

## Afterword And Final Thoughts

This section contains some notes about the decisions and experiences involved with creating this book. (It may also include some final thoughts concerning best practices when working with requirements.<sup>224</sup>).

### It Started As A Matter Transporter

Originally I wanted to demonstrate how one could build a Matter Transporter using the Rational Analysis toolsets (RSA, ReqPro, etc), because that is what I was familiar with, and I already had a Vision of what a prototype Matter Transporter would look like.

It didn't take long to realize that a Matter Transporter would not be an appropriate business model to demonstrate the process (too technical and the technology would detract attention away from the real point of the documentation).

A simpler and widely recognizable example was required. That is when I came across a paper named 'A Method of Translating Business Use Cases into System Use Cases', by George Abraham & Il-Yeol Song. This described a process for transitioning from a BUC model to a System Use Case model. It contains an example of automating a restaurant. The paper does not go into anything like the detail that this work does, but it gave me a foundation for some of the ideas that went into this work.

In order to make the examples accessible to a larger audience it was necessary to drop the IBM/Rational toolset and use something more accessible. Since I was writing the text in MS Word, MS Office seemed the natural extension. I also wanted to use a toolset that allowed me to demonstrate the activities in the process, but has minimal influence over how those processes are conducted.

### Where to put the Word recommendations

One of my favorite training courses to teach is Recommendations for Using Properties and Styles with MS Word<sup>225</sup>. I think it is important to

---

<sup>224</sup> In 2 minds whether the add these or simply reference them as an article on my website.

<sup>225</sup> This is not an analysis course, but could be of interest to anyone using MS Word on a regular basis.

understand these practices while trying to follow the process, but these recommendations do not easily fit into the process. In the end I decided to put them at the beginning of the book as an optional Chapter that may be skipped if the information is already familiar to the reader.

## Rework

As I was working through the examples and attempting to model according to the guidelines that I had previously specified, I often found myself needing to step outside of the guidelines in order to deliver the information that I was trying to specify. This would entail going back and modifying the guidelines. Of course because every artifact is inter-related to at least one other artifact, this meant that modifying the rules for modeling of one component would have a knock-on effect for the rules for specifying another component of the model. This involved continuously going back and reworking previous chapters each time a guideline was changed by a subsequent chapter. (Just as one would expect if using an analysis process in the ‘real’ world.)

## State Transition Diagrams

The sections on modeling with STDs were the biggest headache: And STDs are a tool that I have been using probably longer than any other model components described in this book. Many methods use STDs as part of their process, but very few define a formal notation for use with STDs. The 2 that I am aware of are:

- Harel – which is a highly structured notation for executing models, using states and transitions with StateMate.
- Shlaer/Mellor – Which uses a formalization of Moore notation in order to allow STDs to be executed by the BridgePoint modeling tool.

Both notations are due to the fact that the STD is intended to be executable by the tool that promotes their notation. I have no tool to promote with the notation that is provided in this book. Therefore, I had the freedom to come up with the most flexible notation that I could think of, that could be implemented by a tool that may wish to execute the model.

The notation includes the creation and deletion of class instances, purely to allow real-time execution of the model. I wanted to stress again, that it

is not necessary to analyze the creation and deletion of objects, unless real-time execution is a concern.

If verification of the model is going to be manual, then creation and deletion may be specified (for completeness), but it not necessary to ensure that every created object is subsequently deleted.

Similarly, one may verify the model using a batch execution tool where the necessary instances are predefined. Again, it is not necessary to ensure that instances are either created or deleted.

In addition, if you are not intending to execute the models, then I suggest that STDs do not need to be modeled in a rigorous formal manner. STDs are the heart of an executable model, but may also be used to simply derive the scope of a class and the operations that form its requirements.

## **Model Driven Architecture**

The results of the process are intended to be executable. If the class operations are written in an executable language (perhaps as activity diagrams that may be interpreted into some sort of machine code), then the complete model may form a solution to the business vision.

This concept is often known as Model Driven Architecture (or MDA). I want to stress that deriving an executable solution to the problem is not the intention of this work. The model describes a complete requirements solution to the problem, not a design solution. The major differences between the 2 models are that the requirements model does not:

- Take into account efficient use of memory or processing.
- Specify the interfaces to other applications or systems, except where it references interface specifications.
- Include a user interface design.
- Consider re-use of existing software (COTS for example).
- Make any attempt to describe a software architecture. I.e. separation of functions and data into tiers or client/server type architectures.

Manual execution of the logical model through the use of State Transition Diagrams, is extremely painful and time consuming. I have worked with several tools that have made an attempt to alleviate this pain by

introducing some sort of state execution into the model. Unfortunately, all efforts appear to have been abandoned<sup>226</sup>. If you can find an automated method of executing the model states, then it should be a relatively simple task to produce reports or logs of a ‘run’, as sequence diagrams. This is the real power of the executable model – automatically generated diagrams that show the sequence of messages (and events) passing between the model components as a use case is executed.

## How To Use With Agile

The term ‘Agile’ when referring to software development, is quite popular at the moment. One might wonder, how can such a rigid and detailed process be adopted by an ‘Agile’ environment. This explanation uses the Scrum<sup>227</sup> method to describe how the process may be adopted on an Agile project.

Let’s start by defining some Scrum concepts:

**Sprint** – A Sprint is a product release cycle. Sprints are time-boxed, that is to say that release dates do not vary, but the functionality delivered at the end of a Sprint may.

**Sprint Backlog** – The functionality assigned to a Sprint is placed in a Sprint Backlog. At the start of a Sprint the backlog is populated with stories from the Product Backlog.

**Product Backlog** – A Product Backlog is a container for all identified work for the product being developed. The work is cataloged in the form of user stories. User stories are sized (estimated for work effort) and prioritized. They are assigned to Sprint Backlogs as required, according to their size and priority.

**User Story** – All work for a Product Backlog is documented according to a standard template called a User Story. The user story describes the work to be done in terms of what is needed in order to satisfy a user need, or change to a need.

---

<sup>226</sup> Search for the current status of StateMate (Harel notation) and BridgePoint (Shlaer/Mellor).

<sup>227</sup> Scrum being the most popular and well-defined Agile method, to the best of my knowledge.

**Vision** – A vision is an overriding document describing the scope of the product.

**Scrum Master** – The Scrum Master is a sort of team lead/QA/PM role. The Scrum Master ensures that the product work progresses according to plan and that the correct procedures are followed.

**Product Owner** – The Product Owner is the person responsible for making decisions when questions arise about the work in the Product Backlog. The Product Owner acts as the Subject Matter Expert for the needs of the users.

Note that Scrum does not define a complete life cycle from business vision through to user acceptance testing. Scrum is concerned with the needs of a Product, and does not consider the needs of a business enterprise. Neither does the Scrum team include formal user acceptance testing. At the end of each Sprint a product version is released, but only for delivery to acceptance testing. The product release is only deployed once user acceptance testing determines that the product is an improvement over the current deployment.

This of course means that there are many more Sprints than there are releases (or deployments) into production.

Figure 262: shows how the life cycle of a Scrum method compares with that of the Enterprise method, described in this book.

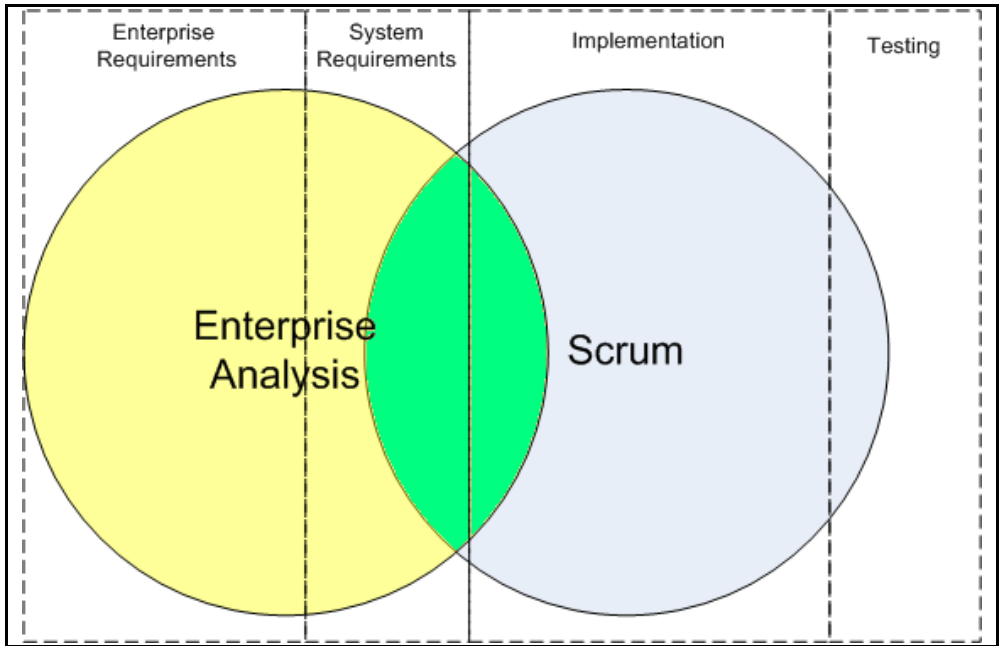


Figure 262: Enterprise Compared To Scrum

The Enterprise Analysis process is concerned with business and system requirements. Scrum is mostly concerned with software implementation, some system requirements and some testing.<sup>228</sup> The green area is where the 2 methods overlap. In this area the Scrum and Enterprise Analysis concepts may be mapped as follows:

**Sprint** – A Sprint is equivalent to an iteration. The difference being that iterations are not necessarily time-locked.

**Sprint Backlog** – These may be mapped to the use cases and stakeholder requests that are assigned to an iteration.

**Product Backlog** – This is the equivalent of the application requirements repository.

**User Story** – A user story is normally a simplified use case. Use cases are more rigid and formalized than user stories. A user story may look more

---

<sup>228</sup> This is my understanding from courses on Scrum, but I am sure that there will be other developers that disagree.

like a use case storyboard. Where the use cases are the responsibility of the analyst, the user stories are defined by the Product Owner.

**Vision** – This is the equivalent to a Product Vision<sup>229</sup> in an Enterprise Analysis.

**Scrum Master** – The quality assurance analyst and project manager traditionally perform many of the responsibilities of the Scrum Master. Although the Scrum Master is more involved with the success of the product than a project.

**Product Owner** – The is expected to be one or SMEs from the business. In an Enterprise Analysis method the Analyst may play the equivalent role. Although the analyst is not necessarily a SME, they are expected to be able to perform an equivalent function.

The Enterprise Analysis process probably provides more deliverables than Scrum normally expects. But there is no reason why the 2 cannot work together. The only conflict may be in the area of the Product Owner; where Scrum expects a business expert. There should however be no reason why an analyst cannot contribute to the Scrum project as a business Product Owner given enough time and experience,

## **Adoption And Modifications**

The guidelines in this work are not perfect and there will be situations arising where it is prudent to add or modify some of the rules for modeling the components. What I can guarantee, is that if you do not yet have a defined requirements analysis method, then this is a good place to start. You do not want (nor should you attempt) to introduce all of these guidelines into a new development effort.

- Perhaps you are wishing to introduce use cases into your approach, and would like some guidelines for documenting use case documents. Then use these, and ignore everything else.

---

<sup>229</sup> The product vision is similar to the project vision, but as it applies to a single product and not across the enterprise.

- Perhaps you are already documenting use case from an established template, but wish to supplement them with activity diagrams. Then the activity diagram modeling guidelines will help with that effort.
- Perhaps you are currently developing system (or application) use cases directly from the business vision, but find yourself having trouble keeping the project scope under control. Then introducing the business use case model into the process might help with your problem.
- Maybe you already have data modeling guidelines in place, but wish to introduce a process for modeling class functionality. Then the chapters on modeling with states might prove useful.

## Final Ramblings

To conclude I would like to describe some basic concepts that I try live by whether working on a requirement/analysis project.

### Anything Is Possible

Given enough time and resources, I believe that if you can imagine it, you can build it.

When a customer comes to the analyst with a solution that sounds impossible, I do not discard their proposal, just because nothing like it has been built before. When you start investigating their problems that led to their proposal for the ‘impossible’ solution, you may find that an alternative solution is entirely feasible.

No matter how outrageous the customer request, go through the motions of gathering requirements anyway, because you may well come up with a partial or even a complete alternative solution.

### The Customer/Stakeholder Is Always Right

Or to put it another way; no-one is ‘completely wrong’. Everyone’s point of view is correct to some extent. The problem is that one point of view may be ‘more correct’ than others. When dealing with stakeholders try to identify the parts of their POV that you can agree with and build from there.

For example, a customer states that the system must be built on a SQL database. Why SQL and why database? The customer explains, because SQL is the best according to a reliability study and a database is where the

business data will be kept. So what they really want is a place to maintain business data that is the ‘best’ in terms of reliability. The customer was not ‘wrong’ when they said that they need a SQL database, because with some interpretation we will eventually both come to a solution that we agree upon. The customer gets their database, but how it is implemented doesn’t matter, so long it is a reliable way of maintaining the data and performs to all intents and purposes like ‘SQL Database’.

### **A Requirement Analysis Makes No Assumptions About Technology**

This is where coming from a technical back ground does not necessarily help with documenting the requirements. I try to forget everything I know about PCs, databases, user interfaces, RAM, ROM, hard-drives, etc. The only thing I need to know is that there is a Central Processing Unit, (that operates at faster than light speeds), Memory (that is infinite) and input and output interfaces. (Interfaces can communicate with just about anything; People, Hardware devices and other software devices<sup>230</sup>.)

I think that is all I need to know. What makes these things work (and their limitations) need on be considered when making design and costing decisions.

### **Only Introduce Design As A Means To Improving Understanding**

When I say ‘design’ when referring to requirements, I normally mean ‘organizational decisions’ or architecture. Ideas such as organizing data into classes. In an analysis model this is only intended as a means for breaking data down into manageable and understandable chunks. It is not intended to specify how a database will be laid out, although using good database design as a foundation for organizing analysis data might just save effort in the long term.

Even organizing business requirements into use cases is an organization of functionality that could be interpreted as design.

Organization is ‘design’ only if the requirements say it is; otherwise its packaging.

### **Verbosity Is Good; Ambiguity Bad**

---

<sup>230</sup> Even creatures from another dimension, if that’s what the documentation says.

Explain everything to the extent that it achieves an understanding, but no more. Also remember who your audience is – an explanation to one person may relay a different understanding to a person from a different background. So keep your explanations audience focused and hide confusing information from an unintended audience.

### **Everything Is Observed From The Outside Of The System**

Related to the comment about ‘technology being agnostic’; not only do we not know how the system is operating, once an input has been, we only see what the system produces. How it produced that output is unknown. Just because a system states that it performed an action as a result of some stimulus does not mean that it actually performed that action, nor should we assume that it did. What we see is the system stating that this was done and that’s what the requirement should state .. not that it is actually going to do.

### **Stakeholders Do Not Know What They ‘Want’**

Stakeholders have a pretty good idea of what the problem is, but if they knew the solution (want), why would they need the services of an analyst?

Stakeholders will tell you what they want in terms of things that they know. An analyst figures out what it is they need from statements of what they want.

Referring to an excellent article on ‘the 5 whys<sup>231</sup>’ (Probably titled Root Cause Analysis). It proposes that in order to get to the root cause of a customer’s desire one should pose the question ‘Why?’ up to 5 times.

An example might be that a stakeholder requests a windowed user interface. This is what the customer thinks they want. The analyst asks why? The answer maybe because the users wish to be able to do 2 or more things at the same time. Why would they wish to do 2 things at the same time? Because calculating some of the responses takes several minutes, and they want the user to be able to do something else while waiting. Why does it take several minutes to respond? .. Eventually we establish that the problem is with the system response times.

---

<sup>231</sup> Probably titled something similar, but different. Maybe it was called ‘Root Cause analysis’.

What the stakeholder ‘needs’ is for the user’s time to be used more efficiently. A windowed system was a solution that the stakeholder had envisioned.

### **Cost Is Not A Modeling concern**

It is the project manager’s job is to worry about cost. Everyone on the project should be concerned about the money that they are spending in order to deliver the solution, be this person-hours or purchased hardware. What the analyst is not concerned about when modeling, is making the model efficient in order to reduce cost to build and upgrade the system. The analyst reduces cost by making the model understandable and eliminating rework.

### **Take Away The Business Technology**

When documenting the business process, why should we describe steps as if they are performed manually, when he business already has an automated solution in place; why not just describe the steps as ‘already automated’ or ‘manual’? The reason is, just because the business has an automated process today, does not mean that the same automated process will be in place tomorrow. Part of your effort may be to replace parts of the automated process or even return it back to a manual process when the automation is not working. The business will describe their day-to-day function in terms of what they do today. Not what they were once doing. It is one of the skills of the analyst to be able to remove the technology from the process and describe what it would look like if performed without automation<sup>232</sup>.

### **Really Finally**

My next project is to attempt to specify a requirements analysis tool that will automate the guidelines in this work and that is developed using these same guidelines. Until then I hope that even if you didn’t agree with everything, then it at least stimulated the reader into wondering, why isn’t this a good idea and how would I do it better?

---

<sup>232</sup> As with many things in my life, I am not 100% convinced of this statement, and reserve the right to change my mind in future revisions of the process.

‘Constructive’ criticism is always welcome.