



Agile Requirements

A WHITE PAPER PROVIDED TO ASPE BY KENNETH PUGH

Agile Requirements

White Paper prepared for ASPE
www.aspe.org
Pugh-Killeen Associates

Overview

Agility is an ability to deliver business value quickly and to respond to changes, particularly in requirements. Requirement change is a natural process. While a system is being developed, the business environment can change. As users employ a new system, they may come up with lots of new ideas.

Another cause of changes is that different personnel become stakeholders and they may bring fresh ideas on requirements.

Since requirement change is a natural aspect of software development, agility is designed to deal with it with low ceremony and overhead. A hallmark of agility is an iterative and incremental process. Gathering requirements for incremental iterations differs from the more traditional requirement-gathering step. This paper describes one successful flow for collaborating on and confirming agile requirements. Agility is not just about responding to changing requirements, but also about changing the development process. Adapt this flow to your own team and environment.

Traditional versus Agile

Traditional Requirements

In traditional software development, business analysts gather requirements from the stakeholders and create a software requirements document. The document details the requirements, both functional and nonfunctional. Non-functional requirements include the “ilities” such as reliability and security. The customer signs off the requirements document prior to the developers starting work on the project. Often the customer signs with trepidation because they feel the sign-off signifies that the requirements completely and accurately reflects their business needs. After sign-off, requirements can change and most likely will, anywhere from 1% a month to much more. In more formalized processes, these changes go through a change control board to assess the need and impact of the change. This in itself can add delays in creating a system that meets the business needs of today, not yesterday.

Agile Requirements

Agile Requirements

Agility recognizes that creating a complete and accurate requirements document is almost impossible. Change is anticipated and planned for. First, a high-level overview of a project's requirements is elicited. Then detailed requirements are gathered during the development process itself, just prior to implementation of those requirements. Implementation takes place in iterations from one to four or more weeks. At the beginning of each iteration, the customer, in collaboration with the developers, determines the requirements to be implemented during the iteration. That set does not change during the iteration, to give stability to the development environment. At the end of an iteration, requirements for the next iteration are planned. Any requirement can be dropped, added, or altered.

Traditional Estimating

With a traditional process, requirements are created upfront. Estimates are placed on the time and resources required to implement those requirements. If the estimates are wrong and the project deadline is absolute, features are typically dropped at the end of the project. A typical story is that a developer estimates the length of time to be eighteen months. The customer says that they have to have it in twelve with the same amount of resources. At the tenth month, it becomes apparent that the complete set of software will not be delivered for the deadline. So quick decisions are made to determine which features are to be dropped.

Agile Estimating

With agile processes, higher priority requirements are scheduled first. If estimates are wrong and time runs out, the project has the priority requirements implemented. Agile processes do not necessarily develop software any more efficiently than traditional processes. However the frequent delivery of software in iterations allows quicker feedback from the customer to the developer. Another facet of agile is prompt bug fixing. Bugs that are discovered within an iteration are fixed within that iteration or the corresponding feature is not marked as done. When bugs escape from the development process, the reported bugs are scheduled along with the current requirements. The customer decides whether a new feature is more important than fixing a bug.

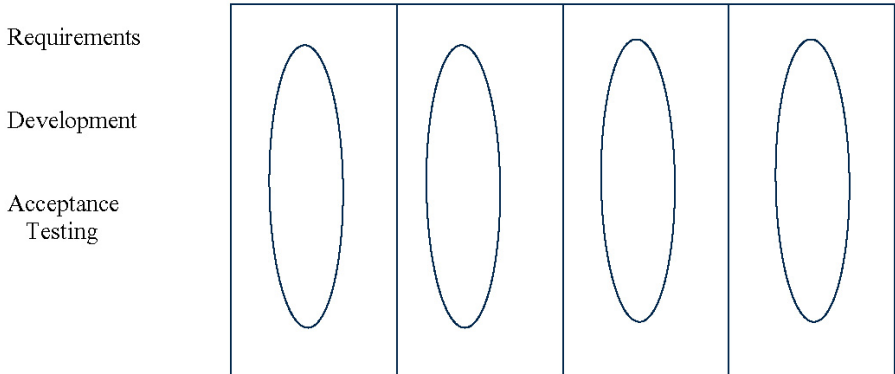
Agile Aspects

What constitutes an agile process? Each flavor has its own particular viewpoints. But in general, agility involves an iteratively and incrementally delivery of a software system. After the first iteration or the first few iterations, a basic system is delivered. Every iteration thereafter implements new functionality. The iterations are time-boxed. If functionality is not implemented during an iteration, it is delayed until the next iteration. Agile processes feature continuous quality control. Developers, customers, and testers create automated tests to ensure that the implementation conforms to the requirements. Requirements are complete for the current iteration and tentative for the next one. Delivery of the iterative output prior to the final system delivery allows the opportunity to validate

Agile Requirements

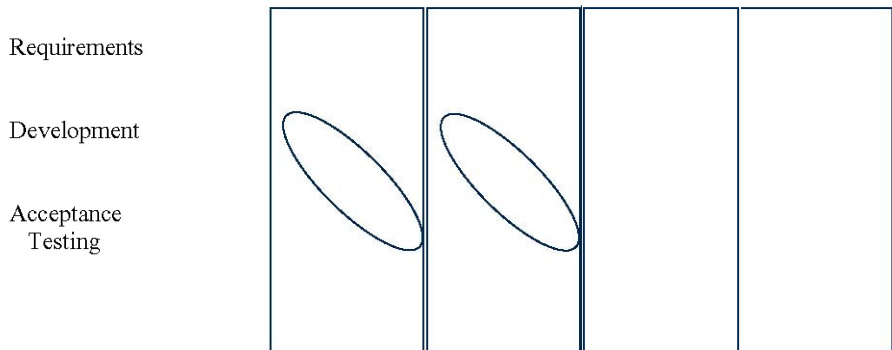
the requirements with the stakeholders and correct the development of the system well before a traditional process permits.

Here is the ideal iteration plan.



Iterations for Larger or Complex Projects

Often gathering requirement details may take longer than the time allowed for an iteration for any number of reasons. Therefore, many projects adapt a rolling wave of iterations. Note although this appears more like a traditional process, each iteration only involves a small number of features. The testing phase may repeat the same automated tests on various versions of the system, as well as adding tests for features delivered during the iteration. Testing may involve actions that require more time than an iteration, such as presenting the system to a group of users to determine its usability,



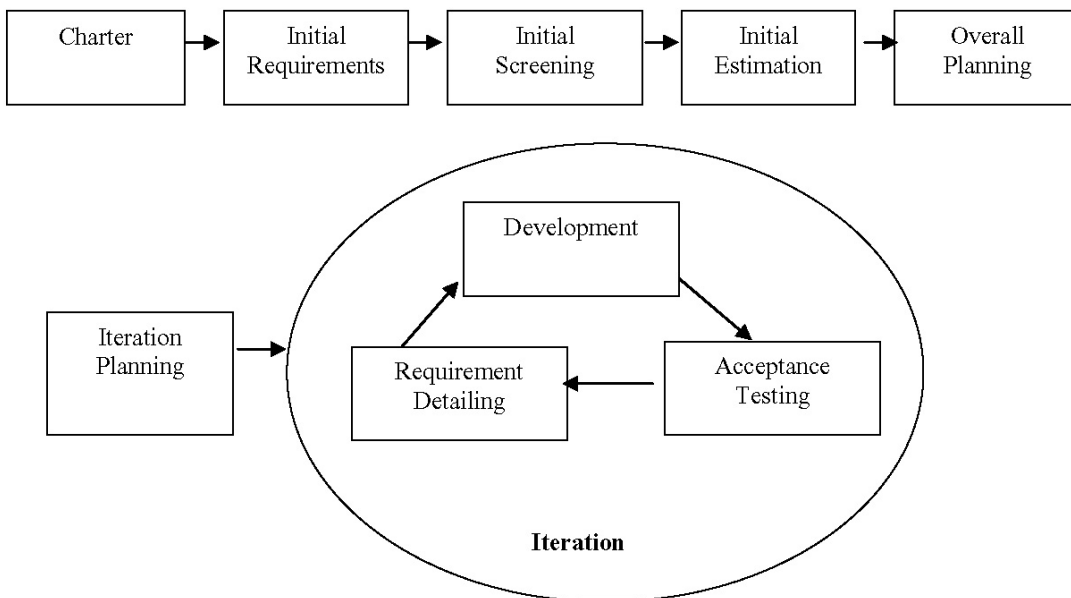
Agile Requirements

Software Development Roles

There are many roles in a software project. The roles can be grouped into stakeholders, customers, and developers. Product stakeholders include the gold owner (the one paying for the project), end users, regulators, and others. The product team is comprised of two units - the customer unit and the developer unit. Each unit has its own responsibilities in developing the project, though in many projects the units consist of many of the same people. The customer unit consists of the product owner or manager, the business analysts, and the independent quality assurance. They interact with the stakeholders to formulate the requirements and their priorities into a "single voice". The customer unit accepts or rejects the results of an iteration. Quality assurance works from the beginning of the project to help develop acceptance tests for the project. They also run tests that involve the project as a whole, such as overall performance, scalability, and portability to multiple operating systems. They perform exploratory or investigative testing to try out situations which developers may not have thought about or which crosscuts across several features. The development unit consists not only of developers, but also testers who can help the developers create their unit and own automated acceptance tests. They are responsible for designing, coding, and testing the software.

Requirements Flow

The flow shown in this paper starts with developing a charter that states the project's vision and an initial requirement gathering to give an overview of the project scope. Next estimates for the requirements help to form a rough overall plan. During each iteration, detailed requirement gathering occurs just prior to implementation.



Agile Requirements

Charter

A charter communicates the vision, goals, and objectives for a project. The term charter often refers to a more complex document. The form shown here contains only that which is needed to communicate why a project is going to be undertaken. The vision states how the project fits into the overall enterprise. The goals are subjective measures of what a successful project will accomplish.

Objectives are SMART - Specific, Measurable, Agreed-upon, Realistic, and Time-boxed. They are the testable part of the charter - the results of the project can be measured against the objectives.

Charter Example

Sam owned Sam's Lawn Mower Repair Shop. When he realized that customers who came in with lawn mowers to be fixed owned portable CD players they listened to while mowing the lawn, Sam decided to rent CD's and renamed his business Sam's Lawn Mower Repair and CD Rental Store.

The paper system for keeping track of CD rentals is becoming burdensome, so he wants a computerized system. He developed this charter:

Vision:

Create a CD rental information system

Goals:

Reduce staff time to process rentals

Offer more services to customers

Objectives

Within six months, CD checkouts will be processed in 20% less time

Within six months, overdue CDs will be listed daily on demand

Initial Requirement Gathering

To initially gather requirements, you can hold a requirements workshop. The workshop ideally involves all the stakeholders, customers, and developers. It can last a few hours to a few days, depending on the size of the project. For larger projects, multiple workshops can involve only those persons concerned with a particular portion of a system.

There are many ways to hold a workshop. A common one is described here. Logistical requirements are a pad of post-it notes (or index cards with tape), a "Sharpie" for each participant and a blank wall covered with flip charts (or a whiteboard). The reason for a "Sharpie", rather than a regular pen, is twofold. Cards written with "Sharpies" can be read from a distance, allowing a more overall view of requirements once they are put on the wall. Second, the thickness of the ink does not allow as many words to be written on the card. The card should represent only a very high level view of a requirement, not the full detail. As a rule of thumb, any requirement that takes more than a card should be broken into two requirements.

A facilitator starts by reviewing the project charter describing the purpose of the project. For a short

Agile Requirements

period of time, each person writes down on individual cards his or her thoughts for features or brief use cases for the system. The idea is to go for quantity, not quality. The cards are then placed on the table or wall. Duplicate ideas are placed on top of each other. One can be tossed, or a new card can be written that combines the phrasing of the duplicate cards.

Ideas can be written down as features, brief use cases, or user stories. Brief use cases represent an individual interacting with the system. User stories, popularized by Extreme Programming, roughly correspond to a brief use case. (See Mike Cohn's book for more details). I'll apply the term "story" to cover any form of requirement - feature, brief use case, or user story.

Story Example

Sam, his wife, brother, and sister-in-law held a requirement's gathering workshop and came up with the following stories.

- Keep track of where each CD is - "in-store" or "on-rental" (and with whom)
- A charge system to bill customers monthly, rather than by each rental
- Be able to offer discounts to frequent renters
- Report when CDs are overdue
- With multiple stores, the system should show which stores have a particular CD
- Have a catalog so the customer can see what CDs are available and what songs are on what CDs

Role Modeling

If you are going to employ use cases, you'll be creating roles in which people may interact with the system. These roles correspond to actors in the Unified Modeling Language Use Case diagram.

An actor interacts with the system to obtain some outcome of value to the actor.

A facilitator can help a group brainstorm user roles in much the same way as coming up with stories. Once roles have been decided upon, they can be used as a starting point to discover other use cases. It is very helpful to have several representatives who act in a particular role take part in the workshop(s) that explores requirements for that role.

Role Modeling Example

Sam and his relatives suggested the following roles.

- Checkout Clerk - a person that processes the rental
- CD Searcher - a customer or a store employee who looks for a CD
- Inventory Maintainer - the person who ensures inventory is correct
- Profit Manager - the person who ensures that the store is making money

Agile Requirements

Brief Use Cases

A brief use case describes an interaction between an actor and the system in a sentence or two. It often has a title that acts as a quick way to refer to the use case. Often the title is sufficient to describe the case for planning purposes.

Use Case Brief Story Example

Sam's features can be restated as use cases:

- Counter Clerk: Checkout CD
- Counter Clerk: Check-in CD
- CD Searcher: Search Catalog
- CD Searcher: Find Store with CD in Stock
- Inventory Maintainer: Request Overdue Report
- Profit Manager: Bill Customers
- Profit Manager: Establish Discounts

Functions and Constraints

Many stories are functional - something the system must do. Other stories may be constraints, such as design, implementation, or performance constraints, not necessarily tied to a particular function. Design and implementation constraints are requirements such as "must use single-
logon", "must use the corporate standard database" or "execute on the corporate standard platform". Performance constraints specify how fast stories must execute, such as "client logon accepted within two seconds".

Constraints should be separated from functional stories. Typically stakeholders do not have issues with design or implementation constraints. Developers, on the other hand, need to consider those constraints when they code the system. Often the mere use of a particular database or library is sufficient to demonstrate compliance with a design or implementation constraint. The customers should specify acceptance tests for other constraints, such as performance. For example, the "client logon" needs to state details such as number of users, network delay, and so forth, to be an objective measure of performances.

In-Out Scope

If the gathered requirements appear to encompass more than the charter for the project suggests, the facilitator can perform an "in and out of scope" exercise. The facilitator selects a card and reads it. He asks for a quick vote - in or out of scope. If there is a lot of discussion as to the meaning or the "scope-ness" of a particular card, it can be set aside for further discussion. He then places the card on either an in-scope or out-of-scope page. The cards on the out-of-scope page are not tossed away, but kept for examination later as to whether the charter might be expanded or another project started with those requirements as the basis.

Agile Requirements

Chunking

If many of the requirements form natural groups, the facilitator can run through the requirements and find out which requirements might be "chunked". Those requirements are placed together on the wall. Once the cards are chunked, the facilitator asks the participants to give a name to each group. Each group can then be explored briefly to see if missing requirements become apparent (e.g. "Client Enters Order", but no "Client Pays for Order").

Having stories on the same level usually makes estimating and prioritizing easier. For example, "Client Enters Order" appears more high level than "Client Pays for Order with American Express" and "Client Pays for Order with Visa". If the requirements seem to be at multiple levels, you could rewrite a chunk of lower-level requirements into a single story, such as "Client Pays for Order". The original stories can act as detailed stories for the new story.

Sometimes a natural hierarchy emerges for the stories. For example, stories that are performed by one role or actor might be grouped together under a higher-level group named by that role. If that is the case, then the facilitator could ask the participants to create a hierarchy. Chunks and hierarchies are two ways of grouping a large number of requirements to reduce the number that need to be dealt with at a given time. For example, either all requirements in a chunk or all requirements in the same group in a hierarchy can be considered together in prioritization.

Initial Screening

Another way of dealing with a large number of requirements is to perform an initial screening. Each participant is given a "Sharpie" as a voting mechanism. The number of requirements under consideration is divided by three. Each participant gets that many votes. First, each person reads all the stories to be sure that he or she understands an overview of each story. If there are questions, a brief explanation of the story is presented.

Then each person marks the story cards with their vote. The dots on each card are counted up and the cards arranged in total vote order. Cards without any votes are placed in a separate area. Getting zero votes does not mean the requirement is discarded from the project. It just means that estimating and prioritizing these requirements are going to be done later. If nobody thought the requirement was important enough to vote on it, then further consideration of it can be delayed. Typically this voting procedure eliminates one third or more of the cards. Voting can clarify what are the essential requirements in the stakeholders' minds. The discussion around why a requirement is important is often as important as the vote itself.

There are many variations of the initial screening. A hallmark of agility is to alter the process to meet the particular people and environment in which you develop. Some groups allow a person to put multiple votes on a single story. Groups which have unequal number of participants from

Agile Requirements

the various stakeholders give fewer votes to each person in an over-represented group or require that the stakeholder group seek to reach consensus. For example, if there are two people from marketing and eight from sales, but the relative weight of the stakeholders should be the same, then each sales person gets one-quarter the number of votes of the marketing persons.

Initial Screening Example

Sam and his three partners voted on the stories. There were six items, four people voting, so each got two votes.

- Keep track of where each CD is - "in-store" or "on-rental" (and with whom) (3)
- Report when CDs are overdue (2)
- Have a catalog so the customer can see what CDs are available and what songs are on what CDs (2)
- A charge system to bill customers monthly, rather than by each rental (1)
- Be able to offer discounts to frequent renters
- When multiple stores, the system should show which stores have a particular CD

Requirement Tests

One rule is that every requirement must have an associated test with it. If a requirement does not have a test, then how will the customer know that the requirement has been completed? The customer has primary responsibility for providing the tests.

Even a requirement such as "user-friendly" can be objectively tested. For example, "the program will be given to 100 randomly selected users. Each user will rank the program on a scale from 1 to 10 as being user-friendly. The program shall meet or exceed an average rating of 8." On the initial requirement card, this might simply be listed as "User group rates user-friendliness"

The tests do not have to be considered during the initial requirement gathering. It is preferable to formulate the overview of the tests prior to getting developer's implementation estimates, as the tests can often clarify the difficulty of the requirement. The tests can be detailed when the requirements are detailed.

Story Card and Details

The story cards are tokens that represent a requirement. They do not have sufficient room to capture all the details of a requirement. As details emerge, they should be captured in a separate document that is updateable by all the parties (such as a wiki or SharePoint page). For example, quality assurance comes up with detailed tests or a business analyst comes up with a use case with exceptions, alternatives, and business rules.

Stories can represent more than the customer's requirements. Developers can suggest stories, such as "get training on the latest version of the server". However these stories should be represented

Agile Requirements

in terms of the customer's benefit, such as "developer training on latest version of server will allow tuning of system performance"

Story Facets

Stories should be the right size for planning purposes. Some stories may be too big to be easily estimated by the developers or they may be larger than what can be completed during an iteration. Mike Cohn terms these types of stories "epics". An epic should be divided into sub-stories that are estimable and completable. These sub-stories may form a "saga", a series of stories that needs to be implemented in a particular sequence. For example, a story that represents a use case with lots of exceptions may be split into two stories - one that represents the basic flow and one that represents handling the exceptions. Another story might be split into a basic interface and one with "bells and whistles". Basic features often are used a great deal more than advanced ones. For example, the standard Google interface is simply a search box and a button. When I've asked customers and students, I've found that about two thirds use only this basic interface. The one third who use the advanced interface (with lots of search criteria) only employ it about ten percent of the time.

Estimating

Developers need to estimate how long it will take to develop a particular story. These estimates start with gross estimates that are refined along the way. Each story needs an estimate so that it can be scheduled. A typical measure for estimates is the "story point". A story point represents an estimate of the amount of effort that it will take to implement a requirement. It takes into consideration both the requirement's size and toughness. Story points are relative, not absolute. Two stories that require the same effort should be assigned the same number of story points.

Time Estimating Example

The developer's estimated the following story points for each of the features. They normally would not estimate for stories that did not receive a vote in the initial selection, but for purposes of this example, they estimated all of them.

- Keep track of where each CD is - "in-store" or "on-rental" (and with whom) (5)
- Report when CDs are overdue (2)
- Have a catalog so the customer can see what CDs are available and what songs are on what CDs (3)
- A charge system to bill customers monthly, rather than by each rental (25)
- Be able to offer discounts to frequent renters (10)
- When multiple stores, the system should show which stores have a particular CD (5)

Velocity and Overall Planning

Developers place an estimate on how many story points of work they can complete during an iter-

Agile Requirements

ation. The measure of story points per iteration is termed the "velocity". The velocity depends on the length of the iteration, the number of developers, and their skill levels. Once a developer unit has completed a few iterations working together as a team, they can better estimate their velocity. . The initial estimate for a project's time to complete all the initial requirements is the total number of story points divided by the velocity times the length of an iteration. Note that this is an initial estimate. As the project goes along, the estimate can increase if additional stories are added, or decrease if the velocity increases.

The stories, based on velocity, are placed into iterations. Depending on the customer's desires for a longer-range view, stories can be tentatively scheduled for just the upcoming iteration, a few iterations, or all the iterations. If stories are placed in all iterations, the stories scheduled for the latter iterations are subject to frequent change, depending on changing requirements or changing velocity.

Overall Plan Example

The total number of story points is 50. The developers have estimated their velocity to be 8 for the developer unit working two-week iterations. Thus the initial estimate for completion time is thirteen weeks (50 divided by 8 (rounded up) times 2 weeks). The customer decides the order of the stories, based on either voting in the initial screening, or on business value estimates (shown in the next section). The following stories are scheduled for the first iteration. Since the catalog system was estimated to be 3 story points, it was broken into two stories - one to explore alternative ways of providing the catalog and one to implement the catalog. The first story has a definite output - a decision as to how to provide the catalog, so its completion can be measured.

- Iteration One
 - o Keep track of where each CD is (5)
 - o Report overdue CDs (2)
 - o Catalog system - exploration of alternatives (1)

Business Value Estimation

Stakeholders are often uneasy with the inexactness of the time estimates. They prefer more precision, even though they often recognize in the back of their mind that more precision does not equate to a more exact estimate.

You can ask the customer in conjunction with the stakeholders to estimate the business value of each story. Stakeholders are often unable to come up with dollar figures, so instead ask for relative business value in "business value" units, which are unit-less. Typically higher business values should reflect higher priority stories.

Agile Requirements

Business Value Example

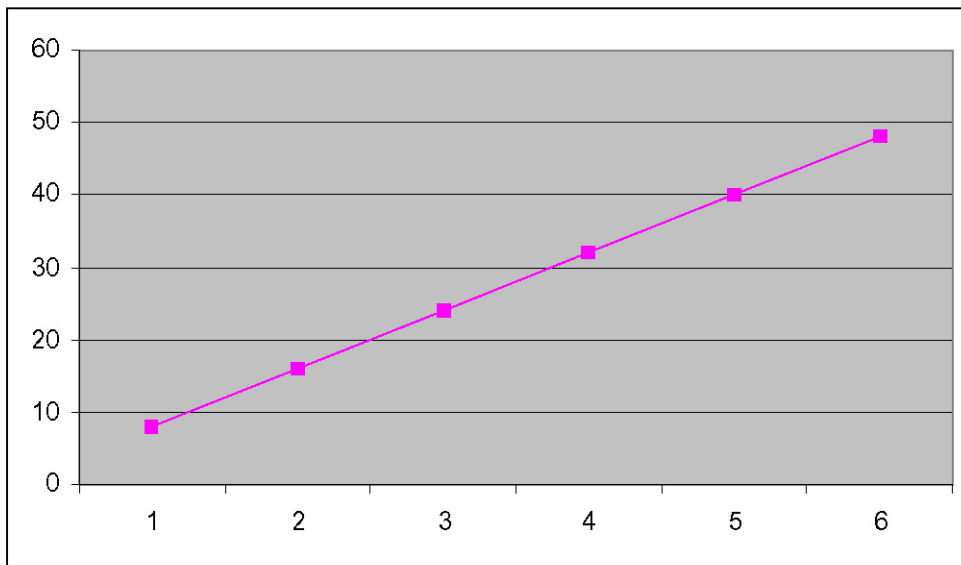
Sam and his partners estimated the business values for each of the features. They assigned a business value of 5 to an important requirement and a 2 to a less important requirement and then ranked the remainder using those two as guidelines.

- Keep track of where each CD is - "in-store" or "on-rental" (and with whom) (5)
- Report when CDs are overdue (2)
- Have a catalog so the customer can see what CDs are available and what songs are on what CDs (3)
- A charge system to bill customers monthly, rather than by each rental (1)
- Be able to offer discounts to frequent renters (1)
- When multiple stores, the system should show which stores have a particular CD (4)

Business Value Chart

One purpose for asking for business values for requirements is to emphasize the positive increase in the value of the enterprise when requirements are implemented. A chart of business value for a project should show either steady or increasing business value.

This chart represents the increase in business value delivered with each iteration, assuming that all stories are delivered according to the velocity.



Bang for the Buck

Once both business value and time have been estimated, the customer can determine "bang for the buck". Bang for the buck is the business value points divided by the story points. It roughly represents a "return on investment". Customers may want to prioritize stories that have highest bang

Agile Requirements

for the buck.

Bang for the Buck Example

Example - Sam's Lawn Mower Repair and CD Rental Shop

- Keep track of where each CD is - "in-store" or "on-rental" (and with whom) (1)
- Report when CDs are overdue (1)
- Have a catalog so the customer can see what CDs are available and what songs are on what CDs (1)
- A charge system to bill customers monthly, rather than by each rental (0.04)
- Be able to offer discounts to frequent renters (0.1)
- With multiple stores, the system should show which stores have a particular CD (.8)

Requirement Change

New stories may emerge continually. Add a story card to the backlog for every changed feature. If appropriate, merge the story with an existing card. Then estimate story points and business value and put into the backlog. At the iteration planning meeting, the new story can be scheduled for the next iteration or delayed further.

- Example:
 - o Add date of release to CD catalog search criteria (BV = 1, SP = .5)

Iterations and Detailed Requirements

To implement stories for an iteration, developers need more details than are on the story cards. Business analysts and quality assurance may have captured details during the concluding iteration. The details are often captured as detailed use cases with exceptions and alternatives. I've found that for many of my clients, it is often easier to develop detailed tests for use cases than for features. During the previous iteration, details are captured, so that estimates can be made more precisely at the beginning of each iteration. Each story usually requires details so that it can be broken into multiple tasks. Developers estimate each task in hours and sign up for tasks based on the amount of time they have during the iteration. During the iteration planning meeting, if the total number of hours exceeds the allotted time for the iteration, the number of stories planned is reduced. The customer decides which stories are put off until a future iteration

To complete the picture, here is an example of a detailed use case and associated customer acceptance tests.

Agile Requirements

Use Case Example

Name: Rent a CD

Description: Clerk checks out CD for customer

Actor: Counter Clerk

Pre-conditions: Customer has ID

Post-conditions: CD is recorded as rented

Main Course:

1. Clerk enters Customer ID
2. Clerk enters CD ID
3. System records CD is rented to Customer
4. System prints rental contract

Exceptions:

- 1a. Customer ID is not recognized
Clerk repeats step 1.
- 1b. Customer violates B.R 1
Clerk informs Customer of B.R. violation
Use case abandoned

Alternatives

- 4a. Printer does not print contract
Clerk manually fills out contract

Business Rules

- B.R. 1 Customer can only have 3 CDs rented at any time

Acceptance Tests Example

Story: Checkout a CD

Scenario: Regular Rental

- Enter a Customer id and CD id
- System should print rental contract
- Check to see that CD recorded as currently rented

Story: Checkout a CD

Scenario: Already Rented (Misuse)

- Enter CD id of CD that is recorded as currently rented
- System should respond with error

Agile Requirements

Summary

Agility allows for changing requirements. Agile requirements are gathered and estimated in progressive stages. In the initial stage, high-level descriptions are created and gross estimates are produced. In later stages, detailed requirements are gathered and estimates become more detailed. Which requirements are to be developed during an iteration are decided just prior to the iteration to deliver what the customer considers the most important business value at that time.

Agile Requirements

References

Books

"Agile Estimating and Planning" , Mike Cohn

"User Stories Applied : For Agile Software Development", Mike Cohn

"Lean Software Development: An Agile Toolkit", Mary and Tom Poppendieck

"Software Requirements " , Karl Wiegers

Web Sites

www.agilealliance.org

www.agilemodeling.com

www.agiledata.org

www.extremeprogramming.com

www.xprogramming.com

www.crystallmethodologies.org

www.jimhighsmith.com

The Story Card

I am hesitant to give a template for a story card. Templates often suggest "completism" - the need to fill out an entry for every field on a form. Often though a diagram gives a clear picture. So here is one for a reference point: 3 dots indicates the votes received for this story

Acknowledgements

Examples in this paper came from my Jolt Award winning book, "Prefactoring - Extreme Abstraction, Extreme Separation, Extreme Readability". Part of the information came from my course, "Collaborating and Confirming Agile Requirements".

Ideas contained in this white paper were gathered from a number of individuals, through books, email, and conversation. This paper represents a composition of their thoughts. The author wishes to thank them for their input:

Karl Wiegers, Kent Beck, Andy Hunt, Dave Thomas, Ron Jeffries, Ken Schwaber, Jerry Weinberg, Martin Fowler, Pete McBreen, Kevan Lyons, Jim Highsmith, Bob Martin, Scott Ambler, Alan Shalloway, David Cohen, Alistair Cockburn, Mikael Lindvall, Patricia Costa, Mike Cohn, Norm Kerth, Tim Lister, Linda Rising, Joshua Kerievsky, Ron Jeffries, Diana Larsen, Christopher Avery, Per Kroll, Bruce MacIssac, Mary Poppendick, Tom Poppendick