Information and Software Technology 52 (2010) 446–461

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Studying the impact of uncertainty in operational release planning – An integrated method and its initial evaluation

Ahmed Al-Emran [a,*], Puneet Kapur [b], Dietmar Pfahl [a,c,d], Guenther Ruhe [a,e]

[a] Department of Electrical and Computer Engineering, Schulich School of Engineering, University of Calgary, 2500 University of Calgary NW, Calgary, AB, Canada T2N 1N4
[b] Chartwell Technology Inc., Suite 400, 750, 11th ST SW, Calgary, AB, Canada T2P 3N7
[c] Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway
[d] Simula Research Laboratory, P.O. Box 134, 1325 Lysaker, Norway
[e] Department of Computer Science, University of Calgary, 2500 University of Calgary NW, Calgary, AB, Canada T2N 1N4

ARTICLE INFO

Article history:
Received 14 May 2009
Received in revised form 20 October 2009
Accepted 5 November 2009
Available online 11 November 2009

Keywords:
Operational release planning
Uncertainty
Impact analysis
Discrete-event simulation
Heuristic optimization
Explorative case study

ABSTRACT

Context: Uncertainty is an unavoidable issue in software engineering and an important area of investigation. This paper studies the impact of uncertainty on total duration (i.e., make-span) for implementing all features in operational release planning.
Objective: The uncertainty factors under investigation are: (1) the number of new features arriving during release construction, (2) the estimated effort needed to implement features, (3) the availability of developers, and (4) the productivity of developers.
Method: An integrated method is presented combining Monte-Carlo simulation (to model uncertainty in the operational release planning (ORP) process) with process simulation (to model the ORP process steps and their dependencies as well as an associated optimization heuristic representing an organization-specific staffing policy for make-span minimization). The method allows for evaluating the impact of uncertainty on make-span. The impact of uncertainty factors both in isolation and in combination are studied in three different pessimism levels through comparison with a baseline plan. Initial evaluation of the method is done by an explorative case study at Chartwell Technology Inc. to demonstrate its applicability and its usefulness.
Results: The impact of uncertainty on release make-span increases – both in terms of magnitude and variance – with an increase of pessimism level as well as with an increase of the number of uncertainty factors. Among the four uncertainty factors, we found that the strongest impact stems from the number of new features arriving during release construction. We have also demonstrated that for any combination of uncertainty factors their combined (i.e., simultaneous) impact is bigger than the addition of their individual impacts.
Conclusion: The added value of the presented method is that managers are able to study the impact of uncertainty on existing (i.e., baseline) operational release plans pro-actively.

1. Introduction

Software release planning is performed at both strategic and operational levels. At the strategic level, managers deal with prioritizing and assigning software features to successive releases so that both technical and resource constraints are met with maximized customer satisfaction. Once strategic planning is done (i.e., determining which features will be implemented in which releases), operational release planning (ORP) takes place focusing on the development of one particular release. It deals with the assignment of developers to tasks that need to be carried out to develop all selected features for that particular release with minimized make-span.

"Uncertainty is inherent and inevitable in software development processes and products" [1] and it is important to know how and by what magnitude uncertainty may impact the original project plan. Since many planning parameters and features are under continuous change [2], release planning becomes a very dynamic endeavor. Related decision-making problems are difficult to resolve under such uncertainty and managers need to know in advance how much deviation from the original plan may arise due to the uncertainty. In addition, Murphy's Law says: "Whatever can go wrong will go wrong, and at the worst possible time, in the worst possible way" [3]. This suggests that worst case scenario analysis addressing uncertainty is an important area of investigation in software product development.

* Corresponding author. Tel.: +1 403 210 5440; fax: +1 403 282 6855.
E-mail address: aalemran@ucalgary.ca (A. Al-Emran).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.11.003

We performed an explorative case study on a real-world project at Chartwell Technology Inc. Chartwell was interested in determining the effectiveness of their staffing policy regarding two aspects: 'goodness' and 'robustness'. Thus, we split the case study project into two sub-projects: (i) measuring and improving the degree of optimality of a given staffing policy (i.e., 'goodness' aspect) and (ii) assessing the degree of stability under uncertainty of project plans as a result of a given staffing policy (i.e., 'robustness' aspect). The work presented in [4] corresponds to the first sub-project whereas the work presented in this paper corresponds to the second sub-project.

Since software process simulation [5] is a well known approach to evaluate scheduling strategies for software projects [6], we propose an integrated method combining Monte-Carlo simulation with process simulation. The purpose of Monte-Carlo simulation is to model uncertainty in the ORP process. The purpose of process simulation is to model the ORP process steps and their dependencies as well as an associated optimization heuristic representing an organization-specific staffing policy for make-span minimization. The proposed method has been implemented as a discrete-event simulation model. Note that when applying the proposed method we neither investigate the causes of uncertainty (as studied in [7]) nor identify determinants for project success (as studied in [8]); rather we measure the impact of uncertainty where the planning parameters are assumed to be stochastic and variant over time.

We consider two major types of uncertainty as defined by Turban et al. [9]: (i) imprecise knowledge and (ii) incomplete information. Imprecise knowledge refers to the collected information on a fact that cannot be guaranteed to be consistent or correct (i.e., most likely to be overestimated or underestimated). Imprecise knowledge about planning parameters that were considered in our case study are: effort estimates of the tasks necessary to develop software features, and productivity of developers who are going to develop these features. Managers need to rely heavily on these estimates (i.e., effort and productivity), however, no one knows whether the estimated values are correct or overestimated or underestimated until completion of associated tasks. Incomplete information refers to the information that "is simply not available or is too expensive or time-consuming to obtain" [9]. Incomplete information about planning parameters that were considered in our case study are: new feature inclusion by the customer, and developer dropout after release development has started. This kind of information is very hard to predict. Ideally an experience database is employed for such prediction but experience databases are expensive to maintain. In order to study the impact of uncertainty (i.e., both imprecise knowledge and incomplete information), we applied distribution functions with a minimum and maximum value range (obtained from past project experience or expert opinion) to our stochastic planning parameters (instead of single/deterministic values). The respective project outcomes were then predicted with the help of a simulation-based analysis by sampling different values from the applied input distributions.

This paper is based upon previous work published in [10]. Compared to our previous work, we now put more emphasis on describing the integrated method ProSim/ORP (Project Simulation for Operational Release Planning). The method is an adaptation of the analysis framework ProSim/RA [11] and implemented using the simulation model DynaReP [12]. ProSim/ORP is easy to tailor to a particular ORP problem that involves estimating the impact of uncertainty due to imprecise knowledge and incomplete information. In addition, we present a more comprehensive analysis studying the impact of the various uncertainty factors, when occurring in isolation and in combination. Based on the integrated solution method, more comprehensive analysis options were explored.

The structure of the rest of the paper is as follows. Section 2 describes related research. Section 3 formalizes the ORP problem followed by an informal description and illustrative example of ORP. Section 4 defines the research questions studied in this paper. Section 5 describes the proposed method and its integrated simulation model. Section 6 gives the context of the case study, the case study itself, its results and the rationale behind this investigation. Section 7 discusses threats to validity. Finally, Section 8 presents conclusions and suggestions for follow-up research.

2. Related work

In this section, we discuss existing work related to ProSim/ORP and draw comparisons with regards to applicability, validity, and methodology. Existing work includes literature on resource allocation and optimization, release planning and make-span minimization, uncertainty analysis, and simulation.

In the area of resource allocation and optimization, the popularity of genetic algorithm based solution approaches is growing constantly. The reasons for this were established by Antoniol et al. [13] who evaluated the use of three different search-based techniques, namely genetic algorithms (GAs), hill climbing and simulated annealing for planning resource allocation in large maintenance projects. The aim was to find an optimal or near optimal order in which to allocate work packages to programming teams such that project duration is minimized (i.e., make-span minimization). According to their observations, GA converged faster than the other approaches. Chang et al. [14] proposed the application of genetic algorithms to software project management. Their proposed method focused on schedule minimization but unlike our case study they did not examine different productivity levels of developers. The GA-based approach proposed by Duggan et al. [15] is a multi-objective (time vs. defect) optimizer that maps software packages to developers where packages come with different complexity levels whereas developers come with different skill levels for different packages. However developers are directly assigned to aggregate work packages (that are similar to features in our case) instead of their constituent building blocks (that are similar to tasks in our case). Therefore, none of the aforementioned related work considers planning aspects at the fine-grained level we propose in this paper.

Ngo-The and Ruhe presented an optimization method called $\text{OPTIMIZE}_{\text{RASORP}}$ [16] which is a two-phased approach that combines the strength of special structure integer linear programming (Phase 1) with the power of genetic algorithms (Phase 2). It can simultaneously generate operational feature implementation plans for individual releases and feature allocation plans for subsequent releases. $\text{OPTIMIZE}_{\text{RASORP}}$ considers feature implementation tasks and their estimated efforts, a pool of developers to carry out these tasks, the productivities of developers to perform these tasks, task overlap and associated dependencies, as well as mappings between tasks and developers for implementing features within individual releases. For our proposed approach, we consider the planning aspects at the same granularity level as $\text{OPTIMIZE}_{\text{RASORP}}$ does.

However, like any other types of method, results of optimization methods are heavily dependent on reliable inputs. That means a method is able to produce a 'true' optimal solution assuming that the supplied input data are accurate. In case of software development, most of the planning input parameters are estimated values (e.g., task effort and developer productivity) with little guarantee as to their accuracy. Therefore, it is obvious that an optimized plan will never be 'exactly' followed and managers need to re-plan again and again due to the presence of uncertainty.

A recent rigorous literature review reports that "more attempts are needed to effectively tackle the uncertainty of software process

in practice" [17]. This clearly indicates that there is a shortage of methods to handle uncertainty. Among the existing research related to this topic, Yang et al. [18] studied the impact of uncertainty on software release time. However, their focus is solely on variability of actual cost whereas our focus is on task effort, developer productivity, unforeseen late request of features, and developer unavailability. The method proposed by Antoniol et al. [19] is a GA-based approach followed by a queuing simulation. The GA portion attempts to minimize project completion time by determining the optimal order of work packages to be processed and staff distribution across project teams. The queuing simulator analyzes the robustness of the project plan suggested by the GA part and feeds this information back into the GA to look for more robust plans. Thus the primary focus is, again, scheduling optimization without considering process heuristic such as conformance to an organization-specific staffing policy as we do.

Padberg [20] attempted computing optimal scheduling policy under uncertainty involved in completion times and rework. The main focus of this approach is on computing an optimal scheduling policy that minimizes the expected project cost. Even though the author succeeded in computing an exact optimal policy for small example projects, later on he concluded that it is unlikely to find an optimal policy under uncertainty for a realistic project instance with dozens of components [21]. In this paper, we are not trying to optimize a staffing policy; rather, evaluating a particular staffing policy.

In our early work on uncertainty analysis, we used the process simulation model REPSIM-1 (Release Planning Simulator, Version-1) [22], a System Dynamics simulation model that can be applied to perform risk analyses on existing operational release plans. This model can evaluate how sensitive existing plans are to possible estimation uncertainties but cannot generate an operational plan by itself based on a strategy or policy. Examples of uncertainty considered in this model include alterations of expected developer productivities, of feature and task specific work volumes (i.e., task effort). The model allows decision-makers to perform 'what-if' analyses on a proposed ORP, helping them prepare for potential manual re-planning.

Later on, we proposed a discrete-event process simulation model called DynaReP to overcome the problem of manual planning and re-planning of releases at operational level [12]. DynaReP supports both initial generation of operational plans as well as automatic re-planning each time a change in planning parameters is identified during the development of a release. It is flexible enough to plug-in any heuristic for developer-to-task assignment that can be guided by either an organization-specific staffing policy or any other optimization module. In addition, its users can sample planning parameter values from probability distribution functions to study the impact of uncertainty on release make-span.

3. Operational release planning

3.1. Informal problem description

The purpose of operational release planning (ORP) is to assign resources to feature implementation tasks such that total release make-span is minimized under given process and project constraints. Re-planning on an operational level involves re-allocation of resources to feature implementation tasks in the face of resource changes, feature changes, and observation of planning mistakes arising from inaccurate estimation. The study presented in this paper focuses on the operational level of release planning (and re-planning) assuming that strategic release planning, i.e., the assignment of a set of features to the release under investigation, has been completed.

In order to implement a single feature, several process steps or tasks need to be carried out. For instance, a feature needs to be designed, implemented and, finally tested. Thus design, code, and test could be three basic tasks that need to be done in order to implement features when developing a software system. Another example of tasks could be 'flash implementation', 'database update', and 'java coding'. However, all of them may not need to be carried out to develop a particular feature.

For each task, we need to estimate the effort, i.e., the effort required by developers to complete the task. Effort estimation can be done through expert judgment, estimation by analogy [23], or a mathematical modeling [24]. The unit of estimated values can be person-hour, person-day, person-week, etc. For example, if the effort required to carry out a task is estimated to be eight person-weeks, it means it will take 8 weeks to perform that task by a single developer with standard (average) productivity.

Developers constitute the workforce that we need to perform feature implementation tasks. Developers may be able to do only one task (e.g., database design) or they may be able to carry out more than one task (e.g., both code and test). Developers may not have the same skills or abilities; the amount of work done per time unit (i.e., productivity) may vary depending on the developer. Even a single developer may have a different productivity (development rate) for each different tasks he/she is able to perform [25]. Thus, we expect that not all developers will have the same average productivity for a specific task and that a single developer may not have the same average productivity for all kinds of tasks. We express productivities as factor values where the value 1 represents the productivity of a developer with average productivity. Recall that the effort estimates for tasks mentioned earlier are determined based on the assumption of having a doing the task, i.e., assuming that the person involved possesses productivity 1 for all feature implementation tasks. Different productivities such as 1.5, 2, 0.5, or 1.2 can be assigned to different developer per task by comparing their average performance with that of the standard developer. For example, a developer with productivity 2 would expect to finish a task in half the time required by a standard developer who possesses productivity 1 for the same task. That is, the developer having productivity 2 is twice as productive as the standard developer. The value 0 for productivity indicates that a developer lacks sufficient knowledge to successfully perform the corresponding task. More on productivity and its measurement can be found in [26].

The execution time of a task not only depends on the effort required to complete the task but also the productivity of the assigned developer for that task. Thus, execution time can be different for the same task if carried out by developers with unequal productivities. Task execution time is calculated by dividing the effort for the task (i.e., estimated task effort) by the productivity of the developer assigned to that task. For example, the execution time for a task with estimated effort of six person-weeks is 6 weeks if performed by a standard developer (i.e., productivity is 1) whereas the it is 4 weeks if the associated developer possesses productivity 1.5 for the same task.

The term 'release make-span' refers to the time starting from the very first task of the release started by a developer until all tasks of the release have been finished. Thus, release make-span is equal to the (relative) end-time of the latest task completed in that release (assuming the projects starts at time 0). It is not necessarily equal to the sum of task execution times associated to all tasks of all features of the release. This is because tasks can be carried out in parallel no matter whether they belong to different features or the same feature. However, there exist dependency constraints between tasks. One possible dependency is the *start–start* and *end–end* constraint between tasks of the same feature. The start–start constraint is based on the start times of tasks: a

predecessor task must start before (or at least at the same time) the successor task of the same feature begins. For example, related to the same feature, the code cannot start before the design starts. The end–end constraint is based on the end times of tasks: a predecessor task of a feature must be finished before (or at least at the same time) its successor task has completed. For example, related to the same feature, the test can never be finished before the code is finished.

There also exist other types of task dependencies which include completion of certain amount of tasks. An *x*% dependency refers to the fact that at least *x*% effort of a task must be completed before its subsequent task can start. 100% task dependency implies, for example, that the coding of a feature needs to be fully completed before its testing can start.

The key elements of the ORP problem, found from the above discussion, are features, tasks and associated estimated efforts, task dependency constraints, numbers of developers and their estimated productivities per task. However, this may not necessarily be the full spectrum of elements characterizing ORP problems. For example, cost elements could be added. We consider the elements that we mentioned above to be the minimum of what is required to formulate an ORP problem.

3.2. Illustrative example

Let us consider the scenario shown in Fig. 1. To keep the example simple, two features feat(1) and feat(2) are considered for a release with start–start and end–end task dependency constraints. A feature feat(n) is said to be implemented as soon as three different associated tasks task(n, 1), task(n, 2), and task(n, 3) (corresponds to design, code, and test tasks, respectively) have been completely carried out. The variables denoted as eff(n, q) (corresponds to task(n, q) that belongs to f(n)) represent efforts for the respective feature implementation tasks, given in person-days. Two developers dev(1) and dev(2) are working to carry out feature implementation tasks. Each developer possesses different productivities for different tasks. Productivities of a developer dev(k) for the three feature development tasks mentioned are denoted as prod(k,1), prod(k,2), and prod(k,3), respectively.

Fig. 1 illustrates an example developer allocation for the scenario in terms of release make-span minimization whereas Fig. 2 shows the corresponding schedule and explains how the make-span minimization has been achieved. The developer with higher productivity for design tasks (i.e., dev(1)) is assigned to the design task that has been estimated to consume higher effort (i.e., task(1,1)). The developer with lower productivity for design tasks (i.e., dev(2)) is assigned to the design task that needs lower effort (i.e., task(2,1)). Since the productivity factor of developer dev(1) is 2.0 for design tasks, he can finish the design task of feature feat(1), which was estimated to have an effort of ten person-days, within 5 days. Developer dev(2) needs 4 days to complete the design task of feature feat(2), having an estimated effort of four person-days, since she possesses a productivity factor of 1.0 for design tasks. Such allocation allows parallel development as well as balanced developer allocation. Since developer dev(1) is not able to perform testing task successfully (his estimated test productivity factor is 0), he is assigned to both of the remaining coding tasks. Developer dev(2) completes all task in parallel to the coding performed by developer dev(1). The test tasks could not be started earlier since this would violate the *start–start* task dependency constraint. The release make-span of 14.33 days is determined by the end-time of the latest completed task (i.e., task(1, 3)).

3.3. Make-span minimization for operational release planning

The primary concern of the make-span minimization of the ORP problem is implementation of features. Different tasks must be carried out in order to implement the features. For each of the feature dependent tasks, there is an associated effort estimates. There are task dependency constraints between tasks of the same feature. For implementation of all the tasks, a pool of developers is needed. Each developer has estimated productivity rates per task (such as design, coding or testing). The overall question becomes to find

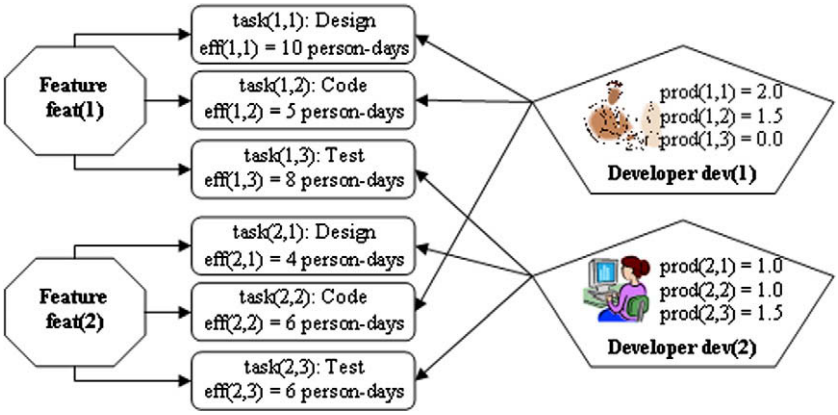

Fig. 1. An example of developer allocation to feature implementation tasks.

Features feat(n)	*Tasks(n)* task(n,q)	*Efforts(n)* eff(n,q)	*Developers* dev(k)	*Productivities(k)* prod(k,q)	Task Execution Times
feat(1)	task(1,1): Design	10	dev(1)	2.0	5
	task(1,2): Code	5	dev(1)	1.5	3.33
	task(1,3): Test	8	dev(2)	1.5	5.33
feat(2)	task(2,1): Design	4	dev(2)	1.0	4
	task(2,2): Code	6	dev(1)	1.5	4
	task(2,3): Test	6	dev(2)	1.5	4

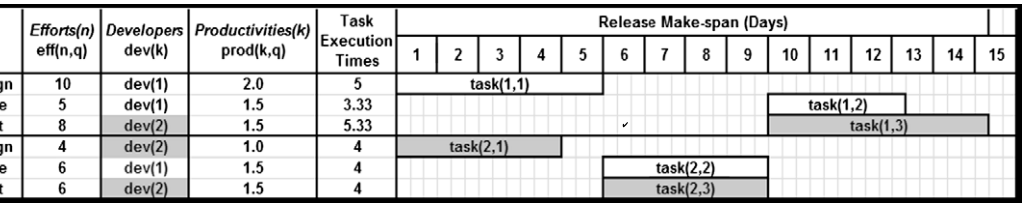

Fig. 2. Gantt chart related to the developer allocation of Fig. 1.

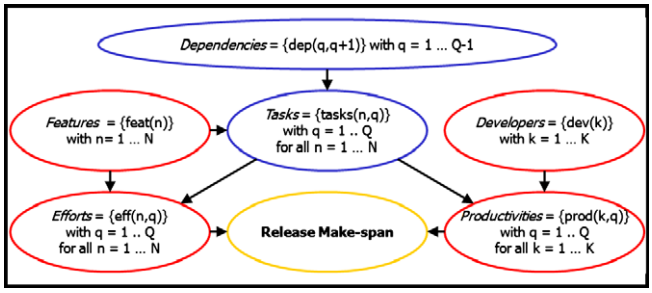

Fig. 3. Dependency graph between the elements of the ORP problem.

assignment of developers to tasks such that the overall make-span is minimized.

Fig. 3 provides a graphical representation of the constituting elements of the ORP problem – i.e., both input and output parameters of the problem. We refer to Sections 3.1 and 3.2 if any further explanation is required for the ORP elements and their notations. Using these key elements, the ORP problem can be formally defined by a 6-tuple ⟨*Features, Tasks, Effort, Dependencies, Developers, Productivities*⟩ where

- *Features* = {feat(1), ..., feat(N)} is the set of features to be implemented in the next release.
- *Tasks(n)* = {task(n, 1), ..., task(n, Q)} is a vector of (possibly Q) tasks necessary to implement feature feat(n) (n = 1, ..., N)
- *Efforts(n)* = {eff(n, 1), ..., eff(n, Q)} is a vector of estimated efforts for the tasks related to feature feat(n) (n = 1, ..., N) .
- *Dependencies* = {dep(q, q + 1) $1 \leqslant q \leqslant Q - 1$, $0 \leqslant \text{dep}(q, q+1) \leqslant 100$} is a vector consisting of dependency values between tasks of the same feature. The degree of dependency applies among the tasks is such that dep(q, q + 1)% of task(n, q) needs to be completed before task(n, q + 1) starts.
- *Developers* = {dev(1), ..., dev(K)} is the set of developers available to develop the features of the release.
- *Productivities(k)* = {prod(k, 1), ..., prod(k, Q)} is a vector of estimated productivities for a developer dev(k) (k = 1, ..., K) to perform tasks task(n, q) (q = 1, ..., Q) for any given feature feat(n).

With all the input data provide above, the make-span problem of operational release planning is defined as finding a schedule of tasks and a feasible assignment of developers to tasks, such that all given features are implemented in minimum *Make-span*. *Make-span* is defined as the total duration to implement all tasks of all defined features. The minimum make-span problem is related to the classical job shop scheduling problem (JSSP), which asks for the scheduling of n jobs job(1), job(2), ..., job(n) on m identical machines. The objective is to minimize the total length of the schedule. Our problem is more general than the JSSP problem because it allows different productivity levels for the different machines (developers) and explicit dependencies between tasks.

4. Problem statement

Our main focus is to analyze the impact of uncertainty on make-span related to the number of features, the estimated efforts, the availability of developers, and the productivities of developers. More precisely, we study the impact of uncertainty in *Features, Efforts, Developers,* and *Productivities* on *Make-span* for an ORP problem ⟨*Features, Tasks, Efforts, Dependencies, Developers, Productivities*⟩. We have formulated the following three research questions.

4.1. Research question RQ1: stochastic impact analysis for individual sources of uncertainty

The first research question investigates how individual changes in *Features, Efforts, Developers,* and *Productivities* changes *Make-span*. RQ1 refines into: How does,

(RQ1.1) a change Δ*Features* in the set of features result in a change Δ*Make-span*?
(RQ1.2) a change Δ*Efforts* in the vector of estimated efforts result in a change Δ*Make-span*?
(RQ1.3) a change Δ*Developers* in the set of developers result in a change Δ*Make-span*?
(RQ1.4) a change Δ*Productivities* in the vector of productivities result in a change Δ*Make-span*?

The Δ in all the input parameters are described by triangular distribution functions TRIANG (Min, Peak, Max) where *Min* stands for the minimum change, *Peak* stands for the most probable change, and *Max* stands for the maximum relative change of the related input parameter. Since input parameters are randomly sampled from their respective triangular distribution, the related analysis is stochastic. Furthermore, Δ for each input parameter is investigated using three TRIANG definitions. The definitions represent three pessimism levels namely *Bad*, *Worse*, and *Worst*.

4.2. Research question RQ2: stochastic impact analysis for multiple sources of uncertainty

The second research question investigates how *Make-span* is impacted, if changes in *Features, Efforts, Developers,* and *Productivities* simultaneously occur in more than one input parameter simultaneously. RQ2 refines into: How do the following combined parameter changes result in a change Δ*Make-span*?

(RQ2.1) Δ*Features*, Δ*Efforts*.
(RQ2.2) Δ*Features*, Δ*Developers*.
(RQ2.3) Δ*Features*, Δ*Productivities*.
(RQ2.4) Δ*Efforts*, Δ*Developers*.
(RQ2.5) Δ*Efforts*, Δ*Productivities*.
(RQ2.6) Δ*Developers*, Δ*Productivities*
(RQ2.7) Δ*Features*, Δ*Efforts*, Δ*Developers*.
(RQ2.8) Δ*Features*, Δ*Efforts*, Δ*Productivities*.
(RQ2.9) Δ*Features*, Δ*Developers*, Δ*Productivities*.
(RQ2.10) Δ*Efforts*, Δ*Developers*, Δ*Productivities*.
(RQ2.11) Δ*Features*, Δ*Efforts*, Δ*Developers*, Δ*Productivities*.

Like for RQ1, the Δ in all the input parameters are described by triangular distribution functions TRIANG (Min, Peak, Max) on three pessimism levels, i.e., *Bad*, *Worse*, and *Worst*. The stochastic analysis is performed to make differences between the influences of input parameter combinations, number of uncertainty sources as well as considered pessimism levels on *Make-span*.

4.3. Research question RQ3: deterministic impact analysis for all sources of uncertainty

The third research question investigates how changes in *Features, Efforts, Developers,* and *Productivities* – both in isolation and in combination (i.e., (RQ1.1) to (RQ1.4) and (RQ2.1) to (RQ2.11)) change *Make-span*. However, in this case, the Δ for each of the input parameters are described by deterministic stepwise changes (instead of distribution functions), and thus the related analysis here is a deterministic impact analysis.

5. Method description

To analyze the impact of uncertainty in ORP as formulated in the above three research questions, we adapted the risk analysis framework ProSim/RA (Project Simulation-based Risk Analysis) [9]. We named the resulting (i.e., after adaptation) method ProSim/ORP (Project Simulation for Operational Release Planning). The framework ProSim/RA requires the availability of a simulation model representing the software development process on an adequate level of granularity. Similarly, the method ProSim/ORP requires an ORP specific simulation model. For our case, we chose DynaReP (Dynamic Re-Planner) [12] as the simulation model, because it can be used for planning and re-planning of a single software release on operational level. In what follows, we present a method called ProSim/ORP being tailored from ProSim/RA towards ORP.

5.1. The method – ProSim/ORP

The method ProSim/ORP is a five step procedure that applies modeling, baseline construction, simulation, and analysis of ORP. These are the steps to be carried out for analyzing a given ORP problem:

- *Step* 1: determining the baseline solution.
- *Step* 2: identifying uncertainty factors and response factors.
- *Step* 3: defining uncertainty ranges and distributions.
- *Step* 4: conducting Monte-Carlo enhanced process simulation.
- *Step* 5: analyzing and interpreting simulation results.

5.1.1. Step 1 – determining the baseline solution

The analysis of the impact of uncertainty is done in relation to a baseline solution. The baseline solution refers to the initial plan that is prepared before development of a release starts. This plan might need to be updated with time due to unforeseen events such as longer completion of a task than expected due to effort underestimation (i.e., imprecise knowledge) or sudden unavailability of developers (i.e., incomplete information). Such unforeseen events may also change the make-span. To pro-actively assess the potential change of make-span, we need to compare make-span of the possible revised plans with the baseline plan before a release development starts. For this reason a simulation model is employed that first produces (simulates) baseline plan and then generates (also by simulation) revised plans by injecting unforeseen events to the baseline solution. More details are provided below.

5.1.2. Step 2 – identifying uncertainty factors and response factors

Uncertainty factors refer to those input parameters of the simulation model whose values are assumed to be stochastic and, thus, are sources of uncertainty and project risk. Uncertainty implies that the correct values of these input variables are not known at the beginning of a release and that their values may change with time during the release development. However, we might not be interested in investigating all uncertainty factors; we rather wish to analyze the impact of those factors that cause major risks for a particular project in a particular organization. In the simulation analysis, uncertainty factors are varied according to a defined distributions (see next step for more detail) which simulates over/under-estimation of effort and/or productivity are taking place before release development start and unforeseen events are happening during release development.

For an ORP problem ⟨*Features, Tasks, Efforts, Dependencies, Developers, Productivities*⟩, the following set of input parameters are candidate uncertainty factors:

- Set of features – *Features*: some (new) features may need to be included/updated/deleted anytime during the release development based on customer feedback. Let us denote this uncertainty factor as F.
- Vector of estimated efforts of feature implementation tasks – *Efforts*: task efforts in order to implement features could have been over/under-estimated before release development starts. Let us denote this uncertainty factor as E.
- Set of available developers – *Developers*: some developers may become unavailable due to illness or being transferred to another project during release development. Let us denote this uncertainty factor as D.
- Vector of estimated productivities of developers for different tasks – *Productivities*: developers' productivities could have been over/under-estimated before release development starts or vary during the development. Let us denote this uncertainty factor as P.

Response factors are the output parameters whose values are affected by the variation of the uncertainty factors. The deviation of response factors from the baseline plan (due to uncertainty factor variations) represents the impact of uncertainty on operational release plans. Therefore, response factors are the main point of interest for managers. For an ORP problem, one important response factor is related to the deviation from the baseline in terms of make-span. While this is the emphasis of the method described in this paper, the method is applicable to both other uncertainty factors and other response factors.

5.1.3. Step 3 – defining uncertainty ranges and distributions

Since the method involves varying uncertainty factor values and observing their corresponding impact on response factors, we need to determine the ranges of variation for uncertainty factors and alter respective values systematically. Systematic variation refers to the fact that a distribution function is constructed for each of the uncertainty factors describing the probability of assuming a particular value. There are two common ways of constructing such distribution functions: (i) experience database (EDB), and (ii) expert opinion.

Construction of experience-based distribution functions is done by fitting generic probability functions (e.g., triangle distribution, normal distribution) to available data from (similar) past projects, for example, extracted from an experience database or EDB. This can automatically be done using commercially available statistical tools such as Stat::Fit (http://www.geerms.com/) or Minitab (http://www.minitab.com/).

If empirical data is not available a distribution function is constructed by interviewing an expert. The triangular distribution TRIANG (Min, Peak, Max) is found to be both efficient and easy to understand and estimate in risk analysis and uncertainty modeling [27]. An expert is asked to estimate the most probable value (peak), the minimal (min), and the maximal (max) value for an uncertainty factor to form a triangular distribution. If more than one expert is available then we define the min, max, and peak values per uncertainty factor either by taking the averages of the experts' estimates or through negotiation between experts. As mentioned for the factors, the method can easily handle other distributions as well.

5.1.4. Step 4 – conducting Monte-Carlo enhanced process simulation

The simulation approach used in ProSim/ORP combines the concepts of Monte-Carlo simulation with discrete-event process simulation. The simulator DynaReP [12] varies stochastic input parameter values as well as injects future possible unforeseen events to the baseline solution according to defined distributions. This is done repeatedly and each time a simulation run is conducted. The values for uncertainty factors are sampled randomly

from the probability functions constructed in STEP 3 to model uncertainty arising from uncertainty factors through Monte-Carlo simulation. This altered information, each time a simulation run is conducted, is then handed over to the process simulation part of DynaReP that models the ORP process steps and their dependencies for developer-to-task assignment and computes release make-span. As a result of multiple replications of simulation runs, generated output form a distribution that reports the induced variation of the response factors.

5.1.5. Step 5 – analyzing and interpreting simulation results

Typical interpretation of simulation results involves summary and descriptive statistical analyses such as mean, standard deviation, median, minimum, and maximum values for response factors. This represents comparison between the baseline plan and revised plans resulting from systematic variation of uncertainty factors. In particular, the magnitude and the nature of impact on the response factors (i.e., deviation from the baseline) are studied. Box plots are an effective tool for such analysis.

The impact on response factors due to variations in uncertainty factors need to be measured appropriately. Al-Emran et al. [28] defined a measurement procedure to compute the impact of uncertainty factors on some uncertainty factors, namely release make-span, schedule structure, and developer assignment.

In our case, the relative change in make-span (response factor) due to the variation of uncertainty factors is computed for a particular simulation run r using the following formula (a positive outcome value means an increment in make-span whereas a negative outcome value indicates a decrement):

$$\text{Make-span Increment}_r = (\text{Make-span}_r - \text{Make-span}_{\text{baseline}})/\text{Make-span}_{\text{baseline}}\%$$

Since uncertainty factors are sampled from probability distributions, we can derive probability distributions for response factors as well. The types and parameters of these distribution functions could be another interesting issue for study. Whatever the response factor is, it should be studied based on the interests of decision makers. For example, a decision maker might be interested in average change, maximum possible impact or range of deviation.

5.2. The simulation model – DynaReP

The simulation model used to perform the case study is DynaReP (Dynamic Re-Planner) originally presented in [12]. DynaReP is a discrete-event process simulation model developed using EXTEND™ (http://www.imaginethatinc.com). The simulation model is able to perform operational release planning as well as automatic re-planning of operational releases in response to imprecise knowledge and incomplete information (c.f., Section 2). Once again, examples of imprecise knowledge are effort and/or productivity under/over-estimation whereas examples of incomplete information are developer dropouts and/or late feature inclusions. All the uncertainty factors specified in Section 5.1.2 i.e., E, P, F, and D are modeled in DynaReP as its set of input parameters.

For the research presented in this paper, DynaReP has been applied as the engine for proactive analyses so that managers can study the impact of future possible pessimistic scenarios mentioned above. The simulator creates the pessimistic scenarios based on the distribution functions defined for the uncertainty factors described in Section 5.1.3. These scenarios inject events such as estimation errors, developer dropouts and late feature inclusions into the baseline plan in different time of the release development period. In response, the simulator performs automatic re-planning based on the remaining workload and available workforce. As a result, the make-span (which is the response factor in our case) may have been changed. We can determine the respective impact by measuring the difference between the new release make-span and original (i.e., baseline) make-span.

In the following, we provide a brief description of the structure of DynaReP and its optimization heuristic for developer-to-task assignment. Detailed information on the model can be found in [12].

5.2.1. Building blocks of DynaReP

DynaReP consists of ten high-level blocks, each of which can be further decomposed. Fig. 4 shows how these blocks are connected to each other. There are two types of connections: (i) Entity link, and (ii) Information link. Entity links are paths through which entities are routed from one block to another. Information links are connections through which data transfer between blocks take

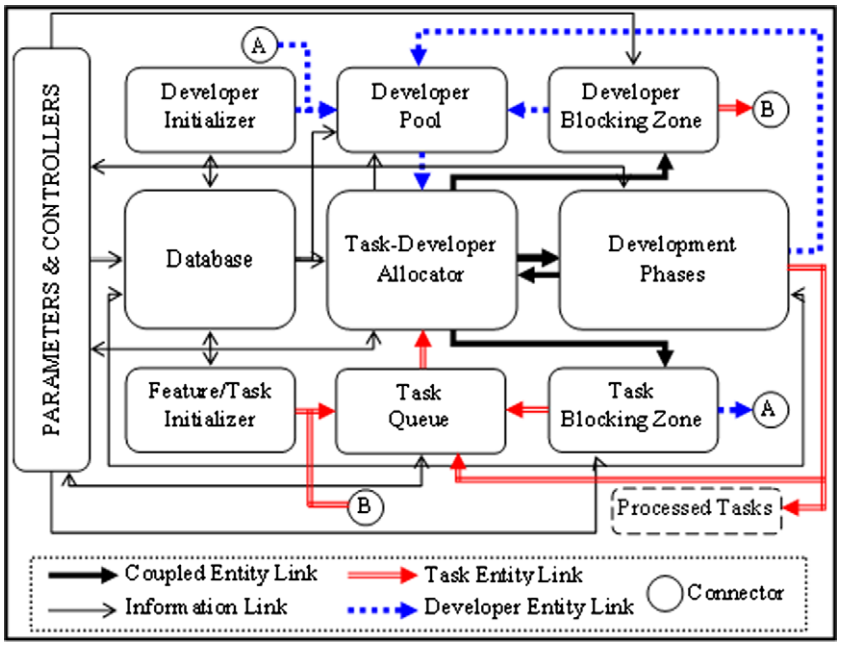

Fig. 4. High level view of DynaReP model structure.

place. Since DynaReP assigns developers to perform feature implementation tasks, there exist three types of entities in DynaReP: (1) Task-Entity (to carry information about features, tasks, and their associated estimated efforts), (2) Developer-Entity (to carry information about developers and their associated productivities for different tasks), and (3) Coupled-Entity (when the first two entities have been merged to form one entity representing that a developer has been assigned to a task).

In what follows, we summarize the functionalities of DynaReP's building blocks:

- *Database*: stores all model input related information i.e., *Features, Tasks, Efforts, Dependencies, Developers, and Productivities*.
- *Feature/Task Initializer*: creates entities to represent features and divides them into Task-Entities to represent all tasks; initializes them with their respective information from *Database*; and sends them to *Task Queue* block.
- *Task Queue*: holds incomplete Task-Entities and forwards them (according to the heuristic described later) to *Task-Developer Allocator* block for assigning developers.
- *Task Blocking Zone*: keeps a Task-Entity away from developer allocation consideration when the task does not meet dependency constraints; and sends it back to *Task Queue* after a successful developer-to-task assignment for later consideration.
- *Developer Initializer*: creates Developer-Entities; initializes them with their respective information from *Database*; and sends them to the *Developer Pool* block.
- *Developer Pool*: holds Developer-Entities and releases them (according to the heuristic described later) to *Task-Developer* allocator block for task assignment.
- *Developer Blocking Zone*: keeps a Developer-Entity away from developer allocation consideration when the corresponding developer does not possess enough productivity to carry out a specific task; and releases it back to *Developer Pool* after a successful developer-to-task assignment for later consideration.
- *Task-Developer Allocator*: combines Developer-Entities with Task-Entities to simulate the assignments of developers to feature implementation tasks according to provided staffing policy; checks whether their combination meets all necessary conditions; and depending on that, sends the Coupled-Entity

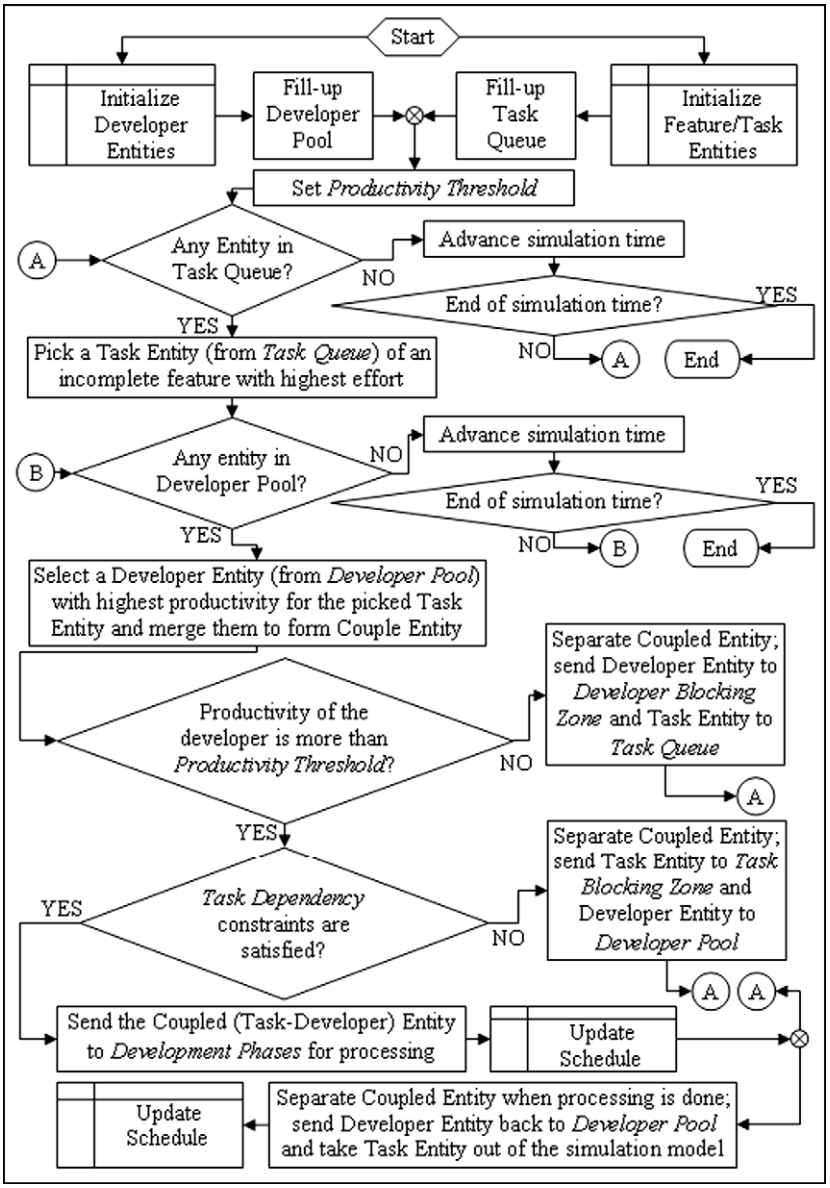

Fig. 5. Flowchart for optimization heuristic of DynaReP.

to either *Development Phases* (if conditions are satisfied) or to either *Task Blocking Zone or Developer Blocking Zone* (depending on type of condition failed).

- *Development Phases*: simulates the behavior of developers working on tasks by holding Coupled-Entities for some calculated simulation time representing task execution time as described at the end of Section 3.3. After completing a task, the respective Coupled-Entity is decoupled to form separate Task-Entity and Developer-Entity. The Developer-Entity is sent back to the *Developer Pool* to be considered for other task assignments and the Task-Entity is taken out of the simulation model (shown as *Processed Tasks* in Fig. 4) since the task is already processed.
- *Parameters & Controllers*: defines and manipulates all model parameters and controllers, for example, number of input entities, their associated values, how and when they will change during simulation runs, etc.

5.2.2. Optimization heuristic of DynaReP

As mentioned before (c.f., Section 5.1.4), the process simulation part of DynaReP models ORP process steps and their dependencies for developer-to-task assignment. We want to make a note here that we do not intend to produce an optimal schedule for make-span minimization using all the available information (from the model's input parameters) like general scheduling algorithms do. Rather, we aim to evaluate (from the perspective of robust planning) the heuristic (i.e., staffing policy) used by Chatwell's managers for make-span minimization. The heuristic here is just a plug-in to the model and interchangeable with any other heuristic for other optimization purposes. For more information on different resource-constrained project scheduling heuristics, we refer to [29,30].

The flowchart for assignment heuristic for available developers to the next tasks to be done at any point in time is shown in Fig. 5. The heuristic matches the next available developer having highest task-specific productivity to the next waiting task of an incomplete feature requiring the largest effort. A developer can only be assigned to a single task at a time and one task is carried out by only one developer unless the developer is dropped in the middle of a task completion. Features are treated independent to each other; however, there exists at least a start–start and end–end dependency between two consecutive tasks of the same feature. One example could be testing of a particular feature cannot be started before coding of that feature starts (start–start dependency) and testing of that feature cannot be finished before coding finishes (end–end dependency).

The only phenomenon that could not be modeled as part of the heuristic is a project managers' intuition-based rationale for not assigning tasks to developers with too low productivities for those tasks (and who may possess higher productivities in other tasks which, however, are currently not available for them). In order to capture such intuition when using the heuristic, a threshold variable is defined for each kind of tasks called *Productivity Threshold*, and the heuristic does not allow developers to be assigned to tasks for which their productivity is not greater than the corresponding threshold. The optimal threshold values for different kind of tasks in terms of make-span minimization following the specified staffing policy are computed by the simulation model itself. For this purpose, a built-in optimization library block called 'Evolutionary Optimizer' is included in the model that comes with the EXTEND ™ (http://www.imaginethatinc.com) simulation modeling tool. Since models are abstractions of the real world and everything cannot be reproduced by a model, we have to live with this limitation. In addition, we deemphasize the role of 'exact' figures or numbers in the proposed impact analysis; instead, we place an emphasis on impact tendency and potential negative effects.

To recapitulate, a developer dev(k) will be assigned to a task task(n, q) if all of the following assumptions or conditions are fulfilled:

- No developer is currently assigned to task(n, q).
- There is work still pending for the task task(n, q), i.e., the related feature feat(n) is incomplete.
- The effort of the associated feature feat(n) given by eff(n, q), with $q = 1, \ldots, Q$, is the maximum over all incomplete features.
- The candidate developer dev(k) is currently idle.
- The productivity prod(k, q) of the candidate developer dev(k) is the maximum over all currently idle developers and greater than the threshold productivity for task task(n, q) for any feature feat(n).
- For $q \neq 1$, task(n, q) cannot be completed before task(n, $q-1$) is completed.
- For $q \neq 1$, at least a work-volume of $\text{eff}(n, q-1) * \text{dep}(q-1, q)$ of the task task(n, $q-1$) must be completed.

6. Case study

6.1. Context

To demonstrate the applicability of the proposed method, we conducted a case study based on real-world release planning data of ongoing product development at Chartwell Technology Inc. Chartwell is a leading developer and supplier of Internet gaming software systems for the online and mobile gaming industry.

Chartwell's project managers were intuitively aware of the uncertainty arising from different planning factors but lacked the means to quantify and accordingly mitigate the associated risks. Consequently, the process of ORP has been a principally a manual activity until recently. In the absence of tool support for managing uncertainty, project managers resorted to lengthy planning sessions and past experience to generate candidate operational plans. The manual approach had numerous shortcomings including lack of repeatability, high labor cost and no provision for easy re-planning.

6.2. Objectives

As mentioned earlier, Chartwell is mainly interested in assessing robustness of the operational release plans, resulting from their current staffing policy that they use for make-span minimization, in the face of uncertainty involved in two uncertainty factors: *Efforts* (E) and *Productivities* (P). Strictly speaking Chartwell is not immediately concerned with uncertainty factors *Features* (F) and *Developers* (D) as the number of features is largely controlled by Chartwell via contractual commitments and very few changes in number of developers have happened in the past, However, Chartwell remains interested in studying the uncertainty associated to *Features* (F) and *Developers* (D) as a prelude to potential changes in their organizational policies. Potential impact of uncertainty factor variations are measured by comparing the values of response factors of the baseline plan with those of the revised plans.

The objective of the study was to better understand the behavior of response factor due to uncertainty involved in different uncertainty factors for operational release plans. The most interesting response factor for Chartwell is release make-span. In order to provide information aligned to Chartwell's needs, three levels of pessimism were considered: *Bad*, *Worse*, and *Worst*. These levels of pessimism represent three categories of worst case scenarios involving uncertainty in increasing order. Each level of pessimism comes with its own probability distribution functions for the ORP uncertainty factors: Effort, Productivity, Feature, and Developer

to represent different levels of uncertainty. In case of Effort and Productivity the probability distributions represent inaccuracy in the estimation process i.e., imprecise knowledge. Probability distributions for Feature addition and Developer dropout represent unforeseen changes during the ongoing release development process.

6.3. Data collection

Acquisition of data for the case study was done with the assistance of two senior staff members at Chartwell with an average of 15 years of industrial experience. Both individuals were technical managers within the organization and thus intimately familiar with the technical details of the 35 features included in our case study in addition to being responsible for the management of those developers assigned to implement said features.

Beginning with the entire set of development tasks (>100) faced by their team for the upcoming product release the two managers selected 35 features for the release using a strategic release planning method. Suitability for inclusion was determined when a full set of requirements were available for the feature – thus permitting realistic effort estimates – or when the feature was already under development and current progress data could be used to extrapolate the total effort involved. Operating independently each manager determined effort estimate for the three tasks associated with each feature. The mangers then met to review their estimates and generate a consensus work estimates for all 35 features. For those features currently under development the time already expended by developers was included in the estimates.

The available pool of 15 developers represents the union of the development team(s) which report to the two managers. To determine productivity values for different kinds of tasks, results of the previous years' performance reviews were used. In the annual performance reviews both of the managers rated each staff members' productivity in a number of key competency areas (e.g., Core Java programming, QA skills) on a scale of 0 (no expectations) to 2 (exceptional performance) with 1 being average performance. In addition, each employee provides a self evaluation of their own abilities in each area during the review. All these values are averaged to define productivity of a developer for a different feature implementation tasks.

6.4. Case study data

In this section, we characterize the release under investigation. Due to corporate confidentiality, we cannot release the complete dataset. However, we report partial data in Fig. 6 for the following ORP uncertainty factors (i.e., F, E, D, and P) so that readers have a working understanding of the dataset. The studied release includes:

- A set of 35 features to be developed: *Features* = {feat(1), ..., feat(35)}.
- Each feature involves three tasks per features: *Tasks(n)* = {task(n, 1), ..., task(n, 3)} for $n = 1, \ldots, 35$.
- A pool of 15 available developers: *Developers* = {dev(1), ..., dev(15)}.
- The estimated efforts for all feature implementation tasks i.e., *Efforts*(n) and the estimated productivity for all developers for different tasks i.e., *Productivities*(k) are reported in Fig. 6. Note that we drop the indices n and k in the following when a relative change applies equally to all features N and all developers K.
- The type of task dependencies considered for this project are start–start and end–end constraints between tasks of the same features, i.e., task(n, q) can start if and only if task(n, $q-1$) has started and task task(n, q) cannot be finished before task(n, $q-1$) has been completed.

Each of the 12 cases shown in Table 1 represents different probability distribution sampling patterns for effort and productivity estimates, as well as for the numbers of newly added features and the numbers of developer dropouts. The variations in these uncertainty factor values are defined based on triangle distributions of the type TRIANG (Min, Peak, Max). 'Peak' represents the most probable value while 'Min' and 'Max' represent the minimum and maximum possible values, respectively.

In the cases of factors Effort and Productivity, variations are expressed in terms of relative change (percentage) with respect to their original values (baseline). For example, most of the effort values in the *Worse* pessimism level remain close to the baseline estimation, some were underestimated by up to 40% and some were overestimated by up to 10%. These distribution functions were defined by the two senior managers mentioned in Section 6.3 based on their observations from past years' records.

In the cases of the factors Feature and Developer, the values represent the number of additional features and the number of dropped out developers, respectively. Again, variations are expressed as relative changes (percentage) with respect to the total numbers of features and developers in the baseline plan (i.e., 35 features and 15 developers). For these two factors, the distribution range is always from 0% to 30% but the peak is 0%, 15%, and 30% for *Bad*, *Worse*, and *Worst* levels, respectively. Since Chartwell did not

feat(n)	eff(n,1)	eff(n,2)	eff(n,3)	feat(n)	eff(n,1)	eff(n,2)	eff(n,3)
feat(1)	*	20	5	feat(19)	0	25	*
feat(2)	*	20	5	feat(20)	0	25	*
feat(3)	*	40	5	feat(21)	0	25	*
feat(4)	*	30	10	feat(22)	0	20	*
feat(5)	*	50	10	feat(23)	10	5	*
feat(6)	*	50	0	feat(24)	10	5	*
feat(7)	10	*	5	feat(25)	5	*	0
feat(8)	0	*	5	feat(26)	5	*	0
feat(9)	0	*	5	feat(27)	5	*	0
feat(10)	0	*	5	feat(28)	5	*	0
feat(11)	0	*	5	feat(29)	5	*	0
feat(12)	0	*	5	feat(30)	5	*	0
feat(13)	5	15	*	feat(31)	*	15	0
feat(14)	0	20	*	feat(32)	*	20	0
feat(15)	5	25	*	feat(33)	*	55	0
feat(16)	0	20	*	feat(34)	*	10	0
feat(17)	0	20	*	feat(35)	*	60	15
feat(18)	0	25	*				

(a)

dev(k)	prod(k,1)	prod(k,2)	prod(k,3)
dev(1)	*	1.25	1
dev(2)	0	*	1.25
dev(3)	0.75	1	*
dev(4)	0.75	1.25	*
dev(5)	0.75	*	1.25
dev(6)	*	1.25	1.25
dev(7)	*	0.75	1
dev(8)	0	*	1
dev(9)	1.25	0.75	*
dev(10)	0.75	1.25	*
dev(11)	0.75	*	1.25
dev(12)	*	1.25	1.25
dev(13)	*	1.25	0
dev(14)	0.75	*	1.25
dev(15)	0	1.25	*

(b)

Fig. 6. Case study data for: (a) *Efforts* and (b) *Productivities*.

Table 1
Definition of triangular distributions at different levels of pessimism.

Case	Levels of pessimism	Variation distributions			
		Uncertainty factor	Min (%)	Peak (%)	Max (%)
1	Bad	Effort (E)	−20	0	30
2		Productivity (P)	−30	0	20
3		Feature (F)	0	0	30
4		Developer (D)	0	0	30
5	Worse	Effort (E)	−10	0	40
6		Productivity (P)	−40	0	10
7		Feature (F)	0	15	30
8		Developer (D)	0	15	30
9	Worst	Effort (E)	0	50	50
10		Productivity (P)	−50	−50	0
11		Feature (F)	0	30	30
12		Developer (D)	0	30	30

possess any past record on these two factors, opinions of two managers were used to define the corresponding distribution functions. The occurrence times for developer dropouts and feature inclusions are expressed as uniform distribution over the baseline make-span. The effort values for the tasks associated with additional features were sampled from a probability distribution derived from past project history by the two Chartwell managers.

6.5. Baseline plan

Based on the data provided by Chartwell, the initial (i.e., baseline) plan was generated. Fig. 6 provides case study data for *Efforts* (E) and *Productivities* (P). Due to confidentiality reasons, we could only report partial data; some of the original data has been replaced by an '∗'. Fig. 7 shows part of the baseline plan that includes schedule and developer assignment for tasks for second half of the features i.e., feat(19) to feat (35). This baseline plan estimates that the release should be completed within 75 workings days i.e., 15 calendar weeks, if all estimations are correct and no unexpected event occurs.

6.6. Stochastic impact analysis for individual sources of uncertainty (RQ1)

6.6.1. Question

As a first step towards the stated case study objective, an analysis of the impact of single factor variation was done. More precisely, for all the four selected uncertainty factors, simulation runs were done to investigate the impact of variation on make-span. Each of the cases 1–12, as specified in Table 1, was executed 50 times with varying values. Each time, the Monte-Carlo simulation samples a value from the defined triangular distribution for the factors.

6.6.2. Results

The box plots provided in Fig. 8 summarizes the statistical information of the resulting deviation for response factor – make-span. Here, the reported deviation refers to increase in make-span compared to the baseline case. This increase in make-span is caused by variation of only one of the four uncertainty factors (i.e., D, E, F,

Features	Tasks(n)	Developers	Start Times	End Times
feat(19)	task(19,2)	dev(1)	37	57
feat(20)	task(20,2)	dev(11)	37	57
feat(21)	task(21,2)	dev(8)	40	60
feat(22)	task(22,2)	dev(10)	50	66
feat(23)	task(23,1)	dev(3)	51	65
	task(23,2)	dev(8)	61	66
	task(23,3)	dev(2)	62	66
feat(24)	task(24,1)	dev(4)	53	67
	task(24,2)	dev(13)	64	68
	task(24,3)	dev(1)	64	69
feat(25)	task(25,1)	dev(5)	28	35
	task(25,2)	dev(10)	28	48
feat(26)	task(26,1)	dev(3)	0	7
	task(26,2)	dev(2)	0	30
feat(27)	task(27,1)	dev(9)	56	60
	task(27,2)	dev(5)	56	68
feat(28)	task(28,1)	dev(9)	39	43
	task(28,2)	dev(6)	40	56
feat(29)	task(29,1)	dev(11)	28	35
	task(29,2)	dev(14)	28	48
feat(30)	task(30,1)	dev(1)	57	64
	task(30,2)	dev(11)	57	69
feat(31)	task(31,1)	dev(9)	61	65
	task(31,2)	dev(6)	61	73
feat(32)	task(32,1)	dev(9)	44	48
	task(32,2)	dev(13)	48	64
feat(33)	task(33,1)	dev(1)	0	7
	task(33,2)	dev(15)	0	44
feat(34)	task(34,2)	dev(8)	67	75
feat(35)	task(35,1)	dev(9)	0	32
	task(35,2)	dev(13)	0	48
	task(35,3)	dev(7)	33	48

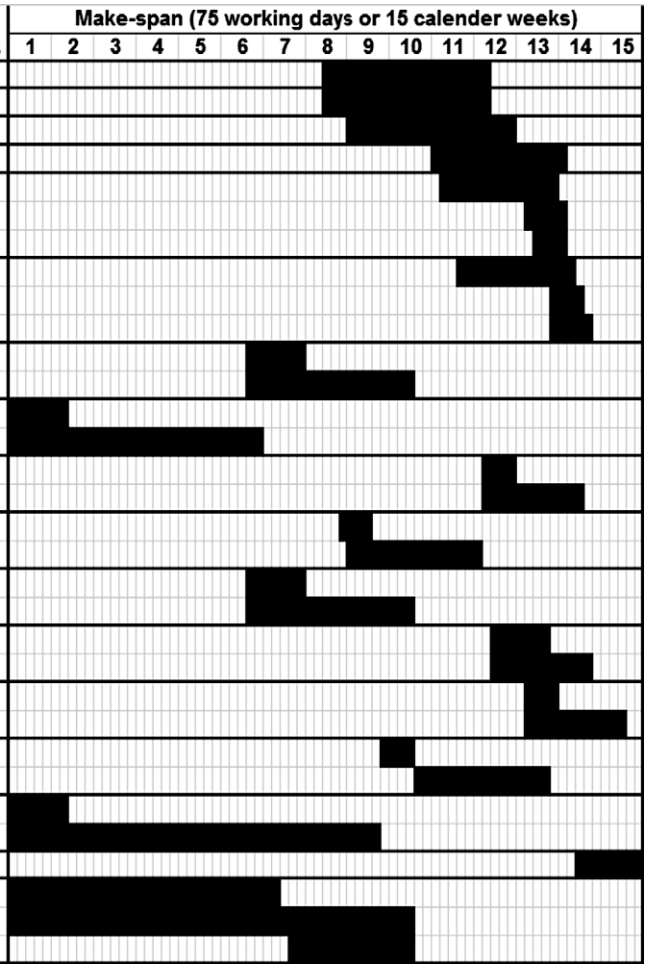

Fig. 7. Baseline plan (feat(19)–feat(35)).

and P) at a time and the analysis is done at all three levels of pessimism.

The process simulation follows the developer-to-task allocation heuristic and uses the sampled values from the Monte-Carlo simulation to compute the make-span deviation profile. For example, make-span increment due to Feature factor variation in the *Worse* pessimism level is 28% on average with the following properties:

- lower quartile (Q1) is at 11.67%,
- higher quartile (Q3) is at 40%,
- inter quartile range (Q3–Q1) is 28.33%
- median (Q2) is at 20%,
- smallest non-outlier observation is at 0%,
- largest non-outlier observation is at 66.67%, and
- two outliers are at 100% and 120%.

6.6.3. Discussion

For each of the four factors, we observe a tendency of increase in the make-span when going from *Bad* to *Worse* to *Worst*. This tendency of increase applies to both the mean value and the total make-span range. It is difficult to make a comparison between the influences of uncertainty factors on response factor make-span because the triangular functions from which uncertainty factor values were sampled are not identical.

For the functions defined in the case study, the strongest impact came from the variation of the number of features (being added during the process of already agreed upon features). In this case, the mean value increases by 21%, 28%, 49% for the three levels of pessimism, respectively. The higher quartile level (Q3) increases by 28%, 40%, 73%, respectively for the three pessimism levels.

At present Chartwell limits the number of features to be incorporated in a release prior to the commencement of development. Rigid limitations on the number of features are a common means of limiting scope creep in 'fixed-price' contracts. However, some customers my request flexibility in the number of features provided so that they can quickly respond to changing business circumstances (e.g., a new feature to complement a recent promotional campaign). Our proactive analysis is supportive for answering what would happen if Chartwell made changes in policy and allowed customers to change their minds even after release development had already started. If the resulting impacts are not excessively high then they may allow on-demand feature inclusion in order to increase customer satisfaction. Alternatively, if the resulting impact is excessively high, then Chartwell incurs significant risk under a 'fixed-price' contract and may advocate a 'time and materials' type of agreement.

Another interesting observation is that the impact of developer dropout was not as catastrophic as was expected (the first three cases of Fig. 8). In fact, the corresponding impact is at about the same level as the impact of the variation in effort (the last three cases of Fig. 8). Form this we conclude that the project is overstaffed and, as a result, developer dropout has a negligible impact. This conclusion is consistent with the results published in [4] where the analysis showed that replacement of Chartwell's current project staffing policy by an optimization method would yield a better resource allocation with even a smaller development team.

6.7. Stochastic impact analysis for multiple sources of uncertainty (RQ2)

6.7.1. Question

As a second direction for analysis, we investigated the impact of combined factors. More precisely, for all combinations of the four selected uncertainty factors, simulation runs were done to investigate the impact of variation on make-span. Each time, the Monte-Carlo simulation samples values from the defined triangular distribution for the uncertainty factors. To limit the number of possible combinations, we only studied combinations of uncertainty factors at the same levels of pessimism.

6.7.2. Results

Fig. 9 reports mean impact due to individual effect of uncertainty factors and, in order to draw a comparison, their combined effect (two, three and four factors are active, respectively) for the *Bad*, *Worse* and *Worst* levels of pessimism. In addition, the impacts of different numbers of combined factors as well as different levels of pessimism are compared in Figs. 10 and 11, respectively, using a dot plot, i.e., a special kind of frequency distribution graph.

6.7.3. Discussion

A scenario posed by one manager at Chartwell was that 'a large increase of features in a given project (i.e., pessimism level Worse) prompts some developers to leave for other projects thus affecting both the effort and productivity of the developers who remain behind'. Translated into the context of our case study, the related analysis addresses the question 'What happens with regards to make-span, if multiple things go wrong at the same time?'

Several observations can be made looking at the analysis results. Fig. 9 shows that the mean impact increases when a uncertainty factor is added to another uncertainty factor or an already existing combination of uncertainty factors. The tendency of increasing impact becomes clearer from Fig. 10 where the impact

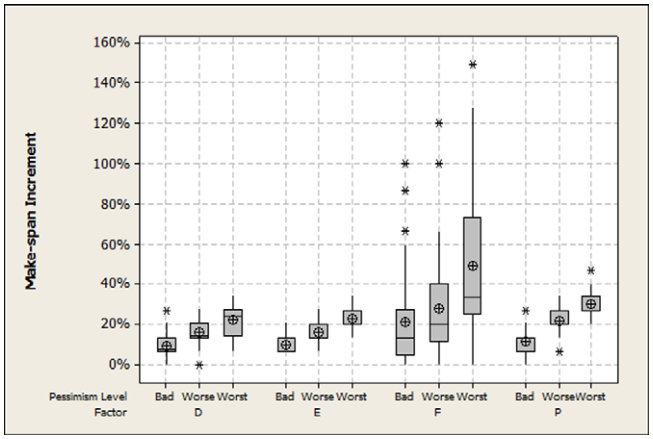

Fig. 8. Impact of single factor variation on make-span.

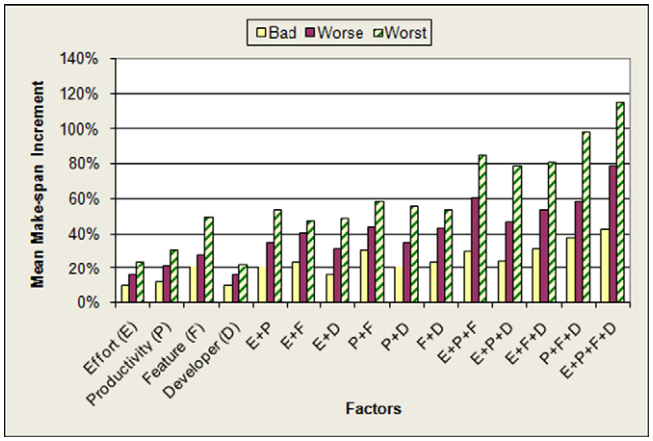

Fig. 9. Mean impact when uncertainty factors are effective in isolation and in combination for different levels of pessimism.

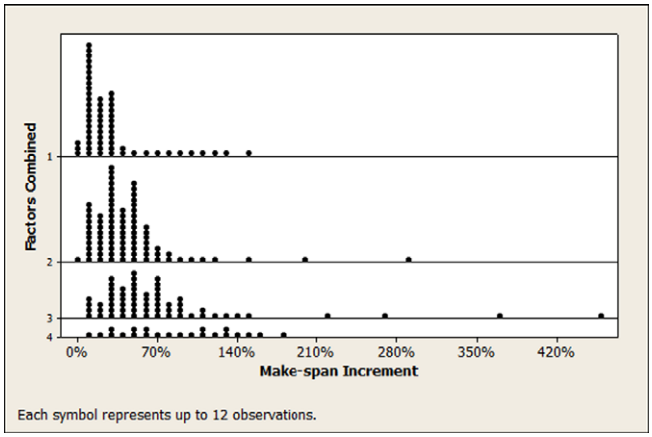

Fig. 10. Distribution of make-span increase for 1, 2, 3, and 4 factors varied simultaneously.

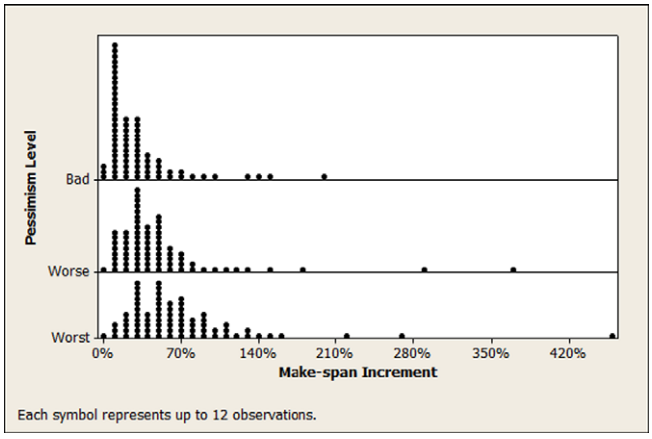

Fig. 11. Distribution of make-span increase for three pessimism levels.

distribution (i.e., the distribution of make-span increase) is spreading out as the number of combined factors increases. The same type of trend can be seen in Fig. 11 where the impact distributions are getting wider as levels of pessimism increase.

6.8. Deterministic impact analysis for all sources of uncertainty (RQ3)

6.8.1. Question

In addition to the stochastic impact analysis, we also performed deterministic impact analysis where single or multiple factors are varied deterministically. Different to the stochastic analysis, we assume a specific (deterministic) relative increase of effort, a specific percentage of features arriving late, a specific relative decrease in productivity, and a specific percentage of developer dropouts. These specific data points lie between the lowest and highest data points of the defined triangular distribution functions for uncertainty factors.

6.8.2. Results

In Fig. 12, the relative increase in make-span at five levels of deterministic variation is presented for each individual uncertainty factor. The five levels are defined in Table 2.

The same type of analysis is done for studying the impact of (deterministic) variation of combined uncertainty factors. The results of this analysis are shown in Fig. 13.

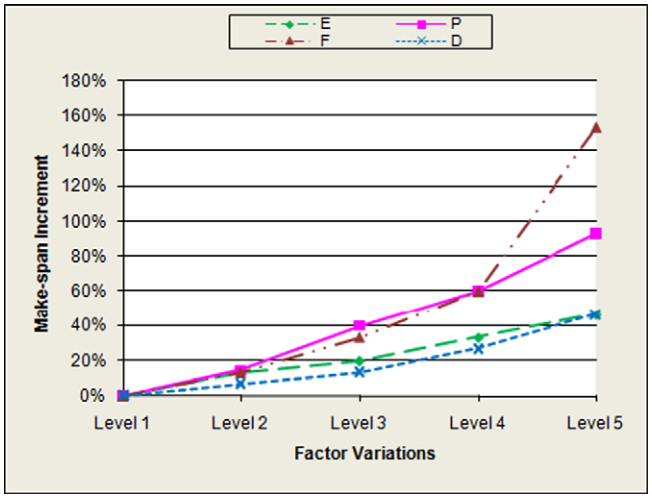

Fig. 12. Deterministic impact analysis for single factor variation.

Table 2
Definition for variation of uncertainty factors at different levels of deterministic cases.

Uncertainty factors	Relative increase of uncertainty factors				
	Level 1 (%)	Level 2 (%)	Level 3 (%)	Level 4 (%)	Level 5 (%)
E	0.0	12.5	25.0	37.5	50.0
P	0.0	12.5	25.0	37.5	50.0
F	0.0	7.5	15.0	22.5	30.0
D	0.0	7.5	15.0	22.5	30.0

6.8.3. Discussion

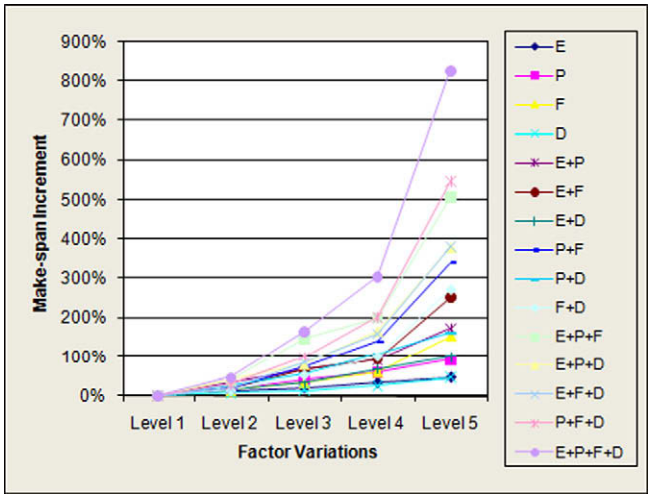

Fig. 13. Deterministic impact analysis for multiple factor variation.

In addition, we compared the actual impact of multiple uncertainty factors (i.e., their simultaneous impact) vs. the impact calculated by adding up the respective single uncertainty factor impacts (called 'actual impact' vs. 'additive impact', respectively). For example, for uncertainty factors x and y, actual impact can be expressed as Impact($x + y$) whereas additive impact can be expressed as Impact(x) + Impact(y). The results are shown in Fig. 14. From Fig. 14, we observe that the actual impacts of combining all the uncertainty factors are stronger than adding up individual impacts. We also observe that the gap becomes the larger the more uncertainty factors are involved in the analysis.

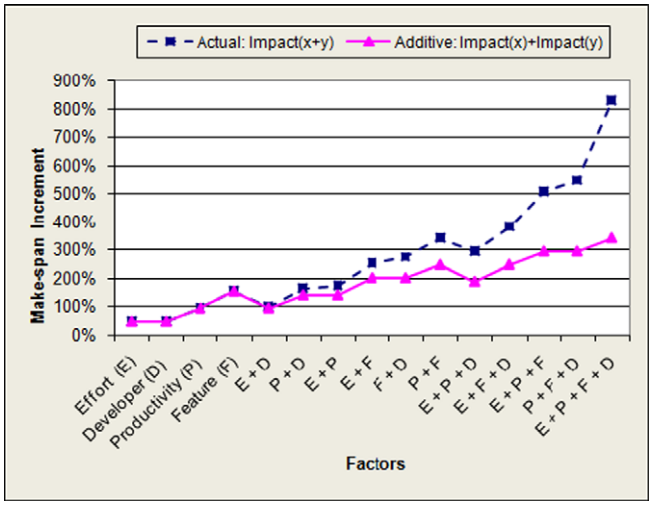

Fig. 14. Comparison between: (i) additive impact, and (ii) actual impact.

7. Threats to validity

From running a series of simulation experiments we gained insight into the nature of sensitivity of release operational planning parameters of Chartwell Technology Inc. While the results from the case study have been confirmed to be meaningful for Chartwell, we need to discuss the different aspects of validity of the results. For that, we follow the validity classification as presented in [31].

Construct validity pertains to how well the measures in a study reflect the concepts under investigation and also to how well-defined the concepts are. We have selected four uncertainty factors to capture uncertainty in the context of release planning. These factors have been confirmed by Chartwell to be of high relevance. The selection of make-span as the response factor also arose from suggestions by Chartwell. Other criteria (such as maximizing the value) in case of a fixed release scenario, are possible and would not change the principal method.

The nature of uncertainty is described by triangular distributions for all stochastic uncertainty factors. However, it is not the only possible one. Since neither the authors nor the experts in Chartwell were 100% sure what type of distribution is the most realistic to pick, the most frequently used (i.e., triangular distribution) was selected.

All solutions of the ORP problem were calculated by running process simulations with an embedded heuristic that represents a specific policy for assigning developers to tasks. The applied heuristic has previously been shown to estimate about 5–10% more make-span for developing a single release as compared to an optimization algorithm for five randomly defined cases [12]. However, as is the case for any heuristic, we cannot precisely evaluate the quality of the solution for a specific scenario. This issue might also limit the comparability between different scenarios.

Internal validity concerns the extent to which observed differences can be attributed to an experimental manipulation, and thus requires the consideration of competing explanations for an observed impact. Since our results are heavily relying on a computerized simulation model, in principle, this should be the easiest type of validity to maximize. The simulated environment offers the experimenter a sterile setting in which entities adhere strictly to whatever heuristics they are assigned and within selected parameter bounds. Competing explanations should be completely controllable and the influence of confounding variables minimized.

External validity is the degree to which the findings in a local setting, containing a single set of sampling units, are applicable to the population of sampling units as well as other settings. Traditionally replication is the default means of maximizing external validity. In our particular case, external validity of our findings is limited in many ways. For the most part of our case study, we used a simulation model that is supposed to capture properties of real-world developers as well as the mechanisms that are governing the decisions how developers are assigned to tasks and the ways, how developers once assigned to a task perform this task. One way of increasing the external validity of our results would be to replicate the case study using other types of simulation models. Another way to increase the external validity of our findings would be to replicate the case study with other sets of features and associated properties.

Another threat to validity is the underlying model assumptions. As any model, our model is an abstraction not able to accommodate all aspects of real-world software release planning. For example, productivity might vary over time and, in addition, might vary not only depending on the task, but also on the specific feature. Without specific evidence that supports or refutes or model assumptions we argue that there is currently no reason to expect fundamentally different results from more detailed models. Moreover the reason that our current estimates are assumed to represent the most typical change is because variations over time and between tasks are approximated by a normal distribution.

While stressing the limitations of the validity of the results, we also want to emphasize that the overall methodology is applicable more broadly in the context of simulation-based analysis. It is neither restricted to discrete-event simulation, nor to the application for ORP. In order to apply the method to other simulation-based analysis, necessary adjustments to the simulation model and the inherent algorithms (heuristics) would be required.

8. Summary and conclusions

The very recent CHAOS report [32] claims that this year's project failure rate is the highest over the last decade with a significant downturn in project success rate compared to the previous study. Triggered by this result, The Standish Group conducted 307 surveys and 30 interviews, to determine the participating organizations' predominant concerns with regard to achieving project success. Their investigation says "resource management, with the ability to do project-level resource allocation" and "project management with . . ., risk assessment" are among the most often mentioned for achieving effectiveness and monetary value [33]. Therefore, both operational software release planning (which deals with project level staffing) and uncertainty analysis (which represents risk assessment) are of significant practical importance. Deciding on the staffing for product releases can take about one third of the time of a product or project manager [4]. In order to be better prepared for risk mitigation strategies, it is important to pro-actively evaluate the possible impact of different forms of pessimistic scenarios.

The main result of the research presented in this paper is an integrated and customizable method for analyzing and estimating the impact of uncertainty of planning parameters (which are defined as uncertainty factors). This was done for both stochastic analysis (using Monte-Carlo simulation) as well as deterministic (sensitivity) analysis. While the method is applicable for any combination of uncertainty factors, we specifically studied the impact of variation in productivity, effort, arrival of new features, and availability of developers (i.e., uncertainty factors) on make-span (i.e., response factor). The method combines Monte-Carlo with process simulation. In each run, an optimization heuristic for assignment of developers to tasks is applied which is applied by the Chartwell managers for their manual release planning process.

The proposed method is customizable in the sense that it allows adaption to other parameters (i.e., both uncertainty factors and response factors), other forms of probability distributions, and other levels of defined pessimism. For the case study project from Chartwell, its applicability, and usefulness was demonstrated.

As it is often the case in simulation, the emphasis of the results should be on output range, most likely result, and shape of distribution; not the exact numbers. For example, if the delta in distribution of an response factor is not noticeable after a shift in uncertainty factor to an upper pessimism level, uncertainty involved in that parameter are not of importance (e.g., results for developer dropout reported in the case study). In that sense, we think the presented results based on scenarios derived from a real-world case study are relevant and provide new insights. However, more comprehensive empirical evaluation with real-world data is necessary to further study external validity of the results.

The results of the simulation-based planning will be the more meaningful the more qualified the data are. Proper design and analysis of measurement data will qualify productivity data (productivity measurement is a research topic of its own) as well as effort estimation. Knowing more about the worst case scenarios does not mean being overly pessimistic. Instead, it means to support the planning and initiation of preventive action. The expected practical benefit of the simulation method is early risk detection and development of mitigation strategies. These benefits were acknowledged by the industrial case study conducted at Chartwell Technology Inc.

Reinforced by the recent financial downturn, project risk analysis has become an even more crucial issue. Before the start of a project, it now becomes worthwhile to check results of all possible worst case scenarios upfront rather than start a project and cancel it later due to lack of mitigation strategies in response to uncontrollable changes in project parameters. The strength of our proposed approach is that it not only helps addressing the concerns mentioned above. Instead, it also allows managers to analyze the impacts of multiple sources of uncertainty acting simultaneously, and it indicates which areas are the largest contributors to potential deviations from baseline plans. The practicability of the proposed method has been demonstrated by an industrial case study.

Future research will be devoted to extending the method to other response factors (i.e., outcome criteria) than make-span, e.g., stability of schedule structure and fluctuation of resource utilization. From the modeling perspective, varying productivity of a developer over time as a result of learning could be a good addition. The idea for variation of task dependencies is promising as well for being taken into account.

Another direction of future research is to integrate other developer-to-task assignment heuristics to replace the current policy. As mentioned in Section 1, another sub-project with Chartwell is improving Chartwell's staffing policy. A potentially better policy (after improvement) in terms of resource utilization further needs a robustness check. For this evaluation, we plan to adapt the evaluation framework suggested and demonstrated by Kitchenham et al. in [34,35].

Acknowledgements

Part of the work presented was financially supported by the Informatics Circle of Research Excellence (iCORE) of Alberta in Canada, and the Natural Sciences and Engineering Research Council (NSERC) of Canada under Discovery Grant No. 327665-06 as well as Discovery Grant No. 250343-07. Ahmed Al-Emran would like to thank NSERC, iCORE, and the Killam Trust for their postgraduate/pre-doctoral scholarships and to Imagine That Inc. for supporting implementation of the research prototype. Chartwell's Jenny Ye and the late Andrew Smith provided valuable contributions for the acquisition of the simulation data and portions of Section 5. Thanks to Jim McElroy and Kornelia Streb for their efforts and comments on the paper.

References

[1] H. Ziv, D.J. Richardson, R. Klösch, The Uncertainty Principle in Software Engineering, Technical Report UCI-TR-96-33, University of California, Irvine, 1996.
[2] G. Stark, A. Skillicorn, R. Ameele, An examination of the effects of requirements changes on software maintenance releases, J. Softw. Maint. Res. Pract. 11 (1999) 293–309.
[3] A. Bolch, Murphy's Law, 26th Anniversary ed., The Berkley Publishing Group, New York, 2003.
[4] P. Kapur, A. Ngo-The, G. Ruhe, A. Smith, Optimized staffing for product releases and its application at Chartwell technology, J. Softw. Maint. Evol. 20 (2008) 365–386.
[5] M.I. Kellner, R.J. Madachy, D.M. Raffo, Software process simulation modeling: Why? What? How?, J Syst. Softw. 46 (2/3) (1999) 91–105.
[6] F. Padberg, Using process simulation to compare scheduling strategies for software projects, in: Proceeding of the 9th Asia-Pacific Software Engineering Conference, 2002, pp. 581–592.
[7] T.M. Gruschke, M. Jørgensen, The role of outcome feedback in improving the uncertainty assessment of software development effort estimates, ACM Trans. Softw. Eng. Methodol. 17 (4) (2007).
[8] J.M. Verner, W.M. Evanco, N. Cerpa, State of the practice: an exploratory analysis of schedule estimation and software project success prediction, Inform. Softw. Technol. 49 (2007) 181–193.
[9] E. Turban, J.E. Aronson, T.P. Liang, Decision Support Systems and Intelligent Systems, Pearson Prentice Hall, 2005.
[10] A. Al-Emran, P. Kapur, D. Pfahl, G. Ruhe, Simulating worst case scenarios and analyzing their combined effect in operational release planning, in: Q. Wang, D. Pfahl, D.M. Raffo (Eds.), ICSP 2008, LNCS 5007, Springer, Berlin, 2008. 269–281.
[11] D. Pfahl, ProSim/RA – software process simulation in support of risk assessment, in: S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, P. Grünbacher (Eds.), Value-based Software Engineering, Springer, Berlin, 2005, pp. 263–286.
[12] A. Al-Emran, D. Pfahl, G. Ruhe, A method for re-planning of software releases using discrete-event simulation, Softw. Process Improv. Pract. 13 (2007) 19–33.
[13] G. Antoniol, M. Di Penta, M. Harman, Search-based techniques applied to optimization of project planning for a massive maintenance project, in: Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005, pp. 240–249.
[14] C. Chang, M. Christensen, T. Zhang, Genetic algorithms for project management, Ann. Softw. Eng. 11 (2001) 107–139.
[15] J. Duggan, J. Byrne, G.J. Lyons, A task allocation optimizer for software construction, IEEE Softw. 21 (3) (2004) 76–82.
[16] A. Ngo-The, G. Ruhe, Optimized resource allocation for software release planning, IEEE Trans. Softw. Eng. 35 (2009) 109–123.
[17] H. Zhang, B. Kitchenham, D. Pfahl, Software process simulation modeling: facts, trends and directions, in: Proceedings of the 15th Asia-Pacific Software Engineering Conference, 2008, pp. 59–66.
[18] B. Yang, H. Hu, L. Jia, A study of uncertainty in software cost and its impact on optimal software release time, IEEE Trans. Softw. Eng. 34 (6) (2008) 813–825.
[19] G. Antoniol, M. Di Penta, M. Harman, A robust search–based approach to project management in the presence of abandonment, rework, error and uncertainty, in: Proceedings of the 10th International Software Metrics Symposium, 2004, pp. 172–183.
[20] F. Padberg, Computing optimal scheduling policies for software projects, in: Proceedings of the 11th Asia-Pacific Software Engineering Conference, 2004, pp. 300–308.
[21] F. Padberg, On the potential of process simulation in software project schedule optimization, in: Proceedings of the 29th Annual International Computer Software and Applications Conference, 2005, pp. 127–130.
[22] D. Pfahl, A. Al-Emran, G. Ruhe, A system dynamics model for analyzing the stability of software release plans, Softw. Process Improv. Pract. 12 (2007) 475–490.
[23] M. Shepperd, C. Schofield, Estimating software project effort using analogies, IEEE Trans. Softw. Eng. 23 (1997) 736–743.
[24] B.W. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D.J. Reifer, B. Steece, Software cost estimation with COCOMO II, Prentice Hall, New Jersey, 2000.
[25] S. Acuña, N. Juristo, A.M. Moreno, Emphasizing human capabilities in software development, IEEE Softw. 23 (2006) 94–101.
[26] D. Anseimo, H. Donald, Measuring productivity in the software industry, Commun. ACM 46 (2003) 121–125.
[27] D. Johnson, The triangular distribution as a proxy for the beta distribution in risk analysis, The Statistician 46 (1997) 387–398.
[28] A. Al-Emran, K. Khosrovian, D. Pfahl, G. Ruhe, Simulation-based uncertainty analysis for planning parameters in operational product

management, in: Proceedings of the 10th International Conference on Integrated Design and Process Technology, Antalya, Turkey, 2007, pp. 191–201.
[29] R. Kolisch, S. Hartmann, Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis, in: J. Weglarz (Ed.), Project Scheduling: Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Berlin, 1999, pp. 147–178.
[30] R. Kolisch, S. Hartmann, Experimental investigation of heuristics for resource-constrained project scheduling, Eur. J. Oper. Res. 174 (2006) 23–37.
[31] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Massachusetts, 2000.
[32] The Standish Group, CHAOS Summary 2009, Boston, MA, April 23, 2009.
[33] The Standish Group, Solutions for Enterprise Project and Portfolio Management, Boston, MA, April 23, 2009.
[34] B.A. Kitchenham, L. Pickard, S. Linkman, P. Jones, A framework for evaluating a software bidding model, Inform. Softw. Technol. 47 (2005) 747–760.
[35] B.A. Kitchenham, L. Pickard, S. Linkman, Experiences of using an evaluation framework, Inform. Softw. Technol. 47 (2005) 761–774.