Sprint Planning Optimization in Agile Data Warehouse Design

Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia

DEIS - Univ. of Bologna,
V.le Risorgimento 2, 40136 Bologna, Italy
{matteo.golfarelli,stefano.rizzi,elisa.turricchia2}@unibo.it

Abstract. Agile methods have been increasingly adopted to make data warehouse design faster and nimbler. They divide a data warehouse project into sprints (iterations), and include a sprint planning phase that is critical to ensure the project success. Several factors impact on the optimality of a sprint plan, e.g., the estimated complexity, business value, and affinity of the elemental functionalities included in each sprint, which makes the planning problem difficult. In this paper we formalize the planning problem and propose an optimization model that, given the estimates made by the project team and a set of development constraints, produces an optimal sprint plan that maximizes the business value perceived by users. The planning problem is converted into a multi-knapsack problem with constraints, given a linear programming formulation, and solved using the IBM ILOG CPLEX Optimizer. Finally, the proposed approach is validated through effectiveness and efficiency tests.

Keywords: Agile methods, Optimization, Data warehouse design.

1 Introduction

As empirical studies suggest [9,2], agility is one of the most promising directions to overcome the problems of traditional software engineering approaches. The twelve principles stated in the Agile Manifesto [3] are followed by several agile methods, such as *Scrum* and *eXtreme Programming*, that have been adopted by an increasing number of companies to make the software development process faster and nimbler. Agile principles also find large application for designing data warehouses (DWs), that are characterized by a particularly long and expensive development process, so some agile approaches to DW design have been devised in recent years [13,11].

A key practice shared by all agile methods is incremental and iterative design and implementation. The DW system is described in terms of detailed user functionalities (*user stories*) [13]; a user story can correspond for instance to a set of correlated reports, a piece of ETL, or a conceptual schema for a fact. At each iteration (*sprint* in the Scrum terminology), the team should deliver the set of user stories that maximizes the utility for the users and fulfills a set of development constraints [22]; typical constraints include limiting the duration of

A. Cuzzocrea and U. Dayal (Eds.): DaWaK 2012, LNCS 7448, pp. 30–41, 2012.

an iteration, respecting dependencies and correlations among user stories (e.g., logical design must follow conceptual design), reducing the non-delivery risk.

Clearly, the sprint planning phase is critical to ensure the project success. User story prioritization and definition of sprint boundaries are obtained by sharing and averaging the estimates given by the different team members about story complexity, utility, and dependencies. For example, advancing high-valued stories (e.g., those that implement critical analysis reports) could lead to an early significant result for users; similarly, risky user stories (e.g., those that implement cleaning procedures for very dirty data) can be advanced to avoid late side-effects, but at the price of a higher probability of delays in the initial stages. The success of a sprint planning phase mainly depends on the accuracy of estimates and on the capability of properly taking several variables and constraints into account. While the first issue is mainly related to the team experience, the second one can be formulated as an optimization problem whose complexity increases with the project size. Clearly, a non-optimal solution to this problem leads to inefficiencies that easily turn into extra-costs and project delays.

Though some commercial tools support agile project management for generic software systems [23,8], they do not provide any support to optimal sprint planning. In this direction, in this paper we formalize the sprint planning problem for DW projects and propose an optimization model that, given the team estimates and a set of development constraints, produces an optimal sprint plan that maximizes the business value perceived by users. Remarkably, though our approach fully complies with the agile principles that give the team experience and knowledge a key role in delivering an effective plan, it also relieves the team from the difficult task of quickly producing an optimal plan. The optimization problem is formalized as a multi-knapsack problem with constraints, and is solved using the IBM ILOG CPLEX Optimizer [14].

The paper is organized as follows. Section 2 reviews the related literature; Section 3 summarizes the key agile practices in DW design; Section 4 formalizes the sprint planning problem and proposes an optimization model that takes agile principles into account; Section 5 presents a set of tests on both synthetic and real projects to prove efficiency and effectiveness of our approach; Section 6 draws the conclusions and sketches our future work.

2 Related Literature

A pioneering work in the field of agile DW design is [13], that breaks with strictly sequential approaches by applying two Agile development techniques, namely *scrum* and *eXtreme Programming*, to the specific challenges of DW projects. To better meet user needs, the work suggests to adopt a *user stories decomposition* step based on a set of architectural categories for the back-end and front-end portions of a DW. In [11] the potential advantages arising from the application of modern software engineering methodologies to a DW project are analyzed, and a design methodology called *Four-Wheel-Drive* (4WD) is proposed. 4WD aims at making the DW design process more reliable, robust, productive, and

timely; to this end it adopts six key principles (incrementality, prototyping, user involvement, component reuse, light documentation, and automated schema transformation), most of which perfectly fit the agile paradigm.

More generally, in the software engineering field several approaches inspired by agile principles have been proposed [2]. In [9], the authors propose a systematic review and comparison of different agile methods, focusing on both organizational and technical features. They also emphasize the increasing penetration of Scrum and Extreme Programming practices in industries. The Scrum approach is deeply discussed in [22], where its key ideas and its life-cycle are described. A more pragmatic work is presented in [7], that focuses on user stories and gives practical hints for estimating their complexity and business value.

As to tools for agile project management, a few solutions are available. AgileFant [1] offers a set of basic functionalities to monitor the progress of project iterations. *Mingle* [23] and *ScrumWorks* [8] provide a more complete set of agile parameters to deal with user story risk, complexity, and business value. However, all these solutions lack in providing an automated solution to the sprint optimization problem; similarly, to the best of our knowledge, no research prototypes have been developed to this purpose.

In the broader context of project scheduling, several models and algorithms have been proposed in the literature (see [15] for a survey). According to the classifications proposed in [12] and [5], our problem is categorized as resource-constrained with renewable resources (i.e., manpower) available on a period-by-period basis. As in the basic PERT/CPM model, finish-start precedences with zero time lag are considered and no preemption of activities is allowed.

The project scheduling literature provides no model that perfectly fits the problem discussed here, and this is where operational research comes into play. Sprint optimization can be formulated as a multi-knapsack problem [17], where sprints are the knapsacks, while user stories are the items. The sprint optimization problem is made original by its objective function and the way how affinity and risk affect the solution. Though the multi-knapsack problem is NP-hard [10], branch-and-bound techniques can efficiently compute exact solutions for medium-sized instances. For large problems, heuristic methods can be used to find approximate results.

3 A Summary of Agile Data Warehouse Design Practices

The success of a DW project is directly related to customer satisfaction, so agile methods strive to better comply with user requests. In particular, agile principles aim at reducing the delivery time and making the development process more flexible; indeed, accelerating the time-to-market leads to overcoming the business pressure, while flexibility ensures fast reactions to both technology evolution and user requirement changes. To achieve these goals, agile methods propose several complemental practices:

– *Incremental process*: The DW system is broken up into smaller portions which are scheduled, developed, and integrated when completed; each

portion represents an increment in business functionality, that users can validate. For instance, 4WD is based on nested iteration cycles: a *data mart cycle* that defines and maintains the global plan for the development of the whole DW and, at each iteration, designs and releases one data mart; a *fact cycle* that refines the data mart plan and incrementally designs and releases its facts; a *modeling* and an *implementation* cycle that include the activities for delivering reports and applications concerning a single fact [11].

- *Iteration*: The DW system is built in iterations, where each cycle expands the product until the project is completed. Since the process is also incremental, each iteration includes analysis, design, coding, and testing. Noticeably, a stepwise refinement based on short iterations increases the quality of projects by supporting rapid feedback and quick deliveries [4,18].
- *User involvement*: Analysis specifications are difficult to be understood during the preliminary life-cycle phases. Continuous interaction with users is promoted to progressively refine the specifications, reduce inadequate requirements, and increase the trust between users and developers. In more general terms, a user-centered design increases customer satisfaction [11].
- *Continuous and automated testing*: To facilitate requirement validation and obtain better results, a DW is developed by refining and expanding an evolutionary prototype that progressively integrates the implementation of each increment [20]. Unit tests are written for each release of the prototype and automated tests are used whenever possible to accelerate error detection.
- *Lean documentation*: A well-defined documentation is a key feature to comply with user requirements. Small and simple formal schemata are preferred to extensive specifications; thanks to continuous user involvement, up-to-date and clear documentation can be achieved [16,21].

Figure 1 shows the general life-cycle of an agile DW project. Depending on the specific methodology adopted, the macro-analysis returns a high-level description of the requirements in terms of facts to be designed, functional areas to be covered, or analysis applications to be developed. The project team and the users break these requirements into user stories and assign a utility and a development complexity (measured in terms of *story points*) to each of them. Typical examples of user stories include: one or more forms for manual input of data to ETL; a report or a group of related reports; the conceptual schema of a fact or a conformed hierarchy; an ETL unit; the glossary for a functional area; a security profile.

Then, the team assigns a priority to each user story and defines possible correlations (*affinities*) among stories. The resulting list composes the *data warehouse backlog*, that must be partitioned into sprints to produce a *plan*. Sprints should be short and regular enough to guarantee a prompt feedback from users. During each sprint, the team carries out a micro-analysis of the user stories involved, then its members take charge of one or more user stories that are then designed, implemented, and tested. After closing a sprint, the users verify if the stories developed match the requirements they expressed. The approved stories are delivered, while the remaining ones are reinserted in the backlog; noticeably, new requirements may arise at this stage from user feedbacks.

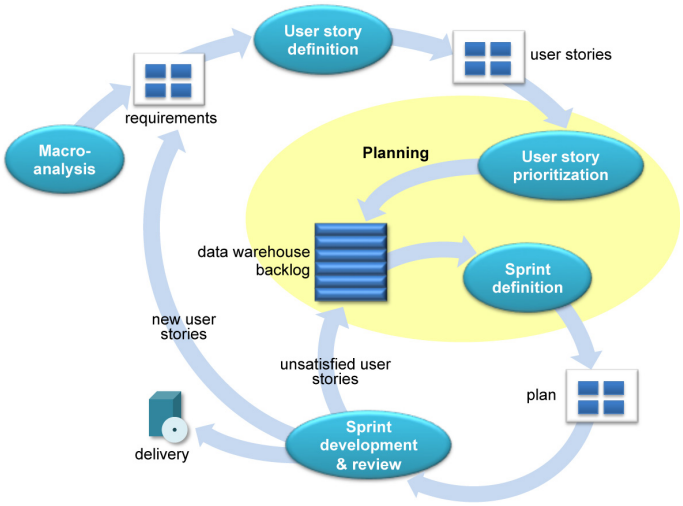


Fig. 1. Agile life-cycle for DW design

4 An Optimization Model

Our formulation of the sprint planning problem takes into account the main variables that affect user stories prioritization and sprint composition. The underlying concepts are:

- *Plan*: a sequence of sprints. All modern DW design methodologies agree on incrementally releasing one data mart at a time, so we will assume the scope of a plan is that of a data mart.
- *Sprint*: the time-bound unit of iteration, typically a one- to four-week period, depending on the project complexity and risk assessment. A sprint includes a set of user stories, and it normally ends with a delivery. A maximum duration is fixed for each sprint by the project team.
- *User story*: a relatively small piece of functionality valuable for users [7]. It represents a light specification that can be later detailed thanks to a continuous communication with the user, but at the same time it must be sufficiently described to estimate its development complexity.
- *Utility*: the business value of a user story as it is perceived by the user that defines it. In general it is quantified through a positive numerical score (typically ranging between 10 and 100 [19]).
- *Story point*: a unit of measurement for the development complexity of user stories. Team members assign story points to each user story based on their experience and knowledge of the domain and project specificities. Story points are non-dimensional and are preferred to time/space measures to avoid subjective and incomparable estimates. Typical complexities of user stories range between 1 and 10 story points [19].

- *Risk*: the risk that the project is not completed as desired. We consider risks related to two different characteristics of user stories: (i) A *critical* story is one that has a strong impact on the other stories, so that taking a wrong solution for it can dramatically affect the success of the project (e.g., the conceptual design of a conformed dimension); (ii) An *uncertain* story is one for which it is somehow hard to estimate the complexity due to unexpected problems that could arise (e.g., changes in the analysis requirements or faulty/incomplete source data). Both types of risk are estimated by positive numbers; here we adopted four classes of risk: 1 (no risk), 1.3 (low risk), 1.7 (medium risk), and 2 (high risk).
- *Affinity*: the degree of correlation between user stories. Similar stories have higher utility if they are included in the same sprint, because users better perceive the overall business value of the functionality delivered. For instance, an "incremental data extraction" story may have low utility on its own, but its utility increases if delivered together with the complemental "incremental data loading" story. The affinity range we adopted is [0, 0.5], meaning that the utility of a story can be increased at most by 50%.
- *Dependence*: a development constraint between two user stories, indicating that a user story (post-condition) cannot start before the other (pre-condition) is completed. Though agile methods discourage user story relationships to improve the project flexibility, some development dependencies must necessarily be preserved (e.g., logical design must follow conceptual design). The *dependency type* of a user story takes value AND (all the pre-condition stories must be completed) or OR (at least one of the pre-condition stories must be completed).
- *Development speed*: the estimated number of story points the team can deliver per day. It is used to convert the sprint duration into the *sprint capacity* (i.e., the maximum number of story points the team can deliver in a sprint).

We can now list the goals an optimal plan should pursue:

♯1 *Customer satisfaction.* It can be obtained by delivering user stories with higher utility first. In the agile philosophy, this also increases the user awareness and trust.
♯2 *Affinity management.* Similar stories should be carried out in the same sprint to increase their value for users.
♯3 *Risk management.* It can be achieved by (i) advancing critical user stories to avoid late side-effects, on the one hand; (ii) distributing uncertain stories in different sprints and postponing them to reduce the risk that the sprint delivery is delayed, on the other hand.

Besides, all constraints related to the sprint capacity and inter-story dependencies must obviously be met.

The problem of determining an optimal plan, i.e., one that achieves these goals, can be converted into a multi-knapsack problem [17], where the knapsacks are the sprints and the items are the stories. Story points measure the weight of an item, while utility represents its value. Knapsack capacity is measured as the story points that the team can deliver given the sprint duration and

the team development speed, i.e., as the sprint capacity. The objective function to be maximized is the cumulative utility of the project (goal ♯1), where the utility of each story is increased if some similar stories are included in the same sprint (goal ♯2) and/or if that story is critical (goal ♯3-i). Finally, in the formulation of the capacity constraint, the story points of user stories are increased by their uncertainty, which discourages the inclusion of two uncertain stories in the same sprint (goal ♯3-ii). The multi-knapsack problem is NP-hard [10]; the linear programming formulation we adopt is shown in the following.

Definition 1 (Sprint Planning Problem). *Given a set of m sprints S and a set of n user stories U, let:*

- x_{ij} *= 1 iff story j is included in sprint i, 0 otherwise;*
- u_j *be the utility of story j;*
- p_j *be the number of story points of story j;*
- p_i^{max} *be the capacity of sprint i, measured in story points;*
- r_j^{cr} *be the criticality risk of story j;*
- r_j^{un} *be the uncertainty risk of story j;*
- a_j *be the affinity of story j;*
- $Y_j \subset U$ *be the set of stories similar (i.e., with some affinity) to story j;*
- y_{ij} *be an accessory variable related to the number of stories in Y_j included in sprint i;*
- $D_j \subset U$ *be the set of stories the story j depends on;*
- $U \supseteq U^{AND} \cup U^{OR}$*, where U^{AND} and U^{OR} are the subsets of stories having dependency type AND and OR, respectively.*

The sprint planning problem consists in determining an optimal assignment of the x_{ij}'s, i.e., in finding which stories compose each sprint in an optimal plan. Its linear programming formulation is as follows:

$$z = Max \sum_{k=1}^{m} \sum_{i=1}^{k} \sum_{j=1}^{n} u_j (r_j^{cr} x_{ij} + a_j \frac{y_{ij}}{|Y_j|}) \tag{1}$$

$$s.t. \sum_{j=1}^{n} p_j r_j^{un} x_{ij} \leq p_i^{max} \qquad \forall i \in S \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall j \in U \tag{3}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j} x_{kz} \geq x_{ij} \qquad \forall i \in S, j \in U^{OR} \tag{4}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j} x_{kz} \geq x_{ij} |D_j| \qquad \forall i \in S, j \in U^{AND} \tag{5}$$

$$y_{ij} \leq \sum_{k \in Y_j} x_{ik} \qquad \forall i \in S, j \in U \tag{6}$$

$$y_{ij} \leq |Y_j| x_{ij} \qquad \forall i \in S, j \in U \tag{7}$$

The explanation of the elements of this formulation is as follows:

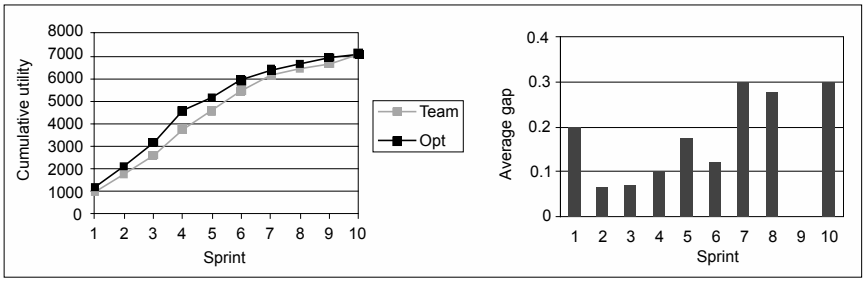
(1) The objective function z states that the optimal plan maximizes the cumulative utility function. The criticality risk r_j^{cr} increases the utility u_j of a critical story j, thus encouraging an early placement of critical stories. Affinity is managed through term $a_j \frac{y_{ij}}{|Y_j|}$, that increases the utility of a story j proportionally to the fraction of similar stories included in sprint i.
(2) These inequalities ensure that the sum of the story points of the stories included in each sprint i does not exceed the sprint capacity p_i^{max}. The story points p_j of story j are increased according to the uncertainty risk r_j^{un} of that story, so as to fairly distribute uncertainty risk among the sprints.
(3) This constraint imposes that each story is included in exactly one sprint.
(4) These inequalities handle OR dependencies by stating that at least one story in D_j is placed before each story j.
(5) These inequalities handle AND dependencies by stating that all stories in D_j are placed before each story j.
(6) These inequalities manage affinity by bounding the number of stories similar to j in sprint i. Using an inequality is necessary to accommodate the fact that, if sprint i includes stories similar to j but j is not part of i, it is $y_{ij} = 0$ (see constraint 7).
(7) These inequalities state that y_{ij} is zero if story j is not part of sprint i, otherwise it cannot be greater than the number of stories similar to it.

CPLEX solves this optimization problem using a branch-and-cut approach [6], that is, a method of combinatorial optimization for solving integer linear programming problems (i.e., linear programming problems where some or all the unknowns are restricted to integer values —the x_{ij}'s and y_{ij}'s in our case). The method is an hybrid of branch-and-bound and cutting plane methods that dramatically improves the performance of classic branch-and-bound methods by incorporating *cutting planes*, that is, inequalities that improve the linear programming relaxation of integer linear programming problems.

5 Model Validation

5.1 Effectiveness Tests

To verify the effectiveness of our model we applied it to a real DW project in the area of pay-tvs, carried out by an Italian system integrator who has successfully been adopting agile methods for five years. The subproject we describe here had an overall duration of 8 months; it included 44 user stories —mostly related to the development of reports, complex ETL units, and forms for manual input of data— and consisted of 10 sprints with an average duration of 17 days. 52 dependencies and just one affinity were involved. The project team included 4 members, but in a few cases one additional programmer was added to support the team.



(a) (b)

Fig. 2. Comparison of cumulative utilities (a) and difference in sprint composition (b) for the optimal and the team plans

The goal of the test presented here is to compare the sprint plan manually defined by the project team with the one generated by our model, using a development speed of 2.43 story points per day. Figure 2.a compares the cumulative utilities of the optimal plan (Opt) and of the plan defined by the team (Team). The curve of the optimal plan is always higher mainly due to a better optimization of sprint composition, but also to a better handling of risk. Indeed, in the team's plan some critical stories with low utility (essentially related to infrastructural needs) were advanced too much.

To better understand how the two plans differ in terms of sprint composition, we measured their difference as the average of the gaps of all user stories:

Definition 2 (User Story Gap). *Let j be a story. Let i^{team} and i^{opt} be the sprints j belongs to in the team plan and in the optimal plan, respectively. The* gap *of story j is*

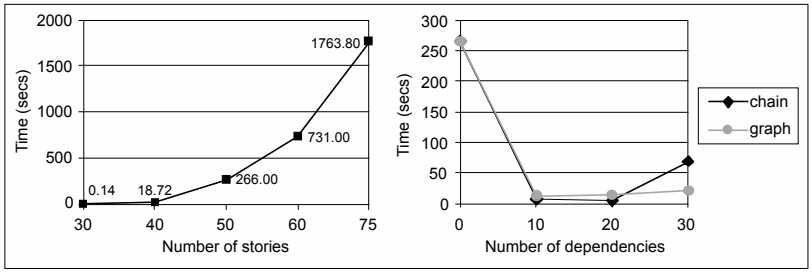
$$gap(j) = \frac{1}{N-1}|i^{team} - i^{opt}|$$

where N is the maximum number of sprints in the two plans.

The user story gap ranges from 0 to 1, where 0 means that the story belongs to the same sprint in both plans. As shown in Figure 2.b, the average gap is always lower then 0.3, denoting a good correspondence between the two plans. The main difference arises in sprints 1, 7, 8, and 10. In particular, in sprint 1, the team plan aimed at anticipating critical stories, thus exceeding the sprint capacity. The strong difference in the composition of the first sprint necessarily affected the subsequent sprints. Noticeably, both plans made good use of affinity relationships.

In order to have a further evaluation of the optimal plan, we discussed it with the team chief after the project end. Here are the main outcomes:

- The team spent a couple of days in defining their plan, while the optimal plan was generated in a few seconds.
- The team was used to collecting user story estimates using standard forms, but the level of detail required by our framework is slightly higher. This was



(a) (b)

Fig. 3. Time for computing the optimal plan for projects (a) with an increasing number of stories and no dependencies, and (b) with an increasing number of dependencies and 50 stories

perceived has a positive aspect since it leads to more refined estimates, thus producing a better plan.
- The team chief recognized that his plan failed in properly distributing risks, which led to some delay in the first sprint.
- The optimal plan was judged to be feasible and realistic, showing that the elements considered in our model provide a good distribution of user stories.
- Most of the differences in sprint compositions were evaluated as improvements over the team plan. In particular, the team plan did not take into account the side effects of postponing some stories, thus causing the stories depending on them to be delayed too much.

5.2 Efficiency Tests

These tests were carried out on an Intel Core 2 Duo platform with 3 Gb of RAM, running at 3 GHz under Windows XP professional. To test the model behavior on a broad benchmark we generated a set of 58 synthetic projects; utility and story points of the user stories were randomized in the intervals [10,100] and [1,10], respectively. The maximum sprint duration was set to 15 days, while the development speed was set to 3 story points per day (i.e., sprint capacity was 45 story points). All planning problems were solved using CPLEX; performances were measured in seconds.

First of all we evaluate performances in function of the total number of user stories on projects that do not include dependencies. Figure 3.a reports the average time needed to compute the exact solution. As expected for a multi-knapsack problem, the computation time grows non-linearly, reflecting an exponential increase in the search space.

The presence of dependencies makes planning harder for the project team. To study their impact on our model, two types of dependencies were added to our benchmark projects: (1) *chain* dependencies, where each story depends on at most another story; and (2) *graph* dependencies, where a story can depend

on several stories. In both cases dependencies were obviously acyclic. Figure 3.b shows how the computation time changes in function of the number of dependencies. This figure suggests that a small number of dependencies tends to reduce the computation time because dependencies allow a set of unfeasible plans to be pruned, thus reducing the search space. However, when the number of dependencies is high, the computation time increases again because finding a feasible plan becomes harder for the solver. Noticeably, we observed that both chain and graph dependencies show similar trends.

Though the time to obtain an exact solution for very complex problems (more than 100 stories) can be too high, the time to obtain a good feasible solution is always limited. CPLEX can be configured so that it first looks for a feasible solution, then it tries to improve it until the exact one is found; at each step it returns the utility of the best solution found so far (i.e., an upper bound to the utility of the optimal solution) and a lower bound to the utility of the optimal solution. We measure the suboptimality at each step (i.e., how the current solution is far from the optimal one) as the ratio between the lower and the upper bounds. Remarkably, a solution that is less than 1% worse than the optimal one is always produced within 5 seconds.

6 Conclusions and Future Work

In this paper we formalized the sprint planning problem for agile DW design and we proposed a multi-knapsack model to solve it. We tested our model on a set of (both synthetic and real) projects. We found that, for medium-sized problems, an exact solution is determined in a time that is fully compatible with the development process (i.e., from some seconds to a few minutes), while for large problems a heuristic solution that is just a few percentage points far from the exact one can be returned in a couple of seconds. As to effectiveness, the team chief judged the optimal plan to be feasible and realistic, and most of the differences in sprint composition were evaluated as improvements over the team plan. Currently, the optimal plan is delivered to the team in tabular form; however, to present the plan in a more effective way, our optimization module could be coupled with existing softwares for agile project management.

We are currently working on extending our model to better support the planning activity. First of all we will accommodate iterative planning, i.e., given a first solution the team will manually adjust it by pinning some user stories to some sprints, and then run the optimizer again. This requires an extension of our model to deal with further types of constraints while preserving the overall structure of the resulting plan. Further improvements that will make the model best fit for real cases are: (1) allowing different development speeds for different sprints due to a variable team composition; (2) modeling different team capabilities (e.g., design, implement, test) so that, in each sprint, the team will be able to deliver a different number of story points for each capability.

References

1. Aalto University, SoberIT: Agilefant (2011), http://www.agilefant.org/
2. Abrahamsson, P., Warsta, J., Siponen, M.T., Ronkainen, J.: New directions on agile methods: A comparative analysis. In: Proc. ICSE, pp. 244–254 (2003)
3. Beck, K., et al.: Manifesto for agile software development (2001), http://agilemanifesto.org/
4. Boehm, B.W.: A spiral model of software development and enhancement. IEEE Computer 21(5), 61–72 (1988)
5. Brucker, P., Drexl, A., Möhring, R.H., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112(1), 3–41 (1999)
6. Caprara, A., Fischetti, M.: Branch-and-cut algorithms. In: Dell'Amico, M., Maffioli, F. (eds.) Annotated Bibliographies in Combinatorial Optimization. Wiley Interscience Series in Discrete Mathematics (1997)
7. Cohn, M.: User Stories Applied: For Agile Software Development. Addison-Wesley Professional (2004)
8. Collabnet: ScrumWorks (2011), http://www.danube.com/
9. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information & Software Technology 50(9-10), 833–859 (2008)
10. Fréville, A.: The multidimensional 0-1 knapsack problem: An overview. European Journal of Operational Research 155(1), 1–21 (2004)
11. Golfarelli, M., Rizzi, S., Turricchia, E.: Modern Software Engineering Methodologies Meet Data Warehouse Design: 4WD. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2011. LNCS, vol. 6862, pp. 66–79. Springer, Heidelberg (2011)
12. Herroelen, W., Demeulemeester, E., Reyck, B.D.: A classification scheme for project scheduling problems. Tech. rep., Katholieke Universiteit Leuven (1997)
13. Hughes, R.: Agile Data Warehousing: Delivering world-class business intelligence systems using Scrum and XP. IUniverse (2008)
14. IBM: IBM ILOG CPLEX optimizer (2011), http://www-01.ibm.com/
15. Kolisch, R., Padman, R.: An integrated survey of deterministic project scheduling. Omega 29(3), 249–272 (2001)
16. Kruchten, P.: The 4+1 view model of architecture. IEEE Software 12(6), 42–50 (1995)
17. Martello, S., Toth, P.: Knapsack Problems: Algorithm and Computer Implementation. John Wiley and Sons Ltd. (1990)
18. Martin, J.: Rapid application development. MacMillan (1991)
19. Nichols, A.: Agile planning, estimation and tracking (2009), http://www.slideshare.net/andrewnichols/agile-planning-estimation-and-tracking
20. Pomberger, G., Bischofberger, W.R., Kolb, D., Pree, W., Schlemm, H.: Prototyping-oriented software development — concepts and tools. Structured Programming 12(1), 43–60 (1991)
21. Royce, W.W.: Managing the development of large software systems: Concepts and techniques. In: Proc. ICSE, Monterey, California, USA, pp. 328–339 (1987)
22. Schwaber, K.: SCRUM development process. In: Proc. OOPSLA (1995)
23. ThoughtWorks Studios: Mingle: Agile project management (2011), http://www.thoughtworks-studios.com/