSGAII	0.020	0.014	**0.421**	**0.014**	0.0129	0.004
	NSGA$_{NE}$	0.020	0.002	0.338	0.001	**0.0001**	**0.0**
OMET	NSGAII	0.020	0.014	**0.465**	**0.006**	0.0044	0.0
	NSGA$_{NE}$	0.020	0.003	0.336	0.001	**0.0001**	**0.0**
PARM	NSGAII	0.020	0.019	**0.489**	**0.011**	0.0090	0.001
	NSGA$_{NE}$	**0.020**	**0.003**	0.338	0.001	**0.0001**	**0.0**

TABLE V
EFFECT-SIZES FOR COMPARISONS BETWEEN NSGAII AND NSGA$_{NE}$

Instance	$\mathbf{I}_C$	$\mathbf{I}_{HV}$	
ACAD	38%	100%	0%
WMET	40%	100%	0%
WAMS	46%	100%	0%
PSOA	50%	100%	0%
OMET	43%	100%	0%
PARM	38%	100%	0%

the reference front uncovered by NSGAII, lowering its and increasing its I_C. NSGAII contributes with more points than NSGA$_{NE}$ on parts of the reference front that it covers, but these improvements are not sufficient to compensate for the distance between points in the extremes of its fronts and the parts of the reference front defined only by NSGA$_{NE}$. Besides

TABLE VI
AVERAGE DIFFERENCE IN MAKESPAN AND COST FOR SOLUTIONS INVOLVING EXCESSIVE OVERTIME FOR EACH INSTANCE. DATA SHOWS THAT THE FORMULATION WITHOUT OVERTIME ERROR DYNAMICS TENDS TO UNDERESTIMATE BOTH COST AND MAKESPAN FOR MOST PROJECTS.

Instance	**Overtime Range**	**(NSGA$_{NE}$ >NSGA-II)** Makespan	Cost
ACAD	332 - 376 hr	-2%	-1%
WMET	332 - 372 hr	-4%	-3%
WAMS	570 - 638 hr	-5%	-3%
PSOA	1,288 - 1,443 hr	6%	5%
OMET	919 - 1,023 hr	-6%	-4%
PARM	797 - 884 hr	-5%	-3%

quality indicators, it is interesting to consider the differences observed in objective values produced under the formulations with and without overtime error dynamics. Table VI presents the differences in average makespan and cost observed in solutions involving the highest overtime allocation for each instance (about 3 hours of daily overtime over the project life-cycle). Data shows that NSGA$_{NE}$ tends to underestimate project cost and makespan when a large number of overtime hours are imposed to the development team. Such difference, observed in 5 out of 6 instances, is due to longer testing activities required to identify and correct the exceeding errors introduced in the software by tired developers. Based on the analysis above, we have found evidence that optimizing overtime planning without taking into account an increase in error generation rates due to developers working overtime produces distinct results from an optimization process that considers such an increase. Next, we explore the practical implications of these findings.

Lessons Learned: Insights for the decision maker are better appreciated in graphical format. Figure II presents 2D projections of pair-wise combinations for the three objectives comprising our formulation for the OPP. Each chart presents the best Pareto front over 50 optimization cycles under NSGAII (circles), the best Pareto front over 50 cycles under NSGA$_{NE}$ (line), and Pareto fronts for CPM (+), MAR (x), and SH (*). Results are shown for the smallest and the largest instances in our sample (respectively, ACAD and PARM). A first result that can be observed from these charts is the linear shape of the Pareto front calculated for NSGA$_{NE}$. This front is comprised of a set of points distributed along the main diagonal of the hypercube representing the objective space for the OPP. The shape of the Pareto front implies that there exist linear relationships between the number of overtime hours invested on the project, project makespan and cost if the dynamics in error generation rates is not taken into account. These linear relationships are incompatible with previous findings in Software Engineering research, which postulate an exponential relationship between cost (effort) and duration for non-trivial software projects [30]. They may also make the optimization process easier for the genetic algorithm, resulting in a larger pool of non-dominated solutions and probing parts of the objective space which could not be explored by the more complex search accounting for error dynamics.

Other interesting aspect is the convex form of the Pareto fronts resulting from our formulation for the OPP. These fronts also fill the main diagonal of the hypercube representing the objective space, but instead of lines we observe curved distributions of points. These curves can be cut into regions presenting fast decrease/increase in objective values and regions on which changes in objective values are slower. For instance, consider the curves for the PARM instance in the lower part of Figure II. The cost of shrinking makespan below 350 days increases exponentially. To reduce makespan from 350 to 337 days, we have to accept an increase of about 8% in project cost. On the other hand, reducing makespan from 360 to 350 days requires increasing the cost by only 1.2%. On regard of overtime hours, the first reduction in makespan requires adding 38 overtime hours to the project, while the second reduction requires adding only 12 hours.

Knee-points connecting different regions of the Pareto front (such as the one described above) can be explored by decision-makers' as overtime allocation alternatives on which large changes in one objective can be garnered by accepting small compromises on other objectives. This allows decision-makers to answer questions such as *"what must I do to reduce makespan?", "how much it costs to reduce makespan by X days?" or "which activities provide the best leverage for a certain amount of overtime?"*.

One might question whether we need heuristic search to provide answers to these questions, that is, can't we find these answers relying solely on simulation? We argue that this is not possible because we are interested in relationships between different results from the simulation. If we were interested in relations between parameters and results, we could sample the selected parameters and calculate the distribution of the results under interest. But to calculate the joint distribution of two or more results we would have to sample all parameters, which is equivalent to a complete search (all combinations of possible values are assigned to parameters) or random search (a limited number of combinations are randomly picked and assigned to parameters). Complete search is unfeasible for interesting problems because the size of the solution space is 9^m, m being the number of activities in the schedule. On the other hand, results from random search are clearly outperformed by our proposed approach, as shown in RQ1. Therefore, heuristic search complements simulation by finding the points in the solution space that represent (close to) optimal combinations of simulation results (in the objective space).

Regarding the last question presented above, we observe strong concentration of overtime allocation in coding and testing activities in Pareto fronts calculated for NSGAII. This concentration is consistent with error propagation dynamics:

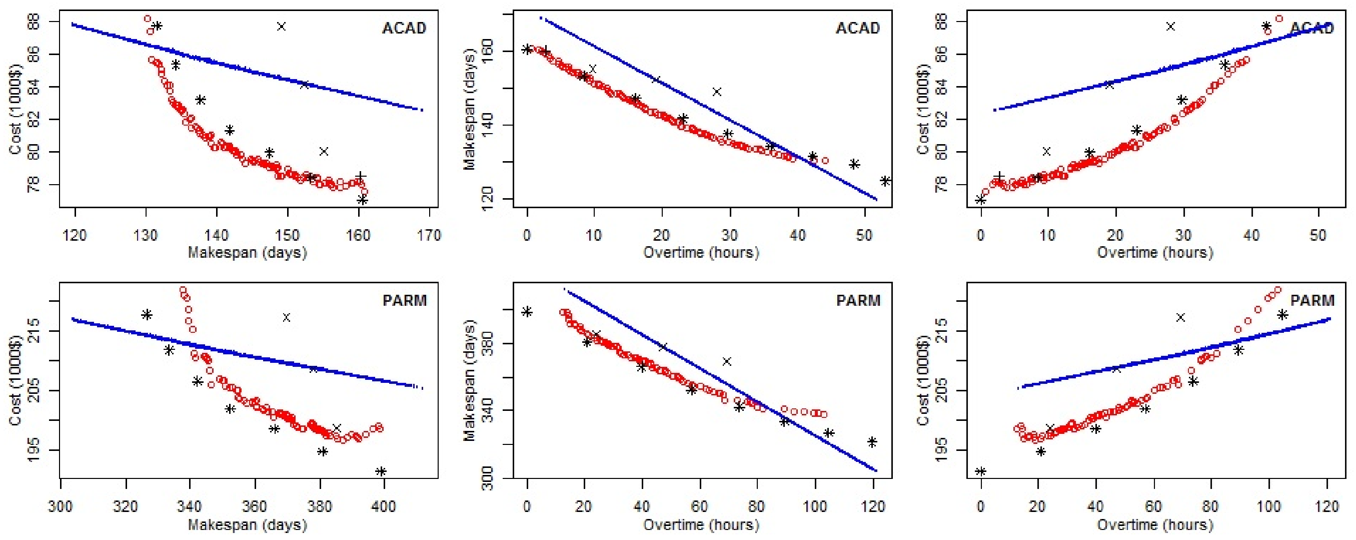

Fig. 2. 2D projections for combinations of the number of overtime hours, makespan, and cost objectives for the smaller (acad) and largest (parm) instances. the linear relations resulting from NSGA$_{NE}$ and the convex shape of nsga-ii can be observed in the charts, as well as the few solutions provided by each oms.

an increase in the number of errors produced in requirement analysis or design activities will snowball through the subsequent activities, increasing the cost of testing later in the schedule. Here the extensible nature of simulation provide us an interesting opportunity: one can break error propagation and regeneration dynamics by using formal reviews or inspections to identify and correct errors earlier than while testing the software. The proposed simulation mechanism allows developing a scenario to represent the effects of such activities [31] and combine them with error dynamics.

The concentration of overtime hours in later activities of the project schedule is also consistent with the quality of the Pareto front built for the SH OMS. As can observed in Figure II, the Pareto front for SH has few points, but it is close to the front built for NSGAII, sometimes even showing better fitness (i.e., PARM instance). Therefore, SH can be used as a starting point for the decision-maker and can be enriched by merging its results with those calculated under our approach.

Finally, despite of having different shapes in the objective space, NSGA and NSGA$_{NE}$ present extremely high inverse correlation (close to -1.0 Spearman rank-order index) between the number of overtime hours and project makespan. NSGA$_{NE}$ also presents extremely high correlation between all pairs of objectives. This leads to actionable results in the sense that algorithms might not be required to pursue optimal values for all three objectives, but only two of them: minimum makespan is observed in solutions having maximum overtime. In practical terms, discarding one objective reduces the cost of optimization, allowing for more generations to be considered and a more diverse number of solutions to be found on Pareto fronts produced by NSGAII.

VI. Validity Threats

In this paper we introduced a formulation for the overtime-planning problem that uses a multi-objective search-based approach and takes into account both positive and negative aspects of overtime. Continuous simulation is used to mimic the behavior of a software project under different conditions of overtime allocation. We compared the proposed approach with practices from the industry and with a similar formulation without the negative effects on quality.

We observed that our approach performs significantly better (high effect-size) than the margarine and critical-path overtime-allocation strategies. However, the practice that suggests allocating overtime in the second half of a project schedule is competitive with our approach and may lead to good overtime allocations, specially for large projects. This behavior confirms the gut-feeling of practitioners who tend to allocate overtime in the last activities of the schedule to compensate for delays introduced earlier. Therefore, by combining search-based optimization and software project simulation we can help researchers to design experimental studies to investigate, evaluate, and provide scientific support for practices and assumptions widely used in the industry, converting intuition into evidence-based knowledge.

For future work, we consider creating new simulation scenarios to model other aspects of software development projects, such as exhaustion due excessive overtime, learning curve, inspections, among others. We also intend to integrate the approach with project management tools, allowing further experimental analysis, particularly to engage humans with the tool and observe their reaction to the overtime allocations it suggests. We also believe that creating means to help visualizing the differences among the alternative allocations proposed by NSGA-II is an important aspect towards adopting our approach in practice.

VII. Related Works

Continuous-time simulation has been extensively used to model software projects, most notably by the seminal work of Abdel-Hamid and Madnick [25], who presented a System Dynamics model of a software project following a waterfall life-cycle and comprising more than 270 equations. Later, this model was extended by Tvedt [32], who added equations to describe a concurrent and incremental life-cycle model. Lin et al. added requirement volatility, milestones, and the possibility to discard certain activities comprising the schedule due to lack of resources [33]. Madachy [31] used System Dynamics to describe the behavior of inspections on software projects. Recently, Cao et al. [34] have used simulation models to evaluate the effects of practices suggested by agile processes. Optimization has also been combined with simulation for project planning purposes. Harman et al. [35] presents a general framework for project planning optimization that uses a simulator to calculate fitness functions according to a project plan and a set of constraints for staff allocation. Antoniol et al. [36] combine a genetic algorithm with a queuing simulator to optimize staff allocation and work package ordering on software projects. The simulator calculates project makespan taking into consideration uncertainties about the effort required for each activity, rework and abandonment of work packages. Recently, Rodriguez et al. [37] used System Dynamics models in conjunction with NSGAII to find ideal team size and schedule estimations for a software project.

VIII. Conclusion and Future Work

In this paper we introduced a formulation for the overtime-planning problem that uses a multi-objective search-based approach and takes into account both positive and negative aspects of overtime. Continuous simulation is used to mimic the behavior of a software project under different conditions of overtime allocation. We compared the proposed approach with practices from the industry and with a similar formulation without the negative effects on quality.

We observed that our approach performs significantly better (high effect-size) than the margarine and critical-path overtime-allocation strategies. However, the practice that suggests allocating overtime in the second half of a project schedule is competitive with our approach and may lead to good overtime allocations, specially for large projects. This behavior confirms the gut-feeling of practitioners who tend to allocate overtime in the last activities of the schedule to compensate for delays introduced earlier. Therefore, by combining search-based optimization and software project simulation we can help researchers to design experimental studies to investigate, evaluate, and provide scientific support for practices and assumptions widely used in the industry, converting intuition into evidence-based knowledge.

For future work, we consider creating new simulation scenarios to model other aspects of software development projects, such as exhaustion due excessive overtime, learning curve, inspections, among others. We also intend to integrate the approach with project management tools, allowing further experimental analysis, particularly to engage humans with the tool and observe their reaction to the overtime allocations it suggests. We also believe that creating means to help visualizing the differences among the alternative allocations proposed by NSGAII is an important aspect towards adopting our approach in practice.

IX. Acknowledgment

References

[1] S. A. Frangos, "Motivated humans for reliable software products," in *IFIP TC5 WG5.4 3rd Internatinal Conference on on Reliability, Quality and Safety of Software-intensive Systems*, ser. ENCRESS '97. London, UK, UK: Chapman & Hall, Ltd., 1997, pp. 83–91.

[2] J. Hyman, C. Baldry, D. Scholarios, and D. Bunzel, "Worklife imbalance in call centres and software development," *British Journal of Industrial Relations*, vol. 41, no. 2, pp. 215–239, 2003.

[3] J. Houdmont, J. Zhou, and J. Hassard, "Overtime and psychological well-being among chinese office workers," *Occupational Medicine*, vol. 61, no. 4, pp. 270–273, 2011.

[4] J. Halliday. (2011) Is facebook playing catchup with google+ and twitter? [Online]. Available: http://www.theguardian.com/media/pda/2011/sep/16/facebook-catchup-google-twitter

[5] M. NISHIKITANI, M. NAKAO, K. KARITA, K. NOMURA, and E. YANO, "Influence of overtime work, sleep duration, and perceived job characteristics on the physical and mental status of software engineers," *Industrial Health*, vol. 43, no. 4, pp. 623–629, 2005.

[6] K. KARITA, M. NAKAO, M. NISHIKITANI, T. IWATA, K. MURATA, and E. YANO, "Effect of overtime work and insufficient sleep on postural sway in information-technology workers," *Journal of occupational health*, vol. 48, no. 1, pp. 65–68, jan 2006.

[7] T. DeMarco, *Controlling software projects : management, measurement & estimation*. New York, NY: Yourdon Press, 1982.

[8] J. C. B. Akula, "Impact of overtime and stress on software quality," in *Proceedings of the 4th International Symposium on Management, Engineering, and Informatics (MEI'08)*, 2008.

[9] A. Riege, "Threedozen knowledgesharing barriers managers must consider," *Journal of Knowledge Management*, vol. 9, no. 3, pp. 18–35, 2005.

[10] A. Amin, S. Basri, M. Hassan, and M. Rehman, "Software engineering occupational stress and knowledge sharing in the context of global software development," in *National Postgraduate Conference (NPC), 2011*, Sept 2011, pp. 1–4.

[11] C. C. CARUSO, "Possible broad impacts of long work hours," *Industrial Health*, vol. 44, no. 4, pp. 531–536, 2006.

[12] I. Donald, P. Taylor, S. Johnson, C. Cooper, S. Cartwright, and S. Robertson, "Work environments, stress, and productivity: An examination using asset." *International Journal of Stress Management*, vol. 12, no. 4, pp. 409–423, 2005.

[13] A. E. Dembe, J. B. Erickson, R. G. Delbos, and S. M. Banks, "The impact of overtime and long work hours on occupational injuries and illnesses: new evidence from the united states," *Occupational and Environmental Medicine*, vol. 62, no. 9, pp. 588–597, 2005.

[14] F. M. Fischer, C. R. de Castro Moreno, F. N. da Silva Borges, and F. M. Louzada, "IMPLEMENTATION OF 12-HOUR SHIFTS IN a BRAZILIAN PETROCHEMICAL PLANT: IMPACT ON SLEEP AND ALERTNESS," *Chronobiology International*, vol. 17, no. 4, pp. 521–537, Jan. 2000.

[15] T. SASAKI, K. IWASAKI, I. MORI, N. HISANAGA, and E. SHIBATA, "Overtime, job stressors, sleep/rest, and fatigue of japanese workers in a company," *Industrial Health*, vol. 45, no. 2, pp. 237–246, 2007.

[16] F. Ferrucci, M. Harman, and F. Sarro, "Search-based software project management," in *Software Project Management in a Changing World*. Springer, 2014, vol. 19, pp. 373–399.

[17] F. Ferrucci, M. Harman, J. Ren, and F. Sarro, "Not going to take this anymore: Multi-objective overtime planning for software engineering projects," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 462–471.

[18] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation (3rd Edition)*, 3rd ed. Prentice Hall, 2000.
[19] J. Forrester, *Industrial Dynamics*. MIT Press, 1961.
[20] S. Ross, *A Course in Simulation*. Macmillan, New York, 1990.
[21] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot, "Statemate: A working environment for the development of complex reactive systems," *Software Engineering, IEEE Transactions on*, vol. 16, no. 4, pp. 403–414, 1990.
[22] A. Drappa and J. Ludewig, "Quantitative modeling for the interactive simulation of software projects." *Journal of Systems and Software*, vol. 46, no. 2-3, pp. 113–122, 1999.
[23] M. de Oliveira Barros, C. M. L. Werner, and G. H. Travassos, "A system dynamics metamodel for software process modeling." *Software Process: Improvement and Practice*, vol. 7, no. 3-4, pp. 161–172, 2002.
[24] C. Jones, *Software Assessments, Benchmarks, and Best Practices*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.
[25] T. ABDEL-HAMID and S. MADNICK, *Software Project Dynamics: an integrated approach*. Prentice-Hall, USA, 1991.
[26] T. DeMarco and T. Lister, *Peopleware-productive projects and teams*. Dorset House Publishing co., 1999.
[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
[28] J. J. Durillo, Y. Zhang, E. Alba, and A. J. Nebro, "A study of the multi-objective next release problem," in *Proceedings of the 2009 1st International Symposium on Search Based Software Engineering*, ser. SSBSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 49–58.
[29] D. A. Van Veldhuizen and G. B. Lamont, "Evolutionary computation and convergence to a pareto front," in *Late Breaking Papers at the Genetic Programming 1998 Conference*, J. R. Koza, Ed. University of Wisconsin, Madison, Wisconsin, USA: Stanford University Bookstore, 22-25 July 1998.
[30] B. Boehm, *Software Engineering Economics*. Prentice Hall, 1981.
[31] R. J. Madachy, "System dynamics modeling of an inspection-based process." in *ICSE*, H. D. Rombach, T. S. E. Maibaum, and M. V. Zelkowitz, Eds. IEEE Computer Society, 1996, pp. 376–386.
[32] J. D. Tvedt, "An extensible model for evaluating the impact of process improvements on software development cycle time," Ph.D. dissertation, Tempe, AZ, USA, 1996, aAI9624837.
[33] C. Y. Lin, T. K. Abdel-Hamid, and J. S. Sherif, "Software-engineering process simulation model (seps)." *Journal of Systems and Software*, vol. 38, no. 3, pp. 263–277, 1997.
[34] L. Cao, B. Ramesh, and T. K. Abdel-Hamid, "Modeling dynamics in agile software development." *ACM Trans. Management Inf. Syst.*, vol. 1, no. 1, p. 5, 2010.
[35] M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," *Department of Computer Science, Kings College London, Tech. Rep. TR-09-03*, 2009.
[36] G. Antoniol, M. D. Penta, and M. Harman, "A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty." in *IEEE METRICS*. IEEE Computer Society, 2004, pp. 172–183.
[37] D. Rodrguez, M. R. Carreira, J. C. Riquelme, and R. Harrison, "Multiobjective simulation optimisation in software project management." in *GECCO*, N. Krasnogor and P. L. Lanzi, Eds. ACM, 2011, pp. 1883–1890.