Learning Overtime Dynamics Through Multiobjective Optimization

Márcio de O. Barros
Federal University of the State of Rio de Janeiro
Av. Pasteur 458, CEP 22.290-240, Brazil
marcio.barros@uniriotec.br

Luiz Antonio O. de Araujo Jr
Federal University of the State of Rio de Janeiro
Av. Pasteur 458, CEP 22.290-240, Brazil
luiz.araujojr@uniriotec.br

ABSTRACT

IT professionals are frequently subject to working overtime, even knowing that excessive overtime has negative effects both on their lives and the software they produce. This contrast creates the need for overtime policies that attend to the demands of a project using as few overtime hours as possible. However, our knowledge about the dynamics of overtime work and the effects of distinct policies on a software project is limited. In this paper we introduce a formulation for the overtime planning problem which extends the state-of-art by considering both the positive effects of overtime on productivity and its negative effects on product quality. We use heuristic search to explore close to optimal overtime allocations under this formulation and report lessons learned by analyzing these allocations. We present an empirical study that compares our approach with practices from the industry and a similar formulation without negative effects. Evidence supports the industrial practice of concentrating overtime in the second half of a project's schedule. Results also show that ignoring the flip-side of the productivity gains brought by overtime may lead to wrong decisions. For instance, excessive overtime may lead a manager to underestimate project cost and duration by 5.9% and 9.2%, respectively.

Keywords

Overtime planning; management; SBSE

1. INTRODUCTION

Software engineers are frequently subject to working overtime, usually without previous planning and in stressful, fire-fighting situations [21][20]. Difficulties related to measuring project progress, late and volatile requirements, compressed schedules, and the need to shrink the time-to-market of new features [18] 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. Nishikitani et al.[26] 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 [24]. Regarding the outcome of developers' work, Akula and Cusick [4] and DeMarco [10] relate overtime work with higher error generation rates in software projects. DeMarco and Lister [11] also relate excessive overtime with increased personal attrition, which may reduce knowledge exchange amongst developers [27] and hamper the ability of a company to hold expertise and properly deal with developer turnover [2].

Problems related to overtime are not exclusive to the IT sector. Caruso [7] identified negative impacts of long work hours on American workers (fatigue, lack of alertness and health degradation), their families (childbearing and delayed marriage), their employers (absenteeism due to illness), and their products (errors due to exhaustion). Donald et al. also found negative correlation between stress and productivity [13]. Finally, overtime problems do not seem related to a specific culture, affecting Americans [7], Brazilians [17], Japanese [29], and Chinese [20], to cite a few.

Despite the recent interest in search-based approaches to software project management [16], we found a single study that addresses overtime planning as an optimization problem [15]. The approach uses a multiobjective genetic algorithm to identify the activities providing the best leverage to reduce the project duration and the risk of schedule overrun for an amount of overtime work. However, it does not consider the negative impact of overtime on product quality. Previous studies [10][4] have presented empirical evidence that the work required to fix extra errors produced by tired developers may subdue the productivity gains brought by working overtime.

This paper advances the state-of-the-art by considering the documented impact of overtime work on the quality of goods produced by developers working in long shifts. It introduces a new formulation for the overtime planning problem (OPP) which uses simulation to propagate the effects of an increased number of errors produced by a tired developer to the project's schedule and cost. The amount of overtime work is optimized to minimize project duration and cost and we discuss the resulting allocation patterns. We compared our results to current overtime management practices and found empirical support for an overtime management practice used in the industry, that is, scientific evidence that confirms project management intuition. We also compared our results to a similar model that does

GECCO '16, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908812.2908824

not increase error generation rates when developers work overtime and show that ignoring such dynamics may lead project managers to wrong decisions. Our main objective is to use optimization and simulation to learn the effects of different overtime planning policies and not to provide an overtime plan for a project manager. Therefore, the primary contributions of this paper are:

1. We introduce a new formulation for the multiobjective OPP that considers an increase in error generation rates due to fatigue. The formulation considers trade-offs among the number of overtime hours worked, time to project completion, and project cost;
2. We present an empirical study based on 6 real-world projects with up to 635 function points. Results show that concentrating overtime hours in the second half of the schedule (a practice used in the industry) is a good policy 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 when selecting the overtime allocation for a project. An overtime policy that may be optimal if such impact is disregarded may not be the best choice if it is considered.

This paper is divided into 7 sections, starting with this introduction. Section 2 presents our formulation for the OPP. Section 3 presents the proposed solution for this problem. Section 4 presents the design of an experimental study carried out to evaluate the proposed solution. Section 5 discusses the results collected from this study. Section 6 presents related work and Section 7 shows conclusions and future research directions regarding overtime planning.

2. PROBLEM FORMULATION

The innovation in our formulation for the OPP with respect of previous works relates to capturing the dynamics of error generation and propagation through the activities of the software project. This allows estimating the effects of an increased number of errors due to overtime and fatigue.

Software Project Dynamics. Let a software project be represented as an acyclic and 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≠j,1≤i≤n,1≤j≤n$\}$ of precedence dependencies amongst work packages. Each wp_i represents a software component that needs to be analyzed, designed, coded, and tested. Each wp_i is described by the amount of effort required for its development, measured in function points (FP). Each dependency in WPDEP represents a work package (wp_j) whose development can only start after the analysis of a second work package (wp_i) is finished.

From these work packages, we develop a project schedule. In our model, the schedule is a partial order of four software development activities created for each work package: analysis, design, coding, and testing. These activities are organized according to a waterfall life-cycle, so that a testing activity is preceded by a coding activity, which is preceded by a design activity, which comes next to a requirements analysis activity. The schedule also includes dependencies amongst work packages: the analysis activity for work package wp_j cannot start before the analysis activity for wp_i is concluded for all (wp_i, wp_j) pairs in WPDEP.

A schedule is represented as an acyclic directed graph <AC, ACDEP> consisting of a node set AC=$\{a_1, a_2, ..., a_m\}$ of activities and an edge set ACDEP=$\{(a_i, a_j)$: i≠j, 1≤i≤m, 1≤j≤m$\}$ of finish-start dependencies amongst activities. The duration of analysis, design or coding activities depends on the amount of work to be performed and the assigned developers' productivity. Jones [23] shows an average effort needed for each phase according to project type and size. For instance, on Information System (IS) projects with about 1,000 FP, requirements analysis activities consume 7% of the total project effort, design activities consume about 10%, coding activities consume about 30%, and testing consumes 30%. The remaining effort is consumed by management, documentation, and other activities, being proportionally distributed among the analysis, design, coding, and testing activities in our model. Given the percentile of effort required for each activity, the size of a work package in FP and an average productivity of 27.8 FP/developer-month [22], we calculate the expected effort for each activity. This effort is divided by the number of developers available to work on the activity to calculate its duration. For simplification purposes and without loss of generality, we assume that a single developer is assigned to each activity.

Regarding error dynamics, when an activity starts it collects all errors produced by its preceding activities (if any) carrying them forward. Abdel-Hamid and Madnick [1] present a model for error regeneration which states that errors inherited by an activity are not only forwarded to next activities but they also contribute to the generation of new errors. That is, by working on an erroneous product, developers will make assumptions that may add errors to the outcome of their work. These *regenerated* errors would not be introduced in the software if the input product was correct. Jones [22] suggests that developers add on average 1 error/FP in analysis, 1.25 errors/FP in design, and 1.75 errors/FP in coding activities. These rates lead to the average of 4 errors/FP after coding and are compatible with the former model [1] and source [22]. Nevertheless, if the number of errors generated in a given activity increases (for instance, due to overtime), regeneration dynamics will escalate the total number of errors.

Testing activities receive the results of coding activities and aim at identifying and correcting errors introduced in the product. The duration of testing activities depends on the effort required to identify and correct an error and on developer's productivity. We assume a linear cost for identifying and correcting errors: given the amount of work expected to be invested in testing [23] (i.e., 30% of the total effort for 1,000-FP IS 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 an error and use this value to estimate the duration of testing activities. However, the effective duration of testing activities depends on the error dynamics: if more than 4 errors/FP reach a testing activity, due to overtime, its duration is increased proportionally.

Overtime Work Dynamics. Working overtime influences developers productivity and error generation rates. We assume that the productivity of a developer assigned to work more than 8 hours/day in a given activity will be increased proportionally to the extra number of work hours, reaching up to 50% increase if working the upper limit of 12 hours/day. This linear relationship is a simplifying assumption: a tired developer at the end of a 12-hours shift is expected to be less productive than during fresh work

hours. But since we are interested in showing that the loss due to an increased number of errors produced during overtime may surpass the gains in productivity, we take the safe-side and assume the topmost gain that can be expected.

Regarding error generation, it is accepted that developers working more than the regular hours may produce software below the quality standard [11]. Developers working in long shifts become less careful while performing their work and tend to introduce more errors in the software. We used data from Abdel-Hamid and Madnick [1] to represent the increase in the number of errors added by developers according to the amount of overtime (Figure 1). This chart suggests a developer working 10 hours/day adds about 20% more errors than those working on regular shifts and one working 12 hours/day introduces 50% more errors than the baseline.

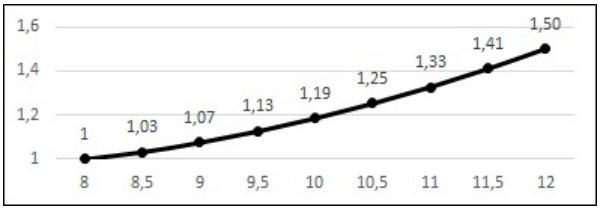

Figure 1: Multiplier for the error generation rate according to daily number of work hours [1].

The non-linear relationship between overtime and error generation rates, along with the error regeneration dynamics, form a complex model on which intuition may lead to wrong interpretation. For a systematic analysis of the effects of overtime, we need an optimizer driven by a simulator that accounts for the error generation dynamics.

Optimization Objectives. OPP is modeled as a three objective decision problem in which the number of overtime hours (NOH), makespan (MKS) and cost (CST) are the objectives to be minimized. A candidate solution for the OPP is a sequence of numbers representing the amount of daily overtime hours (max. 4 hours) to be spent on each activity in the schedule. Our first objective, NOH, is calculated as the sum of the amount of overtime hours consumed by each activity. Project makespan (MKS) is given by the longest precedence-respecting path (critical path) in the graph representing the schedule and is calculated by summing the duration of the activities in this path. The duration of an activity is calculated by the simulator. Finally, project cost (CST) is the sum of each activity's cost. The cost of an activity depends on the number of regular and overtime work hours its developer consumes to produce its expected results. We modeled the problem according to Brazilian laws: if a regular working hour costs X, each of the first two overtime hours costs $120\% \times X$ and the next two hours cost $150\% \times X$ each.

Model Limitations. As any model, the one presented above is subject to simplifications that may influence the results of using it to learn the dynamics of a complex system, such as a software project. First, our model is based on a set of parameters that were used to describe project and overtime dynamics. These parameters were collected from the literature [1] [22] [23] but may not represent the reality of all software development companies.

Next, our model does not account for skills and capabilities of software developers, assigning the same productivity for all developers in all activities. This allowed to investigate the relationship between the positive and negative sides of overtime work without the large number of parameters that would be required to capture the differences among developers. However, more experienced developers may be able to generate less errors and be more resistant to longer shifts, thus possibly affecting our results. The lack of solid information on the literature regarding these issues also inhibited their introduction in our model.

The model also assumes that a single developer is assigned to each activity, disregarding team dynamics such as the cost of the learning curve, cost of communication, and the Brooks Law. For the purposes of our analysis, these are conservative simplifications: team dynamics usually contributes to reduce individual productivity as developers invest part of their time in learning, teaching and communicating.

Regarding testing, instead of a linear cost for identifying and correcting errors, it could be argued that such cost depends on the number of errors in the software: finding an error in a software with few latent errors is harder than in a system crowded with errors [1]. We defend our model pointing that by using averages while dealing with a large number of errors, the costs of very easy/hard to find errors are smoothed. Our model also assumes that all errors are corrected before deployment and that testing will not add extra errors. While relaxing the former might reduce the testing effort and compensate for a higher number of errors due to overtime, the latter would increase the number of errors even more, enforcing our findings.

3. SOLUTION APPROACH

Our solution uses the NSGAII algorithm [9] to find (close to) optimal solutions for the OPP. 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 require experimental settings 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 matches a multiple of 30-minutes overtime. Thus, if the number 4 is assigned to activity, it means two hours of overtime work. Each individual solution evolved by NSGAII is evaluated through simulation. The simulator calculates the effects of overtime according to the number of overtime hours assigned to each activity.

Crossover. We reused the crossover operator proposed by Ferrucci et al. [15] that aims to preserve genes shared by the fittest overtime assignments. Instead of simply exchanging the genetic material of two parents P_1 and P_2, for the genes placed before $C \in [1..m]$ the offspring O_1 and O_2 inherit genes from P_1 and P_2, respectively. The genes after the pointcut (C) are formed by the max/min value from the parents for offspring O_1 and the mean value for offspring O_2, as described in the equations below (given a random variable p with uniform distribution).

$$O_1(g) = \begin{cases} P_1(g) & 1 \leq g < C \\ max(P_1(g), P_2(g)) & C \leq g \leq m, p < 0.5 \\ min(P_1(g), P_2(g)) & C \leq g \leq m, p \geq 0.5 \end{cases} \quad (1)$$

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

The crossover operator is executed with 50% probability. Otherwise, the parents are copied to the next generation. Parents are selected by binary tournament.

Mutation. Mutations are executed for all crossover offspring. Each gene of an offspring O is changed with 30% probability, setting a new value in the [0, 8] integer interval.

4. EXPERIMENTAL DESIGN

Research Questions. The questions addressed by our experiments were designed to provide evidence on the competitiveness of current industrial practices if compared to results produced by an optimizer and the usefulness of the proposed model in raising new information about the allocation of overtime working on software projects.

RQ1 (Competitiveness): How does NSGAII perform if compared to currently used overtime planning strategies? To address this issue, we compare the results produced by NSGAII with overtime-planning practices used in the industry.

RQ2 (Usefulness): Do results of overtime planning change if we consider the dynamics of error generation and the loss of quality caused by overtime? In this sense, we compare the results produced by NSGAII with and without an increase in error generation due to overtime.

Instances. Table 1 presents information about the selected instances used in the experimental analysis. The *TF* and *DF* 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 (*FP*). The *WP* column presents the number of groups on which the data and transaction functions were distributed to form the project schedule, the *ACT* column shows the number of activities in the schedule, and the *DEP* column shows the number of activities dependencies.

Table 1: Instances used in the experimental studies

Name	TF	DF	WP	ACT	DEP	FP
ACAD	39	7	10	40	39	185
WMET	48	7	11	44	33	225
PSOA	65	9	18	72	84	290
WAMS	67	8	15	60	45	381
PARM	98	21	27	108	91	451
OMET	129	22	21	84	63	635

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. PSOA is a personnel management project used for authentication, authorization, and single-point reference from enterprise systems. WAMS is a message routing system responsible for receiving and routing air traffic control messages. PARM stores settings values used by many systems, allowing for fast configuration of user profiles and sharing configurations amongst distinct applications. OMET manages, stores, and delivers meteorological information.

Evaluation. To compare results produced by NSGAII for different formulations, we need to address two properties of the Pareto fronts resulting from these formulations (say F_A and F_B): convergence and diversity. Convergence indicators measure how close a given front F_A is to the reference front, which we build from the non-dominated solutions from both F_A and F_B. Diversity indicators measure the amount of objective space covered by a given front F_A. We chose three indicators: Contributions (I_C), Hypervolume (I_{HV}), and Generational Distance (I_{GD}). They are measured in the [0, 1] interval. I_C is a convergence indicator that represents the proportion of solutions from F_A that pertain to the reference front. Good Pareto fronts have high I_C and contribute heavily to the reference front. I_{HV} mixes convergence and diversity and calculates the volume of the objective space covered by solutions comprising a Pareto front [14]. High-quality fronts present high I_{HV}, thus covering a large extension of the objective space. Finally, I_{GD} is a convergence indicator that measures the distance between a front and the reference front [9]. Good fronts are closer to the reference front and thus present low I_{GD}.

Parameter Setting. Section 3 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 settings to have their values defined: the stop criteria, expressed in terms of a maximum number of fitness function evaluations, and population size, the size of the genetic pool evolved by NSGAII, expressed as a multiplier to the number of activities of the schedule.

To determine the most adequate stop criteria, we executed NSGAII for all selected instances with a maximum of 5,000, 10,000, 20,000, 50,000, 100,000 and 150,000 fitness evaluations. To account for the randomness inherent to heuristic algorithms, the optimization was executed 50 times for each configuration and instance. 150,000 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 evaluation. Afterwards, we built a reference front for each instance based on non-dominated solutions collected from all 50 cycles of all configurations. Then, we calculated I_{GD} for the front resulting from each cycle based on the reference front.

Being bound to the [0, 1] interval, quality indicator data cannot be normally-distributed and thus cannot be subjected to parametric statistical inference tests. Thus, hereafter we 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 pair-wise comparisons through a Wilcoxon test using Bonferroni correction found that there were significant differences among all configurations and the configuration using 150,000 fitness evaluations as stop criteria get smaller I_{GD} for all instances. Thus, we used 150,000 fitness evaluations as stop criteria for all following analysis.

A similar procedure was used to determine the most adequate population size for each instance. We compared population sizes of $2m$, $4m$, and $8m$, m being the number of activities for the instance. NSGAII was executed 50 times for each population size and instance, using 150,000 fitness evaluations as stop criteria. Then, a reference front was built for each instance from the results of all 50 cycles and we calculated I_{GD} for the front resulting from each cycle. A Kruskal-Wallis test was executed upon this data and found significant difference between configurations at 95% level. A post-hoc analysis revealed significantly different results for all configurations on all instances and the configuration using $2m$ as population size led to the best averages. Thus, we chose $2m$ as population size for the analysis. Regarding

the replicability of this paper, all data and the source code are available in a public repository at GitHub [1].

5. ANALYSIS AND DISCUSSION

In this section we present results obtained through our experiments to support answering the proposed research questions. Before proceeding to the questions themselves, we compared the results produced by NSGAII to those yielded by a random search. As expected, the heuristic algorithm outperformed the non-systematic search for all quality indicators under consideration. Thus, our formulation and parameterization passed a basic "sanity check" before being used to support further analysis.

RQ1: To address this question we compared our formulation for the OPP with three overtime management strategies (OMS) presented by Ferrucci et al. [15] as practices from the industry: "margarine" management (MAR) suggests spreading overtime hours evenly over all activities of the schedule; Critical Path (CPM) suggests loading overtime onto the schedule's critical path to reduce completion time; and Second Half (SH) suggests loading overtime onto the later part of the 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 2.

A summary of values collected for each quality indicator and instance is presented in Table 2. 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 because each OMS produced one Pareto front for each instance. I_{HV} and I_{GD} for CPM on instances ACAD, PSOA, and PARM were not calculated because the OMS produced a Pareto front comprised of only two vertices. Bold face values were found significantly different in the comparison between NSGAII and all OMS using Bonferroni-adjusted Wilcoxon-Mann-Whitney inference tests at 95% significance level.

Concerning I_C, NSGAII outperformed all OMS in 5 out of 6 instances. On average, each NSGAII cycle contributed with 4.7 solutions to the reference front, MAR contributed with 1.83 solutions, SH with 3.0 solutions and CPM added 2.0 solutions. Solutions generated under CPM concentrated in the parts of the objective space representing few overtime hours, a region not frequently accessed by NSGAII. All OMS added the solution representing zero overtime to the reference front. For the largest instance, SH provided more solutions than the average run of NSGAII. In respect to I_{HV} and I_{GD}, NSGAII produced the best results for all instances. SH also provided Pareto fronts with high I_{HV} and low I_{GD} though not as high/low as NSGAII due to the number of vertices comprising the latter's fronts.

Thus, we confirm the results presented by Ferrucci et al. [15] in the sense that the CPM and MAR OMS are not competitive with optimization. On the other hand, SH is a good approximation for NSGAII, particularly for large instances. The good performance shown by SH can be explained by the error generation dynamics: if overtime is concentrated towards the end of the schedule, we suppress the snowball-growth in the number of requirements and design errors reaching testing activities.

Table 2: Quality indicator values for each instance under our approach and the OMS.

	Config	NSGAII	MAR	SH	CPM
ACAD	I_C	**0.0198 ± 0.0090**	0.0024	0.0072	0.0048
	I_{HV}	**0,5160 ± 0.0024**	0.2303	0.3589	n/a
	I_{GD}	**0.0024 ± 0.0007**	0.1783	0.0488	n/a
WMET	I_C	**0.0198 ± 0.0110**	0.0055	0.0055	0.0055
	I_{HV}	**0.4980 ± 0.0027**	0.2544	0.3538	0.1913
	I_{GD}	**0.0019 ± 0.0004**	0.2420	0.0418	0.1530
PSOA	I_C	**0.0197 ± 0.0088**	0.0063	0.0063	0.0063
	I_{HV}	**0.3327 ± 0.0033**	0.1496	0.2069	n/a
	I_{GD}	**0.0056 ± 0.0031**	0.2309	0.2725	n/a
WAMS	I_C	**0.0197 ± 0.0100**	0.0053	0.0079	0.0053
	I_{HV}	**0.4922 ± 0.0027**	0.2547	0.3547	0.1718
	I_{GD}	**0.0016 ± 0.0008**	0.1511	0.0418	0.5151
PARM	I_C	0.0190 ± 0.0121	0.0137	**0.0342**	0.0137
	I_{HV}	**0.4211 ± 0.0031**	0.2308	0.3579	n/a
	I_{GD}	**0.0034 ± 0.0003**	0.1518	0.0215	n/a
OMET	I_C	**0.0196 ± 0.0104**	0.0076	0.0115	0.0076
	I_{HV}	**0.4561 ± 0.0032**	0.2545	0.3544	0.2089
	I_{GD}	**0.0017 ± 0.0002**	0.2346	0.0300	0.1356

The A_{12} standardized effect-size was calculated for pairwise comparisons between NSGAII and each OMS for all quality indicators. A_{12} is very high (values were suppressed due to space limitations) and favors NSGAII for all instances for I_{HV} and I_{GD}. It also strongly favors NSGAII in terms of I_C if compared to MAR and CPM. However, A_{12} tells a different story for I_C between NSGAII and SH, clearly favoring SH for the PARM instance. Effect-sizes are in agreement with former results indicating that MAR and CPM are not effective OMS. They also provide evidence on the effectiveness of NSGAII and SH: if there is enough time to run the optimization, NSGAII can produce Pareto fronts which are richer in diversity and closer to optimal than those resulting from SH, but the latter can quickly provide an approximation for the reference front. Thus, the industry has good reasons for concentrating overtime on the second half of project schedules, though it can improve its practices by combining SH with optimization results.

RQ2: 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 our perceptions on the effects of overtime over the duration and cost of a project. Table 3 presents summary values for the quality indicators calculated over 50 optimization cycles for each algorithm. All values are presented in the mean ± standard deviation format. Bold face values are significantly different compared to the other algorithm, as ascertained by Wilcoxon-Mann-Whitney tests at 95% significance level.

A_{12} effect-sizes indicate that $NSGA_{NE}$ consistently obtains better values for I_C (high), I_{HV} (high) and I_{GD} (low) compared to NSGAII. Effect-sizes are equal to 100% for all quality indicators and instances in favor of $NSGA_{NE}$. That implies that optimization results differ significantly if error dynamics associated to overtime working are not considered.

These results are explained by the larger number of

[1] https://github.com/luizaraujojr/GECCO2016

solutions produced by NSGA$_{NE}$: its Pareto fronts have on average 8.8 times the number of solutions of fronts calculated for NSGAII, reaching up to 18.8 times this number for the PARM instance. The larger number of solutions allows NSGA$_{NE}$ to cover parts of the reference front uncovered by NSGAII, increasing its I_C and I_{HV} and decreasing its I_{GD}. NSGAII contributes with more points than NSGA$_{NE}$ on parts of the reference front that it covers, but these improvements are not enough to compensate for the distance between extreme points on its fronts and parts of the reference front defined only by NSGA$_{NE}$.

Table 3: Quality indicators for each instance under our approach and a model without error dynamics.

Instance	Ind.	NSGAII	NSGA$_{NE}$
ACAD	I_C	0.2636 ± 0.1741	**1.7368 ± 0.4060**
	I_{HV}	0.3075 ± 0.0024	**0.3327 ± 0.0011**
	I_{GD}	1.5962 ± 0.3456	**0.0306 ± 0.0051**
WMET	I_C	0.0848 ± 0.1002	**1.9164 ± 0.3881**
	I_{HV}	0.3136 ± 0.0025	**0.3482 ± 0.0007**
	I_{GD}	2.3436 ± 0.4490	**0.0302 ± 0.0038**
PSOA	I_C	0.0024 ± 0.0082	**1.9980 ± 0.2019**
	I_{HV}	0.3439 ± 0.0043	**0.3992 ± 0.0016**
	I_{GD}	0.8928 ± 0.2048	**0.0108 ± 0.0027**
WAMS	I_C	0.0060 ± 0.0297	**1.9960 ± 0.2978**
	I_{HV}	0.3080 ± 0.0023	**0.3531 ± 0.0008**
	I_{GD}	2.0528 ± 0.3486	**0.0204 ± 0.0020**
PARM	I_C	0.0006 ± 0.0042	**1.9996 ± 0.2092**
	I_{HV}	0.2320 ± 0.0047	**0.3628 ± 0.0018**
	I_{GD}	2.8130 ± 0.2549	**0.0102 ± 0.0014**
OMET	I_C	0.0006 ± 0.0042	**1.9994 ± 0.2324**
	I_{HV}	0.2729 ± 0.0034	**0.3528 ± 0.0014**
	I_{GD}	2.0360 ± 0.2711	**0.0106 ± 0.0024**

Besides quality indicators, it is interesting to consider the differences observed in objective values produced under the formulations with and without overtime error dynamics. Table 4 presents the differences on average makespan and cost observed in solutions involving the highest overtime allocation for each instance (about 3 hours of daily overtime over the schedule). Data shows that NSGA$_{NE}$ underestimates project cost and makespan when a large number of overtime hours are imposed to the development team. This difference, observed for all instances, is due to longer testing activities required to identify and correct the extra errors introduced in the software by tired developers.

Table 4: Average difference in makespan and cost for solutions involving overtime for each instance.

Instance	Overtime Range		NSGA$_{NE}$ <NSGA	
	Daily (hr)	Total(hr)	Makespan	Cost
ACAD	2.53 - 2.99	337.6 - 378.8	5.13%	3.51%
WMET	2.88 - 3.49	379.7 - 420.5	8.56%	5.83%
PSOA	2.67 - 3.06	1296.1 - 1451.9	2.23%	0.99%
WAMS	2.88 - 3.51	644.9 - 713.9	9.21%	5.24%
PARM	2.37 - 2.84	812.9 - 902.9	7.27%	5.50%
OMET	2.72 - 3.30	1031.9 - 1138.0	8.99%	5.86%

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 than an optimization process that considers such an increase. Next, we explore the practical implications of these findings.

Lessons Learned: Insights from the analysis are better appreciated in graphical format. Figure 2 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 (∘) in light gray, the best Pareto front over 50 cycles under NSGA$_{NE}$ (∘) in dark gray, and Pareto fronts for MAR (o), SH (△), and CPM (+). Results are shown for the ACAD and PARM instances only, due to space limitations.

A first result that can be observed in the 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 [5]. The exponential relationship between cost and duration is observed in the results produced by our formulation for the OPP, providing evidence that error generation dynamics are required to precisely model the nature of complex software projects. The linear relationships observed in the results calculated for NSGA$_{NE}$ may also make the optimization easier for the genetic algorithm, resulting in a larger pool of non-dominated solutions that probe parts of the objective space which could not be explored by the more complex search that handles error dynamics.

Other interesting aspect is the convex form of the Pareto fronts resulting from our formulation. 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 2. The cost of shrinking makespan below 355 days increases faster than linearly. To reduce makespan from 352 to 342 days, we have to accept an increase of about 5.6% in project cost. However, reducing makespan from 362 to 352 days requires increasing the cost by only 2.0%. Moreover, the first reduction in makespan requires adding 28 overtime hours to the project, while the second reduction requires adding only 15 hours.

We believe this trade-off analysis may be useful for a manager deciding on how much overtime might be invested in a given software project. Knee-points connecting different regions of the Pareto front 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 answering questions such as "what must I do to reduce makespan?", "how much does it cost to reduce makespan by X days?" or "which activities provide the best leverage for a certain amount of overtime?".

One might question why we need heuristic optimization to answer these questions, that is, can't we find answers relying only on simulation? This is not possible because we are interested in relationships between different results of the model. If we were interested in relations between parameters and results, we could sample the parameters and calculate the distribution of the results. But to calculate the joint distribution of two or more results we would have to sample all parameters, which is equivalent

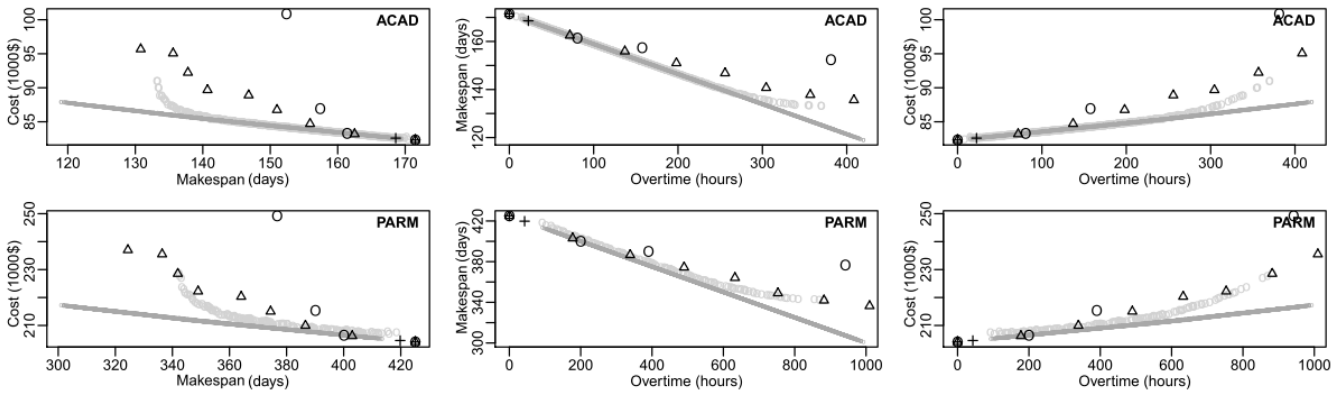

Figure 2: 2D projections for combinations of the number of overtime hours, makespan, and cost objectives for the ACAD and PARM instances. NSGAII is shown in light-gray circles, NSGA$_{NE}$ is shown in dark-gray circles, MAR as circles, SH as triangles, and CPM as plus signs.

to an exhaustive 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 due to the size of the objective space (9^m, m being the number of activities). On the other hand, results from random search are outperformed by our approach. Therefore, heuristic search complements simulation by finding the points in the objective space that represent (close to) optimal combinations of selected simulation results.

Regarding the activities providing the best leverage for overtime work, we observe strong concentration of overtime in coding and testing activities in Pareto fronts calculated for NSGAII. This concentration is consistent with error dynamics: an increase in the number of errors produced in requirements analysis or design will snowball through the subsequent activities, increasing the cost of testing later in the schedule. Here the extensible nature of simulation provides an opportunity: one can break error propagation dynamics by using formal reviews or inspections to identify and correct errors earlier than in testing activities. By using such practices, a manager can reduce the snowball effect of error regeneration, reducing the effort spent on testing. The concentration of overtime in later activities is also consistent with the quality of the Pareto front built for the SH OMS. As can observed in Figure 2, the Pareto front for SH has few points, but it is quite close to the front built for NSGAII.

Finally, despite of having different shapes in the objective space, NSGA and NSGA$_{NE}$ present extremely high inverse correlation between the number of overtime hours and project makespan (close to -1.0 Spearman rank-order index). Indeed, NSGA$_{NE}$ 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 in the same time frame and a more diverse number of solutions to be found on Pareto fronts produced by NSGAII.

6. RELATED WORKS

Continuous-time simulation has been extensively used to model software projects, most notably by the seminal work of Abdel-Hamid and Madnick [1], who presented a System Dynamics model of a software project following a waterfall life-cycle. Later, this model was extended by Tvedt [30], who added equations to describe a concurrent and incremental life-cycle model. Recently, Cao et al. [6] evaluated the effects of practices from agile method with simulation models.

Optimization has been combined with simulation for project planning purposes. Ferrucci et al. [16] present a survey on search-based optimization applied to project management. They were also the first to propose a multi-objective model for overtime work [15]. Harman et al. [19] present a framework for project planning optimization that uses a simulator to calculate fitness functions according to a project plan and constraints on staff allocation. Di Penta et al. [12] use optimization to analyze how project duration and resource allocation varied for different communication overhead models and levels. Luna et al. [25] analyze the scalability of eight multiobjective algorithms applied to a project scheduling problem. Chicano et al. [8] use optimization with a project scheduling problem, analyzing the decision of who does what during a project lifetime. Antoniol et al. [3] combine a GA with a queuing simulator to optimize staff allocation and work package ordering on software projects. Recently, Rodriguez et al. [28] use System Dynamics models and NSGAII to find the ideal team size and schedule estimations for a software project.

7. CONCLUSION AND FUTURE WORK

In this paper we presented a formulation for the overtime planning problem that uses optimization and takes into account both the positive and negative aspects of overtime. Simulation is used to mimic project behavior under different amounts of overtime. We compared the proposed approach with practices from the industry and a similar model devoid of the negative effects of overtime on product quality. Next, we presented lessons learned from studying the overtime allocation patterns produced by the heuristic optimizer.

Our approach performs significantly better than the margarine and critical path overtime policies. However, the practice that suggests allocating overtime in the second half of the schedule is competitive with our approach and may lead to good overtime allocations. This behavior confirms the "gut-feeling" of managers who tend to allocate overtime in the later half of the schedule. Therefore, we showed that optimization and simulation we can be combined

to help researchers in designing experimental studies to provide scientific support for practices used in the industry, converting intuition into evidence-based knowledge. For future work, we consider extending our model to cover other aspects of software projects, such as uncertainties in activity duration, the Brooks' Law, learning curve, and inspections.

8. REFERENCES

[1] T. Abdel-Hamid and S. Madnick. *Software Project Dynamics: an integrated approach*. Prentice-Hall, USA, 1991.

[2] 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*, pages 1–4, 2011.

[3] 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*, pages 172–183, 2004.

[4] J. C. B. Akula. Impact of overtime and stress on software quality. In *Intl Symposium on Management, Engineering, and Informatics*, 2008.

[5] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[6] L. Cao, B. Ramesh, and T. K. Abdel-Hamid. Modeling dynamics in agile software development. *ACM Trans. Management Inf. Syst.*, 1(1):5, 2010.

[7] C. C. Caruso. Possible broad impacts of long work hours. *Industrial Health*, 44(4):531–536, 2006.

[8] F. Chicano, F. Luna, A. J. Nebro, and E. Alba. Using multi-objective metaheuristics to solve the software project scheduling problem. In *Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1915–1922, 2011.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

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

[11] T. DeMarco and T. Lister. *Peopleware-productive projects and teams*. Dorset House Publishing co., 1999.

[12] M. Di Penta, M. Harman, G. Antoniol, and F. Qureshi. The effect of communication overhead on software maintenance project staffing: a search-based approach. In *IEEE Intl Conference on Software Maintenance*, pages 315–324, 2007.

[13] 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*, 12(4):409–423, 2005.

[14] J. J. Durillo, Y. Zhang, E. Alba, and A. J. Nebro. A study of the multi-objective next release problem. In *Intl Symposium on Search Based Software Engineering*, pages 49–58, 2009.

[15] 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*, pages 462–471, NJ, USA, 2013.

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

[17] 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*, 17(4):521–537, Jan. 2000.

[18] J. Halliday. Is facebook playing catchup with google+ and twitter?, 2011.

[19] 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, TR-09-03*, 2009.

[20] J. Houdmont, J. Zhou, and J. Hassard. Overtime and psychological well-being among chinese office workers. *Occupational Medicine*, 61(4):270–273, 2011.

[21] J. Hyman, C. Baldry, D. Scholarios, and D. Bunzel. WorkâĂŞlife imbalance in call centres and software development. *British Journal of Industrial Relations*, 41(2):215–239, 2003.

[22] C. Jones. *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley Longman Pub. Co., Inc., Boston, MA, USA, 2000.

[23] C. Jones. *Estimating Software Costs : Bringing Realism to Estimating: Bringing Realism to Estimating*. McGraw-Hill's AccessEngineering. Mcgraw-hill, 2007.

[24] 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*, 48(1):65–68, jan 2006.

[25] F. Luna, D. L. González-Álvarez, F. Chicano, and M. A. Vega-Rodríguez. The software project scheduling problem: A scalability analysis of multi-objective metaheuristics. *Appl. Soft Comput.*, 15:136–148, Feb. 2014.

[26] 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*, 43(4):623–629, 2005.

[27] A. Riege. Three-dozen knowledge-sharing barriers managers must consider. *Journal of Knowledge Management*, 9(3):18–35, 2005.

[28] D. Rodriguez, M. R. Carreira, J. C. Riquelme, and R. Harrison. Multiobjective simulation optimisation in software project management. In *GECCO*, 2011.

[29] 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*, 45(2):237–246, 2007.

[30] J. D. Tvedt. *An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time*. PhD thesis, Arizona State University, Tempe, AZ, USA, 1996.