Planning Optimal Agile Releases via Requirements Optimization

Joseph Gillain, Ivan Jureta, Stéphane Faulkner
Data Analysis Decisions Advice (DA^2) Research Unit
Department of Business Administration & PReCISE Research Center, University of Namur

***Abstract*—This paper focuses on improving requirements quality in agile projects by determining requirements prioritization. Current methods suggest to take into account business value in order to determine the requirements priority rank. In practice it was observed that many other factors enter into the equation, such as implementation cost and functionality dependencies. Since agile methods suggest that priority should be customer/user's prerogative, taking all relevant factors becomes challenging without decision supporting tools. Our research question is the following: How can we formulate the agile release decision problem, and which computations can we do over requirements models to recommend solutions to that decision problem? Our contributions are the following: (i) we formulate this agile release decision problem as an optimization problem, (ii) we provide a modelling language to represent instances of this problem as requirements models, and (iii) we describe an online tool to make the models and solve the resulting optimization problem instances.**

I. Introduction

A. Context: Requirements Priority in Agile Projects

Agile methods have become popular for organizing software development. Many benefits have been touted [1]: better knowledge transfer, active participation of the customer in the project, incremental deliveries, and so on.

Relative to more established methodologies (e.g., waterfall), which are based on the assumption that the problem is fully specifiable and a solution can be entirely predicted [2], agile methods are based on the assumption that customers, users, stakeholders in general are not able to specify all requirements of the system-to-be at the beginning of the software process, due to requirement or environment change, lack of experience in the problem domain, etc. [3].

Agile tries to treat requirements continuously through the whole project life cycle. Development is iterative, the development team periodically delivers increments of functionality by focusing on requirements with highest priority at each release planning. It ensures that value comes continuously during the complete process and stakeholders are then being able to test and validate the most priority requirements. Those tests helping the customer clarify and refine subsequent requirements, for later releases.

To make such an approach work for customers, requirements prioritization is important. The Agile Requirements Problem (ARP) is driven by elicitation of User Stories. It starts with the definition of high-level statements, *epics*, written from the system user's perspective, then refined into shorter, more narrowly scoped User Stories. This refinement process is ongoing, running throughout the system engineering process. The literature suggests that the priority should be given by the customer, *Product Owner*, to set criteria for the prioritization of user stories.

In practice, it is known that other criteria enter into account when prioritizing user stories. They can be, for instance, the effort estimated to implement each user story, or functional dependencies [4].

B. Scope: Goal Models Agile Requirement Problems

Agile requirement methods and tools have often been opposed to traditional requirements methods [3]. However, we suggest that agile methods could benefit from traditional requirement engineering methods especially from goal modeling (GM). Such models could be used to support agile requirements and specifically prioritization of emerging requirements.

Similarly to user stories, goal models are based on a refinement process in which abstract (goals/user stories) are refined into more concrete ones. In both cases, the refinement process ends up with an operationalization into tasks. Another common point is that both user stories and goals are desired states the system should satisfy, or bring about.

However, goal models differ from user stories since they allows among others to explore alternative solutions or to assess conflicts between goals. They also support modelling of non-functional requirements while it is not obvious how such requirements should be incorporated into user stories[1].

The variability aspect, or the exploration of alternative solutions, of goal models seems truly interesting for agile projects since they do not define the whole product in the early stages. They instead specify a *vision*, which is imprecise and abstract, and gives thereby a space with potentially many alternative solutions (functionalities) that the system could implement. Variability is even more interesting when considering that a first "fast and cheap" version of the system could be delivered, before investing in a more comprehensive solution to be brought to market via subsequent releases.

C. Problem: Which requirements to satisfy when?

As we just discussed, one important principle of agile methods is to provide highest value requirements first on the

[1]One can cite Alistair Cockburn's suggestion to formulate them as constraints [5]. This approach can be transposed to goal models

978-1-5090-3694-3/16 $31.00 © 2016 IEEE
DOI 10.1109/REW.2016.36

RE 2016 Workshops, Beijing, China
AIRE Workshop Paper

basis of business value. We call this the Agile Requirements Problem (ARP) and it involves the following sub-problems.

Evaluation problem: How to assess this business value because it is not independent of the implementation cost. As an example, consider 2 candidate user stories for the next release of a system, let's say a and b (with respective business value of \$8000 and \$5000). Before deciding which one should be selected, it seems reasonable to also take into account their relative cost. If a costs \$5000 to be implemented while b costs \$2000, it is clear that b should be implemented first. Then, business value should be balanced with effort estimation.

Selection problem: A second problem comes from the difficulty to select which alternative should be implemented when facing a lot of variability in requirements. The customer has to select between potentially dozens of combination of features. This is very difficult and time consuming without a decision support tool.

Release problem: A third problem is to determine in which order the selected solution should be delivered. It is about release planning. The main criteria being to provide the most value as soon as feasible in the lifecycle.

D. Contribution: An Optimization Problem and its Resolution

In this paper, we suggest how to model ARP with goal models and provides a mapping to a Mixed-Integer Programming (MIP) optimization problem whose solutions automatically define a planning release taking into account business value, functional constraints and required effort. We also present an implementation of the approach in AnalyticGraph.com, an online tool.

The paper is structured as follows. First, we show in section II how to model ARP instances with goal models. Then, in section III, we present the mapping between the goal model and a mixed-integer program. Section IV presents the solution of the MIP. We present AnalyticGraph and the implementation of the MIP in section V. Eventually, we discuss related works and we conclude in sections VI and VII.

II. Modeling agile requirements with Techne

In this section, we present how we model ARP instances with goal models. The selected Goal Modeling Language (GML) is Techne [6]. This is because, firstly, Techne is an abstract goal modelling language; secondly, it is straightforward how to translate Techne models into models in other goal modelling languages; finally, this work builds on our prior work on requirements optimization in Techne models [7].

This section first discusses the goal refinement process and relates it to the refinement of user stories. Then, we show how to take into account of the business value and task effort in the goal model.

A. Refinement

In order to illustrate the refinement process of user stories with a goal model, we use the following example. The company sends out mystery shoppers, that is, individuals unknown at a shop, to make a purchase and subsequently evaluate the purchasing experience. The data is then used for marketing purposes. The goal model is depicted in Fig. 1, and we review that model below.

There are three abstract user stories (i.e *epics*):

- As a Mystery Shopper (MS), I want to make visit results available to the customer (G0)
- As a Customer[2], I can access visit results in a central repository (G16)
- As a Customer, I can explore data through vizualisations (G37)

Each of these user stories are depicted as a goal in the goal model. However, to make them practical for development, they need to be refined into more concrete ones. For example, I could consider that G0 will be satisfied if a MS can record visit results in an Excel spreadsheet (G5) and then he can send it via mail to the customer (G6). This refinement can be depicted in the goal model. In Techne, we use an inference relation node (I1 in Fig. 1) between G0, G5 and G6. This relation says that

$$G5 \wedge G6 \rightarrow G0. \tag{1}$$

It is also possible to model some dependencies between goals. For example, we cannot consider that G16 is satisfied if G0 has not been. This is done by adding G0 in the premises of all inference relationships which would conclude G16 as satisfied.

In agile methodologies (e.g. SCRUM, XP), once the user stories have been refined to an adequate level of granularity, we can ask developers to define underlying tasks to be done. Those tasks would integrate the Sprint Backlog, i.e. a set of tasks to be done during the next implementation iteration. In our goal model, this is achieved by operationalizing goals into tasks. For example, G5 could be consider as "done" if developers create an Excel template that will be used by mystery shoppers (T31).

This approach fits a SCRUM approach. There, a Product Backlog (PB) is created and iteratively filled. Each item of the PB is a user story that can be refined more and more as the project progresses. User stories are always descriptions of a functionality of the system-to-be from a user perspective. In other words, it is a desired state that the user would like. It is similar to the goal concept in goal modeling. Once a user story is selected for the next sprint, developers filled a Sprint Backlogs with the underlying tasks for a given amount of effort.

The effort required to implement those tasks (or user stories not yet refined into tasks) are estimated in *story points*. For this, one can use poker planning [8]. It is however necessary to take into account these story points in our goal model.

B. Business Value and Story Points as Node Label

Another important concept in agile processes is requirements prioritization. Requirements (user stories) with the highest priority would be implemented first. The agile literature

[2]By *Customer*, it is understood the Mystery Shopper company's customers.

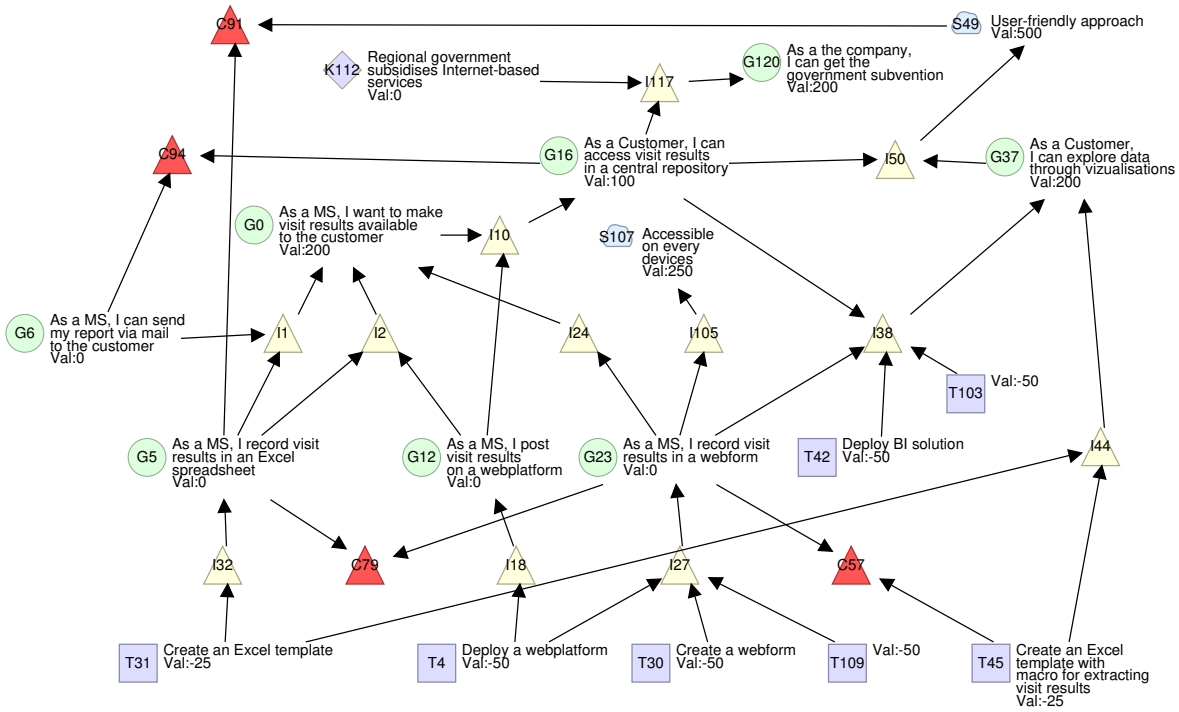

Fig. 1. Modeling of the Mystery Shopper Case. Model accessible and editable at http://analyticgraph.com/dev/?g=R2CinFhvUK
- Shape meaning is the following: Goals are circles, tasks are squares, inferences are yellow triangles and conflicts are red triangles.

insists on the fact that the business value should be the first driver for prioritizing. However, in practice we can see that it is not the only element to be considered for prioritizing [1]. Another important criterion to consider when prioritizing is the required effort, which determines development cost.

Both business value and cost can be integrated in our goal model. It takes the form of a label to associate to each node (goal or task). In the mystery shopper example, value are depicted as a node label prefixed by "Val". For example, the goal G0 will leverage 200 units of value if it is satisfied, while T31 will consume 25 units of resources (it could be story points).

When speaking about revenue acquired from goals, it is important to distinguish two types of revenue: *recurrent revenue* and *unique revenue*.

In the case of the recurrent revenue, the achieved goal will yield revenue continuously as long as it is satisfied. For instance, consider goal G37 stating that a "Customer can explore data through vizualisations". Once this goal is achieved, the mystery shopper company will asks additional $10 per visits since the service offered has higher value. In this case, this goal has a recurrent revenue. For this type of goal, the earlier they are satisfied, the better it is.

With unique revenue, we model goal which yields a single amount of value. For example, the mystery shopper could get a subvention from government as soon as the business has an Internet platform (G120). However, it can be accorded only when goal G16 is satisfied and under the assumption (K112) that the government effectively subsidies it.

Regarding task implementation, some tasks need to be done only once and can be considered as definitely done. We call them *persistent task*. Others do not have this property. A deployment task for instance has to be redone.

Both recurrent revenues and persistence tasks are boolean properties associated with nodes in our goal model.

C. Conflicts between goals

Conflict management is important in this approach, because the solution should not produce a release planning whose increments are individually conflict-free but the integrated solution presents several conflicts.

Nonetheless, it can be interesting to deliver a first version of a system providing some features and removing them when a next release introduces more elaborated features (conflicting with the previous release).

In our example, we know that if mystery shoppers record their visits in a web-based form it will be conflicting with recording them in a spreadsheet. This is depicted with conflict C79. Moreover, the spreadsheet macro would be useless to

preparing visualisation if data are no more recorded in a spreadsheet (conflict C57).

Another interesting conflict is that if customer can access visit results on a central repository and that some analytics are deployed, the solution should be more user-friendly than simple excel reports. Of course, it requires than no more reports should be registered in spreadsheet (conflict C91).

Eventually, the central repository would be usefull iff mystery shopper stop sending their report via mail.

D. Project progress

An important aspect of agile project is that requirements will evolve through the whole project. One should be able to modify the goal model in the same way a product backlog evolves.

After a particular sprint has ended, one should change the story point to 0 if that sprint was completed. If part of tasks is completed, task effort can be reduced.

III. Mathematical model

In previous work, we described how a goal model could be mapped into a mixed-integer program [7]. The complete program is given in Tab. I and we discuss in more details specific modifications bring to the MIP for dealing with ARP.

A. Objective function

Priority in agile methods should be to maximize business value delivered to the customer. If we define u as an *utility function* mapping each goal at particular sprint to a business value, we can define the objective function as:

$$\max \sum_{p \in P} \sum_{r \in R} \mathsf{u}(r,p) * \sigma_{r,p}$$

where:

- P is the set of sprints,
- R is the set of goal model nodes (goals, softgoals, quality constraints and domain assumptions),
- $\sigma_{r,p}$ is a binary decision variable setting if the goal r has been achieved for period p (equals to 1 iff the node is satisfied, 0 otherwise).

For modelling the importance of delivering features as soon as possible (the urgency), a discount rate can be applied on the utility function, resulting in a situation where $\mathsf{u}(r,i) > \mathsf{u}(r,j)$ if $i < j$ with i and j being the number of the sprint. Some features will have no more value if delivered after a particular deadline.

B. Inference relation node

Mapping inference node in the MIP is done by adding two constraints. The first constraint called *Premises constraint* checks if incoming nodes from an inference node are all satisfied:

$$\forall p \in P, \forall i \in I : \sigma_{i,p} \leq \sum_{x \in \mathsf{in}(i)} \frac{\sigma_{x,p}}{|\mathsf{in}(i)|}$$

where:

- I is the set of inference nodes,
- $\mathsf{in}(\mathrm{i})$ is the set of incoming nodes of i
- $|\mathsf{in}(\mathrm{i})|$ is the cardinality set of incoming nodes of i

The second constraint, called *Conclusion constraints*, applying only for nodes having incoming inferences:

$$\forall p \in P, \forall n \in \{x \in N : |\mathsf{in}_I(x)| > 0\} : \sigma_{n,p} \leq \sum_{i \in \mathsf{in}_I(n)} \sigma_{i,p}$$

where:

- N is the set of all nodes
- $\mathsf{in}_I(x)$ is the set of incoming **inferences** of x

C. Constraints for value persistence

As discussed in section II-B, their are different types of goals and tasks regarding their business value/cost. Goals can bring recurent revenues as long as they are considered as satisfied or on the contrary, they can bring a unique revenue even if their satisfaction last several sprints. By default, the model support recurrent revenue. An additional constraint set is needed to model single revenue goal, we call it *Unique Revenue Constraints*.

Regarding task persistence, some tasks need to be done each time it is necessary to use them, while other tasks are required to be executed once. Default behavior of the MIP is the non persistent tasks while an additional set of constraints is required for persistent tasks.

1) Unique Revenue Constraints: Ensuring unique revnue of some goals is done with:

$$\forall g \in G^* : 1 \leq \sum_{p \in P} \sigma_{g,p}$$

where G^* being the set of goals with unique revenue.

2) Persistence of task realization: For persistent task, we need to specify that if the task has been developed during a previous iteration, it can be considered as satisfied for the following iterations. It is done with the following constraints:

$$\forall p \in P, \forall i \in \overline{T} : \sigma_{i,p} \leq \sum_{q \in \{1 \ldots p\}} \alpha_{i,q}$$

where:

- $\overline{T}$ is the set of persistent task
- $\alpha_{i,q}$ is a binary decision variable assessing that the task i has been realized during sprint q.

The constraint is read as following, the task i can be considered as satisfied for period p (i.e. $\sigma_{i,p}$ is set to one) iff it has been realized during the current or a previous sprint.

D. Velocity constraint

Agile methods work iteratively. For example, a SCRUM project is divided into *Sprints* in which developers have to develop a certain amount of story points. This amount is called the team velocity.

Adding this aspect into the MIP is done by adding a new set of constraints called hereafter *velocity constraints*. If we

TABLE I
DESCRIPTION OF THE MIXED INTEGER PROGRAM OF FOR THE ARP MODELLED WITH TECHNE.

Constants		
v		Velocity
Sets		
P	$= \{1, \ldots, p\}$	Iterations
N	$= G \cup T \cup S \cup Q \cup K$	Concept nodes (goals, tasks, softgoals, quality constaints and domain assumptions)
G^*	$\subseteq G$	Unique revenue goals
$\overline{T}$	$\subseteq T$	Persistent tasks
M	$\subseteq N$	Mandatory nodes
C	$= \{c_0, \ldots, c_i\}$	Conflict nodes
I	$= \{i_0, \ldots, i_i\}$	Inference nodes
N^*	$= N \cup I$	Graph nodes
R	$= N \setminus T$	Revenues nodes
Decision Variable		
$\sigma_{N^*,P}$	$= \{\sigma_{i,p} \in \mathbb{B} : i \in N \cup I, p \in P\}$	Binary variables representing satisfaction of nodes.
$\alpha_{\overline{T},P}$	$= \{\alpha_{i,p} \in \mathbb{B} : i \in \overline{T}, p \in P\}$	Binary variables representing satisfaction of persistent tasks.
Functions		
u	$: N \times P \to \mathbb{R}$	Utility function
$\mathsf{in}(x)$	$\subseteq N \cup I$	Incoming nodes function
$\mathsf{in}_Y(x)$	$= \mathsf{in}(x) \cap Y$	Typed incoming nodes function
Objective function		
$\max \sum_{p \in P} \sum_{r \in R} \mathsf{u}(r,p) * \sigma_{r,p}$		
Constraints		
Premises constraints $\forall p \in P, \forall i \in I :$		$\sigma_{i,p} \leq \sum_{x \in \mathsf{in}(i)} \frac{\sigma_{x,p}}{\lvert\mathsf{in}(i)\rvert}$
Conclusion constraints $\forall p \in P, \forall n \in \{x \in N : \lvert\mathsf{in}_I(x)\rvert > 0\} :$		$\sigma_{n,p} \leq \sum_{i \in \mathsf{in}_I(n)} \sigma_{i,p}$
Persistent satisfaction $\forall p \in P, \forall i \in \overline{T} :$		$\sigma_{i,p} \leq \sum_{q \in \{1 \ldots p\}} \alpha_{i,q}$
Unique revenue $\forall g \in G^* :$		$1 \leq \sum_{p \in P} \sigma_{g,p}$
Conflict constraints $\forall p \in P, \forall c \in C :$		$\sum_{x \in \mathsf{in}(c)} \sigma_{x,p} \leq 1$
Mandatory constraints $\forall i \in M :$		$1 \geq \sum_{p \in P} \sigma_{i,p}$
Velocity constraints $\forall p \in P$		$v \geq \sum_{t \in \mathsf{in}_{\bar{T}}} u(t) * \sigma_{t,p} \sum_{t \in \mathsf{in}_{\overline{T}}} u(t) * \alpha_{t,p}$

define v as the team's velocity and P as the set of all sprints in the project, we formalize the constraints as:

$$\forall p \in P : v \geq \sum_{t \in \mathsf{in}_{\bar{T}}} u(t) * \sigma_{t,p} \sum_{t \in \mathsf{in}_{\overline{T}}} u(t) * \alpha_{t,p}$$

IV. MIP Solution

The provided solution of the mathematical program states in which sprint should a task be planned and it also describes how to achieve each goal in each sprint. It is depicted in Table II.

For example, after the first sprint, T31 and T41 would be implemented resulting in the satisfaction of goals G0, G5, G6 and G37. An interesting conclusion from this planning is that it would be interesting for the customer to have a first solution for the system (with excel files) while another more integrated solution would come in the next sprints. This migration would result in two periods where the customer will have no more access to data exploration because of the migration. Nonetheless, it is still the planning maximizing its global satisfaction.

V. AnalyticGraph as Supporting Tool

All previously presented models and modelling primitives are accessible on AnalyticGraph. This is a web-based platform designed to support modelling and reasoning on RE models. In AnalyticGraph, users model RE problems as directed graph. Each graph consists of a set of nodes linked by directed relations. The graph is stored in a graph-oriented database and meta-data (e.g. graph name, author...) are stored in a relational database. Currently, AnalyticGraph comes only with Techne as RML, but other RMLs can be specified and used.

The case presented in Fig. 1 can be accessed at http://analyticgraph.com/dev/?g=R2CinFhvUK. Tutorials are available[3]. Among them is a tutorial related to goal model optimization. This module is aimed at transforming the goal model in the Mixed-Integer Program (MIP) described above. The language used is the GNU MathProg language [4].

VI. Related works

This paper is on the boarder of two research disciplines. First, it deals with requirements optimization since it tries to identified an optimzation set of requirements between different alternatives. Secondly, it is directly related with what is called the Next Release Problem [9].

Several approaches have already suggested for applying optimization techniques on requirements engineering. In the paper, Zhang et al. discussed them in general by presenting advantages and challenges [10].

One of the first application is suggested by [11]. It gives developers a method to balance the cost and value of the requirements, and then implement the most cost-effective set.

Bagnall et al. were the firsts to coined the term *Next Release Problem* [9]. It is about selecting a set of requirements that is deliverable within a budget and which meets the demands of the customers. Our approach goes a step further by providing a mapping between an actual requirement modelling language (Techne) to a MIP. We also further elaborate notion of customer satisfaction by distinguishing two types of revenues.

In their work, Zhang et al. explored the multi-objective next release problem by distinguishing the revenue maximization and the cost minimization [12]. Their work mainly consists of comparing search techniques for multi-objective optimization problems. Although separating cost and customer values seems interesting, our approach can be applied with a profit function (i.e. a fitness function summarizing revenue maximization and cost minimization) [7]. Moreover, our work is less focused on comparing search algorithms but rather on providing a mapping between goal modelling and MIP.

In their work, Ruhe et al. suggested an approach mixing what they call the *art of release planning* and the *science of release planning* [13]. It results in a model taking into account dependencies between features, resource constraints, urgency of features and different stakeholders point of view. Our approach differs from theirs because there is no attempt to balance multiple stakeholders satisfaction and urgency rates. However, urgency is managed with diminishing values of goals through time.

Saliu et al. focused on release planning optimization for evolving systems [14]. They proposed a new release planning framework that considers the effect of existing system characteristics on release planning decisions.

In their paper, Tonella et al. provides an interactive genetic algorithm aimed at minimizing the disagreement between a total order of prioritized requirements and the various constraints that are either encoded with the requirements or that are expressed iteratively by the user during the prioritization process [15].

In comparison with all previously mentioned work, our approach is more focused on providing a mapping between an existing requirement modelling language than a study on performance of various algorithms. It also simultaneously provides a release planning (i.e determines priorities) and select between alternatives while related methods focus only on one of these aspects.

VII. Conclusion and future works

This paper suggested a method able to support optimization of release planning in agile projects. A first contribution was to suggest modeling of the Agile Requirement Problem with goal models. A second contribution was a mapping between the goal model and a Mixed-Intger Program. Eventually, we briefly presented an implementation of the approach on AnalyticGraph.com.

Although our optimization problem focuses on business value delivery (as agile principles prescribes), it also takes into account implementation effort and feature dependencies (through inference relationships).

The approach allows each user to focus on their domain, the product owner defines a product backlog in the form

[3] http://analyticgraph.com/category/tutorial/

[4] https://en.wikibooks.org/wiki/GLPK/GMPL_(MathProg)

TABLE II
SOLUTION OF MYSTERY SHOPPER CASE

		Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6
Tasks	Story Points						
T31	25	x					
T45	25	x					
T4	50		x				
T30	50			x			
T42	50				x		
	Sprint Backlog size	50	50	50	50	50	50
Goals	Revenue						
G0 Register visit	200	x	x	x	x	x	x
G5 Insert xls	0	x	x	x			
G6 Send by mail	0	x					
G12 Post on web	0		x	x	x	x	x
G16 Register central	100		x	x	x	x	x
G23 Insert web	0				x	x	x
G37 Xplore data	200	x	x	x			x
G120 Get subvention	200		x				
S49 User-friendly	500						x
S107 Accessible on many devices	250				x	x	x
	Revenue	400	700	500	550	550	1250
	Cumulated Rev.	400	1100	1600	2150	2700	3950

of a goal model (which replaces the traditional list of user stories), developers identifies implementation tasks to be done during sprints and evaluate their required efforts with story points. Then the release planning (and consequently priorities) is computed by resolution of the mixed-integer program.

REFERENCES

[1] K. Petersen and C. Wohlin, "A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case," *Journal of systems and software*, vol. 82, no. 9, pp. 1479–1490, 2009.

[2] S. Cavaleri and K. Obłój, *Management Systems: A Global Perspective*. Wadsworth, 1993. [Online]. Available: https://books.google.be/books?id=ZAyqQgAACAAJ

[3] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements engineering and agile software development," in *null*. IEEE, 2003, p. 308.

[4] Z. Bakalova, M. Daneva, A. Herrmann, and R. Wieringa, "Agile requirements prioritization: What happens in practice and what is described in literature," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2011, pp. 181–195.

[5] A. Cockburn, "Writing effective use cases," *preparation for Addison-Wesley Longman. www. infor. uva. es/˜ mla-guna/is2/materiales/BookDraft1. pdf*, 1999.

[6] I. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling." in *RE*, 2010, pp. 115–124.

[7] J. Gillain, S. Faulkner, P. Heymans, I. Jureta, and M. Snoeck, "Product portfolio scope optimization based on features and goals," in *Proceedings of the 16th International Software Product Line Conference-Volume 1*. ACM, 2012, pp. 161–170.

[8] V. Mahnič and T. Hovelja, "On using planning poker for estimating user stories," *Journal of Systems and Software*, vol. 85, no. 9, pp. 2086–2095, 2012.

[9] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley, "The next release problem," *Information and software technology*, vol. 43, no. 14, pp. 883–890, 2001.

[10] Y. Zhang, A. Finkelstein, and M. Harman, "Search based requirements optimisation: Existing work and challenges," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2008, pp. 88–94.

[11] H.-W. Jung, "Optimizing value and cost in requirements analysis," *IEEE Softw.*, vol. 15, no. 4, pp. 74–78, Jul. 1998. [Online]. Available: http://dx.doi.org/10.1109/52.687950

[12] Y. Zhang, M. Harman, and S. A. Mansouri, "The multi-objective next release problem," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 1129–1137.

[13] G. Ruhe and M. O. Saliu, "The art and science of software release planning," *IEEE software*, vol. 22, no. 6, pp. 47–53, 2005.

[14] O. Saliu and G. Ruhe, "Supporting software release planning decisions for evolving systems," in *29th Annual IEEE/NASA Software Engineering Workshop*. IEEE, 2005, pp. 14–26.

[15] P. Tonella, A. Susi, and F. Palma, "Using interactive ga for requirements prioritization," in *Search Based Software Engineering (SSBSE), 2010 Second International Symposium on*. IEEE, 2010, pp. 57–66.