# The Next Release Problem Revisited: A New Avenue for Goal Models

Fatma Başak Aydemir, Fabiano Dalpiaz, Sjaak Brinkkemper
Utrecht University
Utrecht, The Netherlands

Paolo Giorgini
University of Trento
Trento, Italy

John Mylopoulos
University of Ottawa
Ottawa, Canada

***Abstract*—*Context.* Goal models have long been critiqued for the time it takes to construct them as well as for their limited cognitive and visual scalability. Is such criticism general or does it depend on the supported task? *Objectives.* We advocate for the latter and the aim of this paper is to demonstrate that the next release problem is a suitable application domain for goal models. This hypothesis stems from the fact that product release management is a long-term investment, and that software products are commonly managed in "themes", smaller focus areas of the product. *Methods.* We employ a version of goal models that is tailored for the next release problem by capturing requirements, synergies among them, constraints, and release objectives. Such goal model allows discovering optimal solutions considering multiple criteria for the next release. *Results.* A retrospective case study confirms that goal models are easier to read and comprehend when organized in themes, and that the reasoning results help product managers decide for the next release. Our scalability experiments show that, through reasoning based on optimization modulo theories, the discovery of the optimal solution is fast and scales sufficiently well with respect to model size, connectivity, and number of alternative solutions.**

***Index Terms*—next release problem, release planning, goal–oriented requirements engineering, constrained goal models, multi–objective optimization, optimization modulo theories**

## I. Introduction

Goal–oriented requirements engineering (GORE) has been extensively studied by the research community since the introduction of early goal–oriented frameworks [1]. Goal models have been used in early requirements engineering [2], agent–oriented software engineering [3], and to analyze software qualities like security [4], privacy [5], risk [6], and trust [7].

GORE has also been criticized due to various reasons. In a recent study, Mavin *et al.* [8] show evidence that GORE approaches have been applied to too few real-life case studies and the industry has been reluctant to adopt GORE. Goal models are perceived as unscalable both in terms of *i.* the effort required to build them and *ii.* visually. To overcome the latter obstacle, Moody *et al.* [9] emphasize modularity as a way to improve visual notations for GORE frameworks.

We identify *software release planning* as a domain where modularity is well practiced. The evolution of software products is managed through stepwise releases so the development efforts are compartmentalized by time. Agile development approaches are modular too, thanks to the adoption of sprints. Furthermore, many companies use *themes* to categorize the focus areas of a (large) software product [10].

The next release for a product is determined by gathering candidate new requirements over a time period and then selecting which of these are going to be implemented, taking into account logical constraints (such as mutual exclusion and precedence), as well as business considerations such as minimize costs, maximize customer value, and the likes. Following the literature [11], we refer to this as the *Next Release Problem* (NRP) for a software product.

The NRP has received considerable attention so far. It has been formalized as a single-objective [11] or a multi-objective optimization problem [12], and genetic algorithms have been applied to solve the problem [13]. These works treat NRP as a *purely numerical problem*. However, many of the business and software qualities for NRP have a qualitative nature and are hard to quantify. Software cost, for example, is notoriously hard to estimate and even approximate [14]. Customer satisfaction, as a competing quality, is even harder to boil down to numerical scores. This calls for approaches that support both numerical/quantitative and qualitative reasoning.

Moreover, existing approaches to NRP represent requirements as flat collections of functions that do not take into account the hierarchical nature of requirements. Thus, they lose valuable information about the requirements inter-relationships such as synergies and conflicts, alternatives, and their rationale too: *why* is a requirement important in the product company's strategy and in the long-term product release plan?

Based on these observations, we *hypothesize* that the NRP may be a suitable domain for the goal–oriented approaches because *i.* goal models can provide the missing expressiveness to the release model and help release managers better understand the synergies among the requirements and *ii.* in this context, goal models will experience less scalability problems thanks to the inherent modular nature of release planning.

In this paper, we frame the NRP in terms of *constrained goal models* [15] where requirements are hierarchically structured and inter-dependent (conflicting/synergistic). The space of alternatives is then explored to discover Pareto-optimal solutions by using combined qualitative and quantitative reasoning techniques founded on automated reasoning technologies, notably Satisfiability and Optimization Modulo Theories (SMT/OMT).

The main contributions of this paper are as follows:

- An expressive goal–oriented language for representing the NRP that supports complex relationships between requirements such as hierarchy, synergy, conflicts (Section III);
- A collection of optimization schemes for expressing objective functions from the literature and from practice that combine qualitative reasoning with quantitative optimization (Section IV);
- The Next Release Tool prototype that supports graphical modeling and—through an encoding of our framework (Section III-D)—reasoning with NRPs by employing a state-of-the-art automated OMT reasoner (Section V-B);
- Experimental results that show the applicability of our approach on a retrospective case study (Section V-A), and the scalability of our reasoning techniques up to and beyond the size of real-world problems (Section V-C).

After introducing our research baseline in Section II, we describe our contributions in Sections III–V, discuss related work in Section VI, and present conclusions and future directions in Section VII.

## II. Research Baseline

### A. Constrained Goal Models

Constrained goal models (CGMs) extend the notion of goal models by *i.* assigning rewards or penalties to goals and their refinements, *ii.* defining constraints, and *iii.* setting optimization objectives for possible solutions [15]. The models are formalized into optimization modulo theories clauses, then an external solver (OptiMathSAT [16]) is used to efficiently search the search space of alternative solutions and discover the optimal ones. Possible applications include assigning costs to tasks, which are the leaf elements, rewards to top goals and set constraints for the budget and aim at discovering the solution that maximizes the reward while minimizing the cost. Aydemir *et al.* [17] employ CGMs to capture and analyse risk in early requirements engineering while Angelopoulos *et al.* [18] utilize them to find the optimal next adaptation for a system. CGMs are also used to facilitate decision making for security engineering [19].

In this paper, we extend CGMs to capture and solve the next release problem.

### B. Requirements for Capturing the Next Release Problem

Bagnall *et al.* [11] model requirements for the next release problem as acyclic graphs where nodes represent requirements and edges denote that the source requirement is a prerequisite of the target one. Toward a more structured approach to modeling requirements for the next release problem, based on a survey with industrial practitioners, Carlshamre *et al.* [20] identify six inter-dependency types for requirements considered for the next releases of a software product (see Table I) and represent these as a matrix using spreadsheets as well as a graph where the nodes represent requirements and labeled edges represent inter-dependencies. We take capturing the inter-dependencies presented in Table I as a requirement for any notation that aims to represent the next release problem. As presented in Sec. III, our proposal accommodates all six of these interdependencies.

Table I: Inter-dependency types between requirements for the next release identified by Carlshamre *et al.* [20]

| Inter-dependency | Meaning |
|---|---|
| REQUIRES | $R_1$ requires $R_2$ to function |
| AND | $R_1$ requires $R_2$ and vice versa |
| TEMPORAL | $R_1$ needs to be implemented before $R_2$ |
| CVALUE | $R_1$ positively or negatively contributes to the customer value of $R_2$ |
| ICOST | $R_1$ positively or negatively contributes to the cost of $R_2$ |
| OR | Only one of $\{R_1, R_2\}$ has to be implemented |

## III. Goal–Oriented Formulation of the NRP

We describe our goal–oriented approach for the NRP. Its key feature is that our goal models capture the rationale behind the requirements for the next release and allow representing complex interdependencies between the requirements.

First, we describe the overall process in Section III-A; second, we detail our proposed goal–oriented requirements modeling language in Section III-B; finally, we define the goal–oriented version of the NRP in Section III-C.

### A. Process Overview

Understanding, capturing, and communicating requirements well has a great impact on the success of software development processes [21]. The same applies to the development of the future releases of a software product. Nevertheless, (goal–oriented) systematic requirements engineering methods in practice are hampered by the high effort that is required, or by the lack of scalability [8].

Our proposed goal–oriented approach to the NRP, which is summarized in Figure 1, overcomes the existing limitations by applying goal orientation to a task having a lengthy time frame; the initial investment is amortized by the fact that a product life time spans over multiple years.

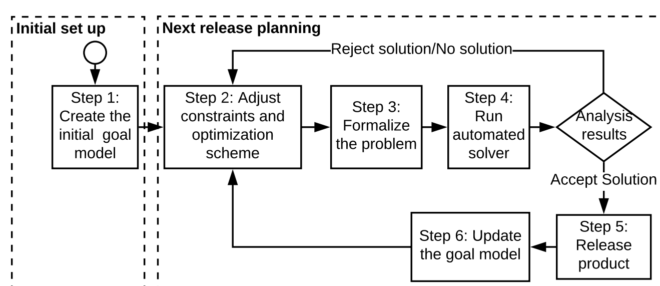



Figure 1: Our goal–oriented approach for handling NRP

*1) Create the initial goal model:* Release management is an iterative process which starts with the initial requirements. This is the step that requires the up-front effort to build the goal model. From this point on, the effort required to update and maintain the model is more limited. Furthermore, our extended notion of CGM overcomes the lack of expressiveness of the flat list representation of requirements, such as spreadsheets. In goal models, high-level, strategic goals are refined into lower-level ones to represent how they can be achieved. The refinement structure in goal models also explains the *raison d'être* of some requirements. Different from the refinement structure, inter-dependencies between goals represent the synergistic relations, capturing how having certain goals together affects positively or negatively certain properties, such as the implementation effort. The inter-dependencies among requirements of Table I are captured by our approach through cost and customer value inter-dependencies among goals such that having goal $G_1$ increases (or decreases) the cost (or the customer value) of goal $G_2$.

*2) Adjust constraints and optimization scheme:* It is important to express constraints for identifying possible and optimal solutions. Hard constraints may be set such as 'the overall cost of the solution should not exceed 500 developer-hours'. Moreover, an optimization scheme for the problem should be defined by using the properties such as minimizing the overall cost, maximizing the overall customer value, or more elaborate combinations of several objectives. We encode common optimization schemes from the literature in Section IV.

*3) Formalize the problem:* To benefit from SMT/OMT reasoning and identify optimal solutions, the goal model and the optimization schemes must be formalized. The formalization–whose basics are explained in Section III-D—is derived automatically from the specifications of the analyst: the goal model, the constraints, and the optimization scheme.

*4) Run automated solver and analyze results:* The formalized problem description is fed into an external SMT/OMT solver. The solver returns the optimal solutions (if any) that respect the constraints and optimize the optimization scheme. There may be multiple optimal solutions, i.e., Pareto-optimal solutions for a given problem description. In other cases, the constraints may be too restrictive and lead to no solution at all. In such cases the release engineer should relax some of the constraints and re-run the solver.

*5) Release product and update the goal model:* After the release of the product (this well-known process falls outside the scope of this paper), the release engineer updates the goal model by adding new requirements and their interdependencies, marking the already implemented requirements, adjusting the properties of goals such as cost and customer value. After the update, the process continues as before from Step 2, by adjusting the constraints and the objectives for the next release.

### *B. A Goal–Oriented Language for the Next Release Problem*

We introduce our goal–oriented modeling language to capture the requirements and inter-dependencies for the NRP. The meta-model of the language is presented in Figure 2.

We deliberately keep the modeling language simple to focus our attention to the selection of requirements, which are represented as goals. As a result, some prominent concepts that are captured in other goal–oriented requirements modeling languages like iStar 2.0 [22] are not included in our meta-model. For simplicity, we exclude *tasks*; as the top-level goals are refined, their semantics become more concrete and leaf

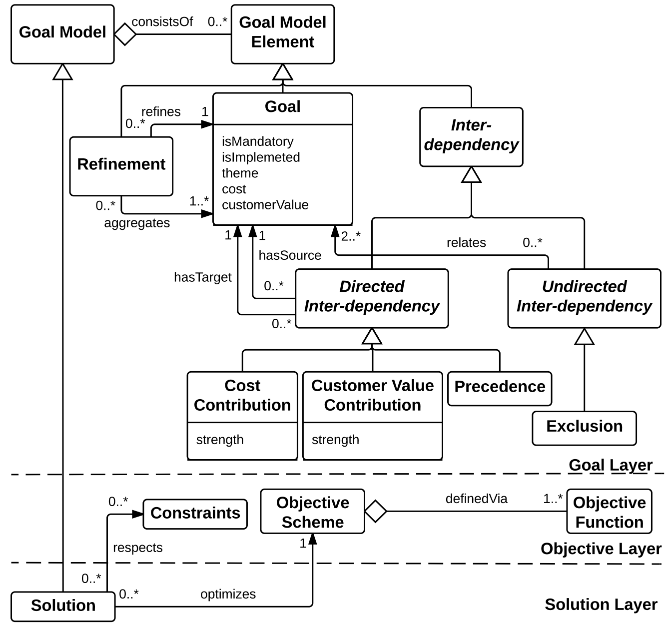


Figure 2: Meta-model for the goal–oriented NRP language

nodes represent immediately actionable items. Since the next release models focus on the product requirements, we also do not capture *actors* and their interactions (*social dependencies*). *Resources*, on the other hand, can be captured as constraints such as the budget for the release but they are not first-class citizens in the models. Below we provide explanations for the elements of the meta-model presented in Figure 2.

A *goal model* consists of *goal model elements*, which are *goals*, *refinements*, and *inter-dependencies*.

**Goal:** Requirements are represented as goals to be achieved. Goals have attributes. A *mandatory* goal must be satisfied in the solution set of a goal model. Goals that have already been implemented in previous releases are marked as *isImplemented*. The *theme* of a goal is a category to which it belongs[1]. Goals can have associated *costs* which can be stated in terms of developer effort–hours or money. Finally, the appreciation of customers are captured as the *customerValue* of a goal.

Assigning values to goal attributes is optional yet useful for determining the optimal solutions. While solutions that do not include all mandatory goals are not valid, *isImplemented* enables the release engineer to keep track of the implemented goals. If the release engineer determines that all previous implementation efforts should be preserved, she could mark them all as mandatory, else she can mark only some implemented goals. Cost and customer value assignments are also used to set constraints (e.g., overall customer value of a valid solution must be higher than $x$) on the solution as well as defining the *optimization scheme* (e.g., minimize the overall cost). Assigned values can be absolute or relative estimations, as long as the units are used consistently in the entire goal model.

[1]Themes are used to organize the requirements for a large system into smaller, more manageable categories.

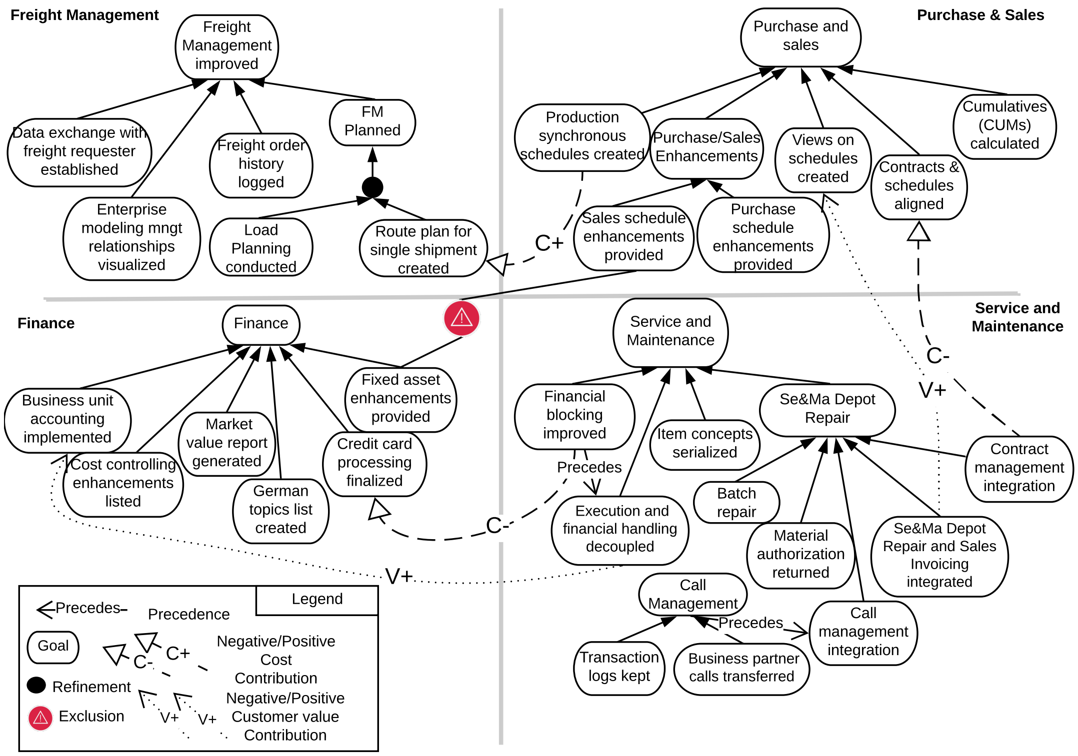



Figure 3: A partial goal–oriented NRP model from our retrospective case study

**Refinement:** A refinement relates a set of child goals to a parent goal, indicating that all the child goals are necessary to fulfill the parent. A parent goal may have multiple refinements, each of which offers an alternative way of achieving that goal. Using refinements, we can capture both REQUIRES, AND, and OR inter-dependencies from Table I. $R_1$ REQUIRES $R_2$ is captured when $R_1$ is refined into $R_2$. $R_1$ AND $R_2$ is captured when both of them are connected to a single refinement node of a parent node. $R_1$ OR $R_2$ is presented when both of them are connected to seperate refinement nodes of the same parent. Figure 4 summarizes these three cases.

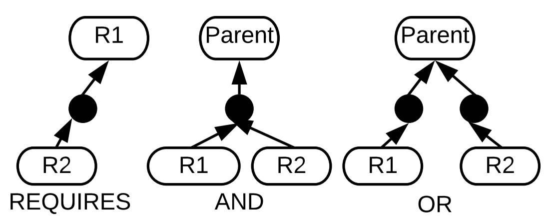



Figure 4: Capturing interdependencies with refinement nodes

Figure 3 shows a fragment of a goal model that is built by the authors from the release data of a software product for managing the rental and service of heavy machinery. The goal model is validated by the product management expert of the software product. Since the company organized the requirements in themes, we reflected that choice in the layout of the model and separated the themes by gray lines. Goal nodes are represented as stadium-shaped nodes. Refinement nodes that link multiple child goals to a parent goal are shown as black-filled circles. Refinement nodes that link only one child node to a parent goal are omitted for visual simplicity. An example of alternative ways of achieving a parent goal concerns the goal '*FM Planned*', which is refined into '*Load planning conducted*' and '*Route plan for single shipment created*'.

**Inter-dependency**: An *inter-dependency* indicates a synergistic relation among two or more goals. An *inter-dependency* is either a *directed inter-dependency* or an *undirected-interdependency*.

**Undirected Inter-dependency**: An *undirected inter-dependency* relates two or more goals. *Exclusion* is an undirected inter-dependency which relates goals that should not be included in the same release. In Figure 3, '*Sales schedule enhancements provided*' and '*Fixed asset enhancements provided*' are related via the exclusion interdependency, which is represented by a white exclamation mark in a triangle that is surrounded by a red-filled circle.

**Directed Inter-dependency**: A directed inter-dependency relates two goals where one goal is the source and the other is the target of the interdependency. There are three types of such links: (i) cost contribution, (ii) customer value contribution, and (iii) precedence. These three inter-dependencies correspond to the ICOST, CVALUE, and TEMPORAL inter-dependencies in Table I. Cost and customer value contribution between goals state that the source goal increases or decreases the cost or the customer value of the target goal. If known, the strength attribute can be assigned to represent the strength of the interdependency in a relative scale or as the absolute value. The precedence inter-dependency indicates a temporal order of implementation between the source and the target goals, where the source goal should be implemented before the target (in the same or a previous release).

In our graphical modeling language, we simply use edges between the source and the target goals instead of having a directed inter-dependency node and connect it with the source and the target goals to prevent visual clutter. Cost contribution inter-dependencies use a $C$ label and the plus or minus sign signifies the positive or the negative impact.

The source and the target goals are linked via a dashed line, and there is an incoming empty triangle arrowhead to the target goal. In Figure 3, '*Financial blocking improved*' decreases the cost of '*Credit card processing finalized*'. Similarly, a dotted line, which is labeled as $V$ together with a plus or minus sign, denotes a customer value inter-dependency.

To further differentiate the two different types of inter-dependencies we use angle arrowhead for the customer value. An example of customer value interdependency from Figure 3 is in between '*Se&Ma Depot Repair and Sales invoicing integrated*' and '*Views on schedules created*' where the former increases the customer value of the latter. Finally, the precedence inter-dependency is shown as a solid line with a simple arrowhead and labeled as precedes. In the model presented in Figure 3, '*Financial blocking improved*' must precede '*Execution and financial handling decoupled*'.

### *C. Goal–Oriented Definition of the NRP*

The meta-model in Figure 2 illustrates the components of the goal–oriented definition of the NRP. In particular, the figure includes three layers of concepts separated by a dashed line.

The upper layer concerns the problem space of the next release problem, that is, the goal model that captures the requirements as goals, and the synergistic relations between them as inter-dependencies. This layer was explained in Section III-B.

The intermediate layer details the other components of the problem statement: the *constraints* and the *optimization scheme*. We provide some examples here, while we discuss how to define typical constraints and optimization schemes in Section IV.

Constraints indicate the limitations set by the release engineer on the valid solutions. Some examples: '*The overall customer value of the solution should be at least $x$*', '*The number of positive cost contributions in the solution should be at most $y$*', '*All goals of theme $z$ must be included in the release*'.

An optimization scheme consists of one or more objective functions. Minimizing the cost is a single objective, where the cost of the solution is the sum of costs of the goals that are included in the solution [11]. Other single objectives could be maximizing the total customer value [23], maximizing the positive customer value inter-dependencies, maximizing the negative cost inter-dependencies, and so on. SMT/OMT reasoning allow us to combine multiple objectives in linear objective functions. Multiple objective functions then can be ordered and lexicographic optimization can be performed.

**The goal–oriented next release problem.** Given a goal model, the goal–oriented NRP concerns finding a sub-model by assigning truth values to leaf goals and propagating the truth assignment to upper level goals (a solution, see the bottom layer of Figure 2) that respects the constraints and optimizes the objective scheme for the model.

### *D. Encoding to SMT*

The encoding that we need consists of obtaining an SMT(LRA) formula $\Psi_M$ of a goal–oriented NRP model $M$ that can be analyzed to derive (optimal) solutions. Formally, we have that $\Psi_M = \Psi \wedge \Psi_{\mathcal{R}} \wedge \Psi_{\mathcal{I}} \wedge \Psi_{\mathcal{W}}$, i.e., the encoding is a conjunction of the constraints in $M$, of the refinements, of the interdependencies, and of the value functions.

**Constraints.** The encoding $\Psi$ includes constraints over elements in $\mathcal{B}$ and $\mathcal{W}$. When a *mandatory goal* $G_i$ is set to be *satisfied* by the release engineer, it is encoded as $(G_i := \top)$ and AND-ed to $\Psi$. Elaborate LRA constraints can be put on solutions such as total cost $\leq x$, where calculation of the total cost of a release candidate is provided in Section IV-A.

**Refinements.** The encoding $\Psi_{\mathcal{R}}$ conveys the fact that every refinement defines a set of sub-goals that together fulfill a parent goal, and that multiple refinements of a same goal introduce alternative ways of satisfying the goal.

Each refinement $R \in \mathcal{R}$ where $\{G_1, \ldots, G_n\} \xrightarrow{R} G_p$ adds the proposition $(\bigwedge_{i=1}^{n} G_i \leftrightarrow R) \wedge (R \rightarrow G_p)$ as a conjunction to the encoding $\Psi_{\mathcal{R}}$.

For all refinements $\{R_1, \ldots, R_n\}$ of a goal $G_p \in \mathcal{G}$, $G_p \rightarrow \bigvee_{i=1}^{n} R_i$ is AND-ed to $\Psi_{\mathcal{R}}$.

**Inter-dependencies.** The encoding $\Psi_{\mathcal{I}}$ consists of co-existence constraints among the goals in a solution.

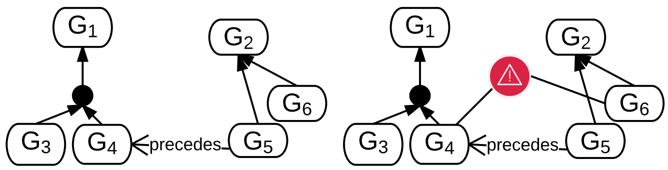
*Exclusion* inter-dependency states that related goals $G_1, \ldots, G_n$ cannot be satisfied all together, and it is encoded as $\neg(\bigwedge_{i=1}^{n} G_i)$ that is AND-ed to $\Psi_{\mathcal{I}}$.

*Precedence* inter-dependency denotes that the implementation of the source precedes the implementation of the target node. The valid cases are that (a) both are included in the solution, (b) both are excluded from the solution, and (c) the source is included but the target is excluded. Given $G_1$ precedes $G_2$, the encoding $(G_2 \rightarrow G_1)$ is added to $\Psi_{\mathcal{I}}$.

*Example.* Figure 5a presents a simple model containing a precedence inter-dependency between $G_5$ and $G_4$, thus non-valid solutions are those that include $G_4$ but not $G_5$. In Fig 5a, assuming that $G_1$ and $G_2$ are mandatory, two solutions are $\{G_3, G_4, G_1, G_5, G_2\}$, $\{G_3, G_4, G_1, G_5, G_6, G_2\}$. Assuming the objective is to minimize cost and each leaf goal has equal costs, the first solution is returned by the solver. On the other hand, the only solution in Figure 5b is $\{G_3, G_4, G_1, G_5, G_2\}$ for $G_6$ cannot be in the same solution as $G_4$, and $G_4$ is required to satisfy $G_1$, which is mandatory.

**Cost and customer value contribution.** These inter-dependencies are also atomic propositions and are encoded as $\Psi_{\mathcal{W}}$. Given that $G_1$ increases the customer value of $G_2$,

1) atomic proposition $I_1$ captures the customer value inter-dependency between $G_1$ and $G_2$. The truth value of $I_1$ is encoded as $(I_1 \leftrightarrow (G_1 \wedge G_2))$,




(a) (b)

Figure 5: Sample models demonstrating precedence relationship

2) the strength of the inter-dependency, $W_{IS}(I_1) = Q$ where $Q \in \mathbb{Q}$. If the release engineer does not specify the strength of the inter-dependency, we assign the strength to zero by default. For the cases in which the release engineer opts for qualitative reasoning by not assigning strength values to cost and customer value inter-dependencies, we keep track of the number of inter-dependencies that are included in the solution (assigned $\top$ by the solver), and use these numbers for optimization.

### *E. (Optimal) Release Candidate*

Based on the definitions and encoding explained above, we can now define a release candidate and an optimal release candidate for a given NR model $M$.

*Definition 1 (Release candidate):* Let $M \stackrel{def}{=} \langle \mathcal{B}, \mathcal{N}, D, \Psi, \mathcal{W} \rangle$ be a next release model, $\Psi_M$ its SMT(LRA) encoding and $\mu$ a LRA-interpretation over $\mathcal{B} \cup \mathcal{N}$ (i.e., a truth assignment over the variables). $\mu$ is a release candidate of $M$ if and only if $\mu \models \Psi_M$.

*Definition 2 (Optimal release candidate):* Let $M \stackrel{def}{=} \langle \mathcal{B}, \mathcal{N}, D, \Psi, \mathcal{W} \rangle$ be a next release model, $\nu$ be a release candidate of $M$, and $O$ an objective schema consisted of LRA–terms occurring in $\Psi_M$. $\nu$ is said to be an optimal release candidate of $M$ iff $\nu$ minimizes [resp. maximizes] $O$, i.e., iff $\nu$ is a solution of the OMT(LRA) minimization [resp. maximization] problem $\langle \Psi_M, \langle O \rangle \rangle$.

In this paper, we will use the term *solution* to indicate an optimal release candidate, for such term better emphasizes the identification of an answer to the NRP.

## IV. Setting constraints and optimization objectives

We show how our framework can be flexibly used to define realistic optimization schemes for the identification of optimal solutions. Our presented schemes are assembled from the literature (Section IV-A to Section IV-C), and also include a customization mechanism to define specific schemes that fit the needs of a company (Section IV-D). Finally, we provide the steps to plan multiple releases in Section IV-E

### *A. Multi-objective quantitative optimization*

This scheme combines multiple objectives, each calculated using attributes of the goals (such as cost and customer value), without considering the inter-dependencies. This corresponds to the multi-objective next release problem formulation [12].

In our goal–oriented formulation, the total cost (or customer value) of a solution is the sum of costs (or customer values) of goals that are included in the solution. Equation 1 returns the total cost (TC) of a release candidate $\mu$ of a next release model $M$, where $ite$ is an *if–then–else* function that returns the cost of a goal $G_i$ of $M$, if $G_i$ is included in $\mu$, 0 otherwise[2].

$$\text{TC}(\mu) = \sum_{G_i \in \mathcal{G}} ite(G_i \text{ in } \mu, W_{GC}(G_i), 0) \tag{1}$$

One optimization scheme that considers both cost and customer value as objectives is the lexicographic optimization, where the objectives are strictly ordered, and the solution is first optimized for the first objective and then for the second one, and so on. In this case, the first objective has strictly a higher priority for optimization than the second one.

Another way of combining multiple objectives is to create an objective function that includes all objectives such as

$$\alpha \times \text{TV}(\mu) - \beta \times \text{TC}(\mu) \text{ where } \alpha + \beta = 1 \tag{2}$$

TV: total customer value

TC: total cost

To get an accurate result for Equation 2, total cost and customer value should be on the same scale. Our framework allows release engineers to assign absolute or relative values (such as story points) to these attributes when building the goal model, so for combining multiple objectives, release engineers must normalize these values.

We only discuss linear normalization due to limited space. Algorithm 1 takes a next release model $M$, a set of objectives $\{obj_i, obj_j, \ldots, obj_k\}$ that are derived from attributes of goals to linearly combine in a normalized objective function, and corresponding co-efficients $\alpha, \beta, \ldots$ for each objective as input. The OMT reasoner is called to calculate the maximal and the minimal objective values for the optimal releases where a single objective is set for the model $M$ (Lines 2-9). TO is the generic representation of calculating a total value of an objective, as in total cost presented in Equation 2. Once these values are returned by the reasoner, the normalized objective function is created using the co-efficients and these values. Overall the reasoner should be called $(2 \times k) + 1$ where $k$ is the number of objectives to be combined. OptiMathSAT provides a boxed optimization option that returns the maximal and minimal values of the same objective in a single call; this enables reducing the total number of reasoner calls to $(k + 1)$.

### *B. Qualitative reasoning with inter-dependencies*

Inter-dependencies provide useful information about the synergistic relations among goals. According to the encoding in Section III-D, a cost or customer value inter-dependency is included in a solution if and only if both source and target goals are included. The number of inter-dependencies (NI) in a release candidate $\mu$ is calculated as

$$\text{NI}(\mu) = \sum_{I_i \in \mathcal{I}} ite(I_i \text{ in } \mu, 1, 0) \tag{3}$$

[2]If the attribute values are not specified, the default value is returned.

**Data:** Next Release Model M;
$\{obj_i, obj_;, \dots, obj_l\}$;
Co-efficients $\alpha, \beta, \dots$;
**foreach** *objective* $o_i$ **do**
  $\nu_{i_{max}}$ = call-omt-solver($\langle \Psi_M$, maximize $o_i \rangle$);
  $max_{o_i}$ = TO($\nu_{i_{max}}$);
  $\nu_{i_{min}}$ = call-omt-solver($\langle \Psi_M$, minimize $o_i \rangle$);
  $min_{o_i}$ = TO($\nu_{i_{min}}$);
**end**
**return** $\alpha \times \frac{o_i - min_{o_i}}{max_{o_i} - min_{o_i}} \pm \beta \times \frac{o - min_{o_j}}{max_{o_j} - min_{o_j}} \pm \dots$;

**Algorithm 1:** Creating a normalized objective function

Equation 3 can be modified to calculate different types of inter-dependencies such as positive cost contribution, negative cost contribution, etc. Then, NI($\mu$) is treated as any other objective in Algorithm 1 and can be used to create a more expressive objective function, blending quantitative and qualitative reasoning. If the strength attributes of a type of inter-dependency are known, 1 is replaced by the corresponding strength value.

### C. Reasoning about specific themes

Themes can group requirements for the NRP into more manageable categories [24]. This categorization can be used to focus or limit efforts spent on a theme. One case is to include every goal of a theme in the next release, by setting all leaf goals of a theme as $\top$. Conversely, to exclude all goals of a theme in the next release, all leaf goals of that theme are set as $\bot$. Further, to limit cost of a theme, a release engineer can define a new function as theme cost (THC):

$$\text{THC}(\mu, x) = \sum_{G_i \in \mathcal{G}} ite(G_i \text{ in } \mu \wedge G_i.\text{theme} = x, W_{GC}(G_i), 0)$$

and add a constraint such that THC$(\mu, x) \leq y$. The same method can be used to set min-max intervals for customer value for a given theme: for example, 'the customer value for theme $T_1$ should be between 10 and 20'.

### D. Beyond the basics: Allocating team effort for specific themes

We show how to extend the framework to suit specific needs of a company. We take the optimization scheme for our case study (its goal model is in Figure 3): given a set of development teams that are experts in one or more themes and have limited availability, allocate team effort in a way that maximizes the utilization of teams in their expertise as well as overall team utilization. Figure 6 captures the extension to the meta-model. Algorithm 2 presents how to keep track efforts of $n$ teams in terms of implementation of leaf goals. First we introduce a new set of atomic proposition ($\mathcal{P}$) in $\mathcal{B}$ for each team–(leaf) goal couple (Line 4). Line 5 states that a goal is selected to be implemented (i.e., included in the solution) iff it is assigned to be implemented by exactly one team[3].

[3]We provide the naïve encoding of 1 from N due to limited space, Klieber *et al.* [25] provide more efficient encodings.

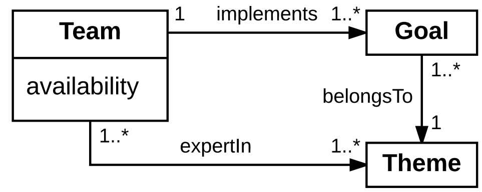



Figure 6: Extension of NRP meta-model with Team and Theme

**Data:** Next Release Model M;
1 **foreach** *Leaf goal* $G_i$ **do**
2   **foreach** *Team* $T_j$ **do**
3     Create new atomic proposition $P_{T_j G_i}$;
4   **end**
5   $(G_i \leftrightarrow ((\bigvee_{k=1}^{n} P_{T_k G_i}) \wedge (\bigwedge_{l=1}^{n-1} \bigwedge_{m=l+1}^{n} (\neg P_{T_l G_i} \vee P_{T_m G_i})))$;
6 **end**

**Algorithm 2:** Keeping track of team efforts on goals

We define expert–effort (EE) of a team $T$ for a release candidate as the sum of implementation costs of goals that are included, implemented by team $T$, and belong to a theme in which $T$ is an expert. Equation 4 formulates the expert effort of a team $T_x$ in a release candidate $\mu$; for simplicity we use the same index value in a team–goal couple $P_i$, and the team $T_i$ and goal $G_i$ in that couple. $P_i$ is set to $\top$ if and only if $T_i$ implements $G_i$ in release candidate $\mu$.

$$\begin{aligned} \text{EE}(\mu, T_x) = \sum_{P_i \in \mathcal{P}} & ite(P_i \text{ in } \mu \wedge T_i = T_x \wedge \\ & T_x.\text{expertIn} == G_i.\text{belongsTo}, W_{GC}(G_i), 0) \end{aligned} \tag{4}$$

Similarly novice–effort (NE) of a team $T$ for a release candidate as the sum of implementation costs of goals that are included in the release candidate, implemented by team $T$, and belongs to a theme in which $T$ is not an expert.

The objective is to minimize novel–effort while fully utilizing the teams (maximizing the total effort where the team availability is the upper bound).

### E. Release planning

The framework can be used for release planning, which is a generalization of the NRP [26]. Once a goal model is built for multiple releases and the reasoner is run for the next release, the model can be re-used to reason on the future releases after following the steps described below.

1) Mark all goals in the solution as *isImplemented* and as *isMandatory* (unless re-implementation is considered).
2) Assign customer value and cost of the goals in the current solution to zero for their implementation cost (customer value) has been spent (used) in the previous release.
3) Assign constraints for the next release such as new maximal cost and mandatory goals.
4) Define the optimization scheme.
5) Re-run the reasoner.

## V. Evaluation and Tool Support

We present the lessons learned from our retrospective case study in Section V-A. Then, we describe the prototype that supports our approach in Section V-B. Finally, we report on scalability experiments that show the ability to efficiently reason on large models in Section V-C.

### A. Lessons learned from an expert evaluation

We conducted an evaluation with an experienced software product manager (over 20 years of experience with large-scale software products) to test the following hypotheses:

**H0.** Executing the process described in Figure 1 is feasible

**H1.** Our goal–oriented approach helps the release manager make decisions for the next release

The domain expert provided us with a real data set of an enterprise application developed for managing heavy machinery. The data set included an Excel file including a flat list of 100 requirements in consideration for the release 5.2, and corresponding themes, effort, and revenue estimations.

One author built separate goal models for each of the six themes; subsets of four of these models are presented in Figure 3. Then, the same author merged these small models into a bigger model while keeping the theme-based layout, and identified trivial interdependencies between the requirements that required no domain knowledge (e.g., '*Call management*' precedes '*Call management integration*').

Based on this initial goal model, the same author had a two-hour face-to-face, hands-on meeting with the domain expert. After a short explanation of the notation, the domain expert was able to identify additional inter-dependencies. Then, we ran our automated analysis multiple times with different objective schemes, and showed the results to the expert. We posed triggering questions to the expert in order to identify pros and cons, and to obtain answers for H0 and H1.

One perceived benefit of building the goal model over a flat spreadsheet is the ability to visualize the requirements in a structured fashion. The visualization helped have a better understanding of the requirements and their inter-dependencies, identify decision points (i.e., alternatives), pinpoint requirements that are highly connected to the others via multiple positive customer value and negative cost inter-dependencies (i.e., requirements are desired to be in the solution), and organize the requirements around themes. The expert also used the vertical positioning of goals to represent relative importance. He mentioned that the model fostered deep thinking and helped discover information that is not explicit in the spreadsheet.

The expert found the visual notation easy to understand, and observed that positive customer value inter-dependencies typically occur more frequently than negative ones. When inquired about visual scalability, the expert commented that a unified extra-large view of all requirements is typically not needed for each product manager focuses on a specific subset of the model. He stated that the approach is feasible (**H0**), but it is necessary to put sufficient effort in updating the goal model throughout the product releases. The extra effort pays off if appropriate tooling allows reviewing model snap-shots of past releases so that the decision rationale can be tracked. The model can bring structure to the discussion between product managers about the next release, and the tool can answer what-if analysis questions such as '*what if we set this goal as mandatory?*', or '*what if we adopt another objective scheme?*' and assist their decision making process (**H1**).

The expert suggested adding multiple abstraction capabilities to the prototype to show/hide cross-functionalities, or selected inter-dependencies. Also, integration with existing development tools is crucial, e.g., with the issue tracker JIRA and with the architectural models. In his experience, release planning is typically coupled with the (re-)definition of the product architecture. To improve visualization, he suggested goal node sizing based on goal cost to provide a visual cue. The expert is confident that his comments are generic for software companies that are organized per Conway's Law [27].

### B. Prototype: the Next Release Tool

Our prototype is a standalone application for Windows, Mac OS and Linux that supports modeling and reasoning.

The graphical editor of the prototype is based on the Eclipse Graphical Modeling Project (GMP)[4]. Well-formedness rules are continuously checked to prevent user mistakes when modeling, e.g., a goal cannot be refined into itself so it is not possible to create both incoming and outgoing edges from a goal to a refinement node.

Next release model elements and the optimization scheme stated by the analyst are automatically encoded as SMT/OMT formulas and fed into OptiMathSAT along with the proper commands for the solver to solve the optimization problem.

The retrieved results are presented in a written report, and the solution is highlighted in the graphical model if found. We are still working at the graphical visualization of multiple solutions, which are currently only listed in the report.

### C. Scalability tests

We conducted three experiments to study the scalability of the prototype, and therefore of the encoding of Section III-D. All experiments are run on a Windows 64 bit machine with Intel(R) Core(TM) i7-3770 CPU 3.40Ghz and 8GB of RAM, and collected the reasoning time reported by OptiMathSAT version 3.5 to find the solution.

**Scalability with respect to problem size.** We define the size of the problem as the number of model elements: number of goals, refinements, and inter-dependencies. To evaluate this dimension of scalability, we have created an input model which includes 13 goal nodes, 9 refinements, and four 4 inter-dependencies. Lexicographic optimization scheme is selected for the optimization. We have generated 75 cases by replicating the input model 10 to 750 times and connecting the replicas to a root goal via the same refinement element. As a result, we have generated test models of size from 255 to 17,275 model elements. We have run the experiment five times.

[4]http://www.eclipse.org/modeling/gmp/

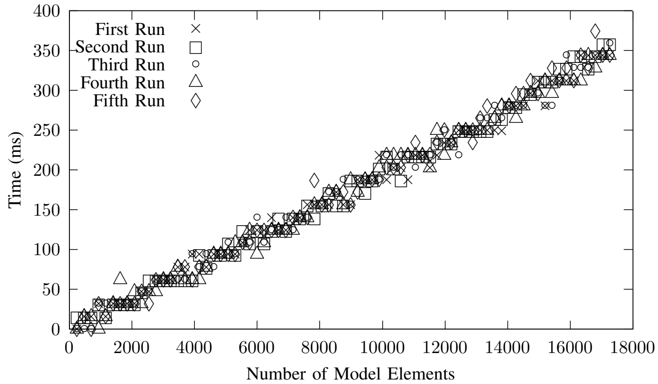



Figure 7: Scalability wrt problem size

To put our experiment in context, one of largest *i** models reported in the literature includes 525 links and 350 elements [7]. Thus, our artificially generated experiment cases are large enough to match industrial case studies. Obviously, creating big models requires time and human effort (see Section V-A).

The results are very positive: the required time to generate the solution grows *linearly* with the model size, and performance is excellent (0.4 seconds for models with 17,275 elements).

**Scalability with respect to problem complexity.** We define problem complexity in terms of the number of inter-dependencies between the model elements. To test this dimension, we generated 400 variants of a fixed-size model (250 goals, 118 refinements, 16 precedence links and 16 exclusion links) with an increasing number of random directed inter-dependencies between goals. Figure 8 presents the results of

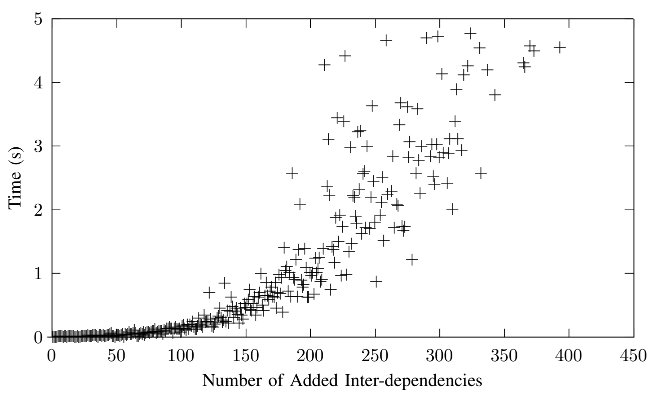



Figure 8: Scalability wrt problem complexity

this experiment. The number of added inter-dependencies are shown on the x-axis whereas the y-axis reports the reasoning time in seconds. There is a smooth increase in reasoning time until approximately 150 added inter-dependencies, the reasoning time increases rapidly after this threshold. Despite the growth, the processing time is still negligible (under 5 seconds) and adequate for a release engineer who is studying the long-term evolution of a software product.

**Scalability with respect to alternatives.** Multiple refinements of a goal introduce alternative solutions for satisfying that goal. Increasing the number of refinements may lead to an exponential increase in the number of solutions, thereby impacting the reasoning time [28]. We created five model variations with fixed number of goals and inter-dependencies, 11,918 and 1,402, respectively. In the first model, all 5,609 refinement elements had two incoming edges, and a parent goal had only one incoming refinement edge. The second model is a variation of the first one, but for the 25% refinement elements, we have created another refinement element pointing to the same parent goal, and directed one of the two incoming edges of the original refinement element to the newly created one, in other words, we have converted 25% of the AND decompositions to OR decompositions. For the rest of the variations, we have increased the conversion rate to 50%, 75%, and finally 100% and run the reasoner for each model.

Table II: Average and standard deviation of reasoning times for the scalability wrt alternatives

| | 0% | 25% | 50% | 75% | 100% |
|---|---|---|---|---|---|
| AVG (s) | 0.2514 | 0.304 | 3.0171 | 3.0532 | 4.8391 |
| STDEV (s) | 0.0045 | 0.008 | 0.0235 | 0.0073 | 0.0169 |

The results are summarized in Table II. The first row reports the average seconds needed to find a solution for 10 runs, and the second row reports the standard deviation. The reasoner finds a solution for the first two models (0%-OR and 25%-OR) in less than one second, for the second two models in around three seconds and for the fifth model in around five seconds. These results reveal that the approach works well even when the number of alternatives increases.

**Discussion.** The overall results confirm the scalability of our tool in all three cases even for models that are way larger than those that can be used in real life. The bigger obstacle to scalability is model connectivity, which is affected by increasing the number of directed inter-dependencies. Even in this case, however, the tool returns the optimal solution for a model of 250 goals and 200 inter-dependencies. The inter-dependency to goal ratio is much higher than the real-life cases reported by Carlshamre *et al.* [20], which is around 20%.

## VI. Related Work

### A. Search–based approaches to NRP

The Next Release Problem is formalized by Bagnall *et al.* [11] as a constrained optimization problem. Following research has adopted quantitative search-based techniques. Jiang *et al.* [29] apply ant-colony optimization whereas Jifeng *et al.* [30] use backbone-based multilevel search. Feather and Menzies [31] propose an iterative approach that takes human-expert decision into consideration. The NRP was formulated as a multi-objective optimization and solved by [12]. Our approach differs in two ways: (i) our goal–oriented representation of the problem shows explicitly the hierarchy of and inter-dependencies among goals; (ii) we apply OMT reasoning to NRP that is proven to be scalable and guarantees an optimal solution. Recently, Pitangueria *et al.* [32] apply SMT reasoning to solve the NRP for a flat list of requirements with excellent scalability results, although their work lacks the benefits of using goal models to structure and interrelate requirements.

The release planning problem generalizes NRP; the goal is to select features for a series of product releases. Du and Ruhe [26] propose machine learning solutions, providing the rationale behind solutions provided by ReleasePlanner®, a commercial tool for release planning. Ngo-The and Ruhe [33] iteratively solve the problem considering also human input, ranking candidate solutions with respect to soft constraints and objectives. Search–based software engineering methods were proposed as well [34]. We focus mostly on the next release problem but we have shown an optimization scheme in Section IV that support release planning. Ameller *et al.* [35] provides a survey on release planning models.

Toward a more structured search space for the NRP, Carlshamre *et al.* [20] identify requirements inter-dependencies and Zhang *et al.* [36] apply multi-objective search based techniques while handling these inter-dependencies. Regnell [37] validates a proposed meta-model through industrial surveys. Those models have AND/OR dependencies among requirements that, however, are less expressive than our hierarchical goal models.

### *B. Goal–Oriented Requirements Engineering*

Goal–oriented requirements engineering has received much attention since the introduction of early goal–oriented frameworks [1]. KAOS [4], NFR[38], and *i**[39] are general-purpose frameworks that have been extended numerous times to handle the overall software development process [3], security [40], [41], [42], [43], privacy [5] and other concerns [44].

Sebastiani *et al.* [45] discover minimum–cost solution in a goal model; similar techniques were later applied to find optimal solutions in goal-risk models [6]. Unfortunately, these approaches can optimize only for a single objective (cost). Multiple reasoning techniques on goal models focus only on finding a solution that satisfies top goals [46], but do not optimize.

CGM-based approaches enable multi-objective optimization in goal models [17], [18], [15]. Nguyen et al. [15] proposes a goal-modelling technique intended for greenfield design. Our proposal is different in that it focuses on next release design and uses an enhanced modelling language that includes an extended set of dependencies over traditional goal models.

Jureta *et al.* support preferences over alternatives and optional goals in the Techne modeling language [47]. SAT/SMT/OMT reasoning techniques over Techne models have not been yet exploited. Liaskos et al. [48] support preferences and utility in goal models and provide a planning based solution. However, AI planning [49] does not scale as well as SMT techniques.

Eliciting and assigning goal cost, customer value, and other numerical values is out of the scope of this paper. However, the literature offers plenty of approaches. The Analytic Hierarchy Process [50] can be used to determine quantitative contribution measures, and it was applied to goal models [51] and to requirements in general [52]. Wiegers [53] details a method for relatively assigning cost, (customer) value, and risk, while Villarroel *et al.* [54] use crowd sourcing to prioritize requirements for release planning. Recently Choetkiertikul *et al.* [55] employed deep learning techniques for effort estimation.

## VII. Discussion and Conclusion

We proposed a new application of GORE to the next release problem. After an initial time investment for building a goal model, such up-front cost can be amortized over time thanks to the re-use of the goal model. The release planning process is carried out with the assistance of efficient automated solvers that automatically identify optimal solutions. Over time, the analyst has to update the model to account for implemented features and additional candidate requirements.

Our models are encoded into SMT/OMT formulas that enable efficient problem resolution via the OptiMathSAT solver. Our optimization schemes show the flexibility of the framework in expressing different objective functions from the literature. The prototype supports the entire process from modeling to automated reasoning and results visualization.

The results obtained from our preliminary evaluation show that *there may be new life for goal models*: (1) despite its simplicity compared to complex framework such as *i** [56], [22] or KAOS [57], the language can supports the different types of inter-dependencies between next-release planning requirements identified by Carlshamre *et al.* [20]; (2) a retrospective case study has evidenced the strengths of our models over flat requirements lists; and (3) the scalability experiments confirmed the applicability of our reasoning to large models.

**Threats to Validity.** *Conclusion validity* is threatened by the use of a single case and the evaluation with a single product manager. Despite the expert's high experience in the software industry, the findings are not conclusive yet. This is why we only claim that there *may* be new life for goal models. The use of a single case does not threaten the conclusions on the scalability of our reasoning, which derive from the scalability results of the employed automated solver. *Internal validity* is threatened by the selection of our case; while representative for the Northern European software industry, the requirements for release planning elsewhere may be less structured in other geographic areas (e.g., our models benefit from the grouping of requirements into themes). *Construct validity* is affected by the choice of a single method to represent the requirements. The expert provided an opinion on how the models compare to a flat representation, but we have not tested other techniques. *External validity*: we conducted a retrospective case, thus, the approach was not tested for a to-be release planning.

**Future work.** We are currently setting up empirical evaluations of the framework within product companies from our network for solving the NRP for future releases. Furthermore, we need to create a simple-to-use language for expressing objective functions and to extend our set of optimization schemes. We also plan to conduct experiments that compare the outcomes of analysts using our framework with the outputs of other approaches from the literature.

## REFERENCES


[1] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, and P. Giorgini, "Goal-oriented requirements engineering: an extended systematic mapping study," *Requirements Engineering*, pp. 1–28, 2017.

[2] A. Fuxman, L. Liu, J. Mylopoulos, M. Roveri, and P. Traverso, "Specifying and analyzing early requirements in tropos," *Requirements Engineering*, vol. 9, no. 2, pp. 132–150, 2004.

[3] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[4] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, vol. 20, no. 1-2, pp. 3–50, 1993.

[5] L. Liu, E. Yu, and J. Mylopoulos, "Security and privacy requirements analysis within a social setting," in *Proceedings of the 11th IEEE International Requirements Engineering Conference*. IEEE, 2003, pp. 151–161.

[6] Y. Asnar, P. Giorgini, and J. Mylopoulos, "Goal-driven risk assessment in requirements engineering," *Requirements Engineering*, vol. 16, no. 2, pp. 101–116, 2010.

[7] J. Horkoff, E. Yu, and L. Liu, "Analyzing trust in technology strategies," in *International Conference on Privacy, Security and Trust*, 2006.

[8] A. Mavin, P. Wilkinson, S. Teufl, H. Femmer, J. Eckhardt, and J. Mund, "Does goal-oriented requirements engineering achieve its goal?" in *Proceedings of the 25th IEEE International Requirements Engineering Conference*. IEEE, 2017, pp. 174–183.

[9] D. L. Moody, P. Heymans, and R. Matulevicius, "Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax," in *Proceedings of the 17th IEEE International Requirements Engineering Conference*, 8 2009, pp. 171–180.

[10] I. van de Weerd, S. Brinkkemper, R. Nieuwenhuis, J. Versendaal, and L. Bijlsma, "Towards a reference framework for software product management," in *Proceedings of the 14th IEEE International Requirements Engineering Conference*. IEEE, 2006, pp. 319–322.

[11] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The next release problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.

[12] Y. Zhang, M. Harman, and S. A. Mansouri, "The multi-objective next release problem," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2007, pp. 1129–1137.

[13] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[14] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 33–53, 2007.

[15] C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Multi-objective reasoning with constrained goal models," *Requirements Engineering*, 2016.

[16] R. Sebastiani and P. Trentin, "Optimathsat: A tool for optimization modulo theories," in *Computer Aided Verification*. Springer Science + Business Media, 2015, pp. 447–454.

[17] F. B. Aydemir, P. Giorgini, and J. Mylopoulos, "Multi-objective risk analysis with goal models," in *Proceedings of the 10th IEEE International Conference on Research Challenges in Information Science*, 2016.

[18] K. Angelopoulos, F. B. Aydemir, P. Giorgini, and J. Mylopoulos, "Solving the next adaptation problem with prometheus," in *Proceedings of the 10th IEEE International Conference on Research Challenges in Information Science*. IEEE, 2016, pp. 1–10.

[19] N. Argyropoulos, K. Angelopoulos, H. Mouratidis, and A. Fish, "Decision-making in security requirements engineering with constrained goal models," in *Computer Security*. Springer, 2017, pp. 262–280.

[20] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. N. och Dag, "An industrial survey of requirements interdependencies in software product release planning," in *IEEE International Symposium on Requirements Engineering*, 2001, pp. 84–91.

[21] A. van Lamsweerde, "Requirements engineering in the year 00," in *Proceedings of the 22nd International Conference on Software Engineering*, 2000, pp. 5–19.

[22] F. Dalpiaz, X. Franch, and J. Horkoff, "iStar 2.0 language guide," arXiv:1605.07767 [cs.SE], 2016.

[23] A. Aurum and C. Wohlin, "A value-based approach in requirements engineering: Explaining some of the fundamental concepts," in *Requirements Engineering: Foundation for Software Quality*. Springer Science + Business Media, 2007, pp. 109–115.

[24] M. R. Karim and G. Ruhe, "Bi-objective genetic search for release planning in support of themes," in *Proceedings of the 6th International Symposium on Search-Based Software Engineering*. Springer International Publishing, 2014, pp. 123–137.

[25] W. Klieber and G. Kwon, "Efficient cnf encoding for selecting 1 from n objects," in *Procedings of the International Workshop on Constraints in Formal Verification*, 2007.

[26] G. Du and G. Ruhe, "Two machine-learning techniques for mining solutions of the releaseplanner$^{\text{TM}}$ decision support system," *Information Sciences*, vol. 259, pp. 474–489, 2014.

[27] I. Kwan, M. Cataldo, and D. Damian, "Conway's law revisited: The evidence for a task-based perspective," *IEEE Software*, vol. 29, no. 1, pp. 90–93, 2012.

[28] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "Adaptive socio-technical systems: A requirements-based approach," *Requirements Engineering*, vol. 18, no. 1, pp. 1–24, 2013.

[29] H. Jiang, J. Zhang, J. Xuan, Z. Ren, and Y. Hu, "A hybrid aco algorithm for the next release problem," in *Software Engineering and Data Mining*, June 2010, pp. 166–171.

[30] J. Xuan, H. Jiang, Z. Ren, and Z. Luo, "Solving the large scale next release problem with a backbone-based multilevel algorithm," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1195–1212, 2012.

[31] M. S. Feather and T. Menzies, "Converging on the optimal attainment of requirements," in *IEEE Joint International Conference on Requirements Engineering*, 2002, pp. 263–272.

[32] A. Pitangueira, P. Tonella, A. Susi, R. Maciel, and M. Barros, "Risk-aware multi-stakeholder next release planning using multi-objective optimization," in *Requirements Engineering: Foundation for Software Quality*, 2016.

[33] A. Ngo-The and G. Ruhe, "A systematic approach for solving the wicked problem of software release planning," *Soft Computing*, vol. 12, no. 1, pp. 95–108, 2007.

[34] G. Ruhe, *Product Release Planning - Methods, Tools and Applications*. CRC Press, 2010.

[35] D. Ameller, C. Farré, X. Franch, and G. Rufian, "A survey on software release planning models," in *Product-Focused Software Process Improvement*. Springer International Publishing, 2016, pp. 48–65.

[36] Y. Zhang, M. Harman, and S. L. Lim, "Empirical evaluation of search based requirements interaction management," *Information & Software Technology*, vol. 55, no. 1, pp. 126–152, 2013.

[37] B. Regnell, "What is essential? - a pilot survey on views about the requirements metamodel of reqt.org," in *Requirements Engineering: Foundation for Software Quality*, 2016.

[38] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using non-functional requirements: a process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, 1992.

[39] E. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, January 1997, pp. 236–235.

[40] D. Alrajeh, J. Kramer, A. van Lamsweerde, A. Russo, and S. Uchitel, "Generating obstacle conditions for requirements completeness," in *Proceedings of the 34th International Conference on Software Engineering*, 6 2012.

[41] A. van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 978–1005, 2000.

[42] H. Mouratidis and P. Giorgini, "Secure tropos: A security-oriented extension of the tropos methodology," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 02, pp. 285–309, 2007.

[43] F. Massacci, J. Mylopoulos, and N. Zannone, "Security requirements engineering: the si* modeling language and the secure tropos methodology," in *Advances in Intelligent Information Systems*. Springer Science + Business Media, 2010, pp. 147–174.

[44] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, 2001, pp. 249–262.

[45] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and minimum-cost satisfiability for goal models," in *Advanced Information Systems Engineering*. Springer Science + Business Media, 2004, pp. 20–35.

[46] J. Horkoff and E. Yu, "Analyzing goal models: different approaches and how to choose among them," in *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 2011, pp. 675–682.

[47] I. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *Proceedings of the 18th IEEE International Requirements Engineering Conference*, 2010, pp. 115–124.

[48] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Representing and reasoning about preferences in requirements engineering," *Requirements Engineering*, vol. 16, no. 3, pp. 227–249, 2011.

[49] F. B. Aydemir, P. Giorgini, J. Mylopoulos, and F. Dalpiaz, "Exploring alternative designs for sociotechnical systems," in *Proceedings of the 8th IEEE International Conference on Research Challenges in Information Science*, 2014, pp. 1–12.

[50] T. L. Saaty, "How to make a decision: the analytic hierarchy process," *European Journal of Operational Research*, vol. 48, no. 1, pp. 9–26, 1990.

[51] S. Liaskos, R. Jalman, and J. Aranda, "On eliciting contribution measures in goal models," in *Proceedings of the 20th IEEE International Requirements Engineering Conference*, 2012, pp. 221–230.

[52] E. Letier and A. van Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 6, p. 53, 2004.

[53] K. E. Wiegers, *Software Requirements*, 2nd ed. Microsoft Press, 2003.

[54] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. D. Penta, "Release planning of mobile apps based on user reviews," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 14–24.

[55] M. Choetkiertikul, H. K. Dam, T. Tran, T. T. M. Pham, A. Ghose, and T. Menzies, "A deep learning model for estimating story points," *IEEE Transactions on Software Engineering*, 2018.

[56] E. S.-K. Yu, "Modelling strategic relationships for process reengineering," Ph.D. dissertation, University of Toronto, 1995.

[57] A. van Lamsweerde, "Goal-oriented requirements engineering: from system objectives to uml models to precise software specifications," in *25th International Conference on Software Engineering, 2003. Proceedings.*, 2003, pp. 744–745.