# Requirements Solicitation and Analysis

An Iterative Process

by: Robert N. Loomas, Jr.

CSCI 5828 – Spring 2012

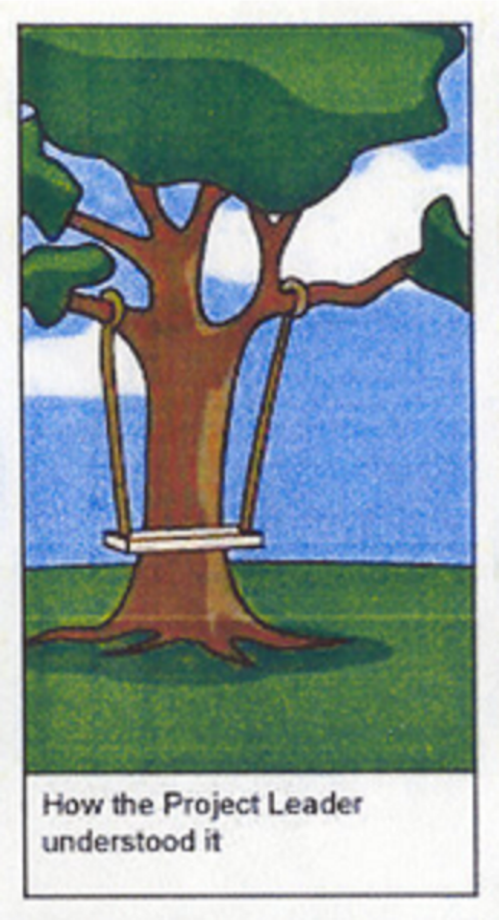
# Does the client know what they need?

How the customer explained it

- The customer starts by explaining what they think they need based on their understanding of the current domain.

- Is this what the customer is REALLY looking for?

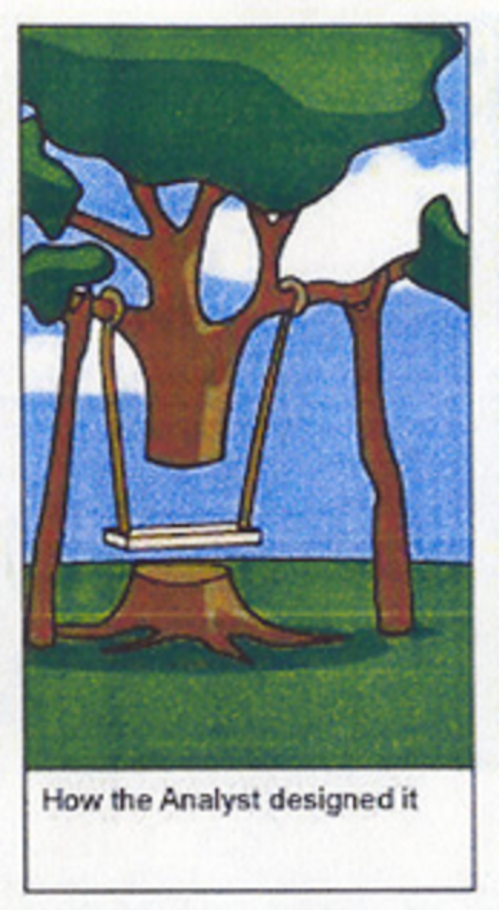
# Does the Project Manager understand what he/she heard?



- The Project Manager develops a “picture” of what the customer explained based on what he/she knows about the customer’s domain.

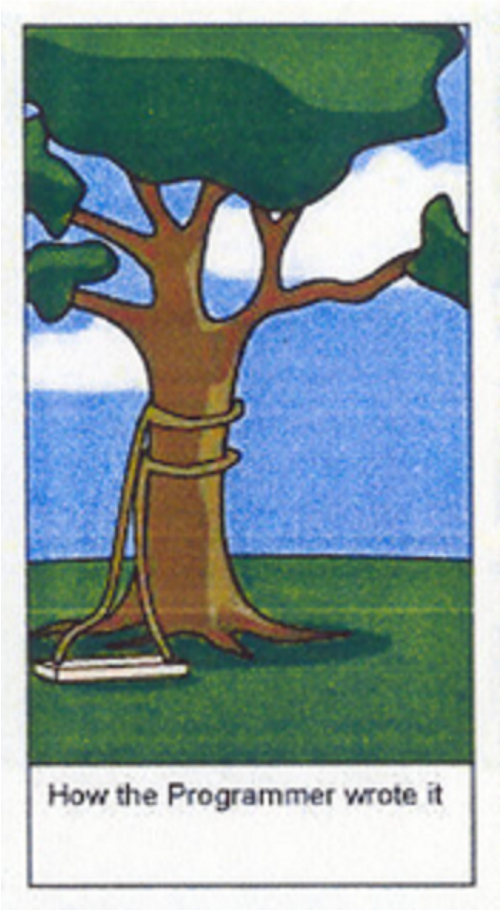
- Is THIS what the customer is looking for?

# Does the Analyst understand what the Project Manager explained?



- The Analyst develops a specification based on how the Project Manager explained it.
- Is THIS what the customer is looking for?

# Does the Programmer understand the specification the Analyst wrote?

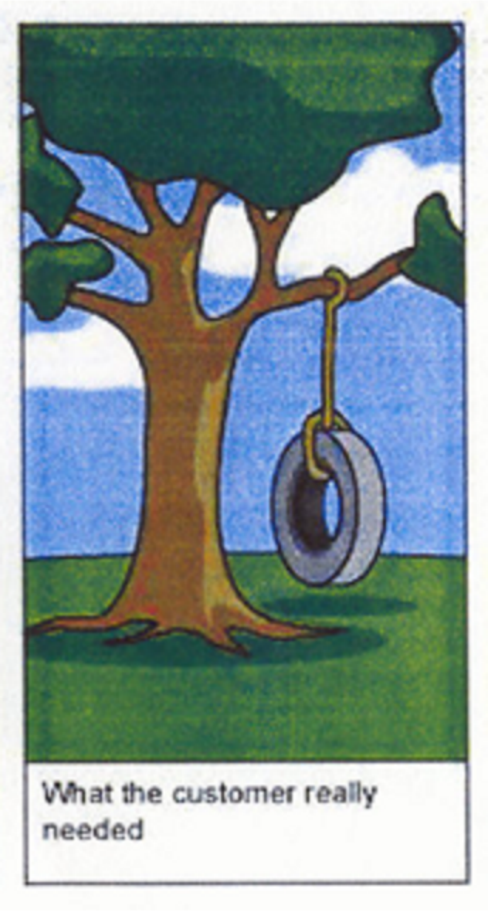


How the Programmer wrote it

- The Programmer develops the product based on the specification written by the Analyst.

- Is THIS what the customer is looking for?

# Did anyone bother to learn how the customer uses the current system and what the customer actually needs?



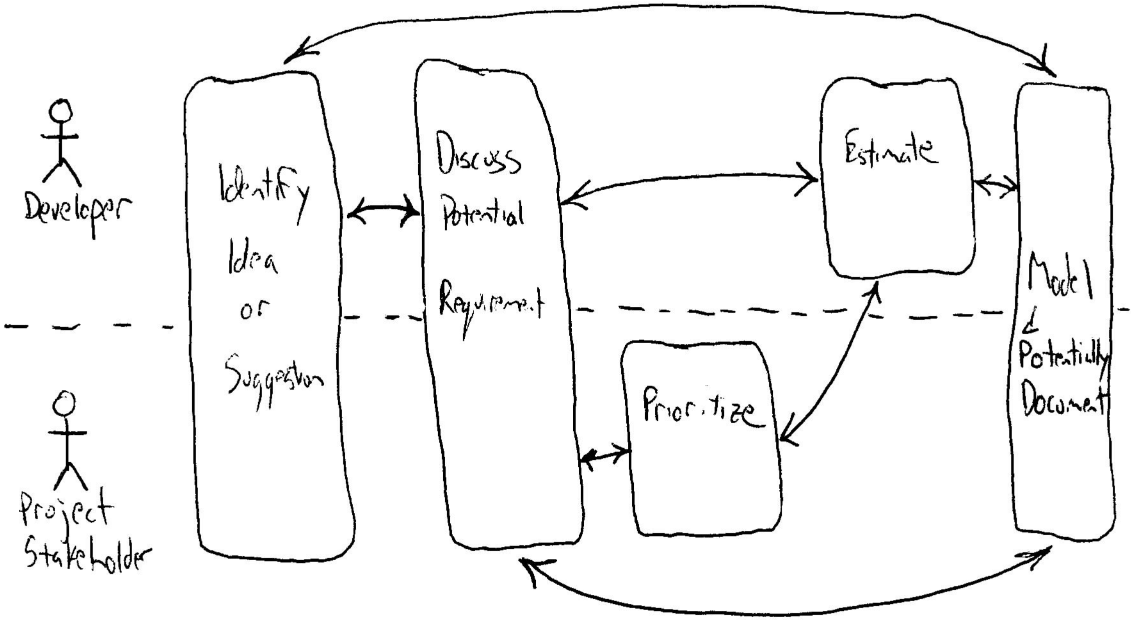
- THIS is what the customer needs!!

# The Agile Process

- Stakeholders actively participate
- Adopt inclusive models
- Take a breadth-first approach
- Model storm details just in time (JIT)
- Treat requirements like a prioritized stack
- Prefer executable requirements over static documentation
- Your goal is to implement requirements, not document them
- Recognize that you have a wide range of stakeholders
- Create platform independent requirements to a point
- Smaller is better
- Question traceability
- Explain the techniques
- Adopt stakeholder terminology
- Keep it fun
- Obtain management support
- Turn stakeholders into developers

## What does it mean for a project stakeholder to actively participate?

There are two issues that need to be addressed to enable this practice – availability of project stakeholders to provide requirements and their (and your) willingness to actively model together.


Developer
Project Stakeholder
Identify Idea or Suggestion
Discuss Potential Requirement
Estimate
Prioritize
Model & Potentially Document

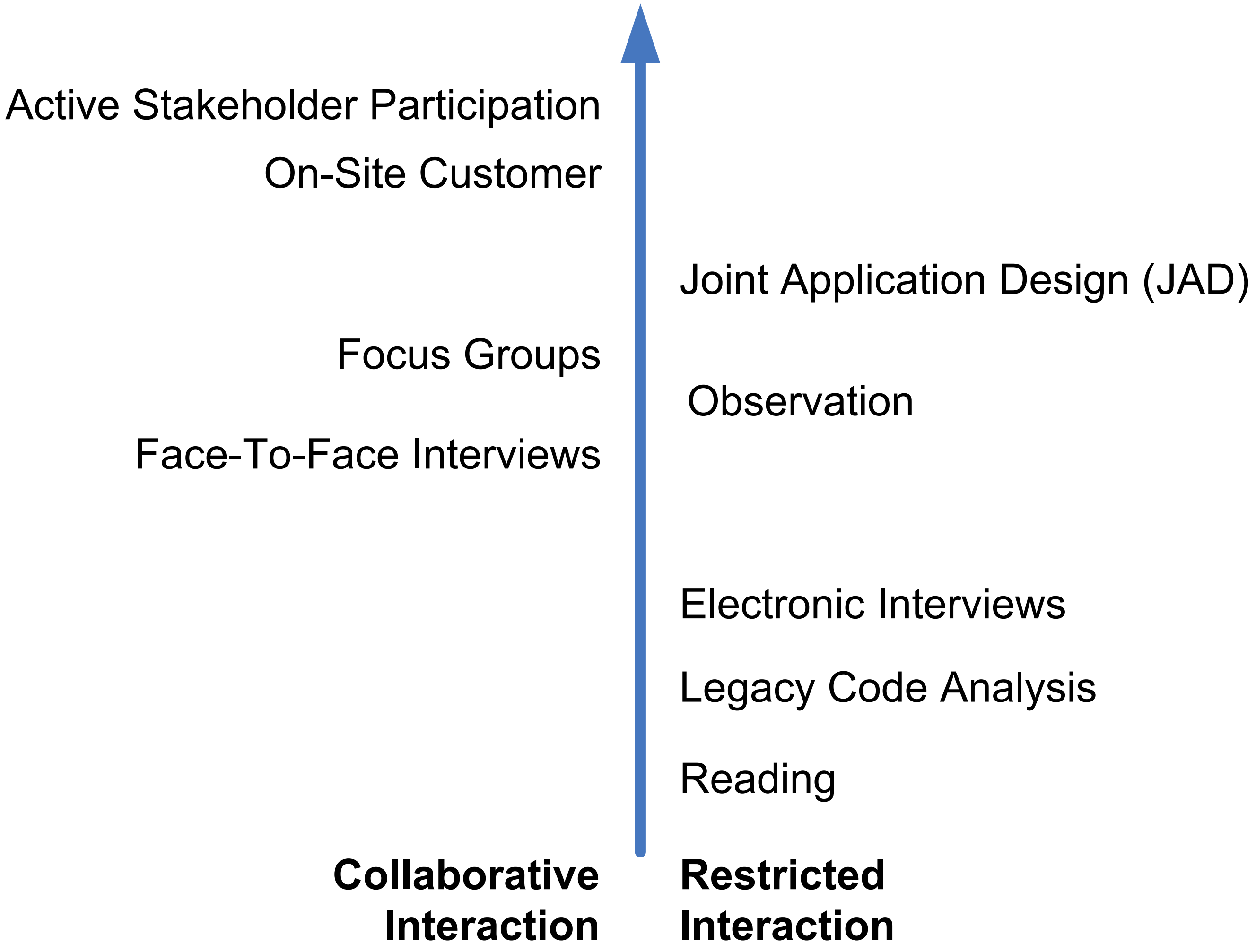
The figure above presents a high-level view of the requirements process, using the notation for UML activity diagrams, indicating the tasks that developers and project stakeholders are involved with.

The dashed line is used to separate the effort into swim lanes that indicate what role is responsible for each process. In this case you see that both project stakeholders and developers are involved with identifying ideas or suggestions, discussing a potential requirement, and then modeling and potentially documenting it.

Project stakeholders are solely responsible for prioritizing requirements, the system is being built for them, therefore, they are the ones that should set the priorities.

Likewise, developers are responsible for estimating the effort to implement a requirement because they are the ones that will be doing the actual work – it isn't fair, nor advisable, to impose external estimates on developers.  Although prioritization and estimation of requirements is outside the scope of Agile Modeling (AM), however, it is within the scope of the underlying process such as Extreme Programming (XP) or Unified Process (UP) that you are applying AM within, it is important to understand that these tasks are critical aspects of your overall requirements engineering effort.

# Effectiveness of Requirements Gathering Techniques



Active Stakeholder Participation

On-Site Customer

Joint Application Design (JAD)

Focus Groups

Observation

Face-To-Face Interviews

Electronic Interviews

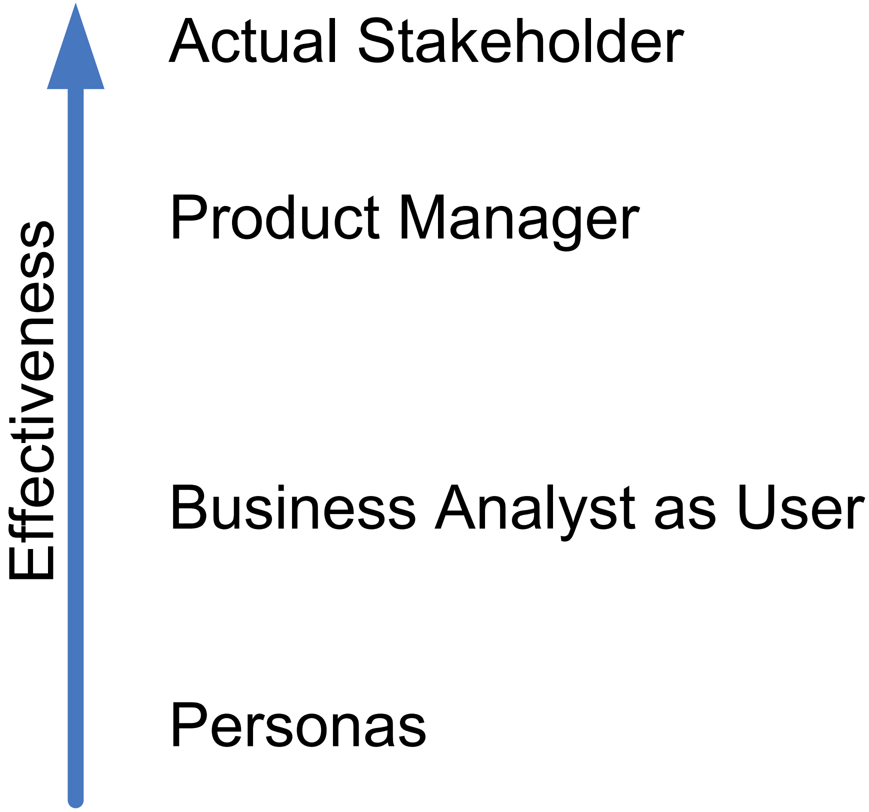
Legacy Code Analysis

Reading

**Collaborative Interaction**

**Restricted Interaction**

# Relative Effectiveness of User Representatives



Effectiveness

Actual Stakeholder

Product Manager

Business Analyst as User

Personas

Copyright 2005 Scott W. Ambler

## Adopt Inclusive Models

To make it easier for project stakeholders to be actively involved with requirements modeling and documentation, to reduce the barriers to entry in business parlance, you want to follow the practice "Use the Simplest Tools."

Many requirements artifacts can be modeled using either simple or complex tools –

- Post It Notes and Flip Chart Paper can be used to model the requirements for a screen/page.

- Index Cards can be used for conceptual modeling.

Whenever you bring technology into the requirements modeling effort, such as a drawing tool to create "clean" versions of use case diagrams or a full-fledged CASE tool, you make it harder for your project stakeholders to participate because they now need to not only learn the modeling techniques but also the modeling tools.

By keeping it simple you encourage participation and thus increase the chances of effective collaboration.

## Take a Breadth-First Approach

It is better to paint a wide swath at first (to get a feel for the bigger picture) than it is to narrowly focus on one small aspect of your system.  By taking a breadth-first approach you quickly gain an overall understanding of your system and can still dive into the details when appropriate.

The point is that you can do a little bit of initial, high-level requirements envisioning up front early in the project to understand the overall scope of your system without having to invest in mounds of documentation.

Through initial, high-level modeling you gain the knowledge that you need to guide the project but choose to wait to act on it.

## Model Storm Details Just In Time (JIT)

Requirements are identified throughout most of your project. Although the majority of your requirements efforts are performed at the beginning your project it is very likely that you will still be working them just before your final code freeze before deployment. Remember the principle "Embrace Change." Agilists take an evolutionary, iterative and incremental, approach to development. The implication is that you need to gather requirements in exactly the same manner.

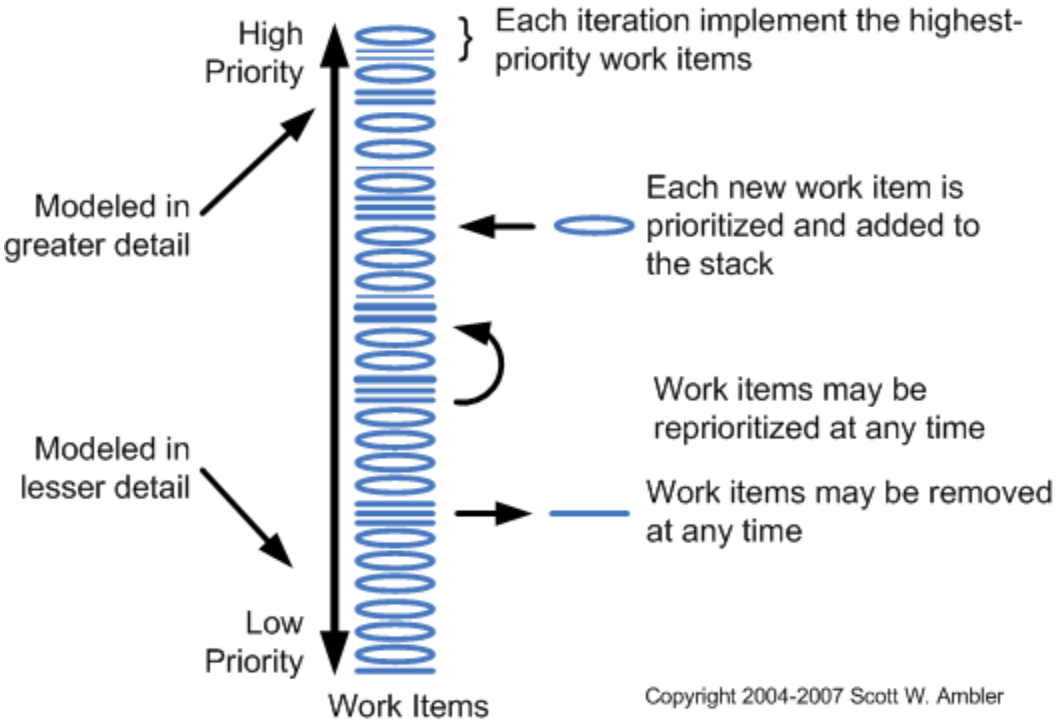
Luckily AM enable evolutionary modeling through the use of practical approaches such as:

Create Several Models in Parallel
Iterate To Another Artifact
Model In Small Increments

The shorter the feedback cycle between model storming a requirement and implementing it, the less need there is for documenting the details.

## Treat Requirements Like a Prioritized Stack

The figure to the right provides an overview of the agile approach to managing requirements, reflecting both Extreme Programming (XP)'s planning game and the Scrum methodology. The software development team has a stack of prioritized and estimated requirements which needs to be implemented (co-located agile teams will often literally have a stack of user stories written on index cards).

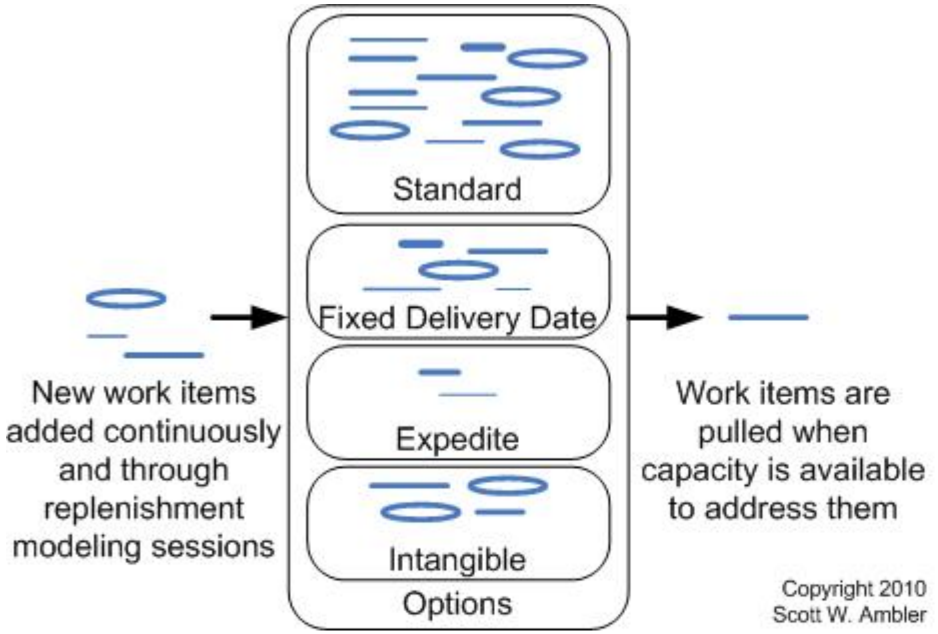

High Priority
Each iteration implement the highest-priority work items
Modeled in greater detail
Each new work item is prioritized and added to the stack
Work items may be reprioritized at any time
Modeled in lesser detail
Work items may be removed at any time
Low Priority
Work Items

Copyright 2004-2007 Scott W. Ambler

The team takes the highest priority requirements from the top of the stack which they begin to implement in the current iteration. Scrum suggests that you freeze the requirements for the current iteration to provide a level of stability for the developers. If you do this, any change to a requirement you're currently implementing should be treated as just another new requirement.

# Lean Work Management Process.

It's interesting to note that lean teams are starting to take a requirements pool approach to managing requirements instead of a stack



Standard
Fixed Delivery Date
Expedite
Intangible
Options
New work items added continuously and through replenishment modeling sessions
Work items are pulled when capacity is available to address them

Copyright 2010
Scott W. Ambler

**Prefer Executable Requirements Over Static Documentation**

During development it is quite common to model storm for several minutes and then code, following common Agile practices (such as Test-First Design (TFD) and refactoring) for several hours and even several days at a time to implement what you've just modeled. This is where your team will spend the majority of its time.

Agile teams do the majority of their detailed modeling in the form of executable specifications, often customer tests or development tests.

Why does this work?

> Because your model storming efforts enable you to think through larger, cross-entity issues and Test-Driven-Design (TDD) allows you to think through very focused issues typically pertinent to a single entity at a time.
>
> With refactoring you evolve your design via small steps to ensure that your work remains of high quality.

TDD promotes confirmatory testing of your application code and detailed specification of that code.

Customer tests (also called agile acceptance tests) can be thought of as a form of detailed requirements and developer tests as detailed design.  Having tests do “double duty” like this is a perfect example of single sourcing information, a practice which enables developers to travel light and reduce overall documentation.

Detailed specification is only part of the overall picture – high-level specification is also critical to your success, when it’s done effectively.

This is why we need to go beyond TDD to consider Agile Model Driven Development (AMDD).

## Your End Goal is To Effectively Implement Requirements, Not Document Them

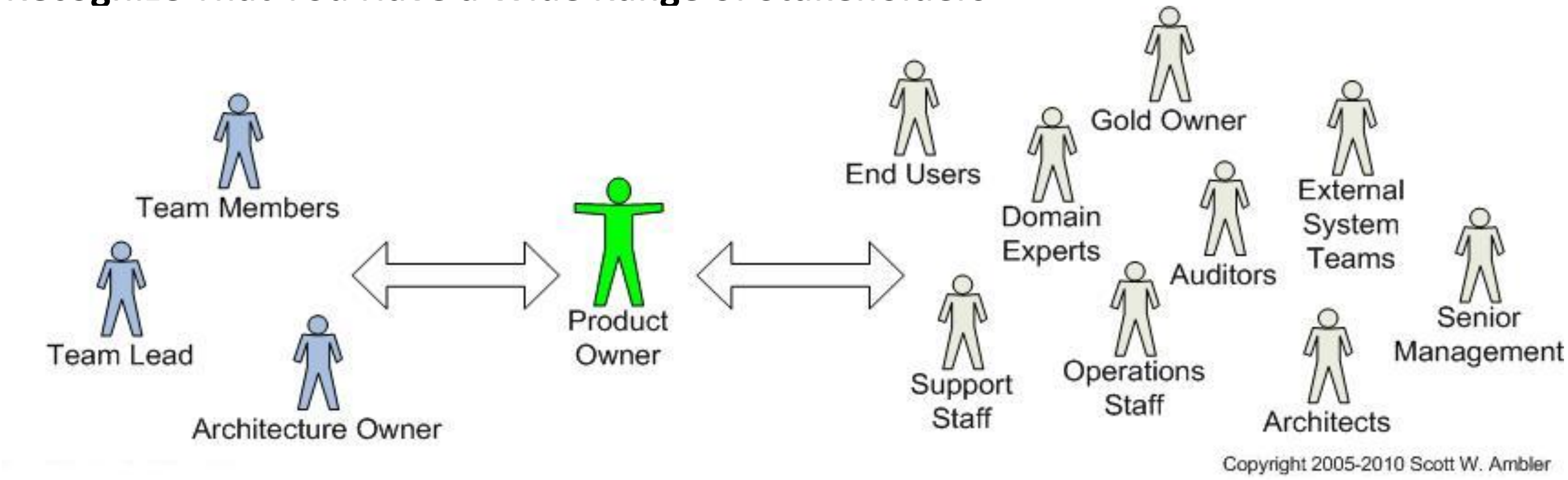
Too many projects are crushed by the overhead required to develop and maintain comprehensive documentation and traceability between it.

Take an agile approach to documentation and keep it lean and effective.

The most effective documentation is just barely good enough for the job at hand.
By doing this, you can focus more of your energy on building working software.

The urge to write requirements documentation should be transformed into an urge to collaborate closely with your stakeholders instead then create a consumable solution based on what they tell you.

## Recognize That You Have a Wide Range of Stakeholders



End users, either direct or indirect, aren't your only stakeholders. Other stakeholders include :

managers of users
operations staff members
support (help desk) staff members

senior managers & auditors
the "gold owner" who funds the project
your program/portfolio manager

developers working on other systems that integrate or interact with the one under development

maintenance professionals potentially affected by the development and/or deployment of a software project

These people aren't going to agree with one another, they're going to have different opinions, priorities, understandings of what they do, understandings of what others do, and visions for what the system should (or shouldn't) do.

The implication is that you're going to need to recognize that you're in this situation and act accordingly. The figure above shows how agile teams typically have someone in a stakeholder representative role (called product owner in Scrum) whom they go to as the official source of information and prioritization decisions. This works well for the development team, but essentially places the burden on the shoulders of this person.

Anyone in this role will need to:

1. Have solid business analysis skills, particularly in negotiation, diplomacy, and requirements elicitation.
2. Educate the team in the complexity of their role.
3. Be prepared to work with other product owners who are representing the stakeholder community on other development teams. (This is particularly true at scale with large agile teams.)
4. Recognize that they are not an expert in all aspects of the domain. Therefore, they will need to have good contacts within the stakeholder community and be prepared to put the development team in touch with the appropriate domain experts on an as-needed basis so that they can share their domain expertise with the team.

**Platform Independent Requirements to a Point**

Requirements should be technology independent.  The terms OO, structured, and component-based are all categories of implementation technologies, and although you may choose to constrain yourself to technology that falls within one of those categories the bottom line is that you should just be concerned about requirements.  That's it, just requirements.

All of the techniques described below can be used to model the requirements for a system using any one (or more) of these categories.

Sometimes you must go away from the ideal of identifying technology-independent requirements.  For example, a common constraint for most projects is to take advantage of the existing technical infrastructure wherever possible.

At this level the requirement is still technology independent, but if you drill down into it to start listing the components of the existing infrastructure (such as your Sybase vX.Y.Z database or the need to integrate with a given module of SAP R/3) then you've crossed the line.

This is okay as long as you know that you are doing so and don't do it very often.

**Smaller is Better**

Remember to think small.

Smaller requirements, such as features and user stories, are much easier to estimate and to build to than are larger requirements, such as use cases.

An average use case describes greater functionality than the average user story and is thus considered “larger”.

They are also easier to prioritize and therefore manage.

## Question Traceability

Think very carefully before investing in a requirements traceability matrix, or in full lifecycle traceability in general, where the traceability information is manually maintained.

> Traceability is the ability to relate aspects of project artifacts to one another, and a requirements traceability matrix is the artifact that is often created to record these relations – it starts with your individual requirements and traces them through any analysis models, architecture models, design models, source code, or test cases that you maintain.

Organizations with traceability cultures will often choose to update artifacts regularly, ignoring the practice "Update Only When it Hurts," so as to achieve consistency between the artifacts (including the matrix) that they maintain. They also have a tendency to capture the same information in several places, often because they employ overly specialized people who "hand off" artifacts to other specialists in a well-defined process.

This is not traveling light. Often a better approach is to single source information and to build teams of generalizing specialists.

The benefits of having such a matrix is that it makes it easier to perform an impact analysis pertaining to a changed requirement because you know what aspects of your system will be potentially affected by the change.

In simple situations (e.g. small co-located teams addressing a fairly straightforward situation) when you have one or more people familiar with the system, it is much easier and cheaper to simply ask them to estimate the change.

If a continuous integration strategy is in place it may be simple enough to make the change and see what, if anything, you broke by rebuilding the system.

In simple situations (in which many agile teams find themselves) traceability matrices are highly overrated because the total cost of ownership (TCO) to maintain such matrices, even if you have specific tools to do so, far outweigh the benefits.

Make your project stakeholders aware of the real costs and benefits and let them decide. A traceability matrix is effectively a document and is therefore a business decision to be made by them.

If you accept the Agile Method principle Maximize Stakeholder ROI, if you're honest about the TCO of traceability matrices, then they often prove to be superfluous.

When does maintaining traceability information make sense? The quick answer is “in some agility at scale situations and when you have proper tooling support”

1. Automated tooling support exists. Some development tools, such as those based on the Jazz platform, will automatically provide most of your traceability needs as a side effect of normal development activities. You may still need to do a bit of extra work here and there to achieve full traceability, but a lot of the traceability work can, and should, be automated. With automated tooling the TCO of traceability drops increasing the chance that it will provide real value to your effort.
2. Complex domains. When you find yourself in a complex situation, perhaps you're developing a financial processing system for a retail bank or a logistics system for a manufacturer, then the need for traceability is much greater to help deal with that complexity.
3. Large teams or geographically distributed teams. Although team size is typically motivated by greater complexity -- domain complexity, technical complexity, or organizational complexity -- the fact is that there is often a greater need for traceability when a large team is involved because it will be difficult for a even the most experienced of team members to comprehend the detailed nuances of the solution. The implication is that you will need the type of insight provided by traceability information to effectively assess the impact of proposed changes. Note that large team size and geographic distribution have a tendency to go hand-in-hand.
4. Regulatory compliance. Sometimes you have actual regulatory compliance needs, for example the Food and Drug Administration's CFR 21 Part 11 regulations requires it, then clearly you need to conform to those regulations.

In short, the question of manually maintained traceability which is solely motivated by:

- "it's a really good idea“
- "we need to justify the existence of people on the CCB“
  (it's rarely worded like that, but that's the gist of it)
- "CMMI's Requirements Management process area requires it“

In reality there's lots of really good ideas out there with much better ROI, surely the CCB members could find something more useful to do, and there aren't any CMMI police so don't worry about it.

In short, just like any other type of work product, you should have to justify the creation of a traceability matrix.

It's requirements analysis, not retentive analysis.
;-)

## Explain the Techniques

Everyone should have a basic understanding of a modeling technique, including your project stakeholders.  They've never seen CRC cards before?  Take a few minutes to explain what they are, why you are using them, and how to create them.  You cannot have Active Stakeholder Participation if your stakeholders are unable to work with the appropriate modeling techniques.

## Adopt Stakeholder Terminology

Do not force artificial, technical jargon onto your project stakeholders.  They are the ones that the system is being built for, therefore, it is their terminology that you should use to model the system.  Avoid "geek-speak."  An important artifact on many projects is a concise glossary of business terms.

## Keep it Fun

Modeling doesn't have to be an arduous task.  In fact, you can always have fun doing it.  Tell a few jokes, and keep your modeling efforts light.  People will have a better time and will be more productive in a "fun" environment.

## Obtain Management Support

Investing the effort to model requirements and, in particular, applying agile usage-centered design techniques, are new concepts to many organizations.

An important issue is that your project stakeholders are actively involved in the modeling effort, a fundamental culture change for most organizations.

As with any culture change, without the support of senior management, you likely will not be successful.

You will need support from both the managers within your IS (Information Systems) department and within the user area.

**Turn Stakeholders Into Developers**

An implication of this approach is that your project stakeholders are learning fundamental development skills when they are actively involved with a software project.

It is quite common to see users make the jump from the business world to the technical world by first becoming a business analyst and then learning further development skills to eventually become a full-fledged developer.

Because agile software development efforts have a greater emphasis on stakeholder involvement than previous software development philosophies we will see this phenomena occur more often (keep a look out for people wishing to make this transition and help to nurture their budding development skills).

You never know, maybe some day someone will help nurture your business skills and help you to make the jump out of the technical world.

# References

- http://www.agilemodeling.com/essays/agileRequirementsBestPractices.htm
- http://www.agilemodeling.com/essays/amdd.htm
- Notes and presentation material from CSCI 5548 / Fall 2011 by Professor Dameron