The Agile 10 Step Requirements Model

In the beginning, when Agile first hit the headlines, it was mainly a story about developers doing technical practices. Weird things like pair-programming, writing tests first and other "extreme" stuff. In time Agile has become a story about processes - iterations, stand-up meetings and such. Extreme went way and it became respectable to "Scrum."

To date Agile has been weaker on requirements. Agile has a message for developers and project managers but less so for Business Analysts and Produce Managers. Agile as we know it currently rests on two pillars: technical practices and iterative processes. Requirements management is the missing pillar.

So far the requirements pillar has one big idea and a whole host of small ideas. The big idea is User Stories. The hinterland includes things like roles and Mike Cohn's INVEST mnemonic (Independent, Negotiable, Valuable, Estimatable, Small and Testable) but not a lot more.

Then there are the small ideas - small because unlike iterations, TDD or even User Stories, they are not so widely adopted or even well know. Many of these exist in isolation; they don't link up to each other or User Stories.

The more I thought about this problem the clearer it seemed to me: these bits weren't joined up. In 2009 I had sketched out a model I call the "Agile 10 Step", its a model I'd like to present here. Explaining the model in depth is beyond the scope of this article - here I will confine myself to a brief overview.

In future articles I hope to elaborate on this model to bring to better link the requirements process up with the rest of the Agile world. Indeed, some of the points raised by the model are already addressed in pieces I have already published

The model can't position every single requirements technique or tool ever created, that would be asking too much, but it can highlight some of the key ones.

Use of the word project

The PRINCE2 project management guide provides one narrow definition of a *project*: "A temporary organisation that is needed to produce a unique and predefined outcome or result at a pre-specified time using predetermined resources." (Commerce, 2005).

Strictly speaking then a "project" is a well defined piece of work. Still, it has become commonplace to refer to almost any piece of software development work as a "project". For the purposes of this document this wider definition is used liberally in keeping with the industry norms.

That said, I am contributing to an industry problem. Again and again in software development teams I see a "project" can be anything from a single bug fix lasting a few days up to major change initiatives with a timeline measured in months if not years.

The Three pillars of Agile

In the early days of Agile it was the technical aspects that received most attention and generated most excitement: test driven development, pair programming, refactoring and so on. As Agile matured the process and project management aspects received more attention: iterations, planning meetings and such.

Requirements form the third pillar of Agile. Yet the requirements element hasn't received the same attention as the first two. Teams can achieve some element of Agile with only one or two pillars but for maximum effect and greatest stability all three are required.

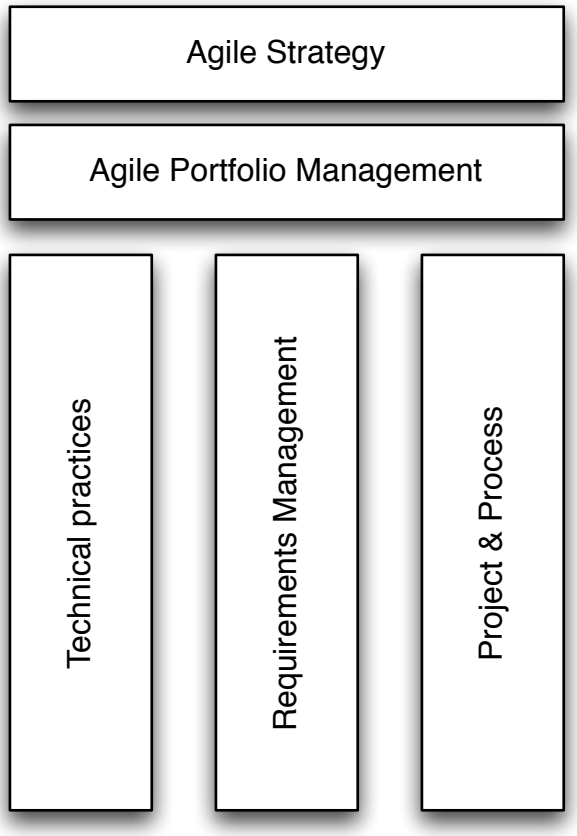

Figure 1 - Three pillars of Agile supporting high Agility

These three pillars provide the operations base on which organizations can push to portfolio and strategic Agile. (See (Kelly, 2010b) for more about Agile at the portfolio and strategy level.)

Principles for Requirements in an Agile world

The good news is Agile doesn't invalidate whole swathes of requirements engineering the way it can project management. There are many good books on requirements: how to find requirements, understand requirements, capture requirements and so on. The analysis side of this stands up. Read a good requirements or business analysis book and most of it is still valid, e.g. (Cadle et al., 2010, Alexander and Beus-Dukic, 2009).

In order to discuss requirements more fully it helps to set down some overarching principles:

- **Business value focused**: requirements are a means to an end. The overall objective is to deliver business value. Creating requirements is an analysis activity that helps identify and understand business value creation so it can be communicated to development teams.
- **Goal directed projects**: because requirements are emerging, being completed and disappearing, and because the environment is changing projects cannot be measured on "scope complete" criteria. Another measure is needed. Instead progress needs to measure as progress towards some overarching goal not fraction complete. (More on goal directed projects in (Kelly, 2010d)).

  With a goal clear requirements can be discovered by working backwards. Requirements are the things that will move the organization from where it is today towards its goal. Thus it makes sense to start with the desired outcome and work back.

  Of course it is much easier to say "goal directed" than to realise it. Many projects start with vaguely defined goals. In these cases discovering the goal is a little like panning for gold. In amongst all the ideas circulating some goal needs to be found. Naturally, this makes working backwards to find the requirements even harder.
- **Customer/End User involvement**: those who will actually use the end product need to have a voice in how it is built, and need to have early sight of what is being created. There are two good reasons for this: as the people who perform the work they are best placed to know how things should work and describe the real-life environment to the development team. Second, as the people who will need to use the software their willingness to use the end product is critical. Involving them early and often is the simplest way to do this.
- **Iterative**: in common with the rest of Agile requirements require an iterative approach. One look will not find all the requirements, multiple passes are required and things will change (Previous writing (Kelly, 2008, Kelly, 2004) contains a discussion of why requirements change.) Thus all aspects of requirements analysis are on going and in parallel with construction.

  Identification, capture and communication starts before the first line of code is cut and continues at least as long as development continues. Modern market economies do not stop while software is created so requirements continue to change and evolve. Priorities and values change.

  Consequently the collection, organization, prioritization needs to be cheap and individual requirements statements disposable. If individual requirements are expensive to create they will take on a life of their own. If they are cheap then there will be less agonising about disposing of them when things change.
- **Just in time**: there is little point in creating and storing masses of requirements in anticipation of the day they will be built. Requirements

sitting on the shelf go off - the market and business environment changes. Since requirements are an ongoing process there is simply no need to create a store so we adopt a just-in-time principle instead.

- **Dialogue over document**: communication of requirements is primarily a dialogue rather than an exhaustive document. Documentation can play several useful roles in the requirements process but it should not attempt to be the definitive word on what needs doing.
- **Analysis not synthesis**: the process of deciding what needs doing is primarily one of analysis. The process of building something is primarily synthesis. No amount of analysis will create synthesis, the individuals who are best suited to analysts are usually different to those who are best at synthesis.

  Requirements should not specify what is to be built, or how it is to be built, i.e. solutions, only what is needed to move towards the goal. Of course some requirements specify constraints on the construction rather than functionality. Similarly the individuals who know most about the needs may work with the development team to design a review a proposed solution.

Who manages requirements?

The subject of just who is responsible for managing the requirements is worth an article in itself. One of the main reason for IT project failure has been lack of user involvement. I'm sure many readers have seen it: a customer asks for a system, some requirements are written and the IT department disappear for six months. When they resurface with the finished system it might bear little resemblance to what was actually wanted - assuming of course that what was actually wanted hasn't changed in the meantime.

Agile's answer to this was to involve the customer, make them central to the development process.

In Extreme Programming the role of the person who specified what was wanted was actually called "the customer." Yet while many XP advocates seek to fill this role with an actual customer this is not always possible. Indeed, the original XP case study, the Chrysler C3 project, staffed this role with a Business Analyst.

Scrum calls the person who really wants the system to get involved and play the Product Owner role. But there are, at least, two problems with this model. Firstly the person who plays the Product Owner may not have the skills and experience necessary to play the role - just because they want the software doesn't mean they know how to work with a development team.

Second, and a common complaint of Scrum teams, is a lack of time from the person playing the Product Owner. If the person in question is important enough to want the software they probably have other things to do. Spending their days working with unwashed developers may not be high on their priority list.

Indeed, the lack of time and consequent stress and pressure was highlighted in a study as long ago as 2004 (Martin et al., 2004). Equally worryingly is the focus on a single "customer" can result in other stakeholder needs being overlooked - a point made by Tom Gilb. Even if customers are put above and beyond stakeholders there is still a need to consider multiple customers.

For example, Microsoft Word has several million customers. While these customers may be segmented in various groups (Home, Business, Education, etc.) something needs to be done to understand competing needs, and priorities still need to be decided.

In short, the "end user" as requirements gatherer and decider model - whether the XP or Scrum version has problems. What is needed is a requirements professional who can take on these issues and proxy for the final customer/users.

Keeping with the Scrum model this article will use the generic *Product Manager* title to refer to the person(s) who decide what the software needs to do. It is common to find a Business Analyst or Product Manager performing the role, at least on a day to day basis. For more on these roles see my earlier Overload articles (Kelly, 2010a, Kelly, 2009b, Kelly, 2009a). Other titles like Requirements Engineer or Analyst are also used for those filling these roles too but the basic skills set remains the same.

One role which is frequently asked to work on requirements but probably should not is that of Project Manager. While some Project Managers move between Business Analysis or Product Manager roles others confine their role to delivery of projects. Sometimes these individuals are asked to take part in the requirements gathering process. The problem is that requirements elicitation is not part of most Project Manager training.

Take for example the UK PRINCE2 standard project manager qualification. PRINCE 2 assume that what is wanted is known, the method and techniques focus on breaking the work down, risk management, scheduling and the like. It does not cover requirements analysis or capture in any depth.

10-Step model overview

The 10 Step Model shown in Figure 2 outlines framework for aligning the work of requirements engineers - usually a Business Analysts or Product Managers, often called a *Product Owner* with Agile development. The model may be considered a process, a checklist or just an aide-memoire. The model attempts to relate various aspects of Agile requirements analysis advocated by different authors.

The model assume the classic Agile/Scrum/XP iteration, or sprint, time boxed development episodes together with the product backlog / sprint backlog mechanism defined in Scrum (Schwaber and Beedle, 2002) and XP (Beck, 2000). These mechanisms can be seen in the lower right of the diagram. Since much has been written about this cycle already this description will focus on the wider requirements process.

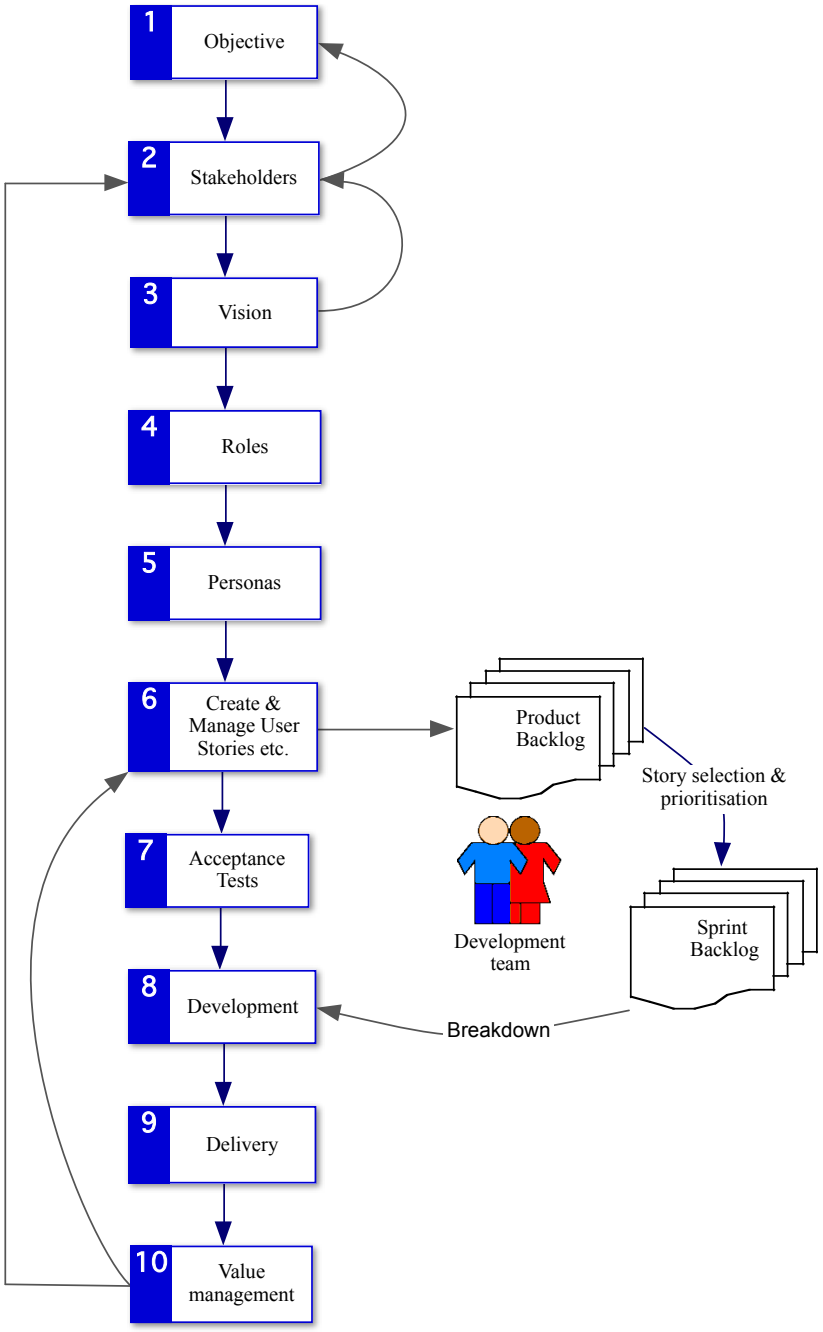

Figure 2 - 10 Step Model

1. **Objective**: the objective is given from outside the model - usually from higher up the management chain. It is the reason the team are brought into being, the reason the project is started, it is the goal the work is aimed towards. (See (Kelly, 2010d) for a longer discussion of this.)
2. **Stakeholders**: stakeholders are those people, and groups of people, who have some interest in the work being undertaken. Stakeholders have their own objectives for the work which might, or might not, align with the objective. Some stakeholders have more stake than others, and some are more significant than others.

This step is not confined to stakeholder identification, this step includes analysis of stakeholder "stake": what stakeholders want from the system, the constrains they impose, how it will create value for them and more.

The stakeholders group includes more than just customers. To start with stakeholders can be split into two large groups: internal stakeholders and external stakeholders. Within corporate IT departments the former will be the larger group while in software companies the latter will be the larger group.

Within the external stakeholders group will be the ultimate customer of the organization. More often than not this group will also benefit from segmentation into specific sub-groups.

3. **Vision**: while the objective is owned by the powers that created the team the vision is created and owned by the team itself. The vision both expands on the objective and answers the objective. If the objective specifies a problem that needs solving the vision gives an answer.
4. **Roles**: roles narrow of the stakeholder base to consider those who will actually interact with the system as envisaged by the vision. It is role holders who interact with the system and thus their needs that need to be considered when determining functionality.
5. **Personas**: personas expand and elaborate certain roles, adding texture so requirements analysts, user design specialists and software developers can better understand and empathise towards those who will use the system. Not all roles will be developed into full personas, and different personas will come to the fore at different times.

   As analysis proceeds from stakeholders through roles to personals there is a natural narrowing shown in Figure 3

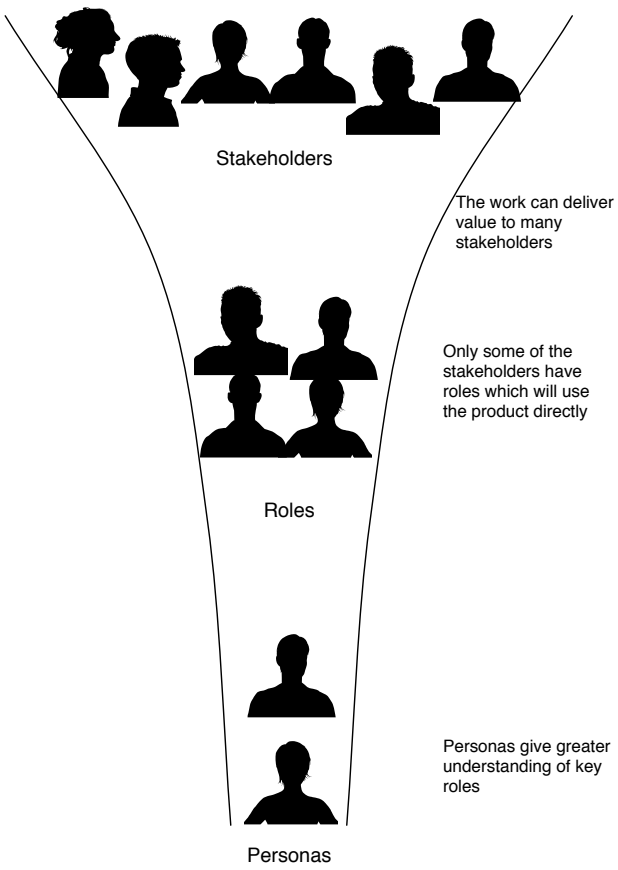

Figure 3 - From stakeholders to personas

6. **Create and manage stories**: when objectives and users are well understood it is time to start specifying what they system will do. Whatever the format used to describe the specifications something needs to be created. Once more than a few requirements have been captured there becomes a need to manage what has been created. This is the step into which much of the existing Agile literature fits: writing User Stories, Managing the Product Backlog and so on. If the 10-step model is being used as a process these process occur in tandem.
7. **Acceptance tests**: once the essence of a story is captured some description of what constitutes done for the story needs be given. How will developers know to stop writing code? Testers know when to pass, or fail, functionality? And requirements specialists know something has actually been done? The answer to all these questions is a set of criteria that determines when a story is complete.
8. **Development**: once a need is identified, understood and acceptance criteria specified it is time to actually do the work, develop the software. (Little needs to be said about this particular step because much has already been written about how development happens in Agile teams.)
9. **Delivery**: once a need is met the product needs to be delivered to the customer. For some systems this is a trivial step, for others it is complicated and involved. Delivering a system in multiple discrete steps is very different from delivering a big-bang all or nothing. Delivering a

system as a shrink-wrapped installable software on a CD is different to a software-as-a-service model.

10. **Value Management**: last but by no means least is the need to close the loop and check that value is actually delivered. The key here is linking the finished product back to stakeholders' needs and objectives. Few organizations can place a dollar amount on a single piece of functionality, for some it may be impossible; but since all requirements start with some stakeholder it should be possible to link return to the stakeholder and check whether the thing that is delivered creates value.

What's the Story?

The basic unit of requirements specification and thus development work, is termed a Story. The format and style of the story can vary widely. Many teams like to use the *User Story* format: "As a [Role] I can [Action] So that [Reason]". This format is commonly associated with Mike Cohn, although Cohn himself credits Rachel Davies (Cohn, 2004), who in turn credits the Connextra development team collectively.

I like to widen this format to allow for Personas and Stakeholders: "As a [Role|Stakeholder|Persona] I can [Action] So that [Reason]." Without Stakeholders some User Stories become tortuous as the writer attempts to give a reason to a role. Personas help bring focus to story and add more background texture.

Although widely taken to be part of Scrum this format is absent from the original Scrum texts (Beedle et al., 1998, Schwaber, 2003, Schwaber and Beedle, 2002). Nor are User Stories present in Beck's Extreme Programming (Beck, 2000). Beck discusses the idea of a "development story" without specifying how it is written.

While User Stories are perhaps the most widely used format some teams still prefer to use *Use Cases* (Cockburn, 2001) or make no attempt to follow a particular format or style. Still other teams use Planguage (Gilb, 2005), while particularly useful for non-functional requirements Planguage is not widely known and requires a particular skill to use effectively.

For the purposes of this discussion the term story will be used generically to cover all possible formats. Story is taken simply to mean: a small piece of development work to be undertaken.

From stakeholders to value management

At first site it may seem odd for value management to appear at the end but this step is about closing the loop, ensuring value was delivered not just promised. There is a symmetry between the stakeholder and value management steps. Stakeholders are ultimately the root of all requirements, no matter how technical. At the end of the day someone, somewhere, must want something from the system. For this person, the stakeholder, there is value (perhaps not financial) to having this thing done.

Value can only be assessed if the stakeholders are known. If nobody wants anything doing to a system then nothing should be done. If value cannot be assigned to work then there is no reason to incur the cost.

The stakeholder might not know what work they want doing, and they are often oblivious to the technical aspects, but then, there is no reason why they

should. The route between stakeholder and change may be complex and non-obvious but it must exist.

Stakeholder analysis and value management are perhaps the two most important steps and the two which certainly deserve more attention in future.

More tools and techniques

There is certainly no shortage of tools and techniques available to the contemporary business analyst or product manager for analysing needs. Whether it is stakeholder analysts, win-loss reports, business analysts modelling, UML diagrams or CATWOE the tools are available. This model does not try to show where each and every tool may be used: not only would it take too long but there are sometimes no clear answers.

What the model does do is firstly, place outputs and expectations at the start of the process: objective and stakeholders should provide a way in here. Secondly it shows where these tools can be used: the stakeholders and roles steps are about understanding customers and needs and it is in these stages that most analysis tools come into play.

While some tools will work within single steps in this model other tools will span multiple steps. The truth is, requirements discovery is not a neat and tidy exercise that occurs in clear cut chunks. Like code development it involves intuition, insight and inspiration which cannot be scheduled.

As a result those charged with discovering, understanding and communicating activities are likely to have several different activity streams occurring at once, overlapping and informing one another.

For example, in tandem with this model forward looking plans and scenarios need to be maintained. Release plans and product roadmaps (Kelly, 2010c) are both informed by the information gathered in the model and feed into the model.

Useful?

This is a deliberately brief explanation of the 10 Step model, I hope readers find the model useful and I would appreciate any feedback on the ideas. For me the model has already filled its original intention of helping explain different aspects of Agile requirements.

I certainly find it helps explain and pull together some of the ideas floating around the discussion on Agile Requirements. Although it is simplest to explain as a process I shy away from calling it that. Rather I prefer to think of it as a check-list and a guide

Perhaps it is better still to view this model as a starting point for your own model. Which steps would you remove? Which would you add? Would you reorder any?

References

ALEXANDER, I. & BEUS-DUKIC, L. 2009. *Discovering Requirements,* Chichester, John Wiley & Sons.

BECK, K. 2000. *Extreme Programming Explained*, Addison-Wesley.

BECK, K. & ANDRES, C. 2004. *Extreme Programming Explained: Embrace Change*, Addison-Wesley.

BEEDLE, M., DEVOS, M., SHARON, Y., SCHWABER, K. & SUTHERLAND, J. 1998. Scrum: A Pattern Language for Hyperproductive Software Development. *Pattern Languages of Program Design "PLoP"* Allerton Park Monticello, Illinois.

CADLE, J., PAUL, D. & TURNER, P. 2010. *Business Analysis Techniques: 72 Essential Tools for Success,* Swansea, BISL (BCS books).

COCKBURN, A. 2001. *Writing Effective Use Cases*, Addison-Wesley.

COHN, M. 2004. *User Stories Applied*, Addison-Wesley.

COMMERCE, O. O. G. 2005. *Managing Successful Projects with PRINCE2,* London, TSO (The Stationary Office).

GILB, T. 2005. *Competitive Engineering*, Butterworth-Heinemann.

KELLY, A. 2004. Why do requirements change? *ACCU Overload*.

KELLY, A. 2008. *Changing Software Development: Learning to Become Agile*, John Wiley & Sons.

KELLY, A. 2009a. On Management #5 - The Product Manager. *ACCU Overload*.

KELLY, A. 2009b. On Management #6 - The BA role. *ACCU Overload*.

KELLY, A. 2010a. "I'm a BA get me out of here" - the role of the Business Analyst on an Agile team. *ACCU Overload*.

KELLY, A. 2010b. *Objective Agility* [Online]. Modern Analyst. Available: http://www.modernanalyst.com/Resources/Articles/tabid/115/articleType/ArticleView/articleId/1502/Objective-Agility-what-does-it-take-to-be-an-Agile-company.aspx [Accessed December 2010 2010].

KELLY, A. 2010c. *Three Plans for Agile* [Online]. Toronto: RWNG. Available: http://www.requirementsnetwork.com/node/2663 [Accessed December 2010 2010].

KELLY, A. 2010d. *Time for Goal Driven Projects* [Online]. Toronto: RQNG. Available: http://www.requirementsnetwork.com/node/2597 [Accessed 23 December 2010 2010].

MARTIN, A., BIDDLE, R. & NOBLE, J. Year. The XP Customer Role in Practice: Three Studies. *In:* Agile Development Conference, 2004 2004 Salt Lake City, Utah.

SCHWABER, K. 2003. *Agile Project Management with Scrum*, Microsoft Press.

SCHWABER, K. & BEEDLE, M. 2002. *Agile Software Development with Scrum*, Addison-Wesley.