NorwalkAberdeen

Welcome

Over the last ten years, Agile has grown from relative obscurity to being the new standard for product development. It has produced better products faster, and teams have enjoyed the more sustainable work pace.

But Agile wasn't designed with Business Analysts in mind. We have had to forge our own path, integrating analysis into the agile environment, and creating a new role for ourselves.

This guide is designed to help you understand your role and to grow in your agile business analysis career.

We hope you find it to be of great use, and we look forward to your feedback.

Don Hussey

Managing Director
NorwalkAberdeen

don.hussey@norwalkaberdeen.com

Concepts

What is Agile?

Agile is a new way to develop products, usually software, but really anything.

It's all about people.

It's all about collaboration.

It's all about mindset.

It's all about continual improvement.

But in the end, it's really all about creating products your customers will love.

There are many ways to be agile. In this guide, we will usually refer to Scrum, the most popular method. Where there are important differences among the methods, we will call those out.

We'll talk a lot about process. But that's not the point. Process is important, but not as important as the items on the left.

Individuals and Interactions

over **processes and tools**

Working Software

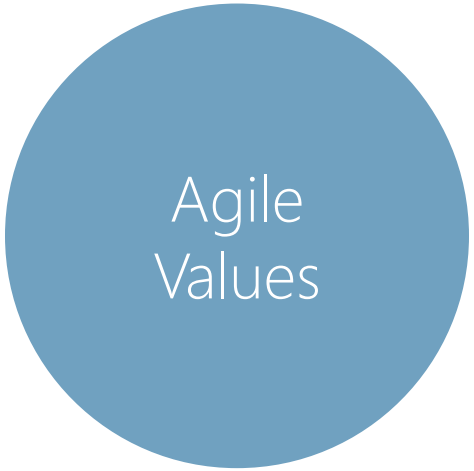
over **comprehensive documentation**

Customer Collaboration

over **contract negotiation**

Responding to Change

over **following a plan**



All of Agile is wrapped up in these four values from the Agile Manifesto (agilemanifesto.org). If you learn only one thing from this guide, make it the values above.

We value all of the things listed above, but we value the things on the left more.

Agile isn't about finding a new process or super-duper new tool to control people's lives. It's about valuing our colleagues, products, customers, and flexibility.

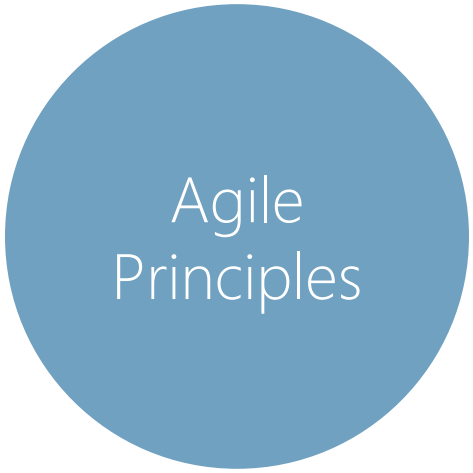
If you follow these four values, it doesn't matter what methodology your organization follows: you are agile.

In your organization, do people run the projects? Or do projects run the people?

As a Business Analyst, do you feel like you can let go of lengthy documents in order to focus on the product your customers will use?

What's more important: working with your customers to create something great, or sticking to the "letter of the law" as defined by a Statement of Work or contract?

When your market environment changes, do your projects rapidly shift course? Or do you stick to the plan, even when it's no longer relevant?



The principles on the right can be found at agilemanifesto.org/principles

They are the principles that the various agile methodologies are based on. Methodologies are important, but remember that our *values* and *principles* are what make us agile.

If you don't work on software, feel free to replace the word with "product."

In the pages that follow, we will look at each one closely.

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity – the art of maximizing the amount of work not done – is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

Everyone agrees that the customer comes first. But if our processes aren't set up to enable it, it won't happen.

Our customers want what they want, how they want it, and as soon as possible. The whole point of Agile is to make that a reality.

Have you ever been on a project status call where everyone reported how things were going? Have you ever said, "requirements are 25% complete?" Customers aren't focused on that. They want to have actual product in their hands. That's what this principle is all about.

"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."

Imagine a project that is two years long, that delivers a product after that time is up. There are so many things that can go wrong:

- The project can run late, preventing the customer from getting the benefit the product provides
- Market change may make the product irrelevant, causing the project to be canceled, and representing a complete financial loss of investment
- The need to deliver on a particular target date may force management to incur unsustainable work hours on the project team, causing inefficiency and burn-out

By delivering a portion of the product every month, on average, each of these risks is dramatically reduced, and the customer is able to take advantage of the incremental benefits delivered.

"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."

This principle turns requirements orthodoxy upside down. Requirement change in waterfall projects can be financially expensive and can wreak havoc on project schedules.

However, we have to reconcile those issues with the reality that businesses, industries, and markets are experiencing a higher degree of change than ever before, making many products obsolete even before they're completed.

By delivering product functionality quickly and piecemeal and giving customers direct control over feature prioritization, we have the opportunity to adapt to change, keeping our teams and customers nimble.

"Business people and developers must work together daily throughout the project."

BAs have often been referred to as "bridges" between business and technology teams. We have certainly created value by helping the two sides to find common understanding, but it has come with a cost.

Bridges are almost always bottlenecks. As the business creates analysis tasks for BAs and developers create hundred-question lists for the BA to take up with the business, the process comes to a standstill. And bridges can collapse – in having one person ferry between the two teams, the enterprise takes on the risk of having a single point of failure in a key process.

Instead, we agile Business Analysts ensure that the *entire* team works closely together daily, freeing us up to do the analysis work that will add the most value to our products and customers.

"Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."

It's a bad management technique to put your people in a difficult situation where they don't have the tools to do the job and then put pressure on them to deliver. But we see this all the time in the work place. Instead, agile teams are given the resources they need to get the job done.

The team is all-important in Agile. They are 100% responsible for creating, testing, and delivering the product. They need management support to get this done.

"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."

Face-to-face communication is critical. Some methodologies even mandate that the team and customers be located in the same *workspace*. Why is it so important? Speeding up collaboration, cutting down on waiting time, and building team identity are only a few of the reasons.

You might say, "...But our developers are in another city/country/time zone!" That's okay – to be agile is to be flexible. Use webcams. Talk with teammates one-on-one by phone. Keep your conversations with them informal. These work-arounds can compensate for not being co-located and will pay dividends over the short-term and long-term.

"Working software is the primary measure of progress."

The customer doesn't care about requirement documents. The customer doesn't care what the code looks like. The customer doesn't care about milestones or story velocity or burn-down charts. The customer only cares about the product features delivered and their quality.

Accordingly, that is how we measure progress on agile projects. Since user stories represent blocks of working product, we use them to gauge how well we are doing.

So, yes, the internal metrics of story velocity and burn-down charts do matter *to us*. But remember that the ultimate goal is getting great product into the customer's hands.

"Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."

Working long hours over the long-term will burn out the team and make you ineffective.

Customers don't care how long or hard you work. They just want their product.

Focus on creating a work environment that is effective. Fine-tune your working processes to maximize the utility of each day and hour. By doing this, you'll increase your productivity and set up a sustainable process that can run indefinitely without crashes.

"Continuous attention to technical excellence and good design enhances agility."

One criticism of Agile that holds *some* degree of merit is that it gives short shrift to technical design. Many teams have jumped right into development without first ensuring that the design is tenable, and many more have failed to maintain it as their projects have evolved.

They've ignored this principle to the detriment of their product and customers. In agile projects, the team must *always* be designing and refining their product.

Similarly, we Business Analysts must constantly be focused on refining *our* work products. Is the backlog properly prioritized? Are there missing elements that we should be working on instead? Are acceptance criteria up to snuff? These all enhance agility too.

"Simplicity – the art of maximizing the amount of work not done – is essential."

"Work smarter, not harder" is often a dictum given by management to teams that are forced to work around constraints and insufficient amounts of resources to get the work done.

But that's exactly what we need to do. We must be lean in our approach to the work. We will not program code that we don't need for this sprint/iteration. We will not write excessive documentation that isn't useful. We will take sensible shortcuts whenever possible.

But we will *never* compromise on the quality of our product.

"The best architectures, requirements, and designs emerge from self-organizing teams."

When you go to the grocery store with your significant other, how do you shop?

Do you have some third party directing you on what to buy and in which order? Do you figure out roles before you go into the store and stick rigidly to them throughout the trip?

Or do you simply have a list of needed items, communicate with each other while shopping, and separate as necessary to pick up items?

That's self-organization. The team members understand the work that needs to be done, and they do it without management directing their steps. It requires communication and trust and is hyper-effective.

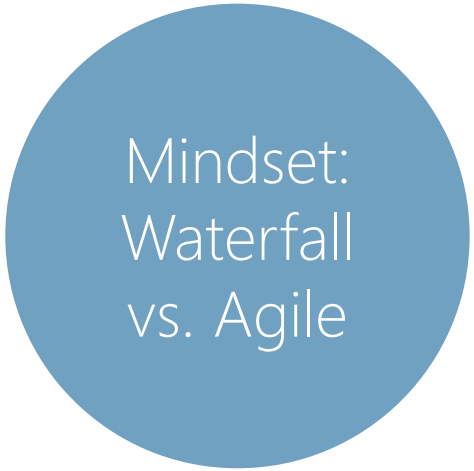
"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

Some things don't work. Some practices are inefficient or counter-productive. Some things that initially seem like a good idea... aren't. As Agilists, we are committed to regularly considering *how* we do the work and improving it.

The more often we do this, the better our process becomes and the less we become attached to bad practices.

The mindset of a project team member varies considerably between agile and waterfall projects.

Agilists tend to value flexibility over predictive planning.



	Waterfall	Agile
Plan	At the beginning of the project, we will determine what needs to be done, predict how long each activity will take, assign each activity to responsible parties, and deliver according to schedule.	At the beginning of the project, we will establish the team and the product vision. We will plan sprints at a high level, but postpone detailed planning – change occurs rapidly, and we need to adapt to it.
Scope	We will define project scope at the beginning of the project. When the scope is delivered, we will know our work is complete. Changes to scope must be approved by senior management.	We will help the Product Owner to set scope as she sees fit. No one knows exactly what the scope should be at the beginning, and she'll need flexibility to keep up with the marketplace.
Requirements	As the Business Analyst, requirements are my most important contribution to the project. I'll devote considerable upfront time to ensuring the requirements are as perfect as possible.	As a team member, I am jointly responsible for delivering the most valuable possible product to our customers. I will work with the PO, customers, and team to contribute however I can.
Change	Change represents both risk and opportunity. We need to keep an eye on trends to make sure we are not blind-sided.	Change happens constantly, and it's often counter-intuitive. We will be nimble and adapt to it as it unfolds.

Roles

The Agile Team

Team Members

Responsible for creating the product

- Design, develop, test, and implement the product
- Estimate the effort of each user story
- Determine sprint scope and own the commitment to the business
- Create and manage delivery plans

Product Owner

Business representative to the team

- Acts as the business owner of the product
- Determines priority and ranking of product backlog items
- Writes the majority of user stories and acceptance criteria

ScrumMaster

Servant leader focused on overcoming team obstacles

- Runs the Scrum process for the team
- Radiates information to the team, ensuring everyone is updated
- Seeks to overcome impediments to help team remain focused on delivering the product

All the agile methodologies lack an explicit Business Analyst role. This is where self-organization comes into play.

The Business Analyst must determine which role he must play, based on team dynamics and the organization's need.

BA as Team Member

Helps create the product

- Performs all the duties of a team member (typically excluding the technical work)
- Works with the Product Owner to develop user stories and their acceptance criteria
- Works with the rest of the team to address business-related spikes and obstacles

BA as PO Assistant/Proxy

Catalyzes product ownership

- Works extremely closely with the Product Owner to enable the PO to effectively handle her role
- Writes the majority of the user stories and acceptance criteria,
- Provides input to the PO on prioritization and dependencies
- Assists the PO with backlog management and story mapping
- Proxies have additional decision-making authority

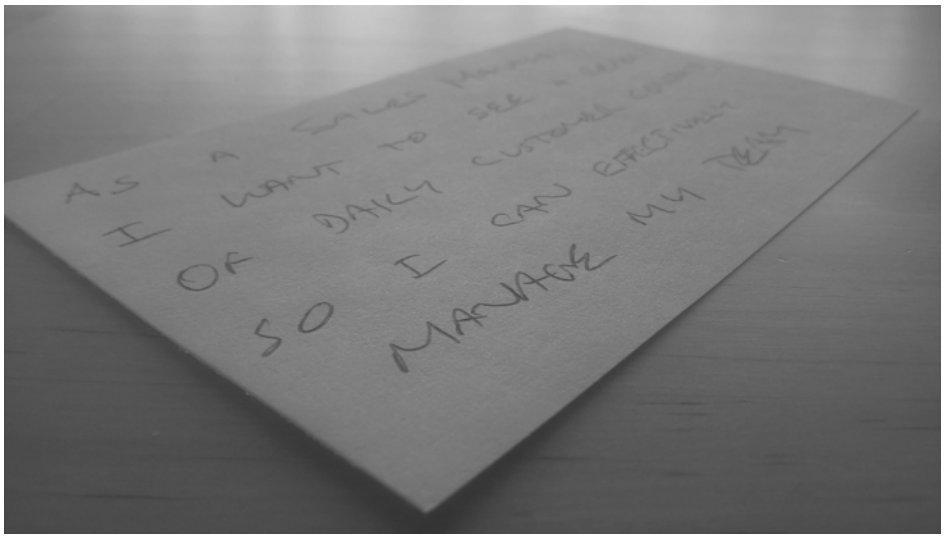
BA as Product Owner

Represents the business

- Performs all the duties of the Product Owner, typically under the direction of a senior product manager

Requirements

User Stories



On agile projects, we typically document requirements in terms of user stories.

User stories are bite-sized statements of functionality desired by a certain type of stakeholder for a particular reason. They are typically a single sentence long, with the following format:

As a _____ (role)
I want _____ (requirement)
So I _____ (rationale)

User stories are not lengthy specifications of every last detail of the feature. Indeed, user stories are often written on index cards. Scott Ambler described the user story as a "reminder to have a conversation with your customer," and that's about right.

They don't have to be perfect, only understandable to the reader.

Examples

As a Sales Manager,
I want to see a report of daily customer contacts,
So I can effectively manage my team.

As a Marketing Analyst,
I want a chart showing web page traffic over time,
So I can gauge how effective content changes are.

As a blogger,
I want to be able to apply HTML styling to my blog posts,
So I can adequately structure my content.

Acceptance Criteria

Just as user stories are high-level expressions of requirements, acceptance criteria specify the details that testers should check when verifying a user story is complete and defect-free.

Developers use them as kind of a checklist when developing the feature and writing automated tests for it. In that way, acceptance criteria act as requirements also.

Try to keep your acceptance criteria succinct and to the point. Just like user stories, they can be as informal as the team can handle. They're not fancy test scripts.

Acceptance criteria are the flip-side to the user story – literally. If you're writing the user story on an index card, the acceptance criteria go on the other side.

Examples

User Story:

As a blogger,
I want to be able to apply HTML styling to my blog posts,
So I can adequately structure my content.

Acceptance Criteria:

User can italicize, bold, and underline text.

User can create numbered and bulleted lists.

User can indent and unindent text.

User can change the font to any of the following: Arial, Helvetica, Times New Roman, Calibri, Courier, Wingdings.

Epics: Splitting Stories

Epics are user stories that are too big to effectively complete in a single sprint. A user story typically comprises no more than 1/6 of the team's velocity.

Instead of working on epics directly, the team decomposes them into user stories… what we usually call "story-splitting."

Here are some strategies to do that.

Functional Decomposition	Breaking a function into smaller sub-functions
Process Decomposition	Breaking up a process into sub-processes or process steps
Structural Decomposition	Breaking a complex item up into smaller pieces

EXAMPLES

Epic	
Blogging	• Author blog post • Edit blog post • Publish blog post
Bank account opening	• Choose account type • Submit identification documents • Request debit card
Marketing dashboard	• Site traffic chart • Social media summary • E-mail metrics table

NorwalkAberdeen

Agile in Action

8:30 I arrive to work, chat with a team member, and go through e-mail.

9:00 Daily scrum. One of the developers said there are inconsistencies between two stories; I'll meet up with her later to figure it out.

9:30 Chatted with the developer. No inconsistency after all, but we ended up clearing up lots of other questions she had.

10:15 Had a conversation with one of our salespeople. He has a lot of blogging requirements. I'll have to chat with the PO.

11:00 Okay, the PO said that blogging is a new high-priority epic. He asked me to break it down into features and stories and review with him later.

11:45 It seems like the core features are: (1) Authoring blogs, reading blogs, and integrating calls-to-action. I'm going to break stories out that way.

1:15 I have lunch in my cubicle today – there is a webinar (from NorwalkAberdeen, of course) that I want to catch up on.

2:00 One of the developers calls me over to help figure out a solution with the data architect. We apparently need some data we don't have. Agile doesn't fix *all* problems, unfortunately.

3:00 I reviewed the blogging epic/features/stories with the PO. He said that we should add tagging as a feature. He wants to get all the functionality done over the next two sprints. I'll have to spend some more time prioritizing the backlog with him over the next couple days.

4:25 Gave a heads-up to the ScrumMaster that we would likely be shifting gears in the next sprint to handle blogging. She shrugged and said, "Okay."

5:00 I go home, instead of working another five hours. Agile is all about maintaining a *sustainable* pace.

The Agile mindset is crucial. Mastering and integrating the values and principles of the Agile Manifesto is more important than learning any methodology.

Never walk into a retrospective without first considering how you and your team can improve. Throughout the sprint, jot down ideas, and share them with the team at the retrospective.

Keep user stories short. Don't depend on them as documentation. Consider them reminders to have a conversation about the requirements.

Don't worry *too* much about agile processes. It's valuable to learn the process, but it's more valuable to create a collaborative spirit in your team.

Keep acceptance criteria simple. Don't depend on them as documentation either. Consider them reminders about things to test and verify.

The most effective Agilists are a combination of purist and pragmatist. Knowing when to strive for adherence to agile methods and when to simply get the job done most expediently is a sign of leadership.

Appendix

Adaptive Planning	An approach to planning in which plan details are more detailed in the short term and less detailed in the long term, enabling flexibility and adaptation to change.
Agile	An approach to product development characterized by close collaboration with customers, short and iterative delivery, and sustainable work pace.
Agile Manifesto	A collection of four values and twelve principles published in 2001, which laid the groundwork for agile software development.
Backlog	A collection of planned work items.
Burn-Down Chart	A chart depicting the remaining amount of work to be delivered in a sprint, usually broken down by day.
Daily Scrum	A daily Scrum meeting in which team members summarize the previous day's accomplishments, their plans for the current day, and any impediments they face.
Daily Stand-Up	Term popularized by Extreme Programming, equivalent to the Daily Scrum.
Epic	A requirement statement or category that is too big to deliver effectively in a single sprint.
Predictive Planning	An approach to planning in which management makes predictions about team productivity and external factors and bases plans around those predictions.
Product Backlog	The list of user stories that have not yet been assigned to a particular sprint.
Spike	A work item representing issue resolution or research.
Sprint	A 1- to 4-week iteration, producing a releasable version of the product.
Sprint Backlog	The list of user stories comprising the scope of the sprint.
Story Point	An arbitrarily-scored metric reflecting a relative amount of product development effort.
Story-Splitting	The process of breaking epics and large user stories into smaller, more manageable stories.
User Story	A short statement of a product user's need, specifying the user type and rationale.
Velocity	The number of story points delivered by a team in the course of a sprint.
Waterfall	A phased approach to software development in which a phase does not begin until the previous phase's work is complete.