Title: Decision Support for Iteration Scheduling in Agile Environments

Author: Szoke, A.

Author's web: [www.mit.bme.hu/~aszoke; www.kese.hu](http://www.mit.bme.hu/~aszoke; www.kese.hu)

Szoke, A.  
*Decision Support for Iteration Scheduling in Agile Environments*  
Frank Bomarius, Markku Oivo, P. J. P. A. (ed.)  
PROFES '09: Proceedings of the 10th International Conference on Product Focused Software Process Improvement  
Springer-Verlag, **2009**, Vol. 32(ISBN: 978-3-642-02151-0), pp. 156-170

**Abstract**

Today's software business development projects often lay claim to low-risk value to the customers in order to be financed. Emerging agile processes offer shorter investment periods, faster time-to-market and better customer satisfaction. To date, however, in agile environments there is no sound methodological schedule support contrary to the traditional plan-based approaches. To address this situation, we present an agile iteration scheduling method whose usefulness is evaluated with post-mortem simulation. It demonstrates that the method can significantly improve load balancing of resources (cca. 5%), produce higher quality and lower-risk feasible schedule, and provide more informed and established decisions by optimized schedule production. Finally, the paper analyzes bene ts and issues from the use of this method.

**Bibtex**

```
@inproceedings{szoke:dsi09,  
  author = {Akos Szoke},  
  title = {Decision Support for Iteration Scheduling in Agile Environments},  
  booktitle = {PROFES '09: Proceedings of the 10th International Conference on Product Focused Software Process Improvement},  
  publisher = {Springer-Verlag},  
  year = {2009},  
  volume = {32},  
  number = {ISBN: 978-3-642-02151-0},  
  pages = {156--170},  
  note = {June 15-17, 2009http://www.informatik.uni-trier.de/~ley/db/conf/profes/profes2009.html},  
  url = {http://www.springerlink.com/content/w351600v46541115/},  
  doi = {http://dx.doi.org/10.1007/978-3-642-02152-7_13}  
}
```

*The original publication is available at the publisher's URL*

DOI
Decision Support for Iteration Scheduling in Agile Environments

Ákos Szőke

Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary
aszoke@mit.bme.hu

Abstract. Today’s software business development projects often lay claim to low-risk value to the customers in order to be financed. Emerging agile processes offer shorter investment periods, faster time-to-market and better customer satisfaction. To date, however, in agile environments there is no sound methodological schedule support contrary to the traditional plan-based approaches. To address this situation, we present an agile iteration scheduling method whose usefulness is evaluated with post-mortem simulation. It demonstrates that the method can significantly improve load balancing of resources (cca. 5×), produce higher quality and lower-risk feasible schedule, and provide more informed and established decisions by optimized schedule production. Finally, the paper analyzes benefits and issues from the use of this method.

Key words: agile planning, iteration planning, scheduling.

1 Introduction

Agile software development represents a major approach to software engineering. Recent surveys showed that in the last 10 years agile methods adoption expanded to the ≈ 70%, which can be explained by the fact that agile teams are generally more successful than traditional ones [1][2]. The most popular agile methods are Extreme Programming (XP) (58%), Scrum(23%), and Feature Driven Development (FDD) (5%) [3]. Several studies pointed out that Extreme programming provides ≈ 60% increase in productivity, quality and improved stakeholder satisfaction, and ≈ 60% and ≈ 40% reduction in products pre-, and post-release defect rates respectively [4].

Despite variety of methods all of them share the common principles specified in the Agile Manifesto [5]. The Declaration of Interdependence (DOI) [6] defines a set of management principles for agile methods [6]. The main practices respect to agile project planning includes i) Continuous improvement ii) Iterative development iii) Staged program delivery iv) Scenario-driven development, and v) Business-driven project pipeline [5][6][7][8].

From the project management point of view, agile software development delivery process is made up of the following phases: 1) conceptualization to define vision, high-level ranked deliverables and project roadmap, 2) release planning
to estimate deliverables and assign them into releases, 3) *iteration planning* to break down selected deliverables into technical tasks, 4) *iteration* to discuss the daily progress concerning writing tests, codes and fixing defects, 5) *iteration review* to demonstrate product increments to stakeholders and conduct iteration retrospective for the next iteration, and finally 6) *release* to package and deploy software to customers [3, 9] (see Fig. 1).

Figure 1 points out that planning functions are generally described by a three-level management hierarchy in agile environments: *release* (coarse-grained), *iteration* (fine-grained), and *daily plans*. Each planning level is responsible for realizing the objectives of both the given and its superior level [3, 9].

**Problem Statement and Analysis.** In 2008, an Agile Tools survey [10] showed that many developers-focused tools were come out (including JUnit testing, sub versioning, auto build, etc.) in the last decade, but most companies (> 52%) are still using old-fashioned project management tools like MS Project [11] or generic tools like spreadsheets. Surprisingly 18% of the respondents do not use any tool for project planning and tracking at all – although many commercial (such as Rally [12]) and open source (e.g. XPlanner [13]) agile project planning tools are available.

The lack of penetration of the modern agile planning tools can be explained by the weak embedded support of traditionally important project scheduling functions such as resource allocations and what-if analysis. Their implemented methods provide ‘quick and dirty’ scheduling solutions [12, 13]: the team can distribute deliverables among releases and iterations in planning meetings – while all explicit and implicit objectives and constraints are taken into account informally. Typical constraints and objectives are P1) *precedences* (to express temporal precedences between realizations), P2) *balancing resource workloads* (to avoid resources overloading), and P3) *optimality* (to choose the best one from different plans). Informal approaches work well in smaller projects, however as the size and complexity increases scheduling becomes a very complex process and advocates tool support [14, 15].

**Related Work.** Scheduling requirements for the upcoming version is complex decision-centric process [15]. Its complexity emanates from increasing market demand and extensive use of high technology while all explicit and implicit objectives and constraints must be taken into account. In order to deal with this decision problem some method have been proposed. Compared to the extensive
research on requirements priorization [16], interdependencies [17, 18], and estimation [19], only few researches investigated the release planning problem. In the early period, researchers focused on the method of assessing requirements value and estimating cost to prioritize requirements [20, 16]. Later, several optimization methods were proposed to select requirements for the next release. In [17] 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 and some managerial steering mechanism that enabled what-if analysis in [14, 21]. The IFM method provides insight into the impact of development decisions with a financially-informed approach to maximize Net Present Value [22]. In [23] a case study showed that integration of requirements and planning how significantly can accelerate UML-based release planning.

All previous methods relate to requirements priorization and selection and none of them bothers with the implementation aspect: how to realize the selected requirements.

Objectives. Our proposed method intends to provide a sound decision support to the P1-P3) by constructing an information model to specify data semantics of agile planning, and an innovative heuristic scheduling algorithm for wide-ranging agile iteration scheduling problems. This method not only supports making delivery decisions even in complex situations but provides a ‘quick and clean’ solution for agile iteration scheduling.

Structure of the Paper. The rest of the paper arranged as follows: Sec. 2 presents the background information on agile planning; Sec. 3 details the information model and the scheduling algorithm; Sec. 4 introduces simulation experiment with our prototypic tools; Sec. 5 discusses our solution and findings; Sec. 6 offers a survey of related work; and finally Sec. 7 concludes the paper.

2 Background

In this section, first we introduce agile release, iteration and daily planning practices to provide the necessary background information for the proposed method.

2.1 Release Planning

Release planning is a fundamental part of any incremental software development process (ISDP). It deals with assigning requirements to releases of evolving software products. Two kinds of release planning are adhered to ISDP: predictive and adaptive planning [24]. Predictive planning produces a detailed plan covering the whole software life cycle. On the contrary, adaptive planning includes two plans: a coarse-grained long-time (release) and a fine-grained short-time (iteration) plan. In the present perpetually changing environment the overall goal of
ISDP is to maximize stakeholders’ satisfaction in least time possible, so adaptive planning is more suitable for ISDP [24, 6, 9].

In agile methods a deliverable system is decomposed into units of customer-valued functionalities, and they are defined as self-contained features [9]. A proper release plan should satisfy customer needs while provide maximal business value by selecting the right set of features (requirements and defect corrections) into the next release(es). Feature selection considers the demands of stakeholders – including users, managers, developers, or their representatives. As a consequence, it is often not obvious which choice is better, because several concurrent aspects must be taken into account. The simplest forms of release planning are done informally and one of the most well known is the Planning game [9, 3]. Sophisticated methods include optimization-based prioritization mechanism while considering different constraint (e.g. technological, resource, system) and optimality criterions (e.g. value, urgency) [14, 15].

2.2 Iteration and Daily Planning

Once the maximal customer-valued features are selected for the next release the following step is to realize them. In agile approaches software is rolled out in increments over time with iterative development approach (c.f. Sec.1 ii, iii) to reduce overall risk of realization [9, 3]. Therefore a release is made up of several iterations (from 1 to 4 and with duration 1 or 2 weeks) which deliver intermediate features (i.e. technical tasks – c.f. Sec.1) to the customers, so they receive both a sense of value and an opportunity to provide early feedback.

Iteration schedule is operational level support for realization of technical tasks, it focuses on resource allocation to these tasks [9]. In traditional approaches scheduling is usually carried out by a project planner software package (e.g. MS Project [11]) that helps dealing with constraints (e.g. scarcity of resources and precedences between features) and objectives (e.g minimal execution time) – but it is constructed mainly manually and takes relatively long time (several hours). However it is too heavyweight for agile approaches since they promise rapid response to the given situation – even on a daily basis (c.f. Sec.1 i, ii, v). Instead, without adequate tool support in agile methodologies, iteration scheduling is based on intuitive human judgements whose inherent discrepancies are resolved during team’s daily and iteration review meetings (see Fig. [1, 9, 24].

3 Decision Support in Iteration Scheduling

In this section first, we construct an information model of agile planning by representing concepts, relations, constraints to specify data semantics for agile iteration scheduling (subset of agile planning). Then we point out that iteration planning problems can be characterized as a special kind of resource-constrained project scheduling problem (RCPSP). Finally, a prototypic tool, and the analysis of our proposed solution is presented.
Iteration scheduling process made up of the following major steps: 1) features are broken down into smaller parts i.e. technical tasks (each task is realized by one developer); 2) durations of tasks are estimated and precedences among them are identified (they affect the realization time and the sequencing of tasks); and finally 3) resources allocation to tasks are performed. Output of this process is an schedule: what task is realized by who and when (see Fig. 2).

3.1 Conceptual Model of Agile Planning

In order to formulate the iteration scheduling model, first, we have to identify the main concepts of agile planning. These concepts are presented in the following list and visualized with UML notation in Fig. 3 [25, 9, 26].

- **Project**: is a planned endeavor, usually with specific requirements and rolled out in several deliverable stages i.e. releases.
- **Release**: produces (usually external) selected deliverable features for the customer, contains 1-4 iterations with start/end date and an iteration count.
- **Iteration**: is a development timebox that delivers intermediate deliverables with the realization of several technical tasks. It is characterized by available resource capacity – often expressed by iteration velocity.
- **Resource**: is human manpower who accomplish the demanded feature for the customer and they are allocated to releases.
- **Feature**: deliverables that the customer values. They can be classified two kind of set of elements: i) (new/change) requirements (functional and non-functional), and ii) defect repairs (fixed defects in former product variants).
- **Technical task**: fundamental working unit accomplished by one developer. In most cases requirements mandates several realization steps that requires cooperation of some developers. Proper coordination requires individually realizable working units thus each requirement and defect repairs should be broken down into several technical tasks. Technical tasks usually requires some working hour (Wh) realization effort that is estimated by developers.
- **Precedence**: realization precedences between features. Precedences emanate from the following sources ([j′, j denotes technical tasks]) [15, 17]: i) functional implication (j demands j′ to function), ii) cost-based dependency (j′ influences the implementation cost of j, so useful to realize j′ earlier),

![Fig. 2. Iteration Scheduling Process Overview](image-url)
iii) *time-related dependency* (expresses technological/organizational demands).

These concepts not only help to identify the objects and the subject of the optimization model but with the precise relationships it can also be used as database schema definition for an agile planning and scheduling application.

### 3.2 Mapping Iteration Scheduling to RCPSP

In the following analogy between iteration planning and resource-constrained project scheduling optimization problem (RCPSP) is presented \[27\]. Generally, scheduling concerns the allocation of limited resources (manpower) to tasks over time in order to fulfill the predefined scheduling objective. In fact, many different objectives are possible – depending on the goals of the decision makers – but our aim is to ‘maximize stakeholders’ satisfaction in least time possible’ (Sec. 2.1), thus the *makespan minimization* (i.e. finding the minimum execution time) is the most adequate. As agile methods recommend collaborative teamwork – without any development role (such as analyst, programmer, tester) – we only identify one kind of resource: the *developer*. The complexity of scheduling arises from the interaction between tasks by *implicit* and *explicit* dependencies. While the previous is given by scarcity of resources, the latter is emerged from different precedences (Sec. 3.1) between tasks that define the routing of the tasks \[17\].

To provide suitable scheduling method for wide-ranging iteration scheduling situations, we extended the ordinary RCPS problem with i) *pre-assignments* (i.e. assigning certain tasks to resources before scheduling) and ii) *timeboxed* iteration duration control. On the one hand, defect corrections and onward development of a formerly delivered functionalities legitimates *pre-assignments*, on the other hand, *timeboxed* iteration execution mandates an upper boundary control in time – which is not allowed to be exceeded otherwise schedule is treated infeasible.

### 3.3 Formulating RCPSP Model

Let $\mathcal{R}$ be the set of resources $i$ and the following typical properties for scheduling be interpreted on technical tasks to schedule them (i.e. $j \in \mathcal{A}$) \[27\]:

**Effort** : $d_j$ – time estimation (in hours) is associated with each task. It is calculated by simple expert estimation (e.g. 2, 4, or 8 working hour (Wh))).
Pre-assignment: In some cases resource pre-assignment is applied before scheduling. It is used by the scheduler algorithm during resource allocation.

Let the vector $S = (S_0, S_1, ..., S_{n+1})$ be start times for tasks’ realizations – where $S_j \geq 0 : j \in A$ and $S_0 = 0$. The vector $S$ is called a schedule of development. In this definition the 0 and $n+1$ are auxiliary elements to represent iteration beginning and termination, respectively.

Temporal and Resource Constraints. Dependencies (temporal constraints – c.f. Sec. 3.1) can be defined by precedence relations (Eq. 1):

$$S_j - S_{j'} + d_{j'} \geq P_{j', j} : j', j \in A$$

(1)

Let the $R_i \in \mathbb{N}$ is a set of capacities of resources that have been assigned to the project. The effort estimation yields resource requirements $r_{j,i} \in \mathbb{Z}$ for each task $j$ and each resource $i$. Now let $S$ be some schedule and let $t$ be some point in time. Then let $A(S, t) \triangleq \{ j \in A | S_j \leq t \leq S_j + d_j \}$ be the active set of tasks being in progress at time $t$. The corresponding requirement for resource $i \in \mathcal{R}$ at time $t$ is given by $r_i (S, t) \triangleq \sum_{j \in A(S, t)} r_{j,i}$. As a consequence, the resource constraints can be treated as follows (Eq. 2):

$$r_i (S, t) \leq R_i : i \in \mathcal{R}$$

(2)

Optimization Model. With the application of previous elements, RCPSP for iteration scheduling can be formulated as follows:

Minimize $z = S_{n+1}$

subject to

$$S_j - S_{j'} + d_{j'} \geq P_{j', j} : j, j' \in A$$

(3b)

$$r_i (S, t) \leq R_i : i \in \mathcal{R}$$

(3c)

$$S_{n+1} \leq c$$

(3d)

where Eq. 3b, 3c are scheduling constraints (c.f. Eq. 1, 2), Eq. 3d is the timebox duration, and Eq. 3a is the makespan minimization objective.

3.4 Solving Iteration Scheduling

For the previous optimization model we developed an innovative scheduling algorithm. It is a constructive heuristic algorithm, which iteratively selects and assigns technical tasks to resources. In the program listing (Algorithm 1) lowercase/uppercase letters with indices denote vectors/matrices (e.g. $r_i, P_{j', j}$). While bold-faced types show concise (without indices) forms (e.g. $\mathbf{P}$).

In the require section the preconditions are given. The vector $\mathbf{r}$ indicates the available resources (developers) in the iteration. Each $d_j$ is the planned effort (duration) for technical task $j$ – both development and defect correction. Every element of vector $\mathbf{a}_j$ contains a reference to a resource index ($a_j \in \{1..|\mathbf{r}|\}$) which
Algorithm 1 List scheduling algorithm with AF strategy

Require:
\[ r_i \in N, c \in N \] /* resources and iteration duration */
\[ a_j \in N : a_j \in \{1, \ldots, |r|\}, d_j \in N \] /* pre-assignments and duration of tasks */
\[ P_{j,j'} \in \{0, 1\} \land P_j = 0 \land P \text{ is DAG} \] /* precedences */

Ensure:
\[ S_{i,j} \in \{0, 1\} \land \forall j \exists! i \text{ } S_{i,j} = 1 \]

1: \[ m \leftarrow \text{length}(r), n \leftarrow \text{length}(d) \] /* number of resources and tasks */
2: \[ S \leftarrow [0]_{m \times n} \] /* assignment matrix initialization */
3: \[ rlist \leftarrow \emptyset, slist \leftarrow \emptyset \] /* 'ready list' and 'scheduled list' initialization */
4: for \( f j = 0 \) to \( n \) do
5: \[ \text{pot} \leftarrow \text{findNotPrecedentedTasks}(P) \] /* find potentially tasks */
6: \[ \text{rlist} \leftarrow \text{pot} \setminus \text{slist} \] /* construct ready list */
7: if \( \text{rlist} \) \( = \) \( \emptyset \) then
8: \[ \text{return} \emptyset \] /* No schedulable task */
9: end if
10: \[ j \leftarrow \max \{a_j : j \in \text{rlist}\} \] /* select a task using AF strategy */
11: if \( a_j = 0 \) then
12: \[ i \leftarrow \text{selectMinLoadedResource}(S) \] /* without assignment */
13: else
14: \[ i \leftarrow a_j \] /* with assignment */
15: end if
16: \[ l \leftarrow \text{sum}(S_{i,\{1,\ldots,n\}}) \] /* calculate load of resource \( i \) */
17: if \( (l + d_j) > c \) then
18: \[ \text{return} \emptyset \] /* Overloaded iteration */
19: end if
20: \[ p \leftarrow \text{findNextPos}(S, i) \] /* index for next task */
21: \[ S_{i,p} \leftarrow j \] /* assign task \( j \) to resource \( i \) at position \( p \) */
22: \[ \text{slist} \leftarrow \text{slist} \cup \{j\} \] /* add task \( j \) to 'scheduled list' */
23: \[ P_{\{1,\ldots,n\},j} = 0 \] /* delete precedence related to scheduled task */
24: end for
25: \[ \text{return} S \]

indicates resource pre-assignment to task \( j \). The \( a_j = 0 \) means that task \( j \) is not pre-assigned, thus the algorithm will find the best resource to its realization. Precedences between tasks (c.f. Eq. 31) can be represented by a precedence matrix where \( P_{j,j'} = 1 \) means that task \( j \) precedes task \( j' \), 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. Iteration timebox is asserted by variable \( c \). It is used as an upperbound in resource allocation to prevent resources overloading. The result of the algorithm is a schedule matrix \( S \), where rows represent resources, and columns give an order of task execution. Thus \( S_{i,p} = j \) means that task \( j \) is assigned to resource \( i \) at position \( p \). The ensure section prescribes the postcondition on the return value (\( S \)): every task \( j \) has to be assigned to exactly one resource \( i \).
During scheduling steps, first the initial values are set (line 1 – 3). The iteration value (n) is equal to the number of technical tasks (line 1). The algorithm uses a ready list (rlist) and a scheduled list (slist) to keep track of schedulable and scheduled tasks. Potentially schedulable tasks (pot) are unscheduled tasks from which the algorithm can choose in the current control step without violating any precedence constraint (line 5). Previously assigned tasks are subtracted from pot to form the ready list (line 6). As long as the ready list contains schedulable tasks, the algorithm chooses tasks from it – otherwise the schedule is infeasible (line 7) and as a consequence the algorithm aborts (line 8).

To select the next task to schedule from concurrently schedulable tasks (i.e. ready list) we constructed the custom ‘Assigned First’ scheduling rule (line 10) (c.f. Sec. 3.2). This rule chooses from the pre-assigned tasks (a_j > 0) before the unassigned ones (a_j = 0). As the selection sequence is discretionary we applied the max function to the choice. After selection the minimal loaded (min summa duration) resource is allocated to the selected task unless the task is pre-assigned to a given resource (line 11–15) (c.f. Sec. 3.2). If the load of the resource i exceeds iteration timebox (c) then the schedule is treated infeasible (line 16 – 19).

The following step is to find the index of next task position (p) (right after the previous task’s index) at resource i (line 20) for task j for assignment (line 21). Finally, scheduled list (slist), is updated with scheduled task (lines 22), and no longer valid precedence relations are also deleted from P (lines 23). Iteration proceeds until all items are assigned to iterations (line 4 – 24). After termination, S contains the task assignments to resources and the makespan is

\[ z \leq \max_{i=1, m} \sum_{p=1}^{n} d_{S_{i,p}} \]  

c.f. Eq. 3a).

Solution Analysis. This greedy strategy makes a series of local decisions, selecting at each point the best step without backtracking or lookahead. Thus local decisions miss the global optimal solution, but produce quick (time complexity is clearly \( O(n + m) \)) and usually sufficient results for practical applications.

Figure 4 illustrates the application of the algorithm on real application development data which was extracted from the backlog of IRIS at Multilogic [28]. The figure shows post-mortem scheduling result of an iteration – visualized by resource aspect Gantt diagram – where tasks’ realizations plotted against time. The diagram points out that 94 tasks (with 2, 4, and 8 working hours (Wh)) are allocated to 6 resources, and the makespan is 78.

3.5 Tool Support

Previously presented theoretical foundation is realized by our MS Sharepoint-based website at Multilogic and our scheduling toolbox on the Matlab platform [29, 28, 30]. Sharepoint is browser-based collaboration and a document-management platform, and its capability includes creating different lists (as database tables) – such as list of technical tasks and resources. The previously constructed agile planning information model (see Fig. 3) were implemented as Sharepoint lists. Thus, the portal was targeted as a collaborative workspace for
developers, and a tool for the management to collect all planning information. With this web-based tool, developers can break-down requirements into technical tasks, indicate precedences, set effort estimation, status of tasks/defect corrections and they also can share these information to facilitate communication. Additionally, we have implemented the presented algorithm in Matlab to support iteration decisions based on data collected through the Sharepoint site.

4 Experiments

To evaluate our proposed scheduling method simulations were carried out. Applying the historical iteration planning data, as an input for the scheduling algorithm, made it possible to compare them [31]. The four past data sets extracted from the backlog of IRIS application that is developed by Multilogic Ltd [28].

In this section, first we set research questions, then present necessary background information, and finally we present and interpret our findings.

4.1 Research Questions

Our initial intend (see Sec. 1 P1-3) was to support decisions in agile iteration scheduling in the following aspects: 1) dealing with precedences, 2) tracking workloads, and 3) providing optimal (makespan minimized) delivery plan.

To validate our proposed method the next questions were addressed: How does optimization-based iteration scheduling compare with informal one in terms of Q1) resource workload over time, Q2) quality and Q3) feasibility of the plans.

4.2 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 planning process were made up of the following steps. First, during release planning, the requirements were selected (expressed in User stories) from
the backlog – considering stakeholders’ demands. Then every User story was estimated by the team and distributed into iterations taking resources, precedences and iteration timebox into account. Second, during iteration planning, each User Story was broken down into technical tasks and important defect corrections were also selected to the next product increment. Finally, resource allocation was determined intuitively by the team in intuitive way and the conflicts (precedences, resource overload) were managed during daily meetings (see Fig. 1).

4.3 Data Collection and Results

Four data sets (four iterations \(I_A^1, I_A^2, I_B^1, I_B^2\)) of two releases \((R_A, R_B)\) were selected to make a comparison between the algorithmic and the intuitive method. All iterations had same project members (6 developers /Dev./), iteration length (80 working hours (2 weeks) /IL/), domain, customer, and development methodology, but they were characterized by different number of technical tasks (development /DT/ and defect correction /CT/ with 2, 4 and 8Wh), User Stories /US/, precedences /Prec./, and pre-assignments /Ass./. Table 1 summarizes state variables that were used to capture facts that were likely affect the findings. These variables were collected from the SharePoint-based backlog.

<table>
<thead>
<tr>
<th></th>
<th>Dev</th>
<th>IL</th>
<th>US</th>
<th>DT</th>
<th>CT</th>
<th>TT = DT + CT</th>
<th>Prec.</th>
<th>Ass.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_A^1)</td>
<td>6</td>
<td>80 Wh</td>
<td>28</td>
<td>91(25,34,32)</td>
<td>3(2,1,0)</td>
<td>94(27,35,32)</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>(I_A^2)</td>
<td>6</td>
<td>80 Wh</td>
<td>35</td>
<td>89(16,46,27)</td>
<td>2(0,2,0)</td>
<td>91(16,48,27)</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>(I_B^1)</td>
<td>6</td>
<td>80 Wh</td>
<td>33</td>
<td>84(29,24,31)</td>
<td>5(2,1,2)</td>
<td>89(31,25,33)</td>
<td>15</td>
<td>22</td>
</tr>
<tr>
<td>(I_B^2)</td>
<td>6</td>
<td>80 Wh</td>
<td>34.5</td>
<td>79(13,31,35)</td>
<td>7(4,2,1)</td>
<td>86(17,33,36)</td>
<td>4</td>
<td>16</td>
</tr>
</tbody>
</table>

We constructed Task effort \((TE_i)\) response variables to test Q1. This simple variable is computed by adding up estimated tasks’ efforts that were assigned to resources \(i\). Explanations of Q2 and Q3 were produced with the utilization of the solution’s inherent properties.

4.4 Analysis

To answer to the questions Q1-3 simulations were performed on the previously described input data to compare the characteristics of the two approaches. The simulation output is summarized in Table 2.

On the left the four historical iteration schedules are presented \((I_A^1, I_A^2, I_B^1\) and \(I_B^2\)). In the table \(D_i\)s denote resources (developers); 2, 4, and 8 values are estimated effort (instead of indeces) of task realizations; and finally the previously introduced response variable \((TE_i)\) can be seen. On the right column simulation results \((*I_A^1, *I_A^2, *I_B^1,\) and \(*I_B^2)\) are presented.
To compare the intuitive and the algorithmic cases quantitative (statistical) analysis were performed on the two response variables (*$TE_1$ and $TE_2$). The result is presented in Table 3 and summarized in boxplot (see Fig. 5).

From these, we conclude that optimized case i) did not exceed the time-box limit (*Max = 78W; < 80W; < Max = 102) which means lower level scheduling risk; ii) has less dispersion in total task allocation (*Std.dev = 3 vs. Std.dev = 14.6); iii) yields more balanced workload on resources – while the means are similar (*Mean = 74 ≈ Mean = 75). As a consequence, in terms of coefficient variation (i.e. normalized measure of dispersion), the optimization-based scheduling provides $c_v/\bar{c}_v = \frac{Std.dev}{Mean}/\frac{Std.dev}{\bar{Mean}} = \frac{0.1976}{0.0410} \approx 5$ times more balanced resource workload over time contrary to the intuitive method (cf. Q1).

The algorithmic method easily resolves complex decision situation – as it handles precedences between tasks and avoids resource workloads – contrary to the intuitive case where these are managed intuitively during daily meeting. As
a consequence these two capabilities of the algorithmic method ensure higher quality and lower-risk feasible plans in contrast to the intuitive case (c.f. Q2-3).

5 Discussion

First we constructed a general agile planning information model – including both releases and iterations – that helped us to identify the objects and the subject of our proposed optimization model. Its precise relationships can also be used as database schema definition for an agile planning and scheduling application such as our Sharepoint-based prototypic tool for collaborative data collection.

Then we formulated iteration scheduling model as a special case of RCPS problem to provide decision support in feature implementation sequencing. The formulated model considers temporal constraints (Sec. 3.1, 3.3), team’s resources, and defines makespan minimization scheduling objective. As a matter of fact many different objectives are possible – depending on the goals of the decision makers – but in our scheduling case (‘maximize stakeholders’ satisfaction in least time possible’) the makespan minimization is the most adequate. This interpretation of iteration schedule makes it possible to adapt extremely successful heuristic algorithms applied for solving RCPSP. To provide suitable scheduling method for wide-ranging iteration scheduling situations we extended the ordinary RCPS problem with i) pre-assignments (i.e. assigning certain tasks to resources before scheduling) and ii) timeboxed iteration duration control.

Generally, RCPS problems are combinatorial NP-hard problems and a variety approximation algorithms are proposed. The most popular heuristics in approximation algorithms are SPT or LTP (Shortest/Longest Processing Time first) [27]. However, we constructed and applied our AF assigned task first scheduling rule demanded by pre-assignments (defect corrections and onward development of a formerly delivered functionalities). Our proposed combinatorial algorithm is capable to provide acceptable results with good time complexity (O(n + m)) for practical applications.

This approach gives the business increased visibility, and it can also provide constantly up-to-date schedule decision support considering changes necessitated by shifting business priorities. Moreover, the decision maker can accommodate quick what-if scenarios and replanning on-the-fly. However, as our simulation carried out post mortem analysis, examination of the method is recommended in real development cases in order to investigate it in dynamical situations.
6 Conclusions

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 release-centered decisions [13]. In agile environments, which recommends small and iterative software releases, the decision is even more difficult due to the perpetual changes in requirements, constraints and objectives.

To address this situation, we have presented a method including an information model to specify data semantics for agile planning, and an innovative heuristic scheduling algorithm for wide-ranging agile iteration scheduling problems. To evaluate our method four simulations were carried out that demonstrated how the method could 1) significantly improve load balancing of resources (cca. 5×), 2) produce higher quality and lower-risk feasible schedule, and 3) provide more informed and established decisions to agile teams.

We think that our proposed method is a plain combination of the present theories and methods, thus it lead us to generalize our findings beyond the result of the simulations.

Acknowledgements. The development is supported in part by the GVOP grant (GVOP-3.3.3-05/1.-2005-05-0046/3.0) and realized by the Multilogic Ltd [28].

References

5. Manifesto for agile software development, http://www.agilemanifesto.org
6. Declaration of interdependence for agile software project management, http://pmdoi.org