Incorporating Learning and Expected Cost of Change in Prioritizing Features on Agile Projects

R. Scott Harris[1] and Mike Cohn[2]

[1] Montana State University–Billings
sharris@msubillings.edu
[2] Mountain Goat Software, LLC
mike@mountaingoatsoftware.com

Abstract. Very little has been written to date on how to prioritize and sequence the development of new features and capabilities on an agile software development project. Agile product managers have been advised to prioritize based on "business value." While this seems an appropriate goal, it is vague and provides little specific guidance. Our approach to optimizing "business value" uses tactics to minimize costs and maximize benefits through strategic learning. In order to provide specific and actionable advice to agile product managers, we present two guidelines. These guidelines are meant to provide a set of considerations and a process by which an agile product manager can achieve the goal of optimizing "business value" while recognizing that different product managers will vary in their notions of what "business value" is.

1 Introduction

Over the past seven years, agile software development processes such as Scrum [1], Extreme Programming [2], Feature-Driven Development [3], and DSDM [4] have emerged and their use has become much more prevalent. Central to these processes is a reliance upon emergent requirements and architecture. On an agile project, there is no upfront requirements engineering effort. Instead, the project begins with very high level requirements, often in the form of "user stories" [5]. The project team builds the software through a series of iterations and a detailed understanding of the requirements is sought only during the iteration in which software supporting those requirements is written.

A key tenet of agile processes is that these requirements are prioritized by a customer [2], customer team [6], or "product owner" [1] acting as a proxy for the end users of the intended system. Throughout this paper we will use the term product manager to represent this role independent of the specific agile process employed.

Product managers are given the relatively vague advice to prioritize based on "business value" [7][8]. Unfortunately, "business value" is both vague and broad whereas prioritization decision must be specific. Elsewhere, we have argued that product managers need to consider specific additional guidelines for prioritizing requirements on agile projects that lead to the fulfillment of maximizing "business value" [9]. This paper outlines those guidelines and discusses their implications for agile software development projects.

P. Abrahamsson, M. Marchesi, and G. Succi (Eds.): XP 2006, LNCS 4044, pp. 175 – 180, 2006.

2 The "Knowledge Problem" Facing Product Managers

Applying the work of Hayek [10], and Jensen and Meckling [11][12] to agile processes, we distinguish between "scientific knowledge" and "specific knowledge." The former is knowledge that is universal and can, for example, be taught in schools. In software development, knowledge of various programming languages and specific algorithms is "scientific knowledge." A challenge on any software development project is obtaining the "specific knowledge" regarding what the customer and users want. This is confounded by the fact that often users do not know precisely what they want and means not only that the customer and users must learn what they want, but that the product manager must also learn what they want.

Learning is the acquisition of knowledge. "Scientific knowledge" is learned outside of the immediate project while the bulk of "specific knowledge" must be learned during the development process and can be roughly divided into two categories: (a) learning what it is that users need and (b) learning the best way to develop software to meet those needs. Participatory design [13], essential use cases [14], and user stories [5] are techniques that have been developed to address the former; educated guessing and experimentation can be efficient ways to generate the latter. Because projects always will have emergent requirements that cannot be defined upfront, experimentation may be the cheapest way to learn what will work to satisfy a user's desires.

Others have studied the issue of prioritizing requirements and have concluded that Saaty's analytic hierarchy process (AHP) is "the most promising approach." [15][16][17]. Their focus is on upfront prioritization that implicitly assumes that ALL knowledge necessary to complete the project is given to the product manager at the beginning. Further, the focus has been on mechanics of the prioritization process and not on discussing the standards used that determine the priority order. Certainly for an agile project this is an overly simplistic view. Through its use of end-of-iteration reviews an agile team will learn more about the relative desirability of each feature and may even alter the criteria by which desirability is judged. This will (or should) alter any previous prioritization, thereby necessitating a new prioritization exercise. If it is anticipated that a significant amount of learning will take place as the project unfolds, expected repetitions of AHP or similar prioritization will be cost-prohibitive.

Our focus has been on how learning if project specific knowledge can affect product management. Any one-time upfront non-iterative approach to doing this ignores the crucial issue of learning. Therefore, we rejected the possibility of discovering or refining a static model to rank features in favor of suggesting guidelines for a dynamic process.

3 Guidelines for Prioritization

We define two issues of concern: "learning" and "the cost of change." We assert that early and low-cost acquisition of project specific knowledge and decreasing the cost of change positively impacts "business value." Though these two concepts are generally interdependent (i.e., the more one learns, the lower will be the cost of change), and related in a manner that depends on specific and particular features, we separate the issues to emphasize how to address each.

3.1 Guideline 1: Defer Features with High Expected Costs of Change

There are two aspects to what we call the expected cost of change for a feature. The first is the risk that a change will be needed; the second is the cost of making the change. The Expected Cost of Change (ECC) for a feature is the arithmetic product of the probability that change will be needed and the cost of making the change.

At any time on a project, every feature to be developed has an associated ECC. Each feature can be ordered from low to high. Those features that are both highly certain to remain unchanged throughout the project and that have a low cost of change will be the ones with the lowest ECC; those features that are very likely to change and that will impose a high cost to change will be the ones with the highest ECC. All others will fall in between.

When considering only ECC, we have demonstrated that total development cost can be minimized by developing features in order from lowest ECC first to highest ECC last [9]. This leads to our first guideline for prioritizing features.

It makes intuitive sense that if a product manager has a choice between developing features that are more likely to be changed and those that are less, it will lower overall expected costs if those that are more likely to be changed are deferred until more and better knowledge about how (or even whether) to develop them is gained. Additionally, one must consider the cost of change and defer developing those features that will be most costly to change. As the project progresses, project-specific learning will increase the probabilities that high cost-of-change features will be done correctly the first time thereby lowering the expectation of ever bearing that cost.

To implement this guideline, if one wants to plan to minimize the total expected cost of change over the scope of the project when learning takes place, sequential decisions will have to be based on (1) prioritizing activities that will have the greatest impact to lower the ECC of the deferred features and (2) deciding which remaining individual feature has the lowest ECC. In doing so, we should note that it is possible that these two criteria may not yield the same immediate priority activity. This possibility is discussed below.

Lowering the ECC of deferred features depends on the amount of specific knowledge that is generated during the immediate activity. Addressing that issue leads to our second guideline.

3.2 Guideline 2: Bring Forward Features That Generate Useful Knowledge

Just as different features will have different ECCs, each feature may have a different impact on learning. For example, developing one feature may greatly inform the product manager about the desirability of a feature set or the usability of the main user interface workflows. Developing different features will impart different amounts of knowledge to the developers creating the product. While the knowledge expected to be generated in any immediate activity will not affect the ECCs used in the prioritization calculations that decided features to develop in that immediate activity, it will affect the ECCs of delayed features. This means (a) the value of acquisition of knowledge can be viewed separately from the issue of ranking ECCs *given current levels of knowledge* and (b) "useful knowledge" may be prioritize by how it is expected to lower the ECC of the deferred features.

Prioritization based on these two guidelines may or may not agree regarding what the immediate next activity should be—in which case the product manager or agile team will have to employ additional criteria to sort out what should be done. However, the more important outcome is that prioritization using these guidelines will indicate a lot of features that should NOT be done immediately. Because the specification (and even the need for) the deferred features will be more nebulous than those to be developed immediately, learning that occurs in the immediate activity could—indeed, should—alter future prioritizations. Therefore, prioritization of features is only useful in deciding what should be done in the immediate next activity and what should be delayed. This leads to our third guideline.

3.3 Guideline 3: Incorporate New Learning Often, but Only to Decide What to Do Next

We cannot emphasize enough that learning is both important and a continuous and cumulative process that will change the priority of what is best to do next. This implies that a product manager and agile team must be nimble and constantly prepared to alter plans based on newly-acquired knowledge. Indeed, it should be clear that becoming wedded to a plan that is any longer than the next activity is both costly to formulate (if any time is spent on it) and could lead one in the wrong direction.

Because learning is a continuous process, decisions are both simplified and bounded. The sequence of decision-making only requires that one decide on the immediate project, user story, or feature to develop next and not concern oneself with the order of deferred activities. Sort the features into just two categories: what to do "now" versus "not now." Those features that are not done "now" will then be reevaluated for the next iteration when there is more knowledge upon which to base the evaluation. This is sequential planning where the "plan" is in the process and not the result. Without it, there is no agility in agile processes.

It should be noted that this guideline is consistent with and supports the agile preference for short iterations. While it is often useful to have a loosely-defined release plan covering the likely set of features to be delivered over the course of a small number of months, the detailed work of prioritizing and sequencing features should only be done an iteration at a time.

4 Implications

In this final section we consider an example of how these guidelines can be applied to the practical decisions of a project. These guidelines are presented to clients in both training classes and in consulting discussions. We have found it best to tell clients to perform a rough, initial prioritization of the desired features based on the nebulous "business value" provided by each. We stress that it is not necessary to prioritize all remaining features and normally guide product managers to plan two or three times as much as they expect the team to be able to complete in a single iteration. For these items product managers are given the guidance to think of expected cost of change and knowledge generated as "sliders" that can move a feature ahead or backward within the prioritization. Product managers then review the selected features sliding

them forward and back based on considerations of expected cost of change and expected knowledge generated.

Following this process, we find that features with architectural implications that will not have exceptionally high expected costs of change but that will increase knowledge dramatically can justifiably be developed in an earlier iteration than would be justified by prioritization solely on business value. We have applied the guidelines in this way to support the early selection of a particular application server. We have also used this on projects to justify the higher prioritization of features that influenced design approaches for a security framework as well as internationalization and localization. Similarly, when applied in this way, the guidelines can support the earlier development of features that generate significant learning about the main metaphors of the user experience being designed.

On the other hand, features with a high expected cost of change that will provide little new knowledge, should be deferred. By deferring such features we put their design off to the point where our knowledge about the product and system has increased and to where we can presumably make better decisions about those features with an initially high expected cost of change. Further, since developing these features would not provide significant new knowledge to the product manager or team, we are able to defer these features while foregoing no opportunities to learn. We have applied the guidelines in this way to a project struggling to choose between three competing client technologies. This decision was deferred while maximizing the team's learning through the development of other features.

Through the application of these guidelines on commercial projects we are able to provide more guidance to agile product managers than the conventional "prioritize based on business value." We have found that instructing them to consider relative changes in the cost of change and, more importantly, the amount of knowledge generated by the development of a feature leads to better decisions. Most importantly, the guideline-based approach described here requires very little effort and allows the product manager to make easier decisions such as "what one thing should be done next" rather than the harder "what is the full set of priorities." This more iterative approach to prioritization acknowledges that learning occurs throughout a development project and is more consistent with the agile management of software development projects.

References

1. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice-Hall, Upper Saddle River, NJ (2001).
2. Beck, K.: Extreme Programming Explained: Embrace change. Addison-Wesley, Upper Saddle River, NJ (1999).
3. Palmer, S.R., Felsing, J.M.: A Practical Guide to Feature-Driven Development. Addison-Wesley, Upper Saddle River, NJ (2002).
4. Stapleton, J.: DSDM: Business-Focused Development, 2nd edn. Pearson Education, Upper Saddle River, NJ (2003).
5. Cohn, M.: User Stories Applied for Agile Software Development. Addison-Wesley, Upper Saddle River, NJ (2004).
6. Poppendieck, T.: The Agile Customer's Toolkit at www.poppendieck.com.

7. Andrea, J.: An Agile Request For Proposal (RFP) Process. Proceedings of the Agile Development Conference, Salt Lake City, UT (2003) 152–161.
8. Augustine, S.: Great COTS! Implementing Packaged Software With Agility. Presentation at Agile Development Conference, Sydney, Australia (2004).
9. Harris, R.S., Cohn, M.: The Role of Learning and Expected Cost of Change in Prioritizing Features on Agile Projects, Ms (2006). Available at www.moutaingoatsoftware.com.
10. Hayak, F.A.: The Use of Knowledge in Society. American Economic Review, Vol. XXXV, No. 4 (Sept. 1945) 519–530.
11. Jensen, M.C., Meckling, W.H., Baker, G.P., Wruck, K.H.: Coordination, Control, and the Management of Organizations: Course Notes. Harvard Business School Working Paper #98-098 (October 17, 1999).
12. Jensen, M.C., Meckling, W.H.: Specific and General Knowledge, and Organizational Structure. In Werin, L., Wijkander, H. (eds.): Contract Economics. Blackwell, Oxford (1992). Also published in Journal of Applied Corporate Finance (Fall 1995) and Jensen, M.C.: Foundations of Organizational Strategy. Harvard University Press, Boston (1998).
13. Schuler, D., Namioka, A. (eds.): Participatory Design: Principles and practice. Erlbaum, Hillsdale, NJ (1993).
14. Constantine, L.L., Lockwood, L.A.D.: Software for Use. Addison-Wesley, Reading, MA (1999).
15. Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. IEEE Software, Vol. 14, no. 5 (1997) 67–74.
16. Saaty, T.L.: The Analytic Hierarchy Process. McGraw-Hill, New York (1980).
17. Karlsson, J., Wohlin, C., Regnell, B.: An Evaluation of Methods for Prioritizing Software Requirements. Journal of Information and Software Technology, Vol. 39, No. 14–15 (1998) 939-947.