

An Agile Request For Proposal (RFP) Process

Jennitta Andrea
ClearStream Consulting
jennitta@clrstream.com

Abstract

The Request For Proposal (RFP) process can be agile and efficient. At a high level, the key to achieving this is to specify requirements just in time and containing just enough detail. This paper applies the following XP practices and concepts to the RFP process: acceptance tests, business value, iterative & incremental delivery, on-site customer, pair development, planning game, spike, story, velocity, and yesterday's weather. In addition, the following concepts are combined with those from XP to achieve maximal benefit: user-goal use case, context diagram, level of detail, and decision tree.

The contributions of this paper to the agile community are two-fold: describing a practical application of XP concepts to a non-programming project; and making use case style requirements processes more agile.

1. Introduction

Current economic conditions and competitive realities are forcing many companies into an unfamiliar state of flux. IT organizations are facing the daunting challenge of delivering software systems while the supported business processes undergo radical change due to downsizing, mergers, or market expansion. Increasingly faced with reduced budgets, tighter timelines, and increased competition, companies are in search of strategies for working smarter and doing more with less. Among the list of successful strategies for achieving this end are purchasing package software, and employing agile methodologies.

1.1. Request for Proposal

Purchasing package software or outsourcing software development rather than building custom systems in-house are viewed as sound strategies for reducing the overall cost of ownership. Request For Proposal (RFP) is the business process a company executes in order to find the vendor and/or product that best meet their criteria. A simplified list of the high level stages in the RFP business process include:

Stage 1: Specification. Company specifies the system requirements (functional and non-functional) and general project constraints (time, cost, architecture, process, etc.) and issues an RFP document to candidate vendors.

Stage 2: Proposal. Vendor assesses the requirements and delivers their response, which includes their proposed solution, constraints, and cost estimates. If the RFP specifies local adaptations to industry standard business rules and processes, customization of an existing vendor product may be proposed.

Stage 3: Evaluation. Company evaluates the responses and selects a vendor based on a consideration of how well their proposal meets the requirements and satisfies the project constraints, notably time and cost.

Stage 4: Implementation. Company and vendor implement the solution.

1.2. Agile Methodologies

For in-house development, companies are experiencing the benefits of agile methodologies as a way to help reduce risk and successfully keep up with the pace of changing business requirements and priorities. Lean Manufacturing [1] summarizes how these goals are achieved: reduce waste (work efficiently) and reduce inventory (work just in time, produce just enough).

When requirements and business priorities are stable, it is considered prudent to look ahead at a up-coming requirements and proactively build flexibility into a software product in order to cost effectively support future capabilities. However, when priorities and requirements are in a state of flux, it pays off to be more reactive. Any work done beyond the current release horizon could end up wasted because tomorrow, brand new requirements may replace the ones that were important today. This is the heart of the philosophy behind agile methods.

Extreme Programming (XP) [2] is one of several agile methodologies, and is a sound strategy for minimizing the amount of work required in order to deliver a high quality, highly valued software product. XP can be summarized by its four core values: communication, simplicity, feedback, and courage. These values are supported by a set of practices: the planning game, small releases, metaphor, simple design, testing, refactoring, pair programming,

collective ownership, continuous integration, 40 hour week, on-site customer, and coding standards. XP thrives in a software development context consisting of a small, co-located team.

1.3. A Blind Date

What if we introduced these two strategies (RFP and agile) to each other? Are they compatible? Would their union achieve synergy and yet preserve their original essence? When should we play matchmaker? When should we stay well enough alone?

This paper explores the application of agile concepts and key XP practices to a non-programming project, the RFP process, to enable making the best product choice while optimizing scarce resources: time and money. The ideas contained in this paper are based on several experiences. The core of the content stems from a project initiated to assess a short list of vendor products being considered as a replacement for an aging, homegrown, safety-critical system. The sophistication and criticality of the system meant that the effort for the RFP requirement specification stage was expected to be substantial. Thus the company explored ways to streamline RFP process, finding a balance between ensuring that all requirements were accounted for, and the level of detail necessary to properly evaluate the candidate vendor products. These initial ideas were further refined after the author experienced the RFP process from the other side of the fence – as a vendor responding RFP's rather than as a company issuing them. These other experiences shed light on how difficult it is to produce an accurate estimate when the requirements within a RFP are too sketchy. All of these experiences have been further refined by reflecting upon how they can be improved with the correct application of established agile practices.

Section 2 covers techniques for optimizing the RFP specification stage, and introduces the concepts of levels of detail, decision tree, and embedding stories into use cases. An example is introduced that is used throughout the rest of the paper. Section 3 describes an agile approach to the RFP specification stage by adding other XP practices to the techniques introduced in Section 2. Section 4 describes an agile approach to the RFP evaluation stage, in particular for evaluating package software compliance to the requirements.

2. Optimizing the RFP Specification Stage

2.1. Use Cases vs. Stories

A typical RFP specification process and a typical agile requirements process are at opposite ends of the spectrum

in terms of sequencing of tasks and the format and content of the requirements.

By necessity, the RFP specification process is a big-up-front-requirements activity. The RFP issued to the vendors must contain all of the system requirements that the company expects the vendor to support. Use cases [3] are a common form of documenting functional requirements for an RFP. A use case captures the essence of a users interaction with the system to achieve a measurable business goal. A use case succinctly describes all of the possible scenarios for this interaction (main, alternative, and failure), and describes the expected outcome. Use cases appear to be out of favor with XP proponents, possibly because they tend to be developed in a waterfall fashion.

XP introduces the concept of stories as a form of functional requirements specification that easily adapts to changing business priorities. A collection of stories is identified for the current release of the system. Initially the stories are sketchy, intended to be 'promises for conversations' that will occur between the customer and developer within the release. During these just in time conversations, the requirements for a story will be explained to the level of detail necessary to enable the developer to implement and test the story. It is not considered necessary to document the requirements in a form other than source code. Stories for future releases remain sketchy until they are actually going to be implemented. Stories are effective for planning and depicting the essence of system requirements, given the ideal situation of having the customer on-site.

2.2. Use Cases Meet Stories

Use cases are appropriate for formally documenting the functional requirements of a system because all related scenarios are succinctly grouped together, providing a big-picture overview. However, use cases are not a good unit for planning. Typically, the various scenarios belonging to the same use case have different priorities, thus, a software release does not develop an entire use case at once. In addition, not all scenarios in the same use case need to be described to the same level of detail. Stories, on the other hand, are a more appropriate unit for planning. Each story, by definition, has a uniform priority, and is small enough to be developed within a single software release. Unfortunately, stories do not have the level of detail or rigor needed for a situation, like a RFP, where the customer and vendor are separated by time and distance, and must rely on documentation rather than conversations.

What happens when we take the best of both worlds? There is no reason why use cases have to be developed in

a pure waterfall fashion, even for the RFP process where all requirements are needed up-front. Instead of specifying all use cases to the same level of detail, a decision-making strategy can be used to determine the minimum detail needed for each use case in order to base a credible estimate upon it. Furthermore, the concept of story can be embedded within a use case to facilitate a finer grained decision-making process. Finally, the concept of business value can be applied effectively to the planning of the requirements specification process.

2.3. Levels of Detail

The Agile RFP Process begins with clear definitions of the various target levels of detail for capturing the use cases. The example use case shown throughout this section is based on the specification of a small system that is used by hospitals to archive inactive patient records (called volumes). A patient is identified by a chart number, and may have one or more volumes. A volume is essentially a file folder that contains treatment details for a patient. If a patient has not received treatment for five or more years, their volume is considered to be inactive. Before boxes of inactive volumes are physically sent to an off-site storage location, a user enters the data associated with each volume into the system. The use case that supports this is called Create Volume. Other use cases associated with this system support the correction of data entry errors, and the tracking of the location of the volume and box as it moves from point to point. Note: this system was not actually part of an Agile RFP Process as described in this paper, but it is one of the few real examples that the author is able to make public.

2.3.1. Identity Level. This level identifies the system behavior. A use case at this level contains enough information to allow the reader to unambiguously recognize the requirement, assuming the reader already understands it. This level is typically used for identifying system scope and for preliminary assessment of the existence of behavior in an existing system. Due to the lack of details, it is not recommended as the basis of estimation or custom development. The use case consists of the following elements:

- Description of the users goal
- Expected system state before the use case begins
- Expected system state after the use case succeeds or fails

For example:

Create Volume: A Volume is entered into the system for

the first time. The user is prevented from creating a duplicate volume. If the box and/or chart specified by the user do not already exist, the system will create them as a side effect.

Pre-Condition	<ul style="list-style-type: none"> • nothing
Success	<ul style="list-style-type: none"> • Volume created (archived state) and associated with the specified Box, and Chart. • Box / Chart is created if it did not exist previously. • User name and date of action placed in audit log.
Failure	<ul style="list-style-type: none"> • nothing

Example 1: Identity Level

2.3.2. Outline Level. This level provides a sketch or outline of the requirement. Typically this level is useful for facilitating a quick estimate when there is significant prior experience in the subject area. The use case consists of all the things contained in *identity level* (shaded in grey and truncated by ellipses), as well as:

- The full main scenario to one level of detail
- The list of important failures, without resolution details
- The list of important alternatives, without details

The standard use case template is modified slightly to accommodate the overlaying of stories onto the use case. For example:

Create Volume: ...	
Pre-Condition	...
Success	...
Failure	...
Story	Description
S1.Basic Functionality	<ol style="list-style-type: none"> 1. User enters volume information. 2. User requests that the information be saved. 3. <i>System validates information. [F2. User Input Validation]</i> 4. System saves information in a new Volume and places Volume in archived state. 5. System adds the Volume to the designated Box. 6. System adds the Volume to the designated Chart. 7. <i>System records the transaction in the audit log [G1. Audit]</i>
S2. User Input Validation	[3] Duplicate Volume
	[3] Missing information

G1. Audit	[7] Log Transaction
S3. Auto-create	[5] Box doesn't exist
	[6] Chart doesn't exist

Example 2 Outline Level

The main scenario represents a single story. For completeness, all steps for the main scenario are described, even though some are optional (e.g. steps 3 and 7). One or more alternate / failure scenarios are grouped together into a story. Some stories span multiple use cases (e.g. Audit). A different numbering scheme may be used to consistently reference the general stories across all of the use cases (e.g. G1 = general story 1).

To further reduce ambiguity and misinterpretation, an initial business object model may be developed to create a common glossary. The details for the important business rules are also documented. Only the concepts mentioned in the use cases and business rules are defined in the glossary.

2.3.3 Detail Level. This level provides the reader with the full details of the system behavior. Typically this level is useful for facilitating an estimate when there is some prior experience in the subject area. This is commonly viewed as the minimum level needed to base out-sourced custom development on. The use case consists of all of the things contained in *outline level* (shaded in grey and truncated by ellipses), as well as:

- All of the failures, with resolution details
- All of the alternatives, with details

For example:

Create Volume: ...	
Pre-Condition	...
Success	...
Failure	...
Story	Description
S1.Basic Functionality	...
S2. User Input Validation	[3] Duplicate Volume a) System detects another Volume with the same chart number, volume number and site. b) System informs the user of the error. c) Use case terminates.
	[3] Missing information a) System informs the user of the missing fields. Use case terminates.
G1. Audit Log	[7] Log Transaction

	a) System asks Logging System to record the transaction providing: creation user and creation date, and sets version number to 1. b) Use case resumes after step [7]
S3. Auto-create	[5] Box doesn't exist a) System creates Box b) System records transaction in the audit log. [G1. Audit] c) Use case resumes at step [5]
	[6] Chart doesn't exist a) System creates Chart b) System records transaction in the audit log. [G1. Audit] c) Use case resumes at step [6]

Example 3 Detail Level

The business object model is enhanced as appropriate. All of the business rules are documented.

2.3.4. Acceptance Tests Level. This level adds specific acceptance test cases for each story contained in a use case specified at *detail level*, thus enabling the recipient to unambiguously determine whether the requirement has been met. This level is valuable when assessing a system's compliance with a requirement. This is the recommended level for estimation of custom development when there is little or no prior experience in the subject area. A beneficial side effect of developing the acceptance tests is to find holes in the use cases. The use cases can then be updated to be more rigorous and correct.

The number of acceptance tests for a story is a function of the number of input parameters and the number of different pre-condition states that apply. A minimum set of tests that provide full coverage can be constructed through careful analysis of the unique and significant combinations of these variables. For example, the list of tests for the sample use case is:

Story	Tests
S1.Basic Functionality	Basic success test
S2. User Input Validation	Duplicate volume test
	Invalid input tests (one per input field, and one per combination of fields)
G1. Audit Log	Enhance the other tests a new post condition indicating how the audit log should have been changed
S3. Auto-create	Auto create chart test
	Auto create box test

	Auto create box and chart test
--	--------------------------------

Example 4 List of Tests for Create Volume

An acceptance test should be written declaratively, as it is a specification for setting up the preconditions and verifying the expected results [5]. It should not include any details relating to how these activities are done, leaving all options open to how the data is created (real-production data, pre-created local / global test fixtures, clean slate approach, etc) and how the tests are performed (e.g. manually, or automated through any number of mechanisms, e.g., FIT, xUnit, record & playback, etc). An example specification for one of the acceptance tests listed above is:

S1. Basic Functionality: Basic Success Test	
Pre conditions	
Box1	Exists, contains Volume1
Chart1	Exists, contains Volume1
Volume2	Does not exist
Process Parameters	
Box #	1
Chart #	1
Volume #	2
Patient Name	<unique string>
Year Last Contact	<ten years ago>
Archive Date	<today>
Post conditions	
Volume2	Exists with appropriate patient name, year of last contact and archive date
Box1	Contains Volume1 and Volume2
Chart1	Contains Volume1 and Volume2

Example 5 Acceptance Test

2.4. Decision Making Strategy

Acceptance tests level represents much more detail and several orders of magnitude more effort than *identity level* one, so it is important to have an effective strategy for assigning a target level to each use case. A number of different forces should be considered when making the assignment, such as the: familiarity, uniqueness, complexity, business-criticality, and life-criticality of the requirement. These considerations are typically linked together into a decision tree to help guide the decision making process. Each RFP process must review the forces relevant to their circumstances and define a decision tree accordingly, as the next two examples illustrate.

Figure 1 illustrates a simple decision-tree that is used to determine the level of detail for each requirement for a custom out-sourced system. The driving force behind the decision making process is ensuring the vendor has enough information to respond with credible, accurate estimates.

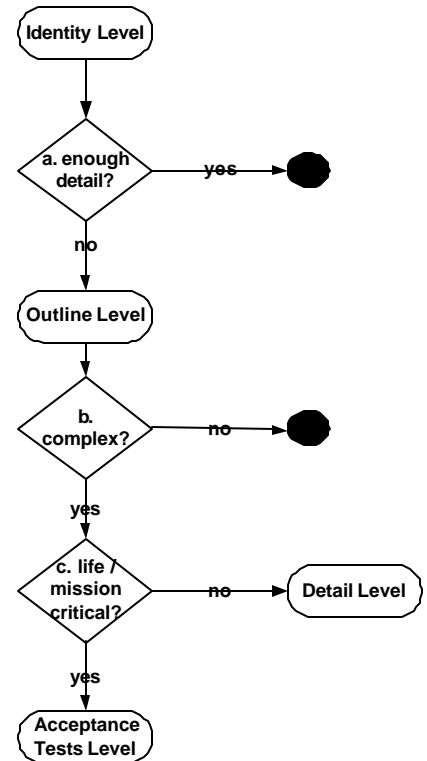


Figure 1 Custom Development Decision Tree

All use cases for the target system are initially developed to *identity level*. There is no need to further develop common stories (e.g. save, print, open, etc) beyond this first level (a). Unique stories that are easily understood (b) are described in less detail than those that are more complex (c). Life / business critical requirements (c) are specified at the greatest level of detail. The number of requirements defined to each level of detail varies from system to system, depending on the nature of the system itself.

Figure 2 depicts a different decision-tree used for making the final vendor product selection from a short list of vendors / products. The driving force behind the decision making process is to facilitate answering two key questions. The first question is: how much information is required to determine whether the vendor product meets a requirement? The follow up question is: if the vendor product does not meet the requirement, how much information is required to produce an accurate estimate for custom development?

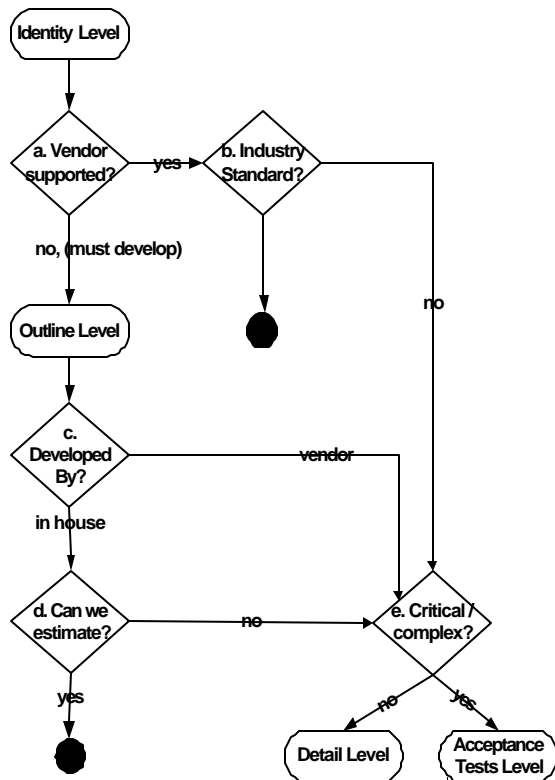


Figure 2 Package Software Decision Tree

As before, all use cases for the target system are developed to *identity level*. The first question, assessing vendor support of the requirement (a), is answered at a high level, for example by reviewing product documentation or talking to the vendor representative. Industry standard stories (b) that are well known, and widely supported are minimally specified. Unique stories, which require custom development (c), are specified in greater detail. In-house development of familiar functionality (d) requires less detailed specifications because prior experience can be used to guide the estimates. Highly complex or life / business critical requirements (e) are specified at the greatest level of detail.

3. An Agile RFP Specification Process

If applied judiciously, the target levels of detail and the decision tree can prevent the waterfall approach to the RFP specification stage. The agility of the RFP process can be further increased by marrying the concepts of use case and story and by applying XP planning game concepts to the requirements specification stage. A high level summary of the Agile RFP Specification Process is:

Step 1: Sketch. Define the breadth of the system's capabilities at a high level.

Step 2: Prioritize. Assign relative business value to each of the requirements as a way to prioritize the rest of the work.

Step 3: Evaluate. Assign target level of detail to each requirement in order to develop the requirement to just enough detail for the purposes of the RFP.

Step 4: Complete. Finish writing each requirement to the target level of detail. This work proceeds in priority order during time-boxed iterations.

The three key roles in this process are customer, business expert, and requirements analyst. The customer knows the business priorities for the system and makes strategic planning decisions. The business expert understands the requirements and can articulate them. The requirements analyst is skilled at facilitating and recording the requirements as use cases / stories. Several passes will be made over the list of use cases in order to properly account for the orthogonal concepts of business value and level of detail. Business value determines *when* a requirement will be captured, level of detail determines *how much* of the requirement is captured.

3.1. Sketch the System

The first step is to sketch out the skeleton of the system in order to clearly understand the breadth of its capabilities. Develop a simple context diagram [4] to identify all of actors (human and computer) that will interact with the system. Capture the goals of the actors as user-goal use cases [3] written at the identity level, as previously described. The user-goal use cases summarize the core capabilities of the system that satisfy measurable business goals in that they automate the steps of a business process. As a rule of thumb, expect roughly ten to thirty user-goal use cases for a typical business system. For example:

Actor	Goals	User Goal Level Use Cases
Archiver	Enter data	<ul style="list-style-type: none"> • Create Volume
Quality Control	Verify data entered correctly	<ul style="list-style-type: none"> • Create Volume • Update Volume • Update Chart • Update Box
Supervisor	Send box to storage	<ul style="list-style-type: none"> • Update Box State
Manager	Track progress	<ul style="list-style-type: none"> • Get Activity Report
Requestor	Retrieve archived volumes / boxes	<ul style="list-style-type: none"> • Update Volume State • Update Box State

Example 6 User Goal Level Use Cases

3.2. Prioritize the Requirements

The customer ranks each use case in terms of its overall business value. At this stage of the requirements planning game, the use case as a *whole* is assigned business value. The priority of a requirement determines when it will be specified. The highest priority requirements will be developed first; if time runs out the lower priority requirements may not be developed to their actual target level. The fact that the use case contains multiple scenarios, each of which potentially has a different business value is taken into consideration later if necessary. An example of a simple business value priority system is:

- Must Have (High): essential functionality to support the core business goals; the system has no value without it.
- Should Have (Medium): necessary functionality but not core; important functionality that improves the efficiency of the business.
- Could Have (Low): functionality that makes the system easier to use; there are alternate ways to meet the need (some of which are manual).
- Not Needed: functionality is not required. These use cases should be dropped from the list immediately.

For example:

Use Case	Business Value
Create Volume	must
Update Volume	must
Update Chart	must
Update Box	must
Update Box State	should
Update Volume State	should
Get Activity Report	could

Example 7 Assign Priority to Use Cases

3.3. Assign Target Level of Detail

According to their business value priority order, the use cases are again reviewed and assigned a target level of detail using the decision tree as a guide. Many different stakeholders will be involved in this decision-making process including: customer, business expert, vendor, and internal IT staff. For a large system with many scenarios associated with each use case, it's worth employing a multi-step approach to the decision making process. The first step is to determine the level of detail for each use case. The following example extends the previous one by adding the 'Level' column:

Use Case	Business Value	Level
Create Volume	must	acceptance tests
Update Volume	must	acceptance tests
Update Chart	must	outline
Update Box	must	outline
Update Box State	should	identity
Update Volume State	should	identity
Get Activity Report	could	identity

Example 8 Assign Level of Detail to Use Cases

All use cases that are assigned *identity level* are now essentially finished. Next, develop the remaining use cases to the outline level of detail in order to expose the stories contained within the use case. Finally, use the decision tree to assign a target level of detail for each story, instead of a single level of detail for the use case as a whole. The following example extends the previous one by including the stories and assigning the level of detail to each story:

Use Case / Story	Business Value	Level
Create Volume	must	
S1. Basic Functionality		detail
S2. User Input Validation		detail
S3. Auto Create		acceptance tests
Update Volume	must	
S4. Basic Functionality		detail
S5. User Input Validation		detail
S6. Auto Create		acceptance tests
Update Chart	must	
S7. Basic Functionality		detail
S8. User Input Validation		outline
Update Box	must	
S9. Basic Functionality		detail
S10. User Input Validation		outline
Update Box State	should	identity
Update Volume State	should	identity
Get Activity Report	could	identity
G1. Audit	should	detail

Example 9 Assign Level of Detail to Stories

Even for a very simple example like this, the finer grained level of analysis results in a considerable timesavings in the overall requirements specification stage. Notably, the initial assessment of the Create

Volume use case resulted in *acceptance tests level*. Further investigation indicated that only in one out of four stories actually requires acceptance tests – a substantial savings in time.

3.4. Complete the Requirements

The team now has enough information to develop a release plan for the RFP specification stage. Consistent and credible estimates for the RFP Specification stage can be achieved by creating benchmark estimates, which are established through a requirements spike. The time taken to capture one of the typical use cases / stories to each of the target levels of detail is recorded. In XP jargon, these benchmark values become yesterday’s weather. The remaining requirements are estimated based on a complexity comparison – use the benchmark estimates if the complexities of the requirements are comparable, otherwise adjust the estimate accordingly.

The remainder of the planning game follows fairly closely to the original. Use cases / stories are ‘fit’ into a time-boxed release based on business value and estimated time. If the estimates exceed the allotted time, a finer grained approach to planning is required. The high priority use cases targeted to *outline level* or greater are given a finer grained priority. Each story within the use case is assigned it’s own business value. The following example extends the previous one by assigning business value to each story:

Use Case / Story	Business Value	Level
Create Volume		
S1. Basic Functionality	must	detail
S2. User Input Validation	should	detail
S3. Auto Create	must	acceptance tests
Update Volume		
S4. Basic Functionality	must	detail
S5. User Input Validation	should	detail
S6. Auto Create	must	acceptance tests
Update Chart		
S7. Basic Functionality	must	detail
S8. User Input Validation	should	outline
Update Box		
S9. Basic Functionality	must	detail
S10. User Input Validation	should	outline

Update Box State	should	identity
Update Volume State	should	identity
Get Activity Report	could	identity
G1. Audit	should	detail

Example 10 Assign Priority to Stories

While the stories will continue to be packaged together into a single use case document, they may be developed at different times in addition to their different levels of detail. Now the finer grained list is re-sorted based on business value and the release plan negotiations resume. The release is broken down into iterations to enable refinement of the plan based on measured velocity.

The actual work of requirements specification proceeds with the customer on-site for interview sessions and just-in-time consultation. The concept of pair-development is also relevant to the requirement specification phase. Pairing two customers having knowledge of the same subject area improves the completeness and accuracy of the information given to the business analyst. Pairing business analysts as they document the requirements improves the quality and consistency of the use cases.

It is useful to capture the requirements as an XML document, and create at least two XSL style sheets to format the information. One style sheet creates a summary of the requirements, containing use case / story name, priority, and a link to the detailed information. This style sheet may also group or sort the stories independently of the use case they are associated with, for example the stories may be sorted based on their priority. This summary view acts as the vendor’s worksheet, and may include other columns, for example, vendor estimate. The other style sheet formats each use case as a whole, with each story taken to its target level of detail, much like Examples 1 – 4.

4. An Agile RFP Evaluation Process

The principle focus of this paper is on the requirements specification stage of the RFP process. This section provides some high level guidance for optimizing the evaluation stage when the RFP is focused on selecting package software. The evaluation stage of an RFP for selecting package software can be time consuming, as the vendor system is evaluated for compliance against the stated requirements. In these cases, it is advisable to take advantage of the business value and target level of detail assigned during the specification stage. Use cases / stories with high business value are evaluated first. These requirements are frequently associated with go/no-go decisions; the remainder of the evaluation may be

terminated if the vendor does not satisfy one or more of these key requirements. Business value also determines who does the assessment: the company typically assesses high business value requirements; the vendor may be asked to self-assess medium or low business value requirements.

The level of detail associated with the requirement suggests the techniques and level of rigor used to assess compliance. For example:

Identity Level. Reading the available product literature and talking to a vendor representative may be sufficient for determining compliance.

Outline Level. Hands-on usage of the product is performed to verify that the main scenario of the use case is properly executed. Simple verification is performed to ensure that the product detects the failure conditions; the handling of the failure conditions is not emphasized.

Detail Level. In addition to the assessment done at *outline level*, the product is used hands-on to verify the proper handling and resolution of: failure conditions, alternate scenarios and the stated business rules. This verification is ad-hoc because formal acceptance tests have not been developed.

Acceptance Tests Level. The acceptance test suite is executed. This involves setting up the data required for the specific scenarios, and using the product to execute the story under test and verify the results. If time, money, and technical considerations permit, the acceptance test suite may be automated to ensure continual compliance as the product evolves.

The results of the assessment are recorded according to whether there was a fit (full compliance of the requirements), or a gap. A gap can be more explicitly categorized as: partially met (part exists and is correct, part does not exist), incorrect (exists but is incorrect), and does not exist. The final stage of the vendor evaluation is the analysis of the list of complete and partial gaps. The business value associated with the unsatisfied requirement guide the assessment of the gap.

The gaps for the highly valued requirements are examined first. These requirements must be met, thus time is allotted to considering if they can be satisfied. Some complete or partial gaps may have to be filled by custom development. The XP planning game can be employed to explore and estimate the cost of this custom work. The developer role for the planning game could be played by internal IT staff, the vendor and/or third party organizations, depending on who is best suited and/or most cost effective to do the work. The use case may have to be taken to more detail in order to facilitate the estimation process.

5. Conclusions

First, a recap of the questions the paper set out to address.

What if we introduced these two strategies (RFP and agile) to each other? Are they compatible? Would their union achieve synergy and yet preserve their original essence? The best qualities of two different requirements approaches were merged together. Stories were embedded within the structure of a use case to facilitate fine-grained control of the timing and level of detail of the requirements specification phase, while preserving the detail and big-picture inherent in a use case. This prepared the way for using the planning game (business value, estimating, velocity, yesterday's weather, releases, iterations, spike, etc) to bring more rigor and accuracy to the planning of a requirements-centric project. Furthermore, it enabled the process to adapt to changes in requirements and/or priorities as they occur during the requirements specification stage. Introducing many of the other non-programming XP practices in the RFP process substantially increased the efficiency and quality of the RFP document. On-site customer and pair development improve the quality of the information captured in the requirements specification. And finally, including acceptance tests in RFP as appropriate, substantially improves the communication of critical requirements.

When should we play matchmaker? When should we stay well enough alone? When contemplating the purchase of a vendor package that represents a significant investment, the company's choice must be made carefully based on an accurate vendor response. Hence the requirements included in the RFP must be complete and unambiguous. At the same time, a cost-conscious company aims to streamline the RFP process as much as possible. The Agile RFP Process attempts to satisfy the competing goals of comprehensiveness and cost effectiveness while specifying the requirements for an RFP.

The good news is that the RFP process can be made more cost effective through the application of agile practices. This paper also presents the case that better vendor choices are made as a result of this approach because there is specific guidance about how to specify and evaluate the key requirements. Finally, although XP was intended for programmers, the concepts and principles are more widely applicable. The Agile RFP Process is just one example where a non-programming activity can make use of the insights and practices offered by XP.

6. Acknowledgements

The author would like to thank the clients whose projects have provided the opportunities to gain the experiences described here. Special recognition goes to Peter Arato and Michael Davis from Canadian Pacific Railway: their engaging questions and insightful comments are at the core of these ideas. Special thanks also goes to The Calgary Health Region (Gladys Glowacki, Peggy Colborne, Diane Talbot, Carmen Kubbernus and Ron Fitzpatrick) for permission to use excerpts from the Offsite Chart Tracking Application analysis model to as the basis of the example used throughout the paper. The author also thanks the colleagues who reviewed and commented on drafts of this paper, in particular Gerard Meszaros and Lougie Anderson.

7. References

1. Ono, Taiichi, *Toyota Production System: Beyond Large-Scale Production*, Productivity Press, 1988.
2. Beck, Kent, *Extreme Programming Explained*, Addison Wesley, 2000.
3. Cockburn, Alistair, *Writing Effective Use Cases*, Addison Wesley, 2001.
4. Fowler, Martin, *UML Distilled: Applying the Standard Object Modeling Language*, Addison Wesley, 1997.
5. Meszaros, Gerard, Shaun Smith and Jennitta Andrea, "The Test Automation Manifesto", XP Agile Universe Conference, 2003.