Product and Release Planning Practices for Extreme Programming

Gert van Valkenhoef[1,2], Tommi Tervonen[1], Bert de Brock[1], and Douwe Postmus[2]

[1] Faculty of Economics and Business, University of Groningen, The Netherlands
{g.h.m.van.valkenhoef,t.p.tervonen,e.o.de.brock}@rug.nl
[2] Dep. of Epidemiology, University Medical Center Groningen, The Netherlands
d.postmus@epi.umcg.nl

Summary. Extreme Programming (XP) is an agile software development methodology defined through a set of practices and values. Although the value of XP is well-established through various real-life case studies, it lacks practices for project management. In order to enable XP for larger projects, we provide the rolling forecast practice to support product planning, and an optimization model to assist in release planning. We briefly evaluate the new practices with a real-life case study.

Key words: Planning, extreme programming, integer programming

1 Introduction

Extreme Programming (XP) is one of the most "agile" software development methodologies. Unlike plan-driven methodologies (e.g. waterfall) that define software development as a process, XP defines it through values and practices proven to work well together in real-life software development [1, 2]. A good project management process and strong customer involvement are critical to project success in XP [3]. Although XP provides a consistent set of practices, it almost completely lacks practices for planning [4]. Therefore, although XP has been reported to be tailorable for large-scale projects [5], it is generally considered more suitable for small projects. Moreover, the 'on-site customer' practice [1] is often hard to implement due to organizational or time constraints [6]. The XP customer is consistently under significantly more pressure than the developers or other participants in the project [7]. This causes the following problems (which become worse as projects get larger):

1. Lack of management context: XP does not address the larger context in which release planning takes place, or the long term project goals [4]. This means that the customer or the developers may loose track of the overall purpose of the system and consequently make sub-optimal planning decisions.
2. User story overload: the number of user stories to be considered in release planning can make the planning process too demanding for the customer.

3. Prioritization stress: the responsibility of prioritizing user stories may cause stress for the customer, even for a small number of stories. It is difficult to foresee the consequences and adequacy of the prioritization [7], and it is unclear whether the customer perceives business value in constantly managing the development priorities [8].

To address these problems, this paper proposes two new planning practices for XP. First, we assist in product planning with the new practice of rolling forecasts (Section 2). This practice helps to provide management context often lacking in XP (Problem 1 above). Second, we introduce an automated planning aid that can be used during release planning to reduce the customer workload by generating a suggested plan that satisfies simultaneously the constraints imposed by the customer and the limited development resources (Section 3). This addresses issues 2 and 3 identified above. After introducing the practices, we demonstrate their use in a real-life study (Section 4), before giving concluding remarks (Section 5).

2 Rolling Forecast for Product Planning

Expectation management is often the key difference between failed and succesful software projects [9]. XP originally proposes the 'system metaphor' practice for expectation management [1]. However, in practice, 'system metaphor' is difficult to apply and not useful, and is therefore often not implemented [6]. The 'system metaphor' has since been removed from XP [2], and there is no replacement practice addressing expectation management. The lack of an expectation management practice that is coherent with the rest of the methodology can cause additional project risks, especially if the customer is not constantly available on-site, as is often the case (see [6]).

Product planning should provide the context in which the release planning takes place [10]. In each release, before stories are elicited, the customers should have a rough idea of the current state of the system and the direction of development. This is promoted in XP by having the customer test and accept implemented stories and by frequently giving system demonstrations. However, it is unclear how a shared vision of the direction of future development can be established, especially when the customer does not clearly know what (s)he wants. As a consequence, an upfront rigid planning of the whole product in concrete terms is often almost impossible and can also become counter productive ('analysis paralysis').

To support product planning, we introduce the practice *rolling forecast*. At project inception, an overview of the *product goals* is drawn up by the customer together with the project manager. The goals should be stated in a functional format but in such a way that they cannot readily be broken into themes without further analysis and elicitation. The goals serve to provide a shared vision of the system and to form a basis for user story elicitation, but they are not requirements *per se*. It is advisable to re-evaluate the overall goals periodically, e.g. after every fourth release.

After defining the product goals, a *theme forecast* is created by the customer, project manager and a development team representative (e.g., an analyst or technical manager). A theme forecast consists of a set of themes, their likely implementation order, and a prediction of which themes will be realized in the coming two or three releases. The theme forecast can be adapted in preparation of every release planning (before story elicitation). Thus, a *rolling forecast* manages the expectations about the software by iteratively developing theme forecasts based on overall product goals. Then, in release planning, the theme forecast is taken into account when deciding on the themes and stories for the next release, while iteration planning takes into account (and adjusts) the release plan in choosing the stories and identifying the tasks for the next iteration, i.e., the normal agile planning practices are applicable at the release and iteration levels [10, 11].

3 Supporting Release Planning Model

Our planning model is aimed to support release planning. The developers elicit stories from the customer and ask him/her to evaluate them with respect to their business value on an interval scale, e.g. 1–5. Then the developers evaluate the stories' implementation complexity in story points. The model provides a planning aid by maximizing the implemented business value, taking into account constraints on implementation complexity and precedence relations. A precedence relation is interpreted as a story not having value unless another (preceding) story is implemented. Moreover, in XP, related stories are often grouped into themes that represent larger pieces of related user functionality, and synergy effects occur when all stories within a theme are implemented [2]. We model such effects by awarding extra value to a theme of stories if they are implemented together in a single release. Note that not all stories need to belong to a theme, and that one story can belong to more than one theme. We don't allow themes to span multiple releases in order to prevent the supporting planning model being used for making longer term plans, that might lower the overall agility of the XP development process. Longer-term product goals should instead be handled with the other proposed practice, rolling forecast.

Our model assumes adherence to the standard best practices regarding story and theme sizes. Stories should be small enough that they can easily be implemented in a single iteration, and themes in a single release. Moreover, a theme should consist of the *minimal set of stories* required to achieve the aforementioned synergy effect. Not adhering to these guidelines may lead to inappropriate results from the model.

The story selection can be formulated as a knapsack problem (the complete integer programming formulation is given in Model 1). Let us denote by n the number of uncompleted stories. Each story i has a business value of b_i and implementation complexity of c_i story points. The total amount of story points that can be implemented during a release is denoted by p. The decision problem is to select the most valuable subset of stories to implement in a release (Model 1:

Model 1 The optimization model as a side-constrained knapsack problem.

1.	max $b_1 x_1 + \ldots + b_{n+m} x_{n+m}$			
2.	s.t. $c_1 x_1 + \ldots + c_{n+m} x_{n+m}$	$\leq p$		
3.	$x_j - x_i$	≤ 0	for all i, j where $x_i \succ x_j$
4.	$\sum_{i=1}^{n} a_{ij} x_i - s_j x_{n+j}$	≥ 0	for $j = n+1, \ldots, n+m$
5.	$\sum_{i=1}^{n} a_{ij} x_i - x_{n+j}$	$\leq s_j - 1$	for $j = n+1, \ldots, n+m$
6.	$x_1, \ldots, x_{n+m} \in \{0, 1\}$.		

1), subject to a budget constraint on the maximum implementation complexity (Model 1: 2). For each story $i \in \{1, \ldots, n\}$, let $x_i = 1$ if story i is selected and $x_i = 0$ otherwise (Model 1: 6). Precedence of story i to story j is denoted by $x_i \succ x_j$ and can be incorporated into the optimization model by adding the following constraint: $x_j - x_i \leq 0$ (Model 1: 3).

To model themes, let m be the number of themes and let s_j ($j \in \{1, \ldots, m\}$) be the number of stories within theme j. Theme j can be included in the model by introducing a dummy story $(n + j)$, such that $x_{n+j} = 1$ if and only if all stories within theme j are implemented (Model 1: 4–5). The business value b_{n+j} associated with story $(n + j)$ represents the additional value that is awarded when all stories within theme j are implemented; its implementation complexity c_{n+j} is set equal to zero.

We implemented the supporting release planning model using R (`http://www.r-project.org`) and lp_solve (`http://lpsolve.sourceforge.net`). Our implementation is freely available online (`http://github.com/gertvv/xpplan`).

4 Real-Life Example

We are involved in a research project with external customers that expect us to develop software artifacts for the application domain of pharmacological decision support. Our development environment consists of 2 teams working part-time. In the following, we detail how we used the rolling forecast practice and our planning model in the development of ADDIS (see `http://drugis.org`).

Rolling Forecast. Although we didn't have clear requirements, we couldn't wait until the research results were present. In order to generate an overall view on the project and the first theme forecast, we interviewed the external customers of the research project. The initial forecast (Figure 1, top) was constructed considering 16 goals, such as "the system should provide drug efficacy and safety information". The theme forecast consists of a detailed set of themes for the next release(s) and a more global set of (likely) themes for the more distant releases. The forecast helped us to elicit stories in release planning meetings with the main external customer, who also chose the stories to implement. Figure 1 shows how our mutual understanding of the project evolved during the first half-year of development (only the most important themes are shown). We initially decided to focus on 'benefit risk' as a long-term goal. This defined our priorities

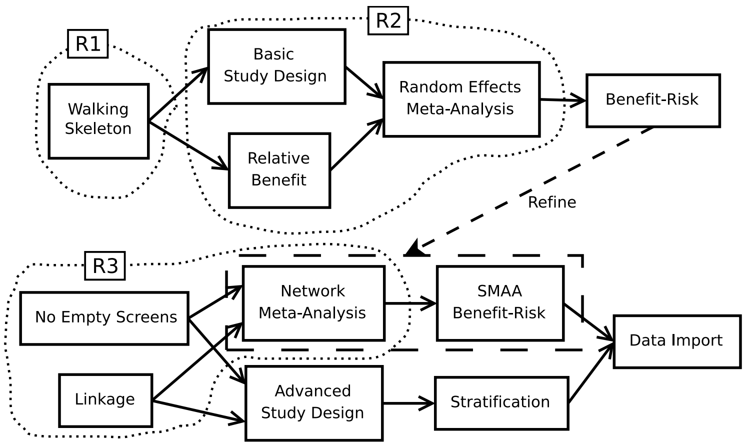

Fig. 1. The theme forecast for the beginning of the project (top) and the updated forecast (bottom) before release 3 (R3). A solid arrow from A to B indicates A has priority over B. The actually implemented themes from release 1 (R1) and 2 (R2) are shown, as well as the expected themes for release 3 (dotted lines). The dashed arrow indicates a high-level theme being refined as more information became available.

for the first two releases. We knew that to actually implement 'benefit-risk', research input would be needed. As these results became available only during the second release, the forecast was refined. Simultaneously, we were able to identify additional themes that also support our long-term goals, as well as two themes ('no empty screens' and 'linkage') that generate interest for our software through usability.

Planning Model. We did the first release (ADDIS 0.2) as a burn-in for velocity estimation and to create an initial end-to-end working system (also known as a 'walking skeleton', see `http://alistair.cockburn.us/Walking+skeleton`). Therefore we didn't estimate story business values while planning the first release. During the second release (ADDIS 0.4), we estimated story business values (scale: 1-5), story complexities (scale: 1,2,3,5,8), technical precedence relations (none were identified) and themes. In this release, we could identify 3 themes as being the most important. After the release was completed, we ran our supporting optimization model for release planning. We tested the sensitivity of the optimization model and differences between the model's solution and the stories we actually implemented by varying the theme value from 0 to 99. The results didn't differ much from our manually planned implementation order and the model showed to be robust with respect to changes in theme value: the only differences emerged when the theme value changed from 0 to 1 and from 10 to 11. When theme values varied between $1 - 10$ the same two out of three themes were included in the optimal solution whereas with theme value > 10 all three themes were included.

5 Conclusions

Lack of management context, user story overload, and prioritization stress cause high workload for the customer and hinder scalability of XP to larger projects. To overcome these limitations, we propose two new practices: rolling forecast for product planning and release planning support through an optimization model. We evaluated the applicability of our new practices in a software development project and found them useful. However, we do not have sufficient evidence to make claims about their suitability for projects with different customer profiles, numbers of developers, or levels of developer competency. Our ongoing development project cannot address these questions, and additional appropriate empirical studies should be initiated. Our future research will investigate how business value should be estimated for themes, and how uncertainty can be made explicit in the planning process.

Acknowledgements This study was partly supported by the Escher project (T6-202), which is a project of the Dutch Top Institute Pharma.

References

1. Beck, K.: Extreme Programming Explained. 1st edn. Addison-Wesley (1999)
2. Beck, K.: Extreme Programming Explained. 2nd edn. Addison-Wesley (2005)
3. Chow, T., Cao, D.B.: A survey study of critical success factors in agile software projects. Journal of Systems and Software **81**(6) (2008) 961–971
4. Abrahamsson, P., Warsta, J., Siponen, M., Ronkainen, J.: New directions on agile methods: a comparative analysis. In: IEEE Proceedings of the International Conference on Software Engineering, Portland, Oregon, USA (2003) 244–254
5. Cao, L., Mohan, K., Xu, P.: How extreme does extreme programming have to be? adapting XP practices to large-scale projects. In: Proceedings of the 37th Hawaii International Conference on System Sciences, Waikoloa, Hawaii (2004)
6. Rumpe, B., Schröder, A.: Quantitative survey on extreme programming projects. In: Proceedings of the Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, Sardinia, Italy (2002) 26–30
7. Martin, A., Biddle, R., Noble, J.: The XP customer role in practice: three studies. In: Agile Development Conference (ADC2004), Salt Lake City, Utah, USA (2004)
8. Grisham, P.S., Perry, D.E.: Customer relationships and extreme programming. In: HSSE '05: Proceedings of the 2005 workshop on Human and social factors of software engineering, New York, NY, USA, ACM (2005) 1–6
9. Boehm, B., Turner, R.: Balancing agility and discipline: a guide to the perplexed. Addison Wesley (2003)
10. Cohn, M.: Agile Estimating and Planning. Robert C. Martin Series. Prentice Hall PTR (2005)
11. Beck, K., Fowler, M.: Planning Extreme Programming. Addison-Wesley (2001)