Agile Release Planning through Optimization

Ákos Szőke

Department of Measurement and Information Systems
Budapest University of Technology and Economics, Budapest, Hungary

Abstract. Agile software development represents a major approach to software engineering. Agile processes offer numerous benefits to organizations including quicker return on investment, higher product quality, and better customer satisfaction. However, there is no sound methodological support of agile release planning – contrary to the traditional, plan-based approaches. To address this situation, we present an agile release planning model and a heuristic optimization algorithm as a solution. Four real life data sets of its application and evaluation are drawn from the lending sector. The experiment demonstrates that this approach can provide more informed and established decisions and support easy optimized release plan productions. Finally, the paper analyzes benefits and issues from the use of this approach in system development projects.

1 Introduction

Development governance covers the steering of software development projects. Traditional governance usually applies command-and-control approaches which explicitly direct development teams. Experiences with these approaches – such as Control Objectives for Information-Related Technology (CobiT) [1], and the Organizational Project Management Maturity Model (OPM) [2] – show that they are too heavy in practice for many organizations, although they provide a wealth of advice [3]. As a reaction to so-called *heavyweight* methodologies [4], many practitioners have adopted the ideas of agility [5]. Agile approaches are quickly becoming the norm, probably because recent surveys showed agile teams are more successful than traditional ones [6, 7]. Several studies pointed out $\approx 60\%$ increase in productivity, quality and improved stakeholder satisfaction [7, 8], and 60% and 40% reduction in pre-, and post-release defect rates [9].

In recent years, several agile methods have emerged. The most popular methods are Extreme Programming (XP) [10](58%), Scrum [11](23%), and Feature Driven Development (FDD) [12](5%) [13]. Despite variety of methods all of them share the common principles and core values specified in the *Agile Manifesto* [5].

Release planning is an activity concerned with the implementation of the selected requirements in the next version of the software. Agile release planning is usually based on a prioritized list of requirements (typically User stories) and is made up of the following major steps: the team i) performs estimation on every requirement, ii) determines the number and the length of iterations using historical iteration velocity (i.e. how much work can be done during each iteration), iii) distributes requirements into iterations considering constraints (Figure 1).

Szőke Á. (2009).
Agile Release Planning through Optimization.
In *Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering - Evaluation of Novel Approaches to Software Engineering*, pages 149-160
DOI: 10.5220/0001865001490160

Fig. 1. Release planning in agile software development.

Problems. The essential aim of release planning is to determine an *optimal* execution plan of development respect to scarcity of available resources and dependencies between requirements. However, distributions of requirements are iteratively selected and assigned *manually* into iterations (see Figure 1). As a consequence, the following factors are managed implicitly: **P1**) *precedences* (temporal constraints between requirements), **P2**) *resource capacities* (resource demands during iterations), and **P3**) *priorities* (importance of each requirement delivery). Therefore, optimality of plans (i.e. maximal business value or minimal cost) is heavily based on the manager's right senses – nevertheless optimized project plans are crucial issues from the economic considerations of both customer and developer sides.

Objectives. Our proposed method intends to mitigate previous problems (**P1-3**) by i) formulating release planning task as an *optimization model* that considers all the previous factors and ii) providing a solution to this model by a *heuristic algorithm* to easily produce release plans.

Outline. The rest of the paper arranged as follows: Sec. 2 presents common notions of agile planning; Sec. 3 introduces our optimization model and algorithm; Sec. 4 describes experiences; Sec. 5 discusses our solution; Sec. 6 focuses on related work; Sec. 7 concludes the paper.

2 Agile Release Planning

In this section, we introduce agile release planning to provide the necessary background information for the proposed method.

2.1 Requirements Specification

Common to all software development processes in any projects is the need to capture and share knowledge about the requirements and design of the product, the development process, the business domain, and the project status. Contrary to the traditional methods (e.g. waterfall), agile methods advocate '*just enough*' documentations where the fundamental issue is communication, not documentation [4, 11]. The agreed requirements not only drive the development, but direct planning of projects, provide basis for acceptance testing, risk management, trade-off analysis and change control [14]. In agile methods *User stories*, *Features*, and *Use cases* (see XP [10], FDD [12]) are the primary models of requirements and the source of effort estimation.

Estimating Effort. Agile teams focus on '*good enough*' estimates and try to optimize their estimating efforts. A good estimation approach takes short time, and can be used

for planning and monitoring progress [15]. Effort estimation models usually based on size metrics: Story points [16], Feature points [12], and Use case points [17]. These metrics are abstract units, express the whole size of development and usually not directly convertible to person hours/days/months.

User Stories and Story Points. The most popular requirements modeling technique in agile methods is the User story technique [16]. User stories are usually materialized on electronic or paper Story cards with the description of i) the feature demanded by stakeholders, ii) the goal of the customers, and iii) the estimated size of the development work is expressed by a Story point and interpreted as person day effort (usually classified into Fibonacci-like effort sequence: 0.5, 1, 2, 3, 5, 8, 13) [15].

Dependencies. The complexity of scheduling arises from the interaction between requirements (User stories) by *implicit* and *explicit* dependencies. While the previous is given by the scarcity of resources, the latter one is emerged from different precedences between tasks' realizations [18, 19]:

i) *Functional Implication* (j demands i to function),
ii) *Cost-based Dependency* (i influences the implementation cost of j, so useful to realize i earlier),
iii) *Time-related Dependency* (expresses technological and/or organizational demands).

3 Optimized Release Planning

In this section we point out release planning can be characterized as a special bin-packing problem. Then we formulate a bin-packing-related optimization model for release planning, and present a solution to this model in the form of a heuristic algorithm.

3.1 Mapping to Bin-packing

Generally, a bin-packing problem instance is specified by a set of items and a standard bin size. The objective is to pack every item in a set of bins while minimizing the total number of bins used [20, 21]. The analogy between release planning and bin-packing problem can be explained as follows.

The team's resource capacity in an iteration stands for a bin size, while a requirement's resource demand represents an item size. In the resource-constrained project scheduling problem (RCPSP) context, we can view each iteration within release as a bin into which we can pack different deliverable requirements. Without loss of generality, we can ensure that the resource demand of each requirement is less than team's resource capacity in an iteration. Then minimizing makespan (i.e. finding the minimum time to completion) of this RCPSP is equivalent to minimizing the number of bins used in the equivalent bin-packing problem [22].

We extend this ordinary bin-packing problem and interpretation with the following elements to provide further computational capabilities for wide-ranging release planning situations (c.f. **P1-3**): i) precedences between items (requirements) (c.f. 2.1), ii) varying capacities of bins (iterations), and iii) item priorities. From now on we call this extended problem as bin-packing-related RCPSP (BPR-RCPSP).

3.2 Formulating BPR-RCPSP Model

Henceforth, without loss of generality, we focus on the User story technique to be more concrete. Given a set of deliverable User stories j ($j \in A : |A| = n$) with *required efforts* w_j, and iterations n with different *capacities* c_i ($i \in \{1, 2, ..., n\}$) within a release. Let assign each User story into one iteration so that the total required effort in iteration i does not exceed c_i and the number of iteration used as a minimum while precedence relation (matrix) $P_{j,j'} \in \{0, 1\}$ (where $P_{j,j'} = 1$ if j precedes j', otherwise $P_{j,j'} = 0$ – c.f. 2.1) holds. A possible mathematical formulation is:

$$\text{Minimize } z = \sum_{i=1}^{n} y_i \tag{1a}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_{i,j} \leq c_i y_i \tag{1b}$$

$$i' - i \geqslant P_{j,j'} \quad : x_{i',j'} = x_{i,j} = 1 \tag{1c}$$

$$\sum_{i=1}^{n} x_{i,j} = 1 \tag{1d}$$

where $y_i = 0$ or $1,$ and $x_{i,j} = 0$ or 1 $(i, j \in N)$, and

$$x_{i,j} = \begin{cases} 1 & \text{if } j \text{ is assigned to iteration } i \\ 0 & \text{otherwise} \end{cases} \tag{2a}$$

$$y_i = \begin{cases} 1 & \text{if iteration } i \text{ is used} \\ 0 & \text{otherwise} \end{cases} \tag{2b}$$

The equations denote minimization of iteration count of release (1a), resource constraints (1b), temporal constraints (1c), and an item j can be assigned to only one iteration (1d). We will suppose, as is usual, that the efforts w_j are positive integers. Without loss of generality, we will also assume that

$$c_i \text{ is a positive integer} \tag{3a}$$

$$w_j \leq c_i \text{ for } \forall i, j \in N \tag{3b}$$

If assumption (3a) is violated, c_i can be replaced by $\lfloor c_i \rfloor$. If an item violates assumption (3b), then the instance is treated as trivially infeasible. For the sake of simplicity we will also assume that, in any feasible solution, the lowest indexed iterations are used, i.e. $y_i \geq y_{i+1}$ for $i = 1, 2, ..., n - 1$.

3.3 Solving the BPR-RCPS Problem

For the previously formulated optimization model we developed a *Binscheduling* algorithm (Algorithm 1). It is a constructive heuristic algorithm, which iteratively selects

Algorithm 1: *Binsched* algorithm with BF strategy.

Require:
$w_j \in N$ /* weights of each User story j */
$P_{j,j'} \in 0,1 \wedge P_{j,j} = 0 \wedge P$ is DAG /* precedences */
$p_j \in N, c_i \in N$ /* priority values and capacity of each iteration */
Ensure: $X_{i,j} \in 0,1 \wedge \forall j \exists! i \; X_{i,j} = 1$
1: $n \Leftarrow length(\mathbf{w})$ /* schedulable User stories */
2: $\mathbf{X} \Leftarrow [0]_{n,n}$ /* assignment matrix initialization */
3: $\mathbf{r} \Leftarrow \mathbf{c}$ /* residual capacities of each iteration */
4: $\mathbf{rlist} \Leftarrow \emptyset$ /* 'ready list' initialization */
5: $\mathbf{slist} \Leftarrow \emptyset$ /* 'scheduled list' initialization */
6: **for** $fj = 0$ to n **do**
7: $\mathbf{pot} \Leftarrow findNotPrecedentedItems\,(\mathbf{P})$
8: $\mathbf{rlist} \Leftarrow \mathbf{pot} \setminus \mathbf{slist}$ /* construct ready list */
9: **if** $\mathbf{rlist} == \emptyset$ **then**
10: **print** `'Infeasible schedule!'`
11: **return** $\emptyset$
12: **end if**
13: $j \Leftarrow \min\{p_j\} : j \in \mathbf{rlist}$
14: $i \Leftarrow selectBestFittingBin\,(w_j, \mathbf{r})$
15: $X_{i,j} \Leftarrow 1$ /* assign User story j to iteration i */
16: $r_i \Leftarrow r_i - w_j$ /* decrease residual capacity */
17: $\mathbf{slist} \Leftarrow \mathbf{slist} \cup \{j\}$
18: $P_{\{1,\ldots,n\},j} = 0$ /* delete scheduled User story */
19: **end for**
20: **return** $\mathbf{X}$

and schedules an item (*User story*) into an *iteration* – where it fits best. In the program listing lowercase and uppercase letters with indices denote vectors and matrices (e.g. c_i, $P_{j,j'}$). While bold-faced letters show concise (without indices) forms (e.g. **c**, **P**).

In the *require* section the preconditions are given. Each w_j is the weight (required effort) for User story j in Story point. Precedences between User stories can be represented by a precedence matrix where $P_{j,j'} = 1$ means that User story j precedes User story j', otherwise $P_{j,j'} = 0$. Both conditions $P_{j,j} = 0$ (no loop) and P is directed acyclic graph (DAG) ensures that temporal constraints are not trivially unsatisfiable. Priorities p_j express stakeholders' demands and are used by the scheduler algorithm as a rule when choosing between concurrent schedulable User stories. Capacities of iterations are calculated by taking the historical values of iteration velocities into consideration. The *ensure* section prescribes the postcondition on the return value ($\mathbf{X}$): every User story j has to be assigned to exactly one iteration i.

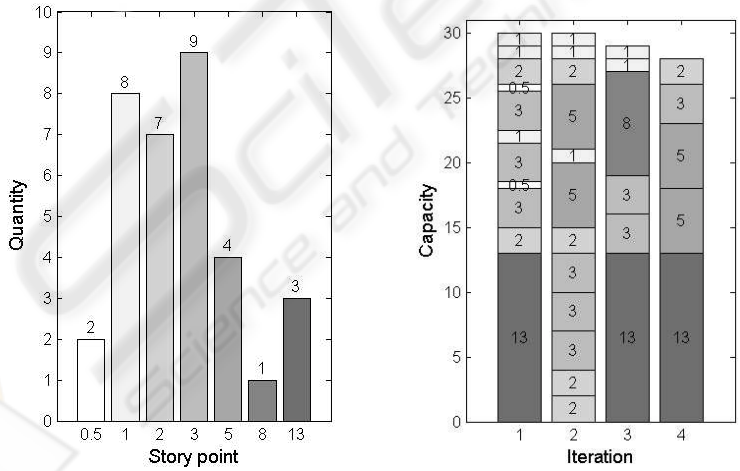
During scheduling steps, first the initial values are set (line $1 - 5$). The iteration value (n) is equal to the number of User stories (line 1). The residual capacity denotes the remained capacity of an iteration after a User story is assigned – so it is initially set to capacity (line 3). The algorithm uses a *ready list* (**rlist**) and a *scheduled list* (**slist**) to keep track of schedulable and scheduled User stories. *Potentially* schedulable items (**pot**) are unscheduled items from which the algorithm can choose in the current con-

trol step without violating any precedence constraint (line 7). Previously assigned items are extracted from potentially schedulable items to form the ready list (line 8). As long as the ready list contains schedulable items, the algorithm chooses items from that list – otherwise the schedule is infeasible (line 9). The minimum priority item is selected from the ready-list to schedule (line 13). To find the proper iteration term for the selected item, the *best fit* (i.e. having the smallest residual capacity) strategy (line 14) is applied, and an item j is assigned to iteration i (i.e. $X_{i,j} = 1$). As a consequence residual capacity of iteration i is decreased by item weight w_j (line 16). Finally, scheduled list (**slist**), is updated with scheduled item (lines 17), and no longer valid precedence relations are also deleted from **P** after scheduling of the given item (lines 18). Iteration proceeds until all items are assigned to iterations (line 6-19).

After termination, $\mathbf{X}$ contains the User story assignments to iterations, where the number of nonzero columns denotes the packed iterations (i.e. $z \Leftarrow length\left(nonZeros\left(\sum_{j=1}^{n} w_j x_{i,j}\right)\right)$ – c.f. (1a)).

There can be used several strategies (e.g. *FirstFit, BestFit*) to find the appropriate release plan, but we used only one (the *best fit*) for simple demonstration (line 14). This greedy strategy makes a series of local decisions, selecting at each point the best step without backtracking or lookahead. As a consequence, local decisions miss the global optimal solution, but produce quick (time complexity is clearly *O(nlogn)*) and usually sufficient results for practical applications [21].

Figure 2 illustrates the packing concept. This example shows the post-mortem release planning result of a real life development situation using the previous algorithm.



Schedulable User stories (i). User story assignments. ($X_{i,j}$)

Fig. 2. Release plan applying the BPR-RCPSP approach.

Figure 3.3 demonstrates the histogram of schedulable User stories. The x-axis enumerates Story point categories (weights), while y-axis shows how many User stories fall into these categories. Figure 3.3 depicts planning results produced by *Binsched* algo-

rithm in stacked bar chart form: the schedulable User stories are packed into four iterations (x-axis) with capacities $30, 30, 29$, and 28 (y-axis). Bar colors on the Figure 3.3 point out how Story points are distributed on Figure 3.3.

4 Experimentation

To obtain a proof-of-concept we implemented a prototype as a scheduling toolbox in Matlab [23]. Four past release data sets – extracted from the backlog of IRIS application developed by Multilogic Ltd [24] – were compared against the results of simulations applying the same inputs [25].

4.1 Context and Methodology

IRIS is a client risk management system (approx. 2 million SLOC) for credit institutions for analyzing the non-payment risk of clients. It has been continual evolution since its first release in the middle of 90s. The system was written in Visual Basic and C# the applied methodology was a custom agile process. The release planning process were made up of the following steps. First, the project manager used intuitive rule for selecting User stories from the backlog into a release. Then the team estimated on every User story and determined the number and the length of iterations within the release – based on iteration velocity. Finally, the team distributed User stories into iterations considering priorities and precedences.

4.2 Data Collection and Results

Four data sets (Collateral evaluation, Risk assumption, Ukrainian deal flow I/II – respectively $R_A - R_D$) were selected to make a comparison between the algorithmic method and the manual release planning carried out previously at Multilogic. The R_A data set is used to present the concept in the previous example (Figure 2). All the releases had same project members (6 developers), iteration length (2 weeks), iteration velocity (30 Story point), domain, customer, and development methodology, but they were characterized by different User story counts (USC), Iteration counts (IC), Buffer per releases (BpR) (for contingencies), and delivered Story point per iteration (SP_i). Table 1 summarizes the variables of $R_A - R_D$ collected from the company's Microsoft SharePoint-based backlog.

To determine the usefulness of our proposed method, we used historical data as input of the Binsched algorithm (Algorithm 1). This method made it possible to compare performance of the algorithmic (optimized) approach against the manual one. Computed values ($R_A^* - R_D^*$) are shown in Table 2 (since USC, IC, BpR were the same as Table 1 they are not shown).

Table 1. Historical release plan values ($R_A - R_D$).

	USC	IC	BpR	SP_1	SP_2	SP_3	SP_4	SP_5	$\sum_1^5 SP_i$
R_A	33	4	3.0	28.0	35.0	24.0	30.0	0.0	117.0
R_B	25	3	4.5	33.0	34.5	18.0	0.0	0.0	85.5
R_C	27	5	12.5	31.5	33.0	23.0	26.0	24.0	137.5
R_D	27	4	3.5	29.5	33.0	27.0	27.0	0.0	116.5

Table 2. Optimized release plan values ($R_A^* - R_D^*$).

	SP_1^*	SP_2^*	SP_3^*	SP_4^*	SP_5^*
R_A^*	30.0	30.0	29.0	28.0	0.0
R_B^*	30.0	28.5	27.0	0.0	0.0
R_C^*	29.5	30.0	30.0	29.0	19.0
R_D^*	29.5	30.0	30.0	27.0	0.0

4.3 Analysis

The analysis goal was to compare the manual and the optimized approaches using the same *input variables*. The following key questions were addressed: **Q1**: *What are the staffing requirements over time?*; **Q2**: *How many iterations do we need per release?*; and **Q3**: *How buffers for contingencies are allocated?*

To answer to these questions, 1) we carried out Exploratory Data Analysis (EDA) [26, 27] to gaining insight into the data sets, then 2) we performed descriptive statistical analysis to compare the main properties of the two approaches.

Qualitative Analysis. The following EDA techniques (called 4P EDA) are simple, efficient, and powerful for the routine testing of underlying assumptions [26]:

1. *run sequence plot* (Y_i versus iteration i)
2. *lag plot* (Y_i versus $Y_i - 1$)
3. *histogram* (counts versus subgroups of Y)
4. *normal probability plot* (ordered Y versus theoretical ordered Y)

where $Y_i \triangleq \sum_{j=1}^{n} w_j x_{i,j}$ (i.e. sum of assigned Story point of each iteration (c.f. 1b) were identified as *result variables* to test or questions (**Q1-3**).

The four EDA plots are juxtaposed for a quick look at the characteristics of the data (Figure 3). The assumptions are addressed by the graphics:

A1: The run sequence plots indicate that the data do not have any significant shifts in location but have significant differences in variation over time.

A2: The upper histogram depicts that the data are skewed to the left, there do not appear to be significant outliers in the tails, and it is reasonable to assume the data are from approximately a normal distribution. Contrary, lower one shows asymmetricity (skewed to the left heavily), data are more peaked than the normal distribution. Additionally, there is a limit in the data (30) that can be explained by the subject of the optimization (c.f. 1b).

A3: The lag plots do not indicate any systematic behavior pattern in the data.

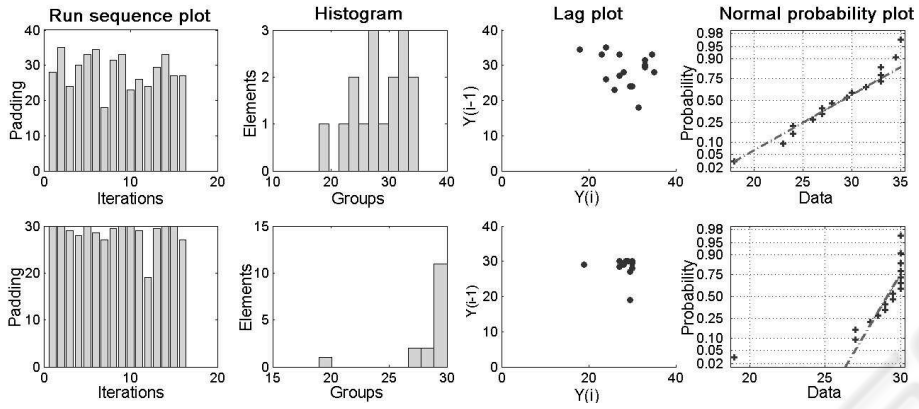


Fig. 3. 4P of historical (upper) and optimized (lower) plans

A4: The normal probability plot in upper approximately follows straight lines through the first and third quartiles of the samples, indicating normal distributions. On the contrary, normality assumption is in fact not reasonable on the right.

From the above plots, we conclude that there is no correlation among the data (**A3**), the historical data follow approximately a normal distribution (**A4**), and the optimized approach yields more smooth release padding and less variance (**A1**,**A2**).

Quantitative Analysis. Due to **A3** data sets could be analyzed with summary (descriptive) statistics (Table 3), and hypothesis test. Table 3 shows important differences between the historical and optimized data:

D1: in the optimized case sample standard deviation is approximately halved, which supports **A1**,

D2: despite of the fact that iteration velocity was 30 Story points the release plan prescribed 35 in the historical case which resulted 17% resource overload (c.f. **A2**),

D3: relatively large skewness of the optimized case (histogram in Figure 3) can be interpreted by the capacity constraints of the optimization (see 1b),

D4: relatively large kurtosis of the optimized case (histogram in Figure 3) can be explained by the subject of the optimization (see 1a).

After statistical analysis, Lilliefors test is carried out to quantify the test of normality (c.f. **A4**) at $\alpha = 95\%$ significance level: historical data comes from a normal distribution ($H_0 : F(Y_i) = \Theta(Y_i)$), against the alternative hypothesis ($H_1 : F(Y_i) \neq \Theta(Y_i)$).

Table 3. Comparison with descriptive statistics.

	Mean	Min	Max	Std.dev.	Skewness	Kurtosis
R_{A-B}	28.53	18.0	35.0	4.78	-0.48	2.50
R^*_{A-B}	28.53	19.0	30.0	2.75	-2.82	10.35

The result yielded *p-value* = 0.5 (observed significance level). As *p-value*$> (1-\alpha)$ so historical data follow normal distribution (so H_0 was accepted at 95% significance level). Since the sample was relatively small, the Lilliefors test was adequate [26].

Finally, maximum likelihood estimation (MLE) procedure was accomplished to find the value of μ (expected value) and σ (standard deviation) parameters of the normal distribution. The estimation resulted $\mu = 28.53$ and $\sigma = 4.63$ values [27].

As a consequence, in the optimized case: staffing requirements (c.f. **Q1**) showed more smooth release padding, with less variance and an upper limit, therefore constituted less risk level regarding resource overload; iteration counts per releases (c.f. **Q2**) did not exhibit any differences contrary to the historical data; finally release buffers (c.f. **Q3**) were moved from the end of iterations to the end of releases which more advisable to mitigate risks [28].

5 Discussion and Related Work

Without loss of generality, we have selected User story as the most popular agile requirements modeling technique as a subject of release planning. User stories have many advantages including i) comprehensible to both customers and the developers, ii) emphasize verbal rather than written communication, iii) represent a discrete piece of functionality, iv) work for iterative development, and finally v) right sized for estimating (i.e. Story points [16]) and planning [10, 15].

We applied the popular Story point method to estimate realization duration of each User story. Up to now, several case studies reported that the Story point is a reliable method to estimate the required effort at the release planning phase [6, 7, 16].

Then we formulated release planning as BPR-RCPSP to provide algorithmic User story distribution considering i) team's resource capacity in an iteration and ii) minimizing the number of iteration used scheduling objective. Our proposed BP-RCPSP is an extension of bin-packing optimization model to cover wide-ranging release planning situations with the expression of: i) precedences between requirements (c.f. 2.1), ii) varying capacities of iterations, and iii) requirements priorities (c.f. **P1-3**). This interpretation makes it possible to adapt extremely successful heuristic algorithms applied to solving bin-packing situations. Generally, bin-packing problems are combinatorial NP-hard problems to which a variety approximation and only a few exact algorithms are proposed. The most popular heuristics in approximation algorithms are *First-Fit* (FF), Next-Fit Decreasing (NFD), First-Fit Decreasing (FFD), where the time complexity is $O(nlogn)$ – considering the worst-case performance ratio of the algorithm [21].

We developed a bin-packing algorithm (*Binsched*) for the BP-RCPSP model which illustrated the iteration capacities are filled more smoothly (c.f. **Q1**) and release buffers are adjusted to the end of the last iterations (c.f. **Q3**) to prevent slippage of schedule by the optimal usage of buffers [28]. Metrics indicated that the algorithmic approach

balanced the workload by halved the dispersion (coefficient of variation ($c_v = \sigma/\mu$): $c_v^{hist} = 0.17 > c_v^{optm} = 0.09$) therefore provided less risky release plans besides satisfying the same constraints. Moreover, the easy and fast computation allows the user to generate alternative selections and what-if analysis to tailor the best plan for the specific project context and considering the stakeholders' feedbacks by altering constraints, capacities and priorities.

The growing pressure to reduce costs, time-to-market and to improve quality catalyzes transitions to more automated methods and tools in software engineering to support project planning, scheduling and decisions [14]. Scheduling requirements into the upcoming release version is complex and mainly manual process. In order to deal with this optimization problem some method have been proposed. Compared to the extensive research on requirements priorization [29, 30], interdependencies [19, 18], and estimation [17], only few researches performed requirements release planning. In [19] release planning was formulated as Integer Linear Programming (ILP) problem, where requirement dependencies were treated as precedence constraints. The ILP technique is extended with stakeholders' opinions in [31], and with some managerial steering mechanism that enabled what-if analysis [32]. In [33] a case study showed that integration of requirements and planning how significantly ($> 50\%$) can accelerate UML-based release planning. Furthermore, all the previous methods relate to the traditional RCPSP.

6 Conclusions

Up to our best knowledge, the proposed optimized model formulation of agile release planning is novel in the field. Although, there are some tenets to manual planning [6, 15] algorithmic solution could not be found. To evaluate our model a simulation was carried out that demonstrated the method could easily cope with the previously manually managed planning factors i.e. precedences, resource constraints and priorities (c.f. **P1-3**) besides providing optimized plans. Additionally, this approach provides more informed and established decisions with application of what-if analysis, and mitigates risks with more smooth and limited requirements allocation and with moving buffers to the end of releases. We believe the results are even more impressive in more complex (more of constraints, user stories etc.) situations.

We think that our proposed method is a plain combination of the present theories and methods, that is demonstrated by the empirical investigation and the prototype. It lead us to generalize our findings beyond the presented experiments.

References

1. Information Systems Audit and Control Association: Control objectives for IT and related technology. http://www.isaca.org/ (2008) Accessed on 28 May 2008.
2. Project Management Institute: Organizational pm maturity model. http://msdn2.microsoft.com (2003) Accessed on 28 May 2008.
3. et al., S.A.: Best practices for lean development governance. The Rational Edge (2007)
4. Chau, T., Maurer, F., Melnik, G.: Knowledge sharing: Agile methods vs. tayloristic methods. (2003) 302–307
5. Manifesto, A.: Manifesto for agile software development. URL: www.agilemanifesto.org (2001) Accessed on 27 Feb 2008.

6. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information and Software Technology 50 (2008) 833–859
7. Ambler, S.W.: Survey says: Agile works in practice. Dr. Dobb's Journal (2006)
8. Layman, L., Williams, L., Cunningham, L.: Motivations and measurements in an agile case study. Journal of Systems Architecture 52 (2006) 654–667
9. Layman, L., Williams, L., Cunningham, L.: Exploring extreme programming in context: An industrial case study. ADC '04: Proceedings of the Agile Development Conf. (2004) 32–41
10. Beck, K., Andres, C.: Extreme Programming Explained : Embrace Change (2nd Edition). Addison-Wesley Professional (2004)
11. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)
12. Palmer, S.R., Felsing, M.: A Practical Guide to Feature-Driven Development. Pearson Education (2001)
13. Chow, T., Cao, D.B.: A survey study of critical success factors in agile software projects. Journal of System and Software 81 (2008) 961–971
14. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: ICSE - Future of SE Track. (2000) 35–46
15. Cohn, M.: Agile Estimating and Planning. Prentice Hall PTR, NJ, USA (2005)
16. Cohn, M.: User Stories Applied For Agile Software Development. Addison-Wesley (2004)
17. Anda, B., Dreiem, H., Sjøberg, D.I.K., Jørgensen, M.: Estimating software development effort based on use cases - experiences from industry. In: 4th International Conference on the UML. Lecture Notes in Computer Science, Springer (2001) 487–502
18. Li, C., van den Akker, J.M., Brinkkemper, S., Diepen, G.: Integrated requirement selection and scheduling for the release planning of a software product. In: REFSQ. Volume 4542 of Lecture Notes in Computer Science., Springer (2007) 93–108
19. et al., P.C.: An industrial survey of requirements interdependencies in software product release planning. (2001) 84
20. Hartmann, S.: Packing problems and project scheduling models: an integrating perspective. Journal of the Operational Research Society 51 (1 September 2000) 1083–1092(10)
21. Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., New York, NY, USA (1990)
22. Schwindt, C.: Resource Allocation in Project Management. Springer-Verlag Berlin and Heidelberg GmbH & Co. K (2005)
23. Mathworks: Matlab homepage. (2008) Accessed on 28 May 2008.
24. Multilogic: Multilogic homepage. URL: http://www.multilogic.hu (2008)
25. Kellner, M., Madachy, R., Raffo, D.: Software process simulation modeling: Why? what? how? Journal of Systems and Software 46 (1999) 91–105
26. Martinez, W.L.: Exploratory Data Analysis with MATLAB (Computer Science and Data Analysis). Chapman & Hall/CRC (2004)
27. Shao, J.: Mathematical Statistics: Exercises and Solutions. Springer (2005)
28. Tukel, O.I., Rom, W.O., Eksioglu, S.D.: An investigation of buffer sizing techniques in critical chain scheduling. European Journal of Operational Research 172 (2006) 401–416
29. Berander, P., Andrews, A.: Requirements Prioritization. In: Engineering and Managing Software Requirements. Springer-Verlag, Inc., Secaucus, NJ, USA (2005) pp.69–94
30. Karlsson, L., Thelin, T., Regnell, B., Berander, P., Wohlin, C.: Pair-wise comparisons versus planning game partitioning–experiments on requirements prioritisation techniques. Empirical Software Engineering 12 (2007) 3–33
31. Ruhe, G., Saliu, M.: The art and science of software release planning. Software, IEEE 22 (Nov.-Dec. 2005) 47–53
32. Marjan van den Akker, Sjaak Brinkemper, G.D.J.V.: Software product release planning through optimization and what-if analysis. Technical Report UU-CS-2006-063 (2006)
33. Szoke, A.: A proposed method for release planning from use case-based requirements. In: Proceedings of the 34th Euromicro Conference. Euromicro SEAA, Parma, Italy, IEEE Computer Society (2008) 449–456 ISBN: 978-0-7695-3276-9.