

# Supporting the Dynamic Reprioritization of Requirements in Agile Development of Software Products

Zornitza Racheva<sup>1</sup>, Maya Daneva<sup>1</sup>, Luigi Buglione<sup>2</sup>

<sup>1</sup> *University of Twente, Netherlands, {z.racheva, m.daneva}@utwente.nl*

<sup>2</sup> *École de Technologie Supérieure (ETS) / Engineering.IT, luigi.buglione@computer.org*

**Abstract.** *Agile requirements engineering is the approach of choice for many software producers whose realities include highly uncertain requirements, use of new development technology, and clients willing to explore the ways in which an evolving product can help their business goals. From customer's perspective, the activity of continuous requirements reprioritization forms the very core of today's agile approaches. However, the freedom for clients to do so does not come for free.*

*This paper presents results of a literature review on agile requirements prioritization methods, derives a conceptual model for understanding the inter-iteration prioritization process in terms of inputs and outcomes, and identifies issues and solutions pertinent to agile prioritization. The latter are derived from the authors' experiences and by using empirical data, published earlier by other authors.*

**Keywords:** agile development, requirements prioritization, inter-iteration decision-making process.

## 1 Introduction

Agile methods, such as Extreme Programming (XP), SCRUM or CRYSTAL, advocate requirements engineering (RE) through the software product development cycle in small and informal stages. That is, instead of engineering the requirements upfront, one lets requirements emerge during development. Agile software process practitioners deem this approach particularly valuable for software producers in a context that includes highly uncertain requirements, experimentation with new development technology, and clients willing to explore the ways in which an evolving product can help their business goals. From customer's perspective, continuous requirements reprioritization forms the key of today's agile approaches. A recent empirical study [6] indicates that, with respect to requirements (re)prioritization, agile RE differs from 'traditional RE' in two ways: (i) (re)prioritization happens at inter-iteration time, which

means the project team anticipates and plans as many reprioritization sessions as the number of project iterations, and (ii) (re)prioritization is based mostly on business value, that is, the highest priority features get implemented early so that most business value gets realized. These two features of agile RE pose at least two challenges: (i) continuous reprioritization more often than not (especially when practiced without caution) leads to project instability, and (ii) customers, by and large, relate the concept of business value to features that meet their functional requirements, so non-functional requirements (such as scalability or security) that might initially appear secondary to clients, turn out critical for the operational success of the product. Redesigning the architecture of the software product at that late stage would add up to an over-expensive or a delayed project. In a context of a supplier network, these challenges aggravate further, as product managers make commitments based on process and product assumptions which are different from the ones of the development team. Yet, these assumptions might be essential to product success.

Our paper is a first attempt to respond to these two challenges. It proposes a conceptual model of the agile prioritization process *from client's perspective*. We complement it with a discussion on the issues in dynamic decision making (reprioritization). We make the note that we do not provide a new prioritization technique. Instead, we (i) redefine our view of requirements and their (re)prioritization by treating them from a clients' perspective, and (ii) we propose a model that reflects this specific focus and represents an unified approach to discussing the prioritization effort independently from the particular method that is used. This allows the clients to spot hidden issues early enough in the project, and helps them make the prioritization decisions.

The paper is structured as follows: Section 2 provides background on the role of clients in agile RE. Section 3 reviews related work on prioritization methods, used in agile software development. Section 4 presents the conceptual model. In Section 5, we reason about the assumptions of its application and identify some issues,

and turn them into open research questions for the future. Section 6 outlines our validation plan and Section 7 concludes the paper.

## 2 Agile Requirements Engineering

### 2.1 The role of clients and its implications

This section could be considered a sidebar for readers who are less familiar with agile development methods for software products, and in particular with the role of clients in agile RE. Our motivation for including a discussion on the client’s requirements (re)prioritization is to help readers understand sections 3, 4 and 5 and to avoid misunderstandings. The agile manifesto [22] positions the customer’s role is critical in making decisions about “what to build”. In the minimalist philosophy of XP, a prominent agile approach, the following is recommended for the customer’s role [4]:

1. The customer is an integral part of the team and should be on-site with the team.
2. The customer writes user stories and then discusses each requirement directly with the programmers.
3. The customer is responsible for all business decisions including prioritizing user story development.
4. The small 2-3 week iterations allow the user to evolve their requirements based on concrete working software.
5. The customer regularly tests the software to confirm it works as expected.

Our focus in this paper is on item 4 of the above list, namely supporting the client when making prioritization decisions. Therefore, in this and the next sections, we narrow the discussion down to the role of clients’ requirements prioritization in agile project management and software development.

When speaking about agile methodologies, a needed premise for any researcher is to recognize that, even if they can share the same principles, not all of them are directly comparable in terms of scope and content. A first distinction can be done between Agile Software Development (ASD) and Agile Project Management (APM). To emphasize the main differences, ASD is definable as “an evolutionary approach which is collaborative and self-organizing in nature, producing high-quality systems that meets the changing needs of stakeholders in a cost effective and timely manner” [2], while APM can be defined as “the work of energizing, empowering and enabling project teams to rapidly and reliably deliver customer value by engaging customer and continuously learning and adapting to their changing needs and environments” [3]. The word ‘transition’ is used to underline the wider scope of the APM methods compared to the scope of the ASD ones. The main drivers

for change in transitioning from ASD to APM are summarized in Table 1. The first column refers to the “what” is under observation in ASD methods such as XP, while the second one is about the related drivers for change and move from the ASD paradigm towards the APM one.

Table 1. Transition from ASD to APM [3]

	ASD (Objects of Interest)	APM Transition (Driver for Change)
Throughput	Flow of Value	Manage the flow, not activities
Teamwork	Small, Integrated Teams	Create an Integrated Team
	Customer Collaboration	Focus on the Project Context, not Content
	Continuous Improvement	More from Lessons Learned to Project Reflections
Leadership	Self-Organization	Coordinate

In the rest of the paper, our attention is paid to those methods classified as APM ones such as SCRUM. In particular, the focus is on the APM activities Flow of Value, Customer Collaboration and Focus on Project *Context*. We want to emphasize, that these activities will be considered from clients’ perspective.

### 2.2 Agile methods from a Product Manager's perspective

The agile literature distinguishes between the agile contexts which are custom programming environments and the agile contexts of software vendors. While in a custom context, the agile approaches rely on the on-site customer, in a vendor model, it’s the product manager who serves as the customer representative in planning and requirements definition.

The question of how APM relates to the roles and responsibilities of a Product Manager have been investigated by a few practitioners [12,14,15,17]. These authors indicate that in the agile context, the role of the project manager is re-positioned with respect to requirements management. This is because in agile context, requirements are the foundation of both project management and product management. The product manager’s mandate is to maintain a prioritized set of market problems to be solved in the next product iteration, as well as a vision for future product generations. Unlike in ‘traditional’ contexts, in which requirements management has been viewed as a software engineering job, in agile contexts the product manager assumes a bigger role in the sense that s/he takes

ownership over requirements prioritization and inter-iteration re-prioritization tasks – based on his knowledge of market problems. As the Product Manager is ‘the voice of the customer’, agile practitioners consider his/her role typically extended to the role of ‘product owner’, which is defined in the agile approaches [26]. This means, the Product Manager is available to answer design questions from customer perspective and updates the priority list before each iteration [17]. Clearly, this also means that Product Manager’s requirements are expressed in a way such that an agile development team could program on them.

More in detail, as per Knudsen,, this extended role of the Product Manager in agile contexts enables him/her accomplish at least three tasks: (i) it lets him/her tie the actual work to the market drivers, (ii) it lets him/her track the project progress so that s/he sets expectations with his/her customers and executives and (iii) it ensures the freedom and the control that the product manager needs so that s/he is able to deliver the right product to the market with the set window of opportunity. For example, the Standish Group [15] illustrates how companies like Webex, Google, Amazon, eBay, implement a practice they call "pipelining", that is, working on a variable scope and users getting small incremental changes instead of releases.

Currently, in response to agile product development contexts, requirements management tools for product managers are being offered to help organizations anticipate requirements (instead of simply react to them). A key support these tools provide is to let product managers list new requirements as they’re entered, existing requirements as they’re changed, and estimates for market requirements as they’re requested from development.

We want to make the note, that in this paper we do not make explicit difference between *on-site customer* and *product manager*, as the discussion of the prioritization process, that we undertake, is relevant to those, who perform prioritization, independently of their role.

### 3 Review of agile requirements prioritization methods

#### 3.1 Traditional versus agile prioritization

Clearly, requirements prioritization is a part of any project, independently from the developing method. Yet, the purpose and the place of this activity are essentially different when we distinguish between ‘traditional’ and agile development. In a ‘traditional’ (e.g. gated or waterfall-style life cycle), it is about which features (i) to implement earlier than others, or (ii) to include in an earlier release. The premise is that the whole functionality can not be implemented in the same time, but it will

eventually be implemented. So it is a project-management activity from the developers’ side. When asked about priorities in a ‘traditional’ project, the customer tends to qualify the majority of the requirements as *high priority*.

In contrast to ‘traditional’ development, agile projects rest on the understanding, that the whole functionality will not be implemented and delivered at once with the first release, and part of it will be eventually *not* implemented. The problem, then, is: (i) how to decide on what to implement in each (next) iteration, and (ii) which requirements will deliver the maximum value to the customers as early as possible. One of the biggest assets of an agile approach is that business value is delivered to the client throughout the project, and the return on investment is generated much earlier. Thus any changes in the requirements can be taken into consideration and implement into the product at an early stage. This highlights the paramount importance of the prioritization activities.

The changes in the list with requirements for an iteration might occur for different reasons – new market or company realities or better knowledge about the value certain features deliver. This requires a dynamic prioritization process as well. This view is supported by Harris and Cohn [11], who use tactics to minimize costs and maximize benefits through strategic learning and provide guidelines on how to optimize business value. They prove the necessity of adopting a dynamic approach to agile prioritization, in order to take into consideration the important aspect of learning in an agile project. Their focus is particularly on incorporating learning and cost of change in the decision-making process. We summarized the differences between the two settings in Table 2.

Table 2. Comparison of traditional and agile requirements prioritization

Aspects of the prioritization process	Traditional (waterfall) development	Agile
When is prioritization performed	Typically once, after the analysis phase and before implementation	Before each iteration, at planning phase, or during iteration
Who is responsible	Developer, with participation of project manager and other stakeholders.	Client/customer is the main driver for choosing, having an aid by the Scrum master (or Agile PM) about the technical feasibility of a schedule.
Goals/Purpose of the prioritization	Project management-vehicle	Vehicle to make sure delivered Business value is maximized at each iteration; Scope definition vehicle at iteration level

While in a traditional project the prioritization is usually performed once and before the implementation phase, in agile context it is an ongoing process, performed in the beginning of each iteration, or even during the iteration; this reflects the dynamics of the project's backlog.

### 3.2. Prioritization approaches in agile projects

In this section, we present a summary of agile requirements prioritization methods that can be found in the literature. We did a semi-systematic literature review to select the related work presented in this section. Our literature review included a broad search of academic and practitioners' information sources. We did a publication search using the five bibliographic databases: IEEEExplore, ACM Digital Library, Google Scholar, InterScience and Citeseer. The key words we used were: agile, requirements, prioritization, inter-iteration, decision-making, business value, features. We traced the references in the identified papers to get access to other relevant sources. To determine the relevance of these sources to our research, for each one, we reviewed the abstracts and the conclusions.

We want to stress that we refer to these methods as agile ones, as we have found evidences about their use in an agile development context – i.e. in an iterative and incremental process, following an agile methodology like SCRUM, XP, Crystal, etc. Nevertheless, the application of those methods is not restricted to agile settings, as many of them have been used in waterfall development as well [13].

Our review yielded 15 methods. Below we describe each of them in terms of its core idea and context of use (whenever details are provided by the authors):

1. **Round-the-group prioritization** [5]: Items are written on cards and placed in random order linearly either vertically or horizontally. The members of the group each take turns placing the items in the order they think is the proper priority order. While doing so, each person moving the cards is welcome to explain their reasoning. However, the other group members refrain from commenting on the new prioritization. This continues around the group as many times as it takes to find a stable order. The context of using this method includes group size of 3 to 8, and item list size less than 15.

2. **Ping Pong Balls** [26]. A fixed number of ping pong ball units are given to the group. The ping pong balls represent units of one dimension for prioritization such as value, risk or cost. The group discusses how to allocate ping pong balls to each item in a dynamic fashion until everyone agrees that the allocation makes sense. For very large lists, this is easiest to do in a spreadsheet with fewer people involved. This method is appropriate for projects where 1 to 12 participants take part in the prioritization effort, and for more than 15 requirements.

3. **\$100 allocation** (cumulative voting) [18]. Stakeholders get a fictitious \$100 to spend on requirements. After they allocate their money, tally the total for each requirement and then that total is divided by the number of stakeholders. The requirements are then rank-ordered, with the highest totals being most important.

4. **Multi-voting system** [27] This method uses elements of cumulative voting. A person can put multiple votes on a single item and can withhold some or all of her votes. After everyone has "finished" voting, the facilitator calls for everyone to step back and think about the results. Some discussion is allowed about the consequences of the results. Finally, everyone is given an opportunity to move their votes. The optimal group of participants might include between 5 and 20, and the item list size should not exceed 50.

5. **MoSCoW** [9] This is a widely used method, close to the *Numerical Assignment Technique* for traditional prioritization, where the items are roughly classified in *Priority groups* depending on importance. The letters stay for: M - MUST have this, S - SHOULD have this if at all possible, C - COULD have this if it does not effect anything else, W - WON'T have this time but would like in the future. The importance of this method is that when prioritising the words mean something and can be used to discuss what is important.

6. **Pair-wise analysis** [10] According to this method, the requirements are ranked by comparing them in pairs until the top requirements emerge at the top of the stack. This method functions for relative small number of requirements, where direct comparison is possible.

7. **Weighted criteria analysis** [10] Criteria are defined, and weights are assigned- so that some have higher value than others. Numbers are assigned to each weighted criterion to arrive at a total score for each requirement.

8. **Analytic Hierarchy Process** (AHP). AHP was developed by Saaty and applied later to software engineering. [25]. AHP is a method for decision making in situations where multiple objectives are present. This method uses a "pair-wise" comparison matrix to calculate the relative value and costs of individual requirements to one another. By using AHP, the requirements engineer can also confirm the consistency of the result. AHP can prevent subjective judgment errors and increase the likelihood that the results are reliable.

9. **Dot voting** [10]. Stakeholders are assigned sticky dots (which can also be colour-coded) to allocate to requirements. One counts the dots to narrow the list of requirements.

10. **Binary Search Tree** (BST) [1]. This is an algorithm that is typically used in a search for information, and which can easily be scaled to be used in prioritizing many requirements. The basic approach includes the following

steps: (1) Put all requirements in one pile. (2) Take one requirement and put it as root node. (3) Take another requirement and compare it to the root node. (4) If the requirement is less important than the root node, compare it to the left child node. If the requirement is more important than the root node, compare it to the right child node. If the node does not have any appropriate child nodes, insert the new requirement as the new child node to the right or left, depending on if the requirement is more or less important. Steps 3-4 are repeated, until all requirements have been compared and inserted into the BST. (5) For presentation purposes, traverse through the entire BST in order and put the requirements in a list, with the least important requirement at the end of the list and the most important requirement at the start of the list.

**11. Ranking based on product definition** [8]. This prioritization technique accounts for three important perspectives on product definition: the business, users, and technology. Using it, stakeholders rank the importance of each feature to the business, UX professionals rank the importance of each feature to the users, and technical analysts rank the feasibility of implementing the feature. Once they have all ranked the features, the product team combines the different rankings and compares them to identify the most important features. The success of this technique is contingent on the involvement of the right people providing input on the right aspect of each item on the list. This simple process provides a quantitative, de-personalized way to arrive at rational, actionable, and — above all — realistic launch priorities.

**12. Planning Game** The planning game is a feature of extreme programming [4] and is used with customers to prioritize features, based on stories. This is basically a variation of the Numeral Assignment Technique, where the customer distributes the requirements into three groups, "those without which the system will not function", "those that are less essential but provide significant business value," and "those that would be nice to have. The process is based on two criteria: business value judged by the customer and technical risk judged by the developers.

**13. Quality functional deployment QFD** [7], [10] is a structured methodology for taking into account customer needs (i.e., the "voice of the customer"). You create a product planning matrix known as the "house of quality," which reflects both what (customer needs) and how (designer needs).

**14. Wieggers' matrix approach.** Karl E. Wieggers [28] describes a semi-quantitative analytical approach that uses a simple spreadsheet model to help estimate the relative priorities for a set of product features. This approach distributes a set of estimated priorities across a continuum, rather than grouping them into just a few

priority levels. This prioritization scheme borrows from the QFD concept of customer value depending on both the customer benefit provided if a specific product feature is present and the penalty paid if that feature is absent. A feature's attractiveness is directly proportional to the value it provides and inversely proportional to its cost and the technical risk associated with implementing it. All other things being equal, those features that have the highest risk-adjusted value/cost ratio should have the highest priority.

**15. Mathematical programming techniques for release planning.** Li et al [19] use formal models to solve the release planning problem based on the precedence dependencies between requirements and the resources/skills constraints. The authors propose mathematical programming techniques that integrate requirement scheduling into software release planning and provide a requirement selection and on-time-delivery project plan simultaneously. This approach goes beyond the process of prioritization.

In addition to the above 15 techniques, our literature review revealed one practice, which can not be treated as separate method or technique, as it could be applied in combination with any other technique - the practice of bucketing requirements [21]. This means "bucketing" groups of major functionality or areas of task support is sometimes easier than feature by feature prioritization. The 15 techniques we've identified can be categorized in two main groups:

(i) techniques, directly comparing requirements pairwise, and

(ii) techniques that group requirements depending on their importance.

The analysis of the methods in the literature indicates that most of the methods are informal and subjective. We observe directions like: 'If the requirement is less important than the root node, compare it to the left child node', or 'participants assign each requirement a number on a scale of 1 to 5 to indicate the importance of those requirements'. These are indications of the assumption that those, involved in the prioritization process, know the importance of each requirement. The purpose of our research effort is to make this decision making process more explicit, objective and systematic, and to increase the awareness about the issues and challenges down the road.

## **4 Conceptual model of agile requirements prioritization**

Based on our review of agile prioritization approaches, we derived the following conceptual model (Fig 1.), which presents a generic prioritization process in terms of its inputs and outputs. The model takes explicitly the perspective of the client, unlike requirements prioritization authors (e.g.[19]) who adopt the perspective

of the development team. We must note that Fig 1 takes a ‘big-picture’ view to make explicit those pieces of information, necessary for the prioritization process.

To create this model we used descriptive coding [20] as the method of organization and analysis of the information we collected through our review. According to Miles and Huberman [20], coding is the part of analysis wherein the researcher differentiates and combines the information retrieved, and reflects upon the knowledge collected. In our study, this included first drawing diagrams and writing notes, then reviewing them and

dissecting them meaningfully, while keeping the relations between the parts (that are dominant concepts, themes, and issues) intact [20]. We followed this process, as it is meant to help the researcher to reduce and analyze data and direct him/her toward trends, themes, and patterns. Due to space limitation, we do not provide a mapping between the literature sources we used to as input to build the model and the parts of the model derived from each source. We, however, plan to publish this in a separate paper in near future.

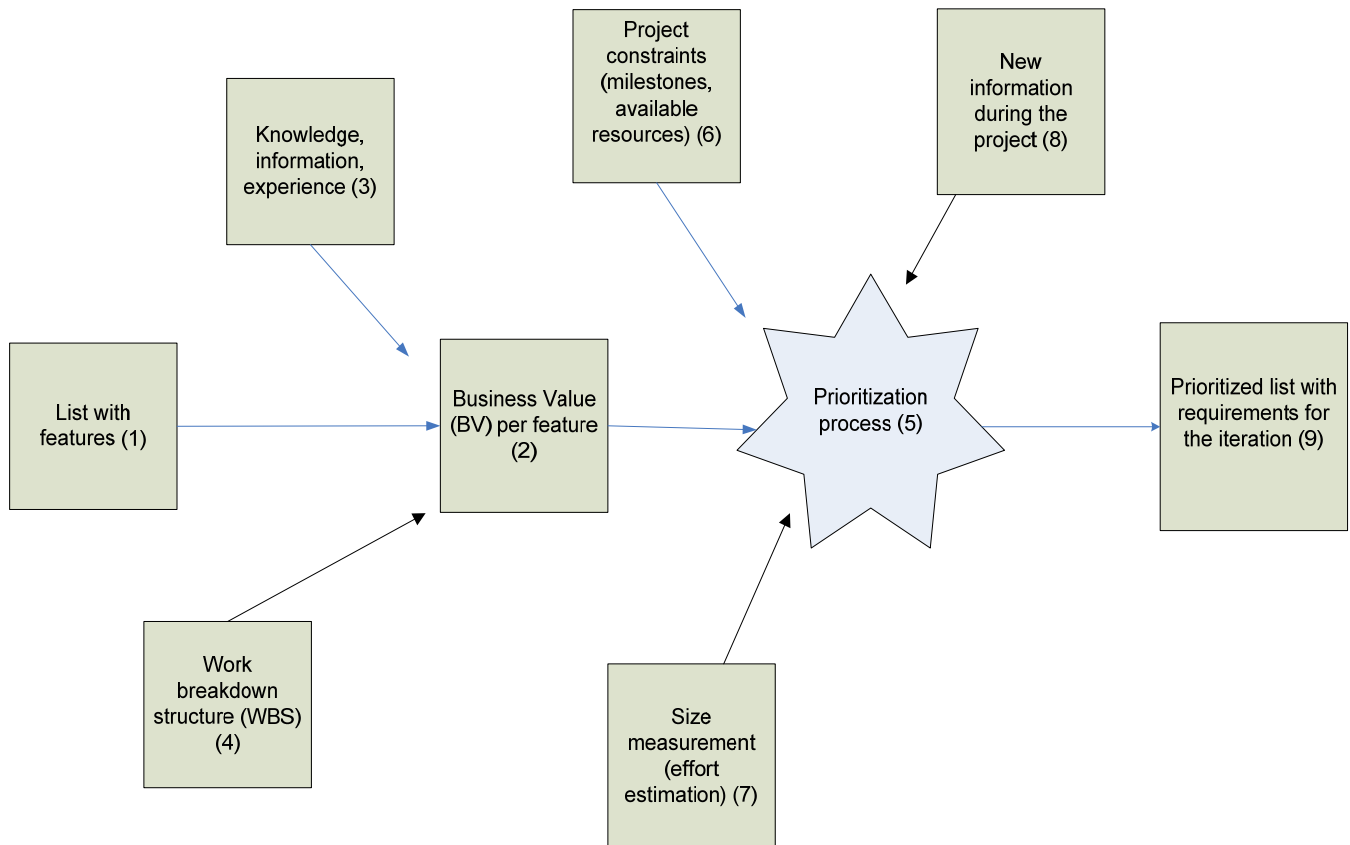


Fig. 1 The prioritization process: the clients’ perspective

We observe, that starting from a list with items (step 1), the prioritization process explicitly or implicitly - as a tacit knowledge, relies on estimation of the value of each item (step 2). Some of the methods used the notion of ‘importance’, or relative importance of a feature, compared to other requirements, instead of ‘value’. Still, we observe that estimations of ‘value’, knowledge about importance or business goals, fulfilled through each feature, are prerequisite for any prioritization process.

The star-like element in Fig 1 stays for a prioritization procedure, and could be replaced with any of the methods we have found in the literature sources. This means, the

prioritization process itself could employ any of the techniques presented in section 3.

The elements on the figure “Work Breakdown Structure”, “Size measurement”, “Project constraints” and “New information” (numbered 3,4, 6,7 and 8) represent external sources of information (or other non-client activities), that the client relies on in order to be able to make prioritization decisions, and that affect the outcome.

We make the note, that not all of these elements are necessarily present in each prioritization effort – i.e. some of them are optional depending on the project’s context or on the method used. For example, the Work Breakdown

structure is used when calculating Business Value as a relative value of a feature among its siblings.

In what follows, we identify issues and solutions that pertain to each of the boxes on our conceptual model. In this way, we use the conceptual model to structure an agenda for future research. The issues and solutions were identified based on the authors' experiences and the reviewed literature.<sup>1</sup>

## 5 Issues and solutions

The issues we identified in the process in Fig 1, and the solutions we propose, are the following:

**Issue 1 – related to step 2.** At the beginning of an iteration the Business Value (BV) of each feature has to be estimated (calculated). The challenge is to make the knowledge or information, used by the experts to perform the estimations, explicit. The sources of such information need to be identified, as well as the criteria that define one requirement as more valuable than another. This would allow for applying objective methods for comparing features, e.g. measurement techniques.

For the sake of simplicity, we'll not consider at this stage dependencies among features, but we are aware of their existence. This will form a cluster of research questions in our agenda for future investigation.

**Solution 1.** In order to make the decision-makers aware of the forces, defining the value of a feature, we propose that the following additional parameters be attached to a feature:

- weight among siblings - attached by the client based on the tree with features (e.g. WBS) and on expert knowledge. These might include marketing or other domain specialist.

- dependencies (e.g. chronological, architectural) – most of the methods described above usually do not take into account dependencies among requirements. Those methods which do acknowledge for dependencies are the ones which describe requirements on several levels of granularity (AHP). We propose that dependencies are considered multi-dimensionally – they might be chronological from developer's and client's perspective, as well as architectural in nature from developer's point of view.

We have made a difference between architectural and chronological dependencies, as we consider both clients' and developers' perspectives. A chronological dependency from clients' point of view could be different from chronological dependency from developers' perspective, as the first one is based on business reasons – and – processes, while the second one has technical

reasons and might require completely different sequence of implementing features. Furthermore, an architectural dependency should not necessarily be of purely chronological character – e.g. architectural decisions about development platform will influence further implementation of features or quality attributes.

- certainty factor - mirrors the certainty about the need to implement the feature at the present moment – this is a function of the feature's value and volatility, and reflects the level of information the client has about the value of the required functionality and the probability that the feature is affected by change in the environment. This factor might be a percentage or a number on a scale. For example, a feature that for sure is needed at the current stage, can have 90%, or '1' on the scale. This parameter has to be revised before each iteration. Thus we address Harris and Cohn's [11] proposition to defer features with high expected cost of change – the ones that are more likely to be changed can be deferred until more and better knowledge about how (or even whether) to develop them is gained. Our solution idea also converges with the solutions proposed by G. Ruhe et al [24] who investigated requirements prioritization in release planning for incremental software product development.

**Issue 2** refers also to step 2. Some user stories do not create business value directly (e.g. understanding what client wants), but are prerequisite for other stories.

**Solution 2.** Clearly, a practical approach should be to consider this as a chronological dependency. We identified two possible solutions to handling the dependencies: (i) Consider such user stories as married to the story that creates (max) business value, or to wrap these stories as a package – from outside is only one story visible, that is, separation is not possible. (ii) Introduce time-ranking within the stories of a feature. This should mean that stories with more-important rank will have to be implemented before the others, irrespective of their business value. We take these dependencies into consideration by introducing the Dependency – parameter in the description of features (e.g. story cards).

**Issue 3.** The quality attributes can not be separated from other features, yet they are of utmost importance for decisions about architecture or choice of implementation framework.

**Solution 3:** To consider them as functional requirement, where the main criteria is the *certainty* parameter. For example, if the client knows for sure, that the system will have many thousands users, the scalability will be addressed at earlier stage. If the estimation about a quality attribute is highly imprecise, this can be considered later in the project.

**Issue 4.** Variation of current BV. The business value of a requirement is different at different points in time. That

---

<sup>1</sup> The third author, Buglione, is actively involved in agile projects.

is, it varies throughout the project, as new information might arrive, changes in the market situation might happen, thus affecting the knowledge the team has about the value of a feature. Furthermore, the requirements volatility will require refactoring or rework, and the estimates for these activities have to be taken into consideration when calculating the business value. These activities are hidden for the client, as they are not explicitly included in the project's backlog or work breakdown structure; they don't provide functionality and, respectively, don't generate business value. Still, they can jeopardise the budget, the schedule, or the scope of a project when not taken into consideration early enough.

**Solution 4:** We propose that the current business value is determined (calculated) when taking into consideration the effort estimations, and this is performed before each iteration. I.e. using the formula:

*Business Value*<sub>feature n, iteration i</sub> =  $f(\text{initial business value, current cost estimate})$

We also mean that this estimation is performed dynamically each time prioritization happens. This innovative proposition addresses the observation made by other authors [6] that in agile context the implementation order is based mainly on the business value. We must acknowledge however, that at this point of time, we do not have any empirical data which suggests a specific form of the relationship among the initial business value and cost, and the business value at following iterations. Establishing such dependency is a topic of future investigations.

**Issue 5.** Deciding on a prioritization technique. The variety of prioritization approaches doesn't suggest that the choice is easy.

**Solution 5.** Similarly to traditional projects, the choice of a prioritization technique in agile projects will depend on the project's context and the previous experience and knowledge the project manager has.

The information, provided in Table 2, suggests that we can consider at least the following criteria when choosing a prioritization approach: (i) number of items to be prioritized, (ii) number of stakeholders involved, (iii) level of requirements volatility, (iv) sources of information available. Karlsson et al [16] state that the technique has to be supported by the software process and other project activities, thus the choice of a technique would depend not only on the time-consumption, ease of use and accuracy, but also on other contextual aspects, e.g. methods and tools used in the organisation.

Still, we assume that there might be other relevant parameters which should be identified and investigated. Whether and how the choice of prioritization approach will influence the results of prioritization is a research

question for future studies. A study by Karlsson et al [16] found that the two particular methods, compared based on 'ease-to-use' and 'time consumption' of the process, do not differ significantly regarding accuracy.

**Issue 6** Defining the scope of next iteration. This will depend on estimated current value of the requirements, and the amount of work that the developer is able to perform in one iteration (for example measured in story points). The choice of what to be implemented in the next iteration can be difficult to make, as different sets of requirements might compete for implementation at the same time, or some stakeholders may exercise over proportional influence.

**Solution 6:** We propose that a formal approach is used for decision making under uncertainty. We have presented such approach, based on elements of the real option theory, elsewhere [23]. Empirical studies show that for some projects an optimization-based prioritization approach yields better results [24].

## 6 Validation plan

As this is a work in progress, future research is needed in order to demonstrate the validity of the solutions we propose. We plan to carry out extensive interviews with experts in software companies, which can help to identify more issues and the applicability of the solution approach. The identification of additional drivers for prioritization decisions, used in practice, could contribute to enrich our conceptual model (Fig. 1). We plan in our immediate future to analyze the fit of the model by using it in case studies.

The relevance of the results to researchers and practitioners is to be judged regarding how the approach fits the situation, that is, whether it helps individuals familiar with the phenomenon (in this study, agile requirements prioritization) to make sense of their experience and to manage the situation better [13, 20].

## 7 Conclusions

This paper identifies and investigates issues and challenges in prioritization decision-making in agile projects. We reviewed existing agile requirements prioritization techniques and made a first attempt to derive a conceptual model for inter-iteration prioritization decision-making from the perspective of the client. We used this conceptual model to structure issues and solutions pertinent to agile prioritization of requirements. This paper is only the beginning of a series of efforts to improve existing agile prioritization practice. Our overall plan for the future is to develop a decision-support vehicle that helps clients complement value-based and cost-based prioritization criteria so that the prioritization is done in a more systematic and objective manner.

## References

- [1] Ahl, V. "An Experimental Comparison of Five Prioritization Methods." Master's Thesis, School of Engineering, Blekinge Institute of Technology, Ronneby, Sweden, 2005.
- [2] Ambler, S., Agile Master Data Management (MDM), 2007, URL: [www.agiledata.org/essays/masterDataManagement.html](http://www.agiledata.org/essays/masterDataManagement.html)
- [3] Augustine, S., Managing Agile Projects, Prentice-Hall, 2005, ISBN 0131240714
- [4] Beck, K. eXtreme Programming Explained: Embrace Change, Addison Wesley, 2000.
- [5] Berteig, M., Methods of Prioritization, March 20, 2006 in Agile Advice online practitioners forum, [http://www.agileadvice.com/archives/2006/03/methods\\_of\\_prio.html](http://www.agileadvice.com/archives/2006/03/methods_of_prio.html)
- [6] Cao, L., Ramesh B., Agile Requirements Engineering Practices: An Empirical Study, IEEE Software, Jan/Feb, 2008 pp. 60-67.
- [7] Crow, K., "Customer-focused Development with QFD", URL: <http://www.npd-solutions.com/qfd.html>
- [8] Fraser, J., Setting Priorities, April 23, 2002, URL: <http://www.adaptivepath.com/ideas/essays/archives/000018.php>
- [9] Getting Started With Use Case Modeling, An Oracle White Paper, May 2007 <http://www.oracle.com/technology/products/jdev/collateral/papers/10g/gswUseCaseModeling.pdf>
- [10] Gottesdiener, E., At a Glance: Other Prioritization Methods, EBG Consulting, Inc. [www.ebgconsulting.com](http://www.ebgconsulting.com)
- [11] Harris, R. S., M. Cohn: Incorporating Learning and Expected Cost of Change in Prioritizing Features on Agile Projects. XP 2006: pp. 175-180
- [12] Heller, G. Moving towards Agile Practice: Requirements Management Experiences at Hewlett Packard,
- [13] Herrmann, A., M. Daneva, "Requirements Prioritization Based on Benefit and Cost Prediction: An Agenda for Future Research", Proc. 16th Int'l Conf. Requirements Eng., 2008, Barcelona, Spain, 08-12th Sept 2008.
- [14] Johnson, J., Driving Requirements Down the Pipe, Software Magazine, March/April, 2007.
- [15] Johnson, J., My Life is Failure, Standish Group International, 2006.
- [16] Karlsson, L., Thelin, T., Regnell, B., Berander, P., Wohlin, C., Pair-wise comparisons versus planning game partitioning--experiments on requirements prioritisation techniques, empirical Software Engineering, Volume 12 Issue 1, 2007
- [17] Knudsen, L., Agile and You: a Product Manager's Guide to Being Successful in an Agile Environment, Hewlett Packard, May 2007, SMT Conference.
- [18] Leffingwell, D., Widrig, D., Managing Software Requirements: A Use Case Approach, 2nd ed. Boston, MA: Addison-Wesley, 2003
- [19] Li, C., Akker, J.M. van den, Brinkkemper, S. & Diepen, G. (2007). Integrated requirement Selection and Scheduling for the Release Planning of a Software Product. In Requirements Engineering: Foundations of Software Quality (REFSQ '07) (pp. 93-108). Springer-Verlag Berlin Heidelberg.
- [20] Miles, M., & Huberman, M. (1994). *Qualitative data analysis* (2<sup>nd</sup> ed.). Thousand Oaks, CA: Sage.
- [21] Patton, J., Finding the forest in the trees, Conference on Object Oriented Programming Systems Languages and Applications, 2005, San Diego, CA, USA, pp: 266 – 274, ISBN:1-59593-193-7
- [22] Principles behind the Agile Manifesto, 2001, URL: <http://agilemanifesto.org/principles.html>
- [23] Racheva, Z.,M. Daneva, L. Buglione, Complementing Measurements and Real Options Concepts to Support Iteration Decision-Making in Agile Projects, to appear in the Proceedings of the Euromicro08 conference, Sept.2008 Parma, Italy
- [24] Ruhe, G., J. McElroy, G. Du, A Family of Empirical Studies to Compare informal and Optimization-based Planning of Software Releases, Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, Rio de Janeiro, Brazil, pp: 212 – 221, ISBN:1-59593-218-6
- [25] Saaty, T.L., The Analytic Hierarchy Process, McGraw-Hill, New York, 1980.
- [26] Schwaber K., Agile Project Management with SCRUM, Microsoft Press, 2004, ISBN 073561993X
- [27] Tabaka, J., Collaboration Explained: Facilitation Skills for Software Project Leaders. Addison Wesley 2006. ISBN-13 978-0321268778
- [28] Wiegers, K., "First Things First: Prioritizing Requirements," *Software Development*, vol. 7, no. 9, Sept. 1999; [www.processimpact.com/pubs.shtml#requirements](http://www.processimpact.com/pubs.shtml#requirements)