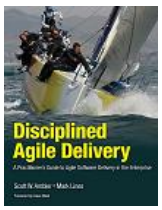
Agile Modeling

Agile Analysis



Agile software developers, just like traditional software developers, perform analysis activities. Unlike traditional developers, agilists approach analysis in a highly collaborative manner and do so on a just-in-time (JIT) basis. Analysis is so important to us we do it every single day. In this article, I discuss:

1. What is Analysis?

The purpose of analysis is to understand what will be built, why it should be built, how much it will likely cost to build (estimation), and in what order it should be built (prioritization).> This is similar to requirements elicitation, the purpose of which is to determine what your users want to have built. The main difference is that the focus of requirements gathering is on understanding your users and their potential usage of the system, whereas the focus of analysis shifts to understanding the system itself and exploring the details of the problem domain. Another way to look at analysis is that it represents the middle ground between requirements and design, the process by which your mindset shifts from what needs to be built to how it will be built.

2. What is Agile Analysis?

Let's begin by describing what agile analysis isn't:

- It isn't a phase in the lifecycle of your project
- It isn't a task on your project schedule
- It isn't a means unto itself>

What is agile analysis? From the previous discussion of what an agile business system analyst does, your agile analysis efforts should exhibit the following traits:

1. **Agile analysis should be communication rich**. Agile developers prefer warm communication techniques such as face-to-face discussion and video conferencing over cold techniques such as documentation and email, as described in the Communication article. Agile developers prefer to work as closely as possible to their project stakeholders, following AM's Active Stakeholder Participation practice wherever possible.
2. **Agile analysis is highly iterative**. As Martin (2002) points out analysis and design activities both rely on each other: estimating is part of analysis yet that relies on some design being performed to identify how something could be implemented and your design efforts rely on sufficient analysis being performed to define what needs to be built. Hence analysis is iterative.
3. **Agile analysis is incremental**.> Martin (2002) also correctly points out that agile analysis is incremental, that you don't need to build systems all in one go. This philosophy supports the incremental approach favored by common agile software processes where the work is broken done into small, achievable "chunks" of functionality. These chunks should be implementable within a short period of time, often as little as hours or days.
4. **Agile analysis explores and illuminates the problem space**. Your primary goal is to identify and understand what your project stakeholders need of your system. Furthermore, this information needs to be communicated to everyone involved with the project, including both developers and stakeholders. This promotes understanding of, and agreement with, the overall vision of your project.
5. **Agile analysis includes estimation and prioritization of requirements**. It is during estimation and prioritization of requirements where software development becomes “real” for project stakeholders. It is very easy to say that you want something but much hard to agree to a price for it, let alone to trade it off for something that is immediately more important to you.
6. **Agile analysis results in artifacts that are just good enough**. If any artifacts are created at all as the result of your agile analysis (you'll be following the AM practice Discard Temporary Models after all) they should be either agile models or agile documents. Such artifacts are typically far from perfect, they just barely fulfill their purpose and no more.

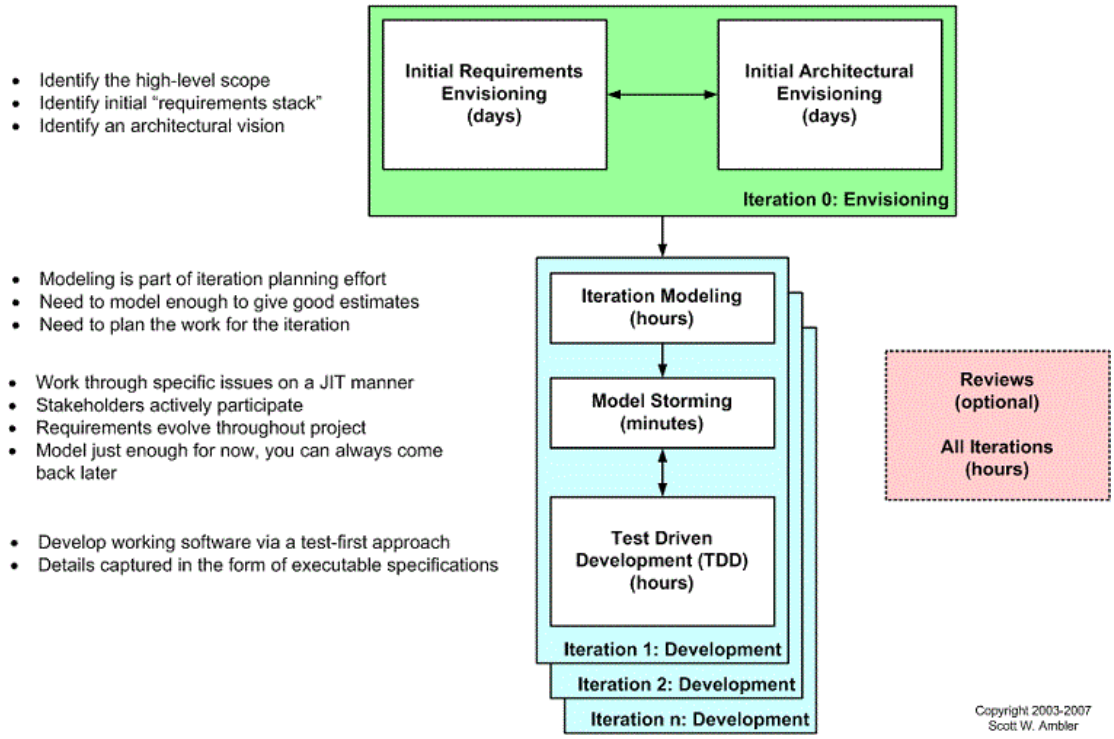
Here is my definition for agile analysis:

> Agile analysis is highly evolutionary and collaborative process where developers and project stakeholders actively work together on a just-in-time (JIT) basis to understand the domain, to identify what needs to be built, to estimate that functionality, to prioritize the functionality, and in the process optionally producing artifacts that are just barely good enough.

3. Analysis Through the Lifecycle

Figure 1 depicts the lifecycle of Agile Model Driven Development (AMDD). During "iteration 0", the first iteration of an agile project, you need to get your project organized and going in the right direction. Part of that effort is the initial envisioning of the requirements and the architecture so that you are able to answer critical questions about the scope, cost, schedule, and technical strategy of your project. Details about the business domain are identified on a just-in-time (JIT) basis during iterations via initial iteration modeling at the beginning of each iteration or by modeling storming throughout the iteration. Analysis is so important to agilists that we do it every day.

Figure 1. The Agile Model Driven Development (AMDD) lifecycle for software projects.



4. Some Potential Analysis Models

Are the artifacts created taking an agile approach to analysis any different than the ones created as the result of traditional analysis? In a way they are. They are in fact the same types of artifacts, a use case diagram is a use case diagram after all, but the way in which they are created are different. The artifacts are created following the principles and practices of AM, they are just barely good enough and are often discarded so as to travel light.

Table 1 lists common artifacts used during analysis modeling and suggests a simple tool with which you could create such an artifact. It is interesting to note that this list is meant to be representative, a more thorough list is presented in the article Artifacts for Agile Modeling and in the book Agile Modeling.

Table 1. Candidate artifacts for analysis modeling.

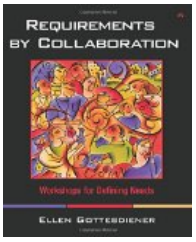
Artifact	Simple Tool	Description
Activity Diagram	Whiteboard	UML activity diagrams are used during analysis modeling to explore the logic of a usage scenario (see below), system use case, or the flow of logic of a business process. In many ways activity diagrams are the object-oriented equivalent of flow charts and data-flow diagrams (DFDs).
Class Diagram	Whiteboard	Class diagrams show the classes of the system, their inter-relationships (including inheritance, aggregation, and association), and the operations and attributes of the classes. During analysis you can use class diagrams to represent your conceptual model which depict your detailed understanding of the problem space for your system.
Constraint definition	Index card	A constraint is a restriction on the degree of freedom you have in providing a solution. Constraints are effectively global requirements for your project.
CRC model	Index cards	A Class Responsibility Collaborator (CRC) model is a collection of standard index cards, each of which have been divided into three sections, indicating the name of the class, the responsibilities of the class, and the collaborators of the class. Like class diagrams, CRC models are used during analysis modeling for conceptual modeling.
Data flow diagram (DFD)	Whiteboard	A data-flow diagram (DFD) shows the movement of data within a system between processes, entities, and data stores. When analysis modeling a DFD can be used to model the existing and/or proposed business processes that your system will support.
Entity/Relationship (E/R) diagram (data diagram)	Whiteboard	E/R diagrams show the main entities, their data attributes, and the relationships between those entities. Like class diagrams E/R diagrams can be used for conceptual modeling, in many ways E/R diagrams can be thought of as a subset of class diagrams.
Flow chart	Whiteboard	Flow charts are used in a similar manner to activity diagrams.
Robustness diagrams	Whiteboard	Robustness diagrams can be used to analyze usage scenarios to identify candidate classes and major user interface elements (screens, reports, ...).
Sequence diagram	Whiteboard	Sequence diagrams are used to model the logic of usage scenarios. Sequence diagrams model the flow of logic within your system in a visual manner, enabling you to both explore and validate your logic.
State chart diagram	Whiteboard	State chart diagrams depict the various states, and the transitions between those states, that an entity exhibits. During analysis modeling state chart diagrams are used to explore the lifecycle of an entity that exhibits complex behavior.
System use case	Paper	A use case is a sequence of actions that provide a measurable value to an actor. A system use case includes high-level implementation decisions in it. For example, a system use case will refer to specific user interface components - such as screens, HTML pages, or reports. System use cases will also reflect fundamental architectural decisions, such as the use of ATMs versus cell phones to access your bank account.
UI prototype	Whiteboard	A user interface (UI) prototype models the user interface of your system. As a part of analysis modeling it enables you to explore the problem space that your system addresses in a manner that your project stakeholders understand. Remember, the user interface is the system to most people.
Usage scenario	Index card	A usage scenario is exactly what its name indicates - the description of a potential way that your system is used.
Use case diagram	Whiteboard	The use-case diagram depicts a collection of use cases, actors, their associations, and optionally a system boundary box. When analysis modeling a use case diagram can be used to depict the business functionality, at a

		high-level, that your system will support. It can also be used to depict the scope of the various releases of your system via the use of color or system boundary boxes.

5. Rethinking the Role of Analysts

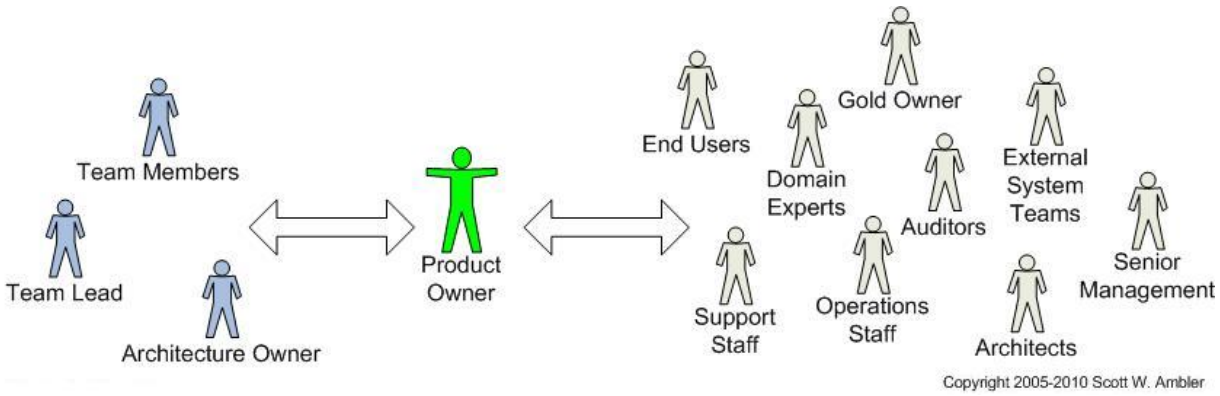
Analysis is a very important activity on any software development project, regardless of paradigm. It is an organizational design decision to define what roles you will have and what those roles will do. In traditional development there is often someone(s) in the role of analyst, and some organizations even choose to distinguish between analyst types and will have system analysts (SAs), business analysts (BAs), business system analysts (BSAs), data analysts, process analysts, and so on. In agile development we make different organizational design decisions. So, although we still perform analysis we haven't made the decision to have someone in that specific role.

If you are an existing BA, here are a few strategies which you may want to consider when moving to agile software development:



1. Recognize that agile teams are made up of generalizing specialists instead of specialists (such as analysts). Your analysis skills are valuable, and are clearly a good start, but you need to be able to do more. Be prepared to work closely with other team members to help transfer your skills to them and to pick up new skills from them.
2. A business analyst, particularly one from the business side, may take on the role of stakeholder/customer/product owner on an agile project, representing the stakeholder community throughout the project (see Figure 2). They may also lead an initial requirements envisioning effort during Iteration 0.
3. For more details, see the article Rethinking the Role of Business Analysts.

Figure 2. BSA as Product Owner.



6. Conclusion

The nature of analysis has changed. Several decades ago analysis was seen as a transformational process within a serial project lifecycle. With the rising popularity of object-orientation in the 1980s and 1990s analysis was soon seen as part of an iterative and incremental process, and in the new millennia the nature of analysis is transforming once again. Today analysis is better envisioned as a highly collaborative, iterative, and incremental endeavor involving both developers and stakeholders.> The evolution of the analysis process necessitates an evolution in the way that business system analysts (BSAs) work, and in many situations this role arguably disappears in favor of developers who are generalizing specialists.

7. Acknowledgements

I'd like to acknowledge the input on the Agile Modeling mailing list of the following people: James Bielak, Adam Geras, Ron Jeffries, Kent J. McDonald, Les Munday, Paul Oldfield, Stephen Palmer, Tom Pardee, Dave Rooney, Gabriel Tanase, and Paul Tiseo.

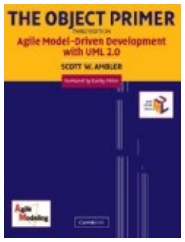
Let Us Help

We actively work with clients around the world to improve their information technology (IT) practices, typically in the role of mentor/coach, team lead, or trainer. A full description of what we do, and how to contact us, can be found at Scott Ambler + Associates.

Recommended Reading

This book, Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise describes the Disciplined Agile Delivery (DAD) process decision framework. The DAD framework is a people-first, learning-oriented hybrid agile approach to IT solution delivery. It has a risk-value delivery lifecycle, is goal-driven, is enterprise aware, and provides the foundation for scaling agile. This book is particularly important for anyone who wants to understand how agile works from end-to-end within an enterprise setting. Data professionals will find it interesting because it shows how agile modeling and agile database techniques fit into the overall solution delivery process. Enterprise professionals will find it interesting beause it explicitly promotes the idea that disciplined agile teams should be enterprise aware and therefore work closely with enterprise teams. Existing agile developers will find it interesting because it shows how to extend Scrum-based and Kanban-based strategies to provide a coherent, end-to-end streamlined delivery process.

The Object Primer 3rd Edition: Agile Model Driven Development with UML 2 is an important reference book for agile modelers, describing how to develop 35 types of agile models including all 13 UML 2 diagrams. Furthermore, this book describes the fundamental programming and testing techniques for successful agile solution delivery. The book also shows how to move from your agile models to source code, how to succeed at



implementation techniques such as refactoring and test-driven development(TDD). The Object Primer also includes a chapter overviewing the critical database development techniques (database refactoring, object/relational mapping, legacy analysis, and database access coding) from my award-winning Agile Database Techniquesbook.