Configuring Release Plans

A. Felfernig[1] and **J. Spöcklberger**[1] and **R. Samer**[1] and **M. Stettinger**[1] and **M. Atas**[1] and **J. Tiihonen**[2] and **M. Raatikainen**[2]

Abstract. Release planning takes place (1) on the *strategic level* where the overall goal is to prioritize (high-level) requirements and (2) on the *operational level* where the major focus is to define more detailed implementation plans, i.e., the assignment of requirements to specific releases and often the assignment of stakeholders to requirements. In this paper, we show how release planning can be represented as a configuration task and how re-configuration tasks can be supported. Thus we advance the state-of-the-art in software release planning by introducing technologies that support the handling of inconsistencies in already existing plans.

1 Introduction

Higher flexibility of software development and better satisfied customer requirements can be achieved by developing and delivering software in an incremental fashion [8]. Release planning is needed to support such development approaches in a structured fashion. Release planning can be regarded as a company-wide optimization problem where stakeholders want to maximize the utilization of often limited resources [8, 10, 11]. Planning often takes place on two levels [1]. First, on a *strategic level* where the major task is to prioritize requirements with regard to criteria such as business relevance (profit), feasibility (risk)[3], and related efforts [9, 11]. On an *operational level*, a detailed planning takes place where requirements are assigned to releases and often also to stakeholders in charge of their implementation. The consequences of poor release planning are low software quality, lost business oportunities, more replanning efforts, and also project cancellation [10].

Figure 1 depicts an overview of a release planning process. First, requirements are prioritized on the basis of a utility analysis [3]. Second, detailed planning takes place where requirements are assigned to releases and a release planner is in charge of assuring the consistency of the plan with regard to a set of additional constraints related to dependencies between requirements and further constraints imposed by stakeholders (see the example release constraints depicted in Table 5).

The major contributions of this paper are the following. First, we show how to represent a release planning problem as a configuration task. In this context, we also show how reconfiguration can be supported on the basis of configuration and diagnosis techniques. Second, we report the results of a performance analysis that has been conducted on the basis of different types of release planning (configuration) problems.

The remainder of this paper is organized as follows. In Section 2, we sketch how utility analysis can be performed on the basis of a given set of requirements. Thereafter, in Section 3 we show how release planning can be represented as a configuration task and how re-configuration can be supported in this context. In Section 4, we report the results of an evaluation of the proposed approach. The paper is concluded with a discussion of issues for future work.

2 Utility-based Prioritization of Requirements

The prioritization of requirements can be performed on the basis of a utility analysis, i.e., the evaluation and ranking of requirements with regard to a predefined set of interest dimensions such as *profit*, *effort*, and *risk* [3]. In the line of the two basic recommendation approaches in group recommender systems [4], prioritization can be performed in two ways (see Figure 1): (1) *aggregated utilities* based approaches collect stakeholder-individual requirements evaluations with regard to a set of interest dimensions, aggregate those evaluations, and calculate requirement utilities thereof, (2) *aggregated prioritizations* based approaches aggregate stakeholder-specific prioritizations into the final prioritization. In the following, we explain both approaches in more detail. The second variant assumes that each stakeholder provides a prioritization (directly or in terms of utility evaluations) wheres the first approach also allows prioritization in situations where not all stakeholders evaluated each of the given requirements.

Prioritization with Aggregated Utilities. In this context, multi-attribute utility theory can be applied to determine a ranking (prioritization) of a given set of requirements (see Table 1). First, each stakeholder s_i evaluates each individual requirement with regard to interest dimensions. In our example, we use the interest dimensions *profit*, *effort*, and *risk*, which are typical interest dimensions in release planning. Interest dimensions can have an assigned weight, for example, it is more important to avoid risky release plans than maximizing the potential profit.

In such a setting, the distribution of weights could be similar to the one defined in Table 2. Second, on the basis of a given set of evaluations and a specification of the importance of individual interest dimensions, requirements can be ranked according to Formula 1 where $imp(d)$ denotes the importance of interest dimension d and $contrib(r, d)$ denotes the contri-

[1] Graz University of Technology, Graz, Austria email: {felfernig,spoecklberger,samer,stettinger,atas}@ist.tugraz.at
[2] University of Helsinki, Finland email: {juha.tiihonen,mikko.raatikainen}@helsinki.fi
[3] We interpret *profit* as the *value* of a requirement for the user [7].

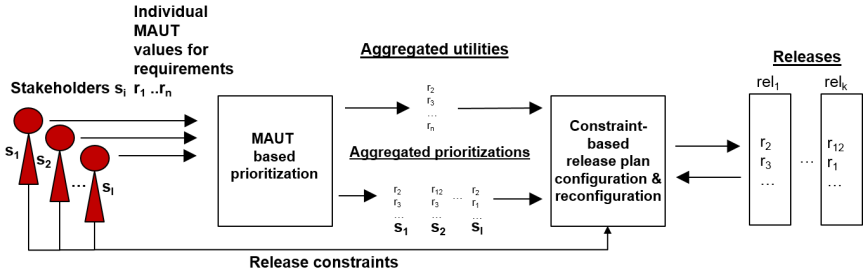

Figure 1. Utility-based prioritization and constraint-based configuration of release plans. Requirements can be prioritized on the basis of a group-based utility analysis (MAUT – multi-attribute utility theory [3]). The resulting prioritization can be determined on the basis of *aggregated utilities* or *aggregated prioritizations*. Release plans are generated on the basis of a given prioritization and corresponding release constraints defined by stakeholders.

bution of requirement r to dimension d.

$$utility(r) = \Sigma_{d \in Dim} imp(d) \times contrib(r, d) \quad (1)$$

When applying Formula 1 to the entries in Table 1 and Table 2, we are able to derive a ranking of the requirements $R = \{r_1, r_2, .., r_n\}$ as depicted in Table 3.

In this paper, we directly evaluate requirements with regard to interest dimensions (on a rating scale [1..10]). Especially for the dimensions effort and profit, alternative evaluation scales can be defined which are then mapped to a utility scale (e.g., [1..10]). For example, instead of evaluating effort directly on a scale [1..10], effort could be specified in man-months which are then translated into a corresponding utility scale. In the context of our examples, high values for *profit* denote a high associated profit, whereas high values for *effort* and *risk* denote low associated effort and risk estimates.

Prioritization with Aggregated Prioritizations. First, multi-attribute utility theory can be applied to determine stakeholder-individual requirement utilities (priorities) (see Table 4). Each stakeholder s_i evaluates individual requirements with regard to the pre-defined interest dimensions. Alternatively, stakeholders can specify prioritizizations "directly", i.e., without a utility-based pre-evaluation. Second, requirement utilities can be determined on the basis of Formula 2 where s represents a stakeholder, $d \in Dim$ represents an interest dimension, and r is a requirement. We assume a globally defined specific weight for the individual interest dimensions (see Table 2).

$$utility(r, s) = \Sigma_{d \in Dim} imp(d) \times contrib(r, d, s) \quad (2)$$

Stakeholder-individual prioritizations (see Table 4) have to be aggregated. One approach to support this aggregation is to apply basic social choice based preference aggregation functions such as *Borda Count* where the winner is the requirement with the best total ranking score where each requirement rank is associated with a score 0 .. #requirements-1 (see Table 6).[4]

Testing Prioritizations. A prioritization derived on the basis of a utility analysis can be tested for plausibility. For example, stakeholders can specify specific prioritization constraints that have to hold in the final prioritization. Such tests could be applied, for example, when different departments are cooperating in a prioritization process and specific constraints have been pre-defined by upper-level management. Such constraints can be regarded as test cases for the prioritization process (see *Definition 1*).

Definition 1: Consistent Prioritization: given a prioritization $P = \{p_1 : r_1 = v_1, p_2 : r_2 = v_2, .., p_n : r_n = v_n\}$ for requirements $REQ = \{r_1, r_2, .., r_n\}$ ($v_i \in domain(r_i)$) and a set of test cases $T = \{t_1, t_2, .., t_k\}$. Then P is a consistent prioritization if $P \cup t_i$ is consistent $\forall t_i \in T$.

Consistent prioritizations determined on the basis of a utility analysis can be regarded as input for release planning. In the following, we show how prioritizations can be exploited in the context of release planning and how release planning can be represented as a constraint-based configuration task.

3 Constraint-based Release Configuration

Constraints $r_i = v_i$ ($v_i \in domain(r_i)$) representing individual requirement prioritizations can be used as one input of a release configuration problem [5] by generating release assignment constraints following the rule $\forall \{p_i : r_i = v_i, p_j : r_j = v_j\} \in P (i \neq j) : v_i > v_j \rightarrow rel_{rj} \geq rel_{ri}$. Since $r_1 > r_2$ holds in our working example, we can derive the constraint $pc : rel_{r2} \geq rel_{r1}$ as an input for our release configuration task (see Definition 2). We denote the set $\bigcup pc_i$ derived from a prioritization P as PC.

Definition 2: Constraint-based Release Configuration Task: a constraint-based release configuration task can be defined as a tuple (R, PC, RC) where $R = \{rel_{r1}, rel_{r2}, .., rel_{rn}\}$ is a set of variables representing potential *assignments of requirements to releases* ($domain(rel_{ri})$=[0..n] – 0 refers to requirements not assigned to a release), $PC = \{pc_1, pc_2, .., pc_m\}$ represents a set of *prioritization constraints*[5], and $RC = \{rc_1, rc_2, .., rc_l\}$ is a set of *release constraints*.

Examples of typically used release constraints are given in Table 5. Further release constraints that will not be taken into account in our working example are *release capacity in hours*, *total capacity in hours*, *release costs*, *total costs*, *assignment of stakeholders to requirements*, *average risk level per release*, *minimum market value per release*, and further optimization

[4] For an overview of different types of preference aggregation functions we refer to [4].

[5] Hard prioritization constraints are often too strict in practice. They can also be interpreted as preferences, i.e., solvers should take these into account as much as possible.

requirement	r_1			r_2			r_3		
	profit	effort	risk	profit	effort	risk	profit	effort	risk
s_1	3	3	4	7	8	6	1	2	1
s_2	5	2	4	4	4	5	3	4	3
s_3	6	3	2	5	5	7	4	1	4
$average(AVG)$	4.67	2.67	3.33	5.33	5.67	6.0	2.67	2.33	2.67

Table 1. Contribution (on a scale 1–10) of requirements $R = \{r_1, r_2, r_3\}$ to the interest dimensions $Dim = \{profit, effort, risk\}$. Following the *"aggregated utilities"* approach, utility analysis determines a prioritization. AVG is one possible aggregation function – for further alternatives we refer to [4].

	profit	effort	risk
$importance(imp)$	1	3	6

Table 2. Importance of interest dimensions in a specific requirements prioritization context.

	r_1	r_2	r_3
$utility(r_i)$	32.66	58.34	25.68
$prioritization$	2	1	3

Table 3. Utility and prioritization of individual requirements $R = \{r_1, r_2, r_3\}$ with regard to the interest dimensions $Dim = \{profit, effort, risk\}$. In this context, we assume that each requirement has its unique prioritization, i.e., prioritization(r_i) = prioritization$(r_j) \rightarrow i = j$.

criteria such as *maximum profit*, *maximum customer value*, and *minimum risk*. For simplicity, Table 5 contains only binary constraints, however, these are generalizable to higher-order constraints [2], for example, coupling$(rel_{ra}, rel_{rb}, rel_{rc})$ would be translated into $rel_{ra} = rel_{rb} \wedge rel_{rb} = rel_{rc}$.

An example of a constraint-based release configuration task is the following. This task includes the three requirements from Section 2. Furthermore, three releases are available.

- $R = \{rel_{r1}, rel_{r2}, rel_{r3}\}$
- $domain(rel_{ri}) = [1..3]$
- $PC = \{pc_1 : rel_{r2} \leq rel_{r1}, pc_2 : rel_{r2} \leq rel_{r3}, pc_3 : rel_{r1} \leq rel_{r3}\}$
- $RC = \{rc_1 : rel_{r_1} = rel_{r_2}, rc_2 : rel_{r1} = 1\}$

Definition 3: Constraint-based Release Configuration: A constraint-based release configuration (solution) for a constraint-based release configuration task can be represented by a complete assignment $RP = \{rel_{ra} = va, rel_{rb} = vb, .., rel_{rk} = vk\}$ where vk is the release number of requirement rk such that $RP \cup PC \cup RC$ is consistent.

In the context of our working example, an example of a constraint-based release configuration is $RP = \{rel_{r1} = 1, rel_{r2} = 1, rel_{r3} = 2\}$.

As it is often the case, prioritization as well as release planning is an iterative process [8]. As a consequence, prioritizations PC of requirements change (resulting in PC') as well as release constraints, i.e., RC is transformed into RC'. In such contexts, situations can occur where $RP \cup PC' \cup RC'$ becomes inconsistent and we are in the need of a reconfiguration of RP (resulting in RP'). Consequently, a release reconfiguration task has to be solved (see Definition 4).

Definition 4: Release Reconfiguration Task: A release reconfiguration task can be defined by a tuple (RP, PC', RC') where PC' represents a set of adapted prioritization constraints, RC' a set of adapted release constraints, and $RP \cup PC' \cup RC'$ is assumed to be inconsistent. The underlying task is to identify a minimal set of constraints $\Delta \subseteq RP$ such that $RP - \Delta \cup PC' \cup RC'$ is consistent. If parts of RP have already been implemented, we can assume $RP = RP_{completed} \cup RP_{open}$ and the task is to identify a diagnosis Δ with $RP_{open} - \Delta \cup RP_{completed} \cup PC' \cup RC'$ is consistent.

A reconfiguration for a given release reconfiguration task can be defined as follows (see Definition 5).

Definition 5: Release Reconfiguration: A release reconfiguration (solution) for a release reconfiguration task can be represented by an assignment $RECONF = \{rel_{ra} = va', rel_{rb} = vb', .., rel_{rk} = vk'\}$ where $rel_{ri} = vi' \in RECONF \rightarrow rel_{ri} = vi \in \Delta$ and $vi \neq vi'$.

In this context, Δ represents a diagnosis, i.e., a minimal set of constraints that, if deleted from RP (RP_{open}), help to restore consistency, i.e., $RP - \Delta \cup PC' \cup RC'$ is consistent.

The set Δ can be determined on the basis of a model-based diagnosis algorithm such as FastDiag [6] which returns a minimal set of constraints that have to be deleted in order to restore consistency. In order to assure the existence of a diagnosis Δ, we have to assume the consistency of $PC' \cup RC'$.

One could also be interested in identifying minimal sets of changes Δ in given prioritizations PC such that $PC - \Delta \cup RC$ is consistent. Table 7 provides an overview of example (re-)configuration services that can be provided in release configuration scenarios. (1) proposed prioritizations (PC) have to be checked with regard to their consistency with a defined set of release constraints (RC). (2) Assuming the consistency of RC

requirement	r_1				r_2				r_3			
	profit	effort	risk	utility (prio)	profit	effort	risk	utility (prio)	profit	effort	risk	utility (prio)
s_1	3	3	4	36 (2)	7	8	6	68 (1)	1	2	1	13 (3)
s_2	5	2	4	35 (2)	4	4	5	46 (1)	3	4	3	33 (3)
s_3	6	3	2	27 (3)	5	5	7	62 (1)	4	1	4	31 (2)

Table 4. Contribution (on a scale 1–10) of requirements $R = \{r_1, r_2, r_3\}$ to the interest dimensions $Dim = \{profit, effort, risk\}$. High values for profit denote a high associated profit, whereas high values for effort and risk denote low associated effort and risk estimates. Following the *"aggregated predictions"* approach, utility analysis determines stakeholder-specific prioritizations.

constraint	formalization	description
coupling(rel_{ra}, rel_{rb})	$rel_{ra} = rel_{rb}$	$\{rel_{ra}, rel_{rb}\}$ have to be implemented in the same release
different(rel_{ra}, rel_{rb})	$rel_{ra} \neq rel_{rb} \vee rel_{ra} = 0 \wedge rel_{rb} = 0$	$\{rel_{ra}, rel_{rb}\}$ have to be implemented in different releases
eitheror(rel_{ra}, rel_{rb})	$(rel_{ra} = 0 \wedge rel_{rb} \neq 0) \vee (rel_{ra} \neq 0 \wedge rel_{rb} = 0)$	$\{rel_{ra}, rel_{rb}\}$ are exclusive
atleastone(rel_{ra}, rel_{rb})	$\neg(rel_{ra} = 0 \wedge rel_{rb} = 0)$	at least one out of $\{rel_{ra}, rel_{rb}\}$ has to be assigned to a release
atleastonea(rel_{ra}, rel_{rb}, a)	$\neg(rel_{ra} \neq a \wedge rel_{rb} \neq a)$	at least one out of $\{rel_{ra}, rel_{rb}\}$ has to be assigned to release a
atmostone(rel_{ra}, rel_{rb})	$\neg(rel_{ra} \neq 0 \wedge rel_{rb} \neq 0)$	at most one out of $\{rel_{ra}, rel_{rb}\}$ has to be assigned to a release
atmostonea(rel_{ra}, rel_{rb}, a)	$\neg(rel_{ra} = a \wedge rel_{rb} = a)$	at most one out of $\{rel_{ra}, rel_{rb}\}$ has to be assigned to release a
weakprecedence(rel_{ra}, rel_{rb})	$rel_{ra} \leq rel_{rb}$	rel_{ra} must be implemented before rel_{rb} or in the same release
weakprecedence(rel_{ra}, rel_{rb})	$rel_{ra} \leq rel_{rb} \wedge rel_{ra} > 0$	rel_{ra} must be implemented before rel_{rb} or in the same release
strictprecedence(rel_{ra}, rel_{rb})	$rel_{ra} < rel_{rb}$	rel_{ra} must be implemented before rel_{rb}
strictprecedence(rel_{ra}, rel_{rb})	$rel_{ra} < rel_{rb} \wedge rel_{ra} > 0$	rel_{ra} must be implemented before rel_{rb}
valuedependency(rel_{ra}, rel_{rb})	$\neg(\lvert rel_{ra} - rel_{rb}\rvert > k)$	rel_{ra} and rel_{rb} must be implemented in nearly the same release
effortdependency(rel_{ra}, rel_{rb})	$\neg(\lvert rel_{ra} - rel_{rb}\rvert > k)$	rel_{ra} and rel_{rb} must be implemented in nearly the same release
fixed(rel_r, a)	$rel_r = a$	requirement r must be implemented in release a
nolaterthan(rel_r, a)	$rel_r \leq a$	requirement r must not be implemented after release a
notearlierthan(rel_r, a)	$rel_r \geq a$	requirement r must not be implemented before release a

Table 5. Example release constraints. For the constraint types *valuedependency* and *effortdependency* we assume a maximum deviation of k.

	r_1	r_2	r_3
s_1	2	1	3
s_2	3	1	2
s_3	3	1	2
score (BRC)	1	6	2
prioritization	3	1	2

Table 6. Aggregation of stakeholder-individual prioritizations based on the *Borda Count* (BRC) aggregation function (highest score receives 2 points, second highest score 1 point, and lowest score receives 0 points [4]).

and the inconsistency of $PC \cup RC$, a minimal set of elements in PC has to be identified such that $PC - \Delta \cup RC$ is consistent. (3) Assuming the consistency of $PC' \cup RC'$, a minimal set of elements in RP has to be identified (i.e., a potential change of the current release plan) such that $RP - \Delta \cup PC' \cup RC'$ is consistent. (4) Given a diagnosis Δ for RP (with regard to $PC' \cup RC'$), a constraint solver can determine a solution for $RP - \Delta \cup PC' \cup RC'$. (5) If there exists a test case $t \in T$ with $inconsistent(t \cup PC)$, a diagnosis Δ represents a minimal set of elements in PC such that $PC - \Delta \cup t$ is consistent $\forall t \in T$.

ID	service
1	consistency check of $PC \cup RC$
2	diagnosis of PC with regard to RC
3	diagnosis of PC with regard to test cases T
4	diagnosis of RP with regard to $PC' \cup RC'$
5	reconfiguration of RP with regard to $PC' \cup RC'$
6	diagnosis of RC

Table 7. Overview of example release (re-)configuration services.

4 Evaluation

Beside performance analyses, there are different alternative ways to evaluate the release planning related services mentioned in Table 7.

Release plans as outcome of a configuration process can be evaluated with regard to different interest dimensions such as profit, risk, and effort. The corresponding utility function is the following (see Formula 3).

$$utility(RP) = \frac{\Sigma_{rel_r \in RP} \Sigma_{d \in Dim} \frac{contrib(r,d) \times imp(d)}{rel_r}}{|rel_r \in RP|} \quad (3)$$

An evaluation of the utility of release plans generated with the Choco constraint solver[6] is depicted in Table 8. A corresponding performance evaluation is depicted in Table 9. For each combination of $|RC| \times \#releases$, we randomly generated $|RC|$ release constraints 10 times. In this context, we did not optimize solution search on the basis of search heuristics – this is regarded as a major task of our future work. In Table 8, we can observe a decreasing utility of release plans along with an increasing size of RC. Table 9 shows increasing runtimes with an increasing size of RC and an increasing number of releases.

[6] choco-solver.org

$\|RC\|$	#releases					
	1	5	10	25	50	100
25	114.6	110.8	123.3	123.2	114.2	118.1
50	125.3	110.9	118.0	121.9	112.9	120.7
100	114.4	99.6	111.3	108.0	111.3	120.9
250	114.3	78.4	827.3	104.2	112.0	114.9
500	123.1	61.8	70.6	88.4	110.6	123.1
1000	112.3	63.1	50.7	71.7	95.3	109.4

Table 8. Avg. utility of release plans without optimization.

$\|RC\|$	#releases					
	1	5	10	25	50	100
25	54.0	19.9	16.5	77.2	142.5	587.9
50	36.1	22.1	33.3	53.5	142.5	622.4
100	75.0	84.1	85.0	144.1	654.2	641.9
250	300.8	721.9	829.4	1161.5	2094.2	3831.6
500	1110.3	2824.9	5053.4	6790.1	9372.7	16677.6
1000	5018.1	12537.6	23975.8	43738.5	58207.6	73785.1

Table 9. Avg. performance of release plan determination in *ms*.

Reconfigurations of release plans can be evaluated with regard to the *similarity* between the new configuration and the old configuration where $a(S)$ denotes the set of variable assignments contained in solution S.

$$similarity(S_{old}, S_{new}) = \frac{a(S_{old}) \cap a(s_{new})}{a(S_{old}) \cup a(s_{new})} \quad (4)$$

An evaluation of the similarity between reconfigurations and original release plans is depicted in Table 10. For each combination of $|RC| \times \#releases$, we randomly generated $|RC|$ release constraints and a corresponding release plan 10 times, randomly changed 10% of the (original) release constraints and determined a reconfiguration (for the given release plan). We can observe a decreasing similarity with a corresponding increasing number of release constraints.

$\|RC\|$	#releases					
	1	5	10	25	50	100
25	.96	.99	1.00	1.00	1.00	1.00
50	.93	.97	.99	1.00	1.00	1.00
100	.87	.96	.97	.99	.99	1.00
250	.96	.87	.93	.98	.99	.99
500	.99	.76	.86	.95	.97	.99
1000	1.00	.75	.78	.90	.95	.97

Table 10. Avg. similarity of reconfigurations.

Diagnoses can be evaluated with regard to their degree of *conservativism* (see Formula 5), i.e., the number of changes needed in a constraint set C compared to the overall number of elements in C. Furthermore, *diagnoses* can be evaluated with regard to their *relevance*: the lower the total relevance of constraints contained in a diagnosis (represented as the sum of the individual relevances ($rel(\delta_i)$) of constraints in Δ), the higher the relevance of the Δ (see Formula 6).

$$conservativism(\Delta, C) = 1 - \frac{|\Delta|}{|C|} \quad (5)$$

$$relevance(\Delta = \{\delta_1, \delta_2, .., \delta_q\}) = \frac{\Sigma_{\delta_i \in \Delta} rel(\delta_i)}{|\Delta|} \quad (6)$$

An evaluation of conservativism and relevance of generated diagnoses is depicted in Table 11. For each combination of $|RC| \times \#releases$, we randomly generated $|RC|$ release constraints 10 times, assigned importance values to these constraints, and randomly changed 10% of the constraints. The resulting (inconsistent) constraint sets were diagnosed with FastDiag [6]. The corresponding evaluation results are depicted in Table 11. We can observe a decreasing degree of conservativism with an increasing number of release constraints RC.

	#releases					
$\|RC\|$	1	5	10	25	50	100
25	.84/1.0	.99/.2	.99/.1	1.00/0	1.00/0	1.00/0
50	.75/1.0	.99/.5	.99/.4	1.00/0	1.00/0	1.00/0
100	.62/1.0	.98/1.0	.99/.7	1.00/.2	1.00/0	1.00/0
250	.58/1.0	.86/1.0	.96/1.0	.99/.9	1.00/.8	1.00/.1
500	.56/1.1	.76/1.0	.89/1.0	.98/1.0	.99/.8	1.00/.8
1000	.55/1.2	.64/1.0	.76/1.0	.92/1.0	.97/1.0	.99/1.0

Table 11. Avg. conservativism and relevance (c/r) of diagnoses.

5 Conclusion and Future Work

In this paper, we have shown how to represent release planning as a configuration problem. This representation is a major basis for supporting re-planning tasks, i.e., the adaptation of plans that become inconsistent due to changing constraints (e.g., a changing availability of resources). In this context, we also focused on introducing concepts that support the automated adaptation of release plans (reconfiguration of release plans) and different variants thereof such as the diagnosis of release constraints and prioritization constraints. To show the applicability of the presented concepts, we have presented initial evaluation results based on a couple of evaluation metrics. Future work will include the development and evaluation of different types of preference aggregation functions (see Section 2) with regard to their capability of generating relevant prioritizations. Furthermore, we will optimize the determination of release plans, reconfigurations, and diagnoses by integrating intelligent search heuristics that help to improve the quality of identified solutions. In this context, we will compare constraint-based reasoning approaches with local search based ones [8] with regard to efficiency and solution quality.

Acknowledgment

The work presented in this paper has been conducted within the scope of the Horizon 2020 project OpenReq (732463).

REFERENCES

[1] D. Ameller, C. Farre, X. Franch, D. Valerio, and A. Cassarino, 'Towards continuous software release planning', in *SANER 2017*, pp. 402–406, Klagenfurt, Austria, (2017).

[2] F. Bacchus, X. Chen, P. van Beek, and T. Walsh, 'Binary vs. non-binary constraints', *Artificial Intelligence*, **140**(1–2), 1–37, (2002).

[3] J. Dyer, 'Multi attribute utility theory', *International Series in Operations Research and Management Science*, **78**, 265–292, (1997).

[4] A. Felfernig, L. Boratto, M. Stettinger, and M. Tkalcic, *Group Recommender Systems – An Introduction*, Springer, 2018.

[5] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Elsevier/Morgan Kaufmann Publishers, 1st edn., 2014.

[6] A. Felfernig, M. Schubert, and C. Zehentner, 'An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets', *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, **26**(1), 175–184, (2012).

[7] D. Greer, D. Bustard, and T. Sunazuka, 'Prioritization of system changes using cost-benefit and risk assessments', in *4th International Symposium on Requirements Engineering*, pp. 180–187, Limerick, Ireland, (1999).

[8] D. Greer and G. Ruhe, 'Software release planning: An evolutionary and iterative approach', *Information and Software Technology*, **46**(4), 243–253, (2004).

[9] H. Jung, 'Optimizing value and cost in requirements analysis', *IEEE Software*, **15**(4), 74–78, (1998).

[10] M. Lindgren, R. Land, C. Norström, and A. Wall, 'Key aspects of software release planning in industry', in *19th Australian Conference on Software Engineering*, pp. 320–329, Perth, WA, Australia, (2008).

[11] G. Ruhe and M. Saliu, 'The art and science of software release planning', *IEEE Software*, **22**(6), 47–53, (2005).