Agile Product & Project Management

Vol. 8, No. 9

CUTTER CONSORTIUM

# Requirements for Managing Requirements

by Suzanne Robertson, Senior Consultant, Cutter Consortium

This *Executive Report* discusses how managers can use consistent and understandable requirements knowledge as input to making decisions and steering a project down its most agile path.

Executive Report

# Requirements for Managing Requirements

AGILE PRODUCT & PROJECT MANAGEMENT ADVISORY SERVICE

**Executive Report, Vol. 8, No. 9**

**by Suzanne Robertson, Senior Consultant, Cutter Consortium**

Progressive organizations recognize that well-understood requirements are at the root of delivering relevant products. These organizations commit resources to improve skills for discovering and communicating requirements and for integrating these techniques with agile development. When requirements at all levels are consistently communicable, the project manager can use them as input for making decisions. Factors about the current state of requirements including their number, level of completion, priority, and dependencies enable managers to make better estimates, monitor progress, allocate responsibilities, and respond to change. This *Executive Report* focuses on how project managers can use requirements knowledge to steer projects down an agile path.

Project teams that can talk consistently about requirements at different levels of detail are in the driver's seat. In this scenario, team members can communicate with each other and with other stakeholders and make relevant decisions about what they know, what they don't know, and where to put the effort to get to where they want to go. These individuals know that requirements are not just a list of statements saying, "The product shall do x or y." Instead they recognize that requirements knowledge exists at a number of levels of detail and that, most importantly, all the details are traceable between levels. The knowledge model presented in this report represents a scheme for keeping track of all the requirements-related subject matter. Suppose you have a consistent way of stating and linking your business goals, stakeholders, project scope, terminology, constraints, assumptions, functional requirements, nonfunctional requirements, and other types of requirements knowledge. The appealing thing about this is that once you have a scheme for organizing knowledge, you also have the freedom to decide which parts of that knowledge — and to what level of detail — need to be investigated and communicated for each of your projects. In other words, some degree of formality frees you to make the choices appropriate for accelerating each particular project.

During the past 10 years, there have been significant improvements in techniques for discovering and communicating requirements. Instead of limiting requirements-trawling techniques to interviewing, people are using apprenticing, simulation, stories, scenarios, collaborative software, creativity workshops, integrated modeling, family therapy, and many other techniques to discover the relevant requirements more quickly.

The most effective requirements practitioners discover requirements by choosing the trawling technique that fits each particular situation and express their findings by integrating a wide range of modeling techniques with natural language techniques. These specialists also have a structure for determining the appropriate level of detail in each case. Agile requirements specialists are well aware that the primary concern is to have an accurate understanding of the requirements and to ensure that understanding is communicated throughout the project team. The degree of formality necessary to communicate that understanding depends on project factors such as geographical proximity of the team, experience of the team, location and experience of all the stakeholders, size of the project, involvement of other organizations, and the impact of getting it wrong.

The earlier you have some consistent way of expressing requirements, the earlier you can quantify the size of the project. Along with this you can identify the areas of certainty and uncertainty as well as the parts that will benefit from further investigation and the parts that are already well understood. In other words, you are in a position to use the high-level requirements (but this does not mean vague) to do a risk and benefit analysis and to decide on an appropriate strategy for your project.

This *Executive Report* looks at high, medium, and low levels of requirements and the connections between them. The gradual buildup of a class diagram identifies classes of requirements knowledge and the relationships among them. An example is presented in the report to help you consider different ways of communicating the requirements within your own projects. Specifically, the report begins with a discussion of how to build a requirements knowledge model. It then guides you through making estimates for your project that are traceable to the requirements knowledge. It next offers a primer on using early requirements knowledge to count function points. The report concludes by helping you consider the agility potential for your project.

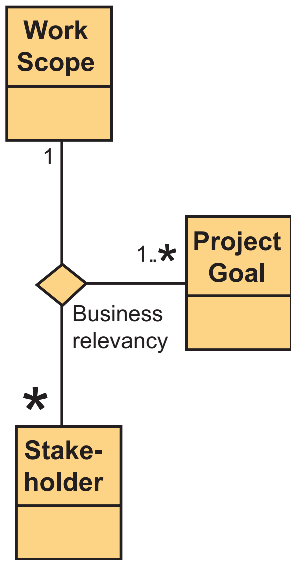

Figure 1 — These three pieces of requirements knowledge provide the first project management countables.

## EARLY COUNTABLES

A good starting point for quickly taking stock of your requirements knowledge is to do a stakeholder, goals, and scope (SGS) analysis. Each of these areas is shown as a different class of requirements knowledge in Figure 1 (we'll be building on this figure throughout the report). The relationship between these classes of knowledge is identified as "business relevancy." In other words, the scope that you study has to be sufficient and relevant to the goals, and the stakeholders you are concerned with need to be relevant and sufficient for the goals you intend to meet and the scope you need to understand. Asterisks (*) in the figure indicate many; for instance,

The ***Agile Product & Project Management Advisory Service Executive Report*** is published by the Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA. Client Services: Tel: +1 781 641 9876 or, within North America, +1 800 492 1650; Fax: +1 781 648 1950 or, within North America, +1 800 888 1816; E-mail: service@cutter.com; Web site: www.cutter.com. Group Publisher: Kara Letourneau, E-mail: kletourneau@cutter.com. Managing Editor: Cindy Swain, E-mail: cswain@cutter.com. ISSN: 1536-2981. ©2007 Cutter Consortium. All rights reserved. Unauthorized reproduction in any form, including photocopying, faxing, image scanning, and downloading electronic copies, is against the law. Reprints make an excellent training tool. For more information about reprints and/or back issues of Cutter Consortium publications, call +1 781 648 8700 or e-mail service@cutter.com.

there are many stakeholders, whereas there is just one work scope.

When you do an SGS analysis, your aim, as quickly as possible, is to identify the following requirements knowledge:

- **For each stakeholder:**
  - — The role you expect the stakeholder to play (e.g., engineer, domain expert, tester).
  - — The knowledge you expect the stakeholder to contribute (e.g., order taking, payroll rules, insurance calculation).
  - — The specific person who will supply this knowledge.
- **For each project goal:**
  - — A one-sentence description of the goal.
  - — A one-sentence statement of the advantage the business seeks to achieve if the goal is met.
  - — A measurement of how you will know if the goal has been met.
- **For the work scope:**
  - — The people, organizations, and other systems that are outside the detailed work/business that you intend to investigate (referred to as adjacent systems).
  - — The inputs and outputs between the adjacent systems and the work/business that you intend to study. In most cases, these inputs and outputs are data, although, depending on the domain, they might also be material or control signals.

### *Questions to Ask*

The requirements knowledge you get from a quick SGS analysis provides input for raising early project management questions.

Look for gaps in your stakeholder analysis. If you have any types of knowledge for which you do not have a stakeholder, then you need to determine how you will get that knowledge. If you have any roles without specific people to fill them, then chances are that nobody will take responsibility for that knowledge. What about people who do not have a role/ knowledge assigned to them? If you truly have this situation, then you have stakeholders who are unnecessary and will probably slow the project down. Also, identify potential conflicts by looking for stakeholders who are interested in the same type of knowledge. Address each conflict by making an agreement on how decisions will be made if/when the conflict occurs.

One of the most common causes for project failure is the lack of a quantified goal. Do you have a description/advantage/ measurement of each goal? If you do not, then you have nothing to guide you in choosing options and priorities. Do you really have a project goal that acts as a guide for the whole project? Or do you have detailed requirements masquerading as goals? If you have more than three to five project goals, then chances are you have descended into detail without stating the overall business problem.

Use the work scope to help analyze the size of the project. How many input data flows are there? How many outputs? How many stores of data are there within the scope? How many adjacent systems are there? Given your history, how long do you estimate that it will take you to understand the requirements for an input flow and its related output flows? Given the budget — time and money — for your project, can you deal with all these inputs and outputs in the time available? Can you put priorities on the inputs and outputs? Which ones contribute most to the project goals?

When using the SGS approach[1] people ask, "Should we first find the stakeholders, analyze the goals, or try to define the work and product scope?" You will get the fastest results if you do not do any one of these things before the other; instead, do them in parallel.

[1]The SGS approach is part of the Volere requirements techniques from Cutter Senior Consultants Suzanne Robertson and James Robertson [1]. Volere is a set of techniques and templates for discovering, communicating, tracing, and managing requirements. The techniques are widely used on commercial, scientific, and engineering projects.

Take an iterative approach as illustrated in Figure 2. This makes a lot of sense because discoveries about one lead you to questions and answers about another. You are looking for a *balance* among the sources of the subject matter (stakeholders), the business reasons for doing the project (goals), and the subject matter you need to investigate (scope).

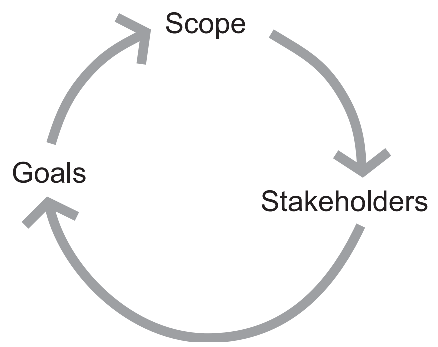

Figure 2 — The stakeholder, goal, and scope cycle iteratively explores and defines the requirements space.

Other questions that you can raise as a result of doing an SGS analysis are:

- Is the benefit of doing this project worth the investment?
- Given our resources, is this a viable project?

Failure to face up to these questions often results in projects that either do not deliver business benefit or stumble along until they are eventually cancelled after having wasted resources that could have been better used elsewhere.

Bear in mind that these are questions that are driven by the "highest-level" requirements knowledge. You are using that knowledge to identify uncomfortable issues early so that you can choose projects that provide real benefits, quantify what you intend to manage, and give the projects a flying start.

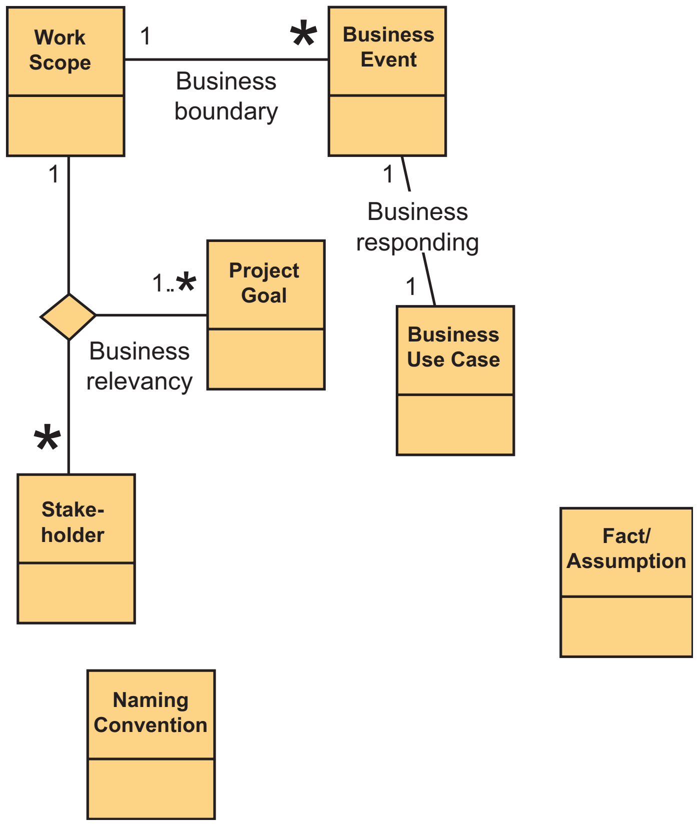

Figure 3 — Business events and business use cases (BUCs) provide a functional partitioning of the work scope.

## LEVELS OF KNOWLEDGE

The team will use many and varied approaches for discovering and capturing the requirements at a more detailed level. The requirements can be in many forms — models, prototypes, or scenarios; the form does not matter. However, if you want to be able to manage the project and respond to change, there is one thing that does matter: it is vital that you are able to trace the requirements at lower levels back to the high-level requirements and vice versa. This section discusses how you can use a requirements knowledge model to ensure traceability, keep track of progress, and improve your potential for agility.

In Figure 3, four more classes of knowledge are added to the knowledge model:

1. Business event
2. Business use case

3. Naming convention
4. Fact/assumption

All of these give the project manager input for making decisions. Notice that the work scope has been partitioned into a number of business events, and the relationship states that each business event must be traceable back to the work scope so that, at the lower level of detail, the business boundary is consistent with the higher level. How do you verify this consistency? Remember the data inputs and outputs defined on the work scope during the stakeholder, goals, and scope analysis? Each one of those inputs and outputs is attached to a business event.

At this stage, it is useful to focus on a specific example so that we can discuss how a project manager can benefit from the additional classes of knowledge. In this discussion, we'll be examining the four classes of knowledge discussed above, shown in Figure 3, as well as additional classes of knowledge to be introduced later in the report.

***Business Events***

Figure 4 is a context diagram showing the work scope of a project concerned with the work of managing the stocking and loaning of books in a library. The circle in the center represents the work to be investigated in order to understand the requirements. The squares around the periphery represent adjacent systems (in this case, book borrower and publisher) that are connected to the work by named interfaces. The diagram defines the scope of the work to be investigated by declaring and naming four input data flows and five output data flows. Each of the interfaces either comes from or goes to one of the adjacent systems.

The knowledge model shows that the business boundary declared by the work scope is partitioned into a number of business events. Table 1 illustrates how the business events would look in our example.

So what does this event partitioning do for the project manager? He or she can see that the work is broken up into five pieces, each of which can be traced back (using the input and output names) to the declared scope. The project manager can use this understanding to talk to the team and address questions like:

- Which of the business events has the highest priority according to the project goals?
- Which event(s) should we concentrate on first?
- Has any investigation work already been done on any of these events? We don't want to repeat anything when we can reuse.

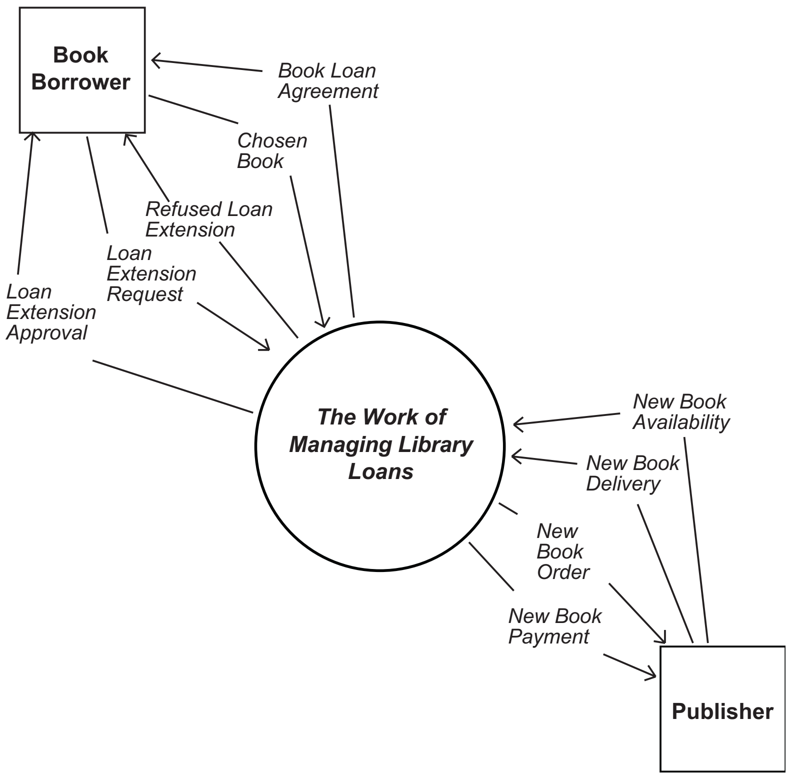

Figure 4 — This context diagram shows the business boundary for investigating the work of managing library loans.

Table 1 — Business Events in Library Example

| No. | Event Name | Input | Output |
|---|---|---|---|
| 1 | Borrower chooses book | Chosen Book | Book Loan Agreement |
| 2 | Borrower wants to extend loan | Loan Extension Request | • Loan Extension Approval<br>• Refused Loan Extension |
| 3 | Publisher has new book available | New Book Availability | New Book Order |
| 4 | Publisher delivers new book | New Book Delivery | *None* |
| 5 | Time to pay for new books | *None* | New Book Payment |

- Which members of the team will work on which events?
- How long does it normally take us to deliver product for events of this size? It might be too early to ask this question, but maybe some member of the team has a gut feeling or maybe it would be worth taking one event and prototyping it to come up with a feel for the complexity.
- Which events do you anticipate will be the most difficult given the stakeholder involvement that you need?

These questions are all concerned with the project manager being able to keep track of the pieces of the project and identify problem areas as early as possible. Then the manager can help the team to balance its resources and do as much as possible to anticipate and address problems. All of these questions are possible early in the project if the project manager and the team have a shared way of talking about their requirements knowledge.

### *Business Use Cases*

Suppose that the team decides that Business Event 2 (borrower wants to extend loan), shown in Table 2, is the one that has the highest priority.

The team then takes that event and does some requirements trawling to understand the details of the business response to the event. This response is called the business use case (often referred to as a BUC).

The BUC can be expressed in many different ways including as:

- An activity diagram
- Interview notes
- A business use case scenario
- A story card
- A tape or video recording
- A sequence diagram
- A process model
- A prototype

Suppose that the analyst decided to write a BUC scenario that goes something like this:

### Business Use Case Scenario for Event 2: Borrower Wants to Extend Loan

- The borrower gives the loan extension request to the duty librarian.
- The librarian looks up the book loan agreement for the requested extension.
- The librarian looks up any other books that are currently loaned to the borrower.
- If the due return date on each of the books is later than today's date

then …

— The librarian tells the borrower there is a loan extension approval.

— The librarian records the loan extension.

otherwise …

— The librarian tells the borrower there is a refused loan extension.

— The librarian records the refused extension.

Table 2 — Business Event 2

| No. | Event Name | Input | Output |
|---|---|---|---|
| 2 | Borrower wants to extend loan | Loan Extension Request | • Loan Extension Approval<br>• Refused Loan Extension |

— The librarian asks the borrower to return the overdue books.

In capturing the BUC for a business event, the analyst has captured the business rules that must be carried out whenever that event takes place. In other words, the analyst has captured the business requirements. With an understanding of the business use case, the analyst has the basis for identifying the most relevant and advantageous product.

It does not matter how the BUC is expressed providing that the business responding relationship (refer back to Figure 3) between business event and business use case is traceable. Once again, it is the input and output flows of data that preserve that traceability. Often when studying the details of a business use case, the requirements analyst discovers another flow of data around the periphery of the business use case. When that happens, the analyst adds that flow to the business event boundary as well as to the overall work scope.

### *Naming Conventions and Facts and Assumptions*

In the growing knowledge model in Figure 3, there are two more classes of knowledge: naming conventions and facts and assumptions. As you start to understand more about the terminology used in the project, you can trap a lot of requirements knowledge by adding the definitions of terms to the naming conventions. For example, while you are investigating the BUC for the business event of "borrower wants to extend loan," you learn more details about what is meant by a "loan extension request." If you define the meaning of the term, then the whole team will have the same understanding.

We can define the loan extension request as follows:

> **Loan extension request:** This flow of information contains the details of a borrower's request to extend an existing loan for a library book.

The data contents are: borrower name, borrower number, book title, return due date, requested extension date.

The other class of knowledge — facts/assumptions — is there to help the project team bring dependencies, issues, and decisions out into the open. For example, if there is a dependency on another parallel project, then that should be stated. And if there is a decision to exclude some functionality, then that decision should be written down along with its rationale; otherwise, it will surface again later and cause confusion and wasted time.

Notice that naming conventions and facts/assumptions do not have relationship links connecting them to other classes of knowledge. The reason for this is that they are potentially connected to all the other classes of knowledge. As an example, loan extension request shows up on the work scope; it is also input to Event 2, and it will show up in BUC 2 and eventually in some of the detailed requirements. The point is that wherever the term is referenced, it should carry the same meaning — the one defined in the naming conventions.

### *Product Use Cases*

The next class of knowledge to add to the model is the product use case (often referred to as a PUC), as seen in Figure 5. The PUC is the part of the business use case that will be implemented as a product. The analyst investigates the BUC to a level of detail that makes it possible to decide, given the constraints and goals, which parts of the BUC should be carried out by the product.

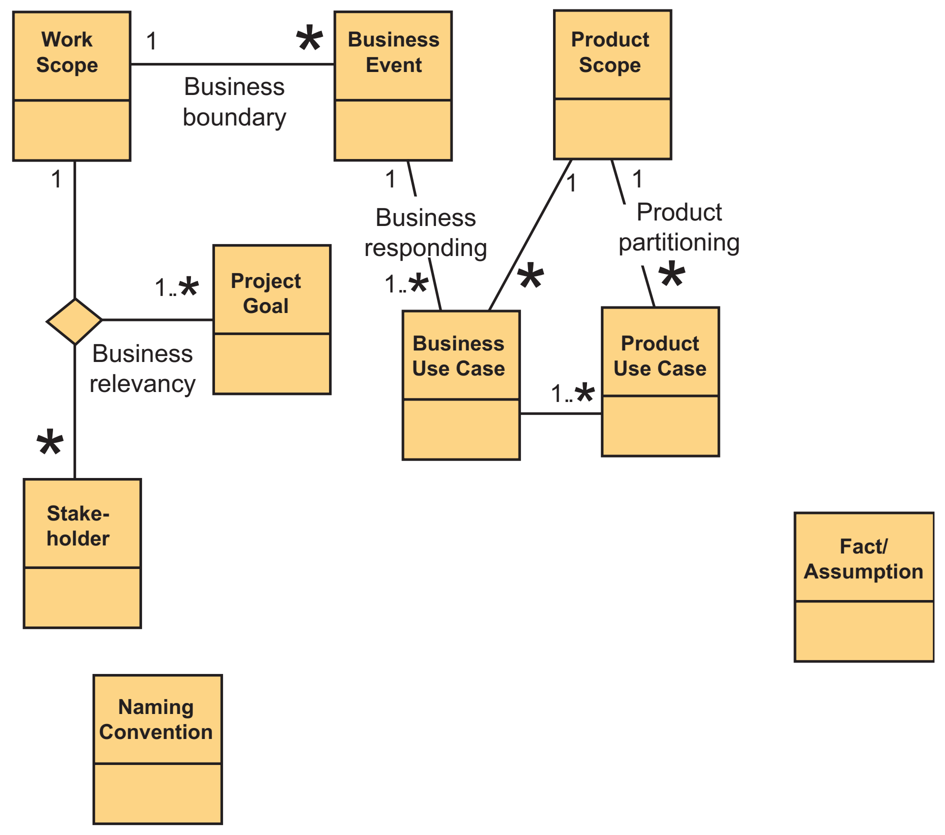

Figure 5 — The addition of the product use case (PUC) to the knowledge model highlights the distinction between the business rules and the parts of those business rules that will be carried out by the product.

The reason for spending time on the BUC is to ensure that there is an understanding of the real business problem before trying to come up with ways of solving it. Analysts can test whether they have a good enough understanding of a BUC by determining whether they can identify a beneficial corresponding PUC. In other words, given the understanding of the business problem, what are the parts of it that would benefit most from help? When asking this question, the analyst is exploring the options for a particular BUC taking into account the project goals and the constraints, as shown in Figure 5.

Continuing with the BUC for Event 2 (borrower wants to extend loan), the analyst can look at each detail and ask: Would there be a business benefit if the product does this, and do the constraints allow us to do this?

Suppose that the analyst annotates the BUC with ideas for what the product will do as follows:

**Business Use Case Scenario for Event 2: Borrower Wants to Extend Loan** *(Annotated with ideas for the PUC)*

- The borrower gives the loan extension request to the duty librarian. *(borrower and librarian)*
- The librarian looks up the book loan agreement for the requested extension. *(librarian and product)*
- The librarian looks up any other books that are currently loaned to the borrower. *(librarian and product)*
- If the due return date on each of the books is later than today's date *(product)*

then …

- — The librarian tells the borrower there is a loan extension approval. *(borrower and librarian)*
- — The librarian records the loan extension. *(librarian and product)*

otherwise …

- — The librarian tells the borrower there is a refused loan extension. *(librarian and borrower)*
- — The librarian records the refused extension. *(librarian and product)*
- — The librarian asks the borrower to return the overdue books. *(librarian and borrower)*

Bear in mind that this is only one of potentially many ideas for what the PUC could be for this BUC. For example, in one alternative PUC, the borrower could have a direct interface with the product, which would mean that more of the functionality of the BUC would be included in the PUC. The chosen PUC depends on the best mix between the goals and constraints and is often best settled on by building a quick version of the PUC to test out the ideas. For the purpose of this example, let's suppose that you have decided to go with the ideas on the annotated BUC. Then the resulting PUC scenario would be as follows:

**Product Use Case Scenario for Event 2: Borrower Wants to Extend Loan**

**PUC name: Extend Loan**

- The librarian enters the loan extension request.
- The product finds all outstanding loans for that borrower.
- The product identifies loans that are overdue: those with a return due date before today's date.
- The product informs the librarian of overdue loans.
- The librarian enters the loan extension or the refused loan extension.

Suppose that you determined the PUCs for each of the BUCs on the list of events, then, as you see in Figure 5, the summary of all the PUCs defines the product scope. Notice how the product scope is related to a number of PUCs by the product partitioning relationship.

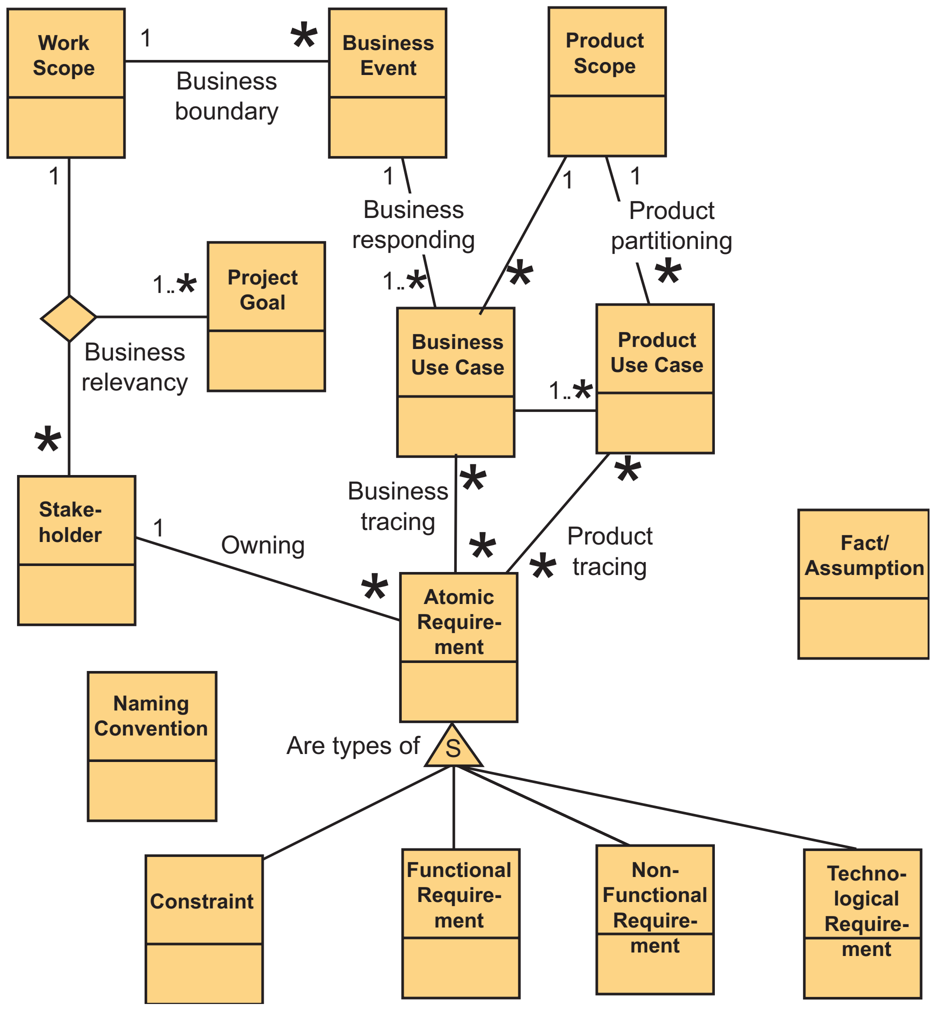

Figure 6 — Functional, nonfunctional, constraint, and technological requirements are all types of atomic requirements.

### *Atomic Requirements*

The next level of detail is the measurable atomic requirements that you derive from the PUCs. The knowledge model in Figure 6 contains a number of new classes of requirements knowledge. It shows the atomic requirement and subtypes (S) of the atomic requirement: functional requirement, nonfunctional requirement, and technological requirement (along with constraint, mentioned above). In other words, these subtypes are all different sorts of atomic requirements. The product tracing relationship between the atomic requirement and the PUC shows that for each product use case there can be many (*) atomic requirements and that each atomic requirement might be related to many (*) product use cases. In other words, an individual atomic requirement might be duplicated in a number of parts of the product.

The functional requirements are concerned with what functions that product is required to carry out (e.g., find overdue loans or record loan extension date). If you are working in an agile environment, then you are unlikely to need to write functional requirements as they will be taken care of by your scenarios.

The nonfunctional requirements[2] deal with how well the product is required to carry out the functionality. What look-and-feel, usability, performance, operational, maintainability, security, cultural, and legal requirements does the product have?

The technological requirements are requirements that do not relate to the business problem and are not the concern of the business specialist. These requirements are there because the designer has made a decision about how to achieve the functional and nonfunctional requirements by using particular technologies.

Notice also that the constraints mentioned in conjunction with the earlier discussion about PUCs are also classified as a type of atomic requirement. It helps to think of a constraint as a requirement that does not have any negotiability (e.g., the product shall be implemented using our existing network of personal computers; the product shall interface with the existing library loans database).

An atomic requirement has a number of attributes (see Figure 7

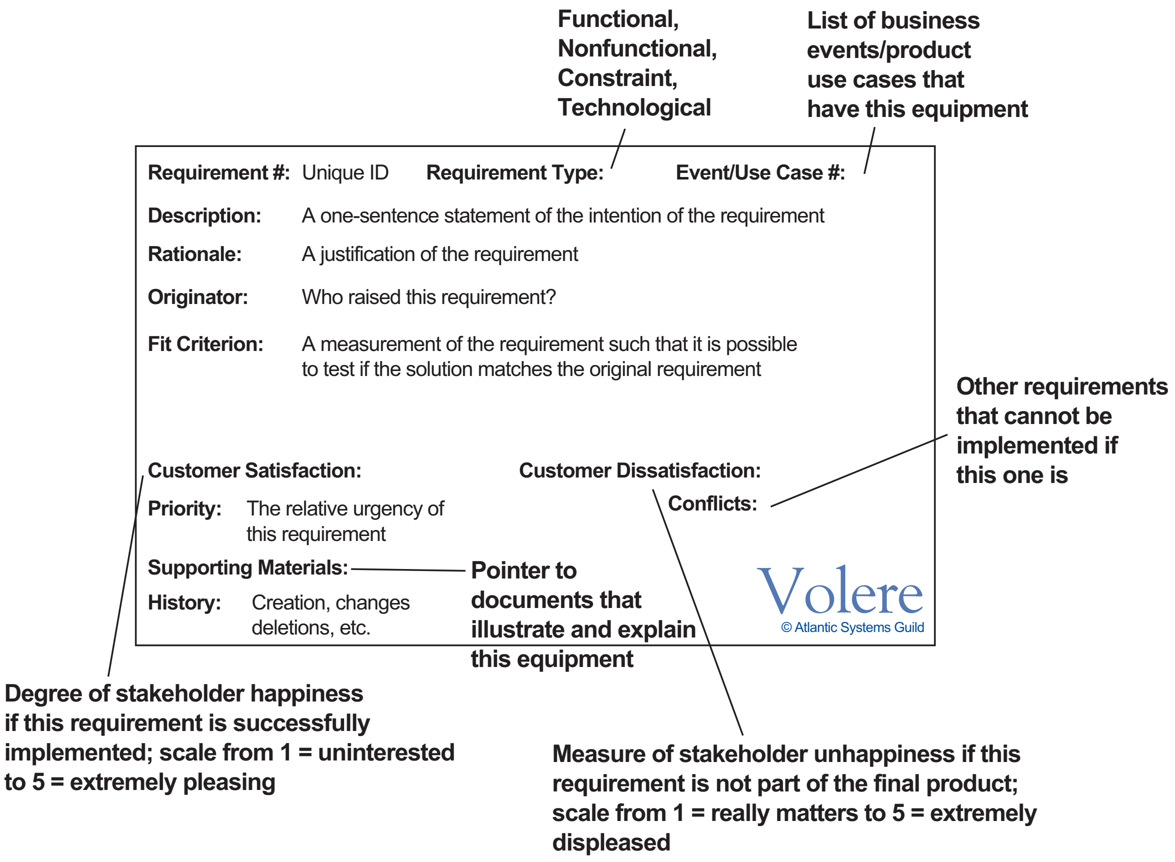
Functional, Nonfunctional, Constraint, Technological
List of business events/product use cases that have this equipment
Requirement #: Unique ID
Requirement Type:
Event/Use Case #:
Description: A one-sentence statement of the intention of the requirement
Rationale: A justification of the requirement
Originator: Who raised this requirement?
Fit Criterion: A measurement of the requirement such that it is possible to test if the solution matches the original requirement
Customer Satisfaction:
Customer Dissatisfaction:
Priority: The relative urgency of this requirement
Conflicts:
Supporting Materials:
History: Creation, changes deletions, etc.
Other requirements that cannot be implemented if this one is
Pointer to documents that illustrate and explain this equipment
Volere
© Atlantic Systems Guild
Degree of stakeholder happiness if this requirement is successfully implemented; scale from 1 = uninterested to 5 = extremely pleasing
Measure of stakeholder unhappiness if this requirement is not part of the final product; scale from 1 = really matters to 5 = extremely displeased

Figure 7 — An atomic requirement is a collection of attributes that together make the requirement testable and traceable. It is useful to have a checklist (such as the Volere requirements shell illustrated in this figure) to help you think of all the attributes that might be relevant to each of your requirements.

[2]Refer to the Volere Requirements Template [3] for a detailed discussion of nonfunctional requirements.

for an example) that provide further input to making decisions. The project manager can use the atomic requirements to get a feel for the size, complexity, and progress of the project. How many atomic requirements are there for each PUC? How many of the requirements have a defined fit criterion (this is what makes them testable)? Do we have any requirements without a rationale? Are the terms used in the atomic requirements defined in the naming conventions?

If you are working in a small, colocated team, then you have a high potential for agility. In this case, a PUC scenario supported by well-maintained naming conventions and nonfunctional requirements might provide enough requirements input for building your product. In other words, you might not need to write atomic functional requirements. However, the larger, more distributed, and more fragmented your team, the lower your potential for agility and the more you need to define unambiguous and measurable requirements at all levels.

### Testing and Implementation

The requirements knowledge model that we have built up contains requirements at a number of levels of detail, and each level is traceable both upward and downward. We have discussed how the project manager can use the classes of requirements knowledge to monitor the completeness of the requirements, raise questions, and make strategic decisions. In Figure 8, three additional classes of knowledge — test case, system architecture component, and implementation unit — provide the project manager with additional input.

Test case is a class of knowledge that belongs to the testers. If you look inside it, you find attributes that keep track of the description of the tests that have been run together with the results. Notice that each test case has a testing relationship with many (*) atomic requirements and also with many PUCs. It is these relationships that ensure that the testers are in fact designing and running tests that match the specified requirements. If you look back at the atomic requirement in Figure 7, you see an attribute called fit criterion. The fit criterion is the measurement of the requirement, and

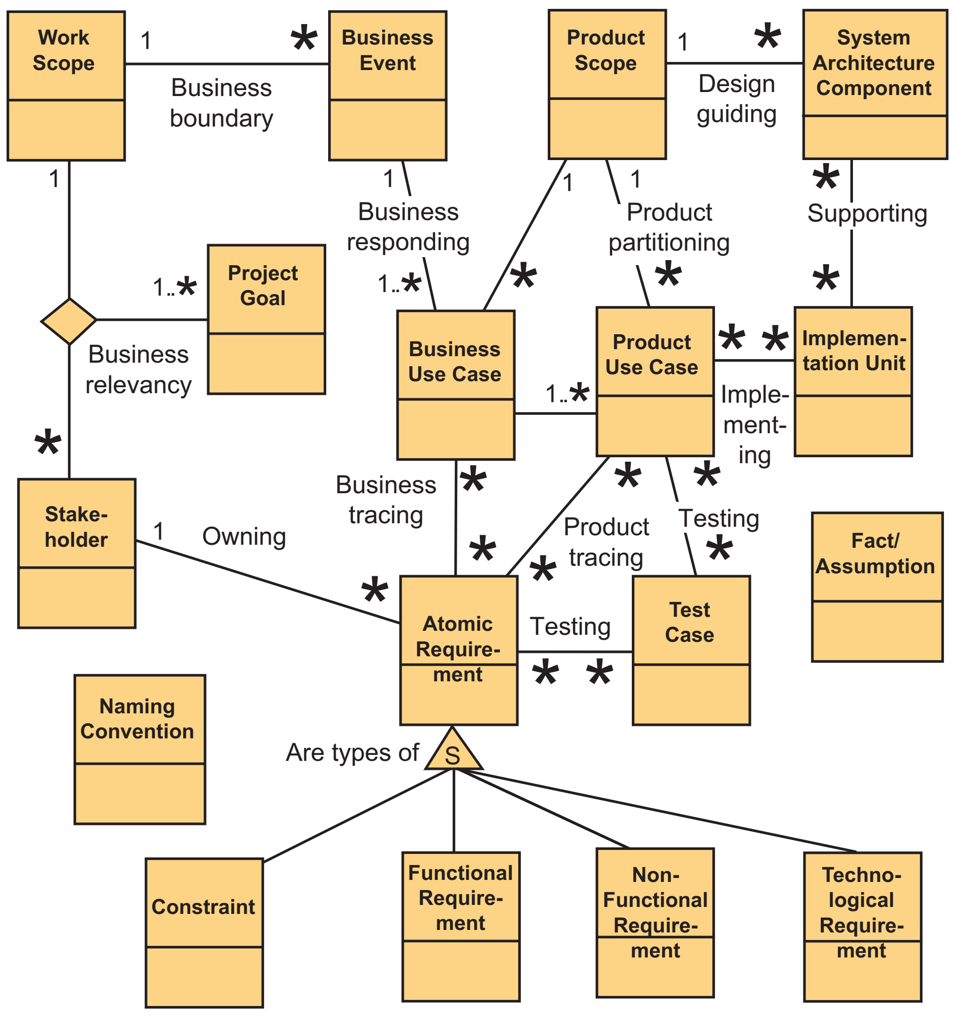

Figure 8 — The test cases belong to the testers, but they have a traceable relationship to the atomic requirements and the PUCs. Similarly, to deal with change, the PUCs need to be traceable to the implementation units.

it is precisely this that the tester needs to test. In other words, by writing the fit criteria for the requirements, the analyst is ensuring that the tester has precise input and does not need to make his or her own interpretation of what the requirement means. If a tester designs a test for a PUC, then he or she designs the test case to test the fit criteria for all of the atomic requirements related to that PUC. If the tester decides to test an alternative grouping of requirements, then once again he or she designs the test case to test each of the atomic requirements in that grouping. This traceability between test cases and the requirements provides the project manager with status statistics about the number of PUCs and atomic requirements that have been successfully/unsuccessfully tested.

The implementation unit added in Figure 8 represents a unit of code and belongs to the developers. It might be a system use case, a module, an object, a class — it depends on how the developers are working. From the requirements point of view, the important thing is the implementing relationship between PUCs and implementation units. In other words, however the developers are working, it must be possible to trace each PUC to one or more implementation units. If you are not working with PUCs, then you would have a relationship between implementation unit and atomic requirement. The point is that if there is a change in an implementation unit, then you need to be able to see which requirements are affected and vice versa.

The class of knowledge called system architecture component naturally belongs to the systems architect. The design guiding relationship with the product scope is there to remind the requirements analyst that it makes sense for the systems architect to have early involvement in decisions about the scope of the product. It is true that you need to understand the requirements before you can come up with a solution. But an understanding of the higher-level requirements, represented by business events, is often enough to be able to communicate the business problem to the architect and to get some guidance about what is possible within the constraints.

If you do this early, you will save time by identifying which parts of the business requirements need to be defined at a lower level of detail and which parts you do not need to take any further.

### *Your Knowledge Model*

The knowledge model discussed in this section represents a scheme for keeping track of requirements knowledge. Think of this as a filing scheme that, at the start of your project, is made up of a number of linked empty folders. The amount of requirements information that you put into each of the folders depends on the characteristics of your project. For example, if you are working closely with a colocated group, then you might decide to represent your PUCs with scenario cards that you pin on the wall of your office. Those, together with defined naming conventions, might be enough for your purposes. A large, distributed team will need to formalize more of the detailed requirements because the risk of being misunderstood is much higher.

You can record your knowledge in any mixture of models, text, simulations, and prototypes that your team has agreed on. The form does not matter. What does matter is that you can identify the different classes of knowledge and trace the relationships between them.

Keep in mind that the knowledge model we have been discussing is a generic filing system. We suggest that you use it as a starting point and then make changes and additions progressively to fit it into your own environment. The sorts of changes that might be made include changing the name of classes and/or relationships to match the company's own terminology, adding new classes and relationships, and adding new attributes to classes to capture other facts specific to the organization.

The knowledge model provides a consistent way for the whole team to talk about requirements knowledge. This means that you greatly improve your potential for agility and iterative development. You know what you know and what you don't know, and you can make choices about where to get the most benefit for your efforts.

The Volere Requirements Template [3] provides detailed guidance on all the classes of knowledge mentioned in the knowledge model. The first version of the template was released in 1995. The template has been applied on a wide variety of projects in commercial, scientific, and engineering domains and is periodically updated to reflect new insights. At the time of writing this report, the template is Version 11.

## EARLY ESTIMATES

A common problem faced by project managers is being asked to make an estimate of how long a project will take before knowing how much effort is involved. People are often expected — on the first day of a project — to state precisely and irrevocably how long the project will take. If the only thing you know is the name of the project, then it is impossible to make a meaningful estimate. Before you can know how long it will take to study a piece of work and derive a product to improve that work, you need to know the size of the piece of work. Obviously, the larger and more complex the scope of the work, the more time it will take. But in order to know its size, we need to be able to quantify the amount and complexity of the work. Earlier in the report, we talked about how the requirements knowledge model organizes requirements into traceable levels of detail. Now let's look at how we can use those high-level requirements as input to doing a meaningful estimate early in the project.

Function point counting is a technique that was originally developed to measure the functionality of software. In this section, you will see a brief primer[3] on how the same technique can be used to measure and estimate the size of the requirements activity. Carol Dekkers of Quality Plus Technologies is a specialist in function point counting and has helped us to apply the technique to requirements. Dekkers points out that this is very similar to what an architect does when you want to know how much it will cost to build your house. The first rough "high-level" plan provides the architect with something to count. Suppose the architect tells you that the area of the building will be 3,000 square feet and that this style of building costs $250 per square foot. This provides you with a reasonable idea of the estimated cost of your house. You can use this technique to estimate the size of the work scope that you intend to study; instead of measuring the square feet that the building will occupy, you measure the number of function points within the work scope that you intend to study.

If you are not familiar with function points, it might be helpful to review some function point counts of established software systems. Capers Jones of Software Productivity Research counted the function points of systems in the following categories, after the systems had been built:

- Airline reservation: 25,000
- Insurance claims: 15,000
- Telephone billing: 11,000
- Visual Basic: 3,000
- Word 7.0: 2,500
- Aircraft radar: 3,000
- Unix v5: 50,000

Suppose you could estimate the number of function points delivered in your last project. Then, given the total cost of your project, you could come up with the average cost for specifying and installing a single function point. Whilst you cannot rely on this cost per function point being identical for all of your projects, at least it provides you with a starting point for your own organization. Now suppose that, instead of waiting until the end of the project to count your function points, you have a way of estimating the number of function points by using

[3]For more on how to use early requirements knowledge to count function points refer to Suzanne Robertson and James Robertson's book *Requirements-Led Project Management: Discovering David's Slingshot* [2].

early requirements deliverables. This provides you with a tangible input to your estimating process.

This section uses the library loan example introduced above to illustrate how you can do a function point estimate early in the project.

***Stored Data***

In Figure 4, you have a work context diagram that defines the scope of the work that we intend to study in order to understand the work of managing library loans. The context diagram shows the input and output data that defines the boundary of the study. As with any piece of work, if you look inside, you will discover that it contains stored data. This stored data — databases, files, archives — that the work references and maintains all adds to the work's functionality. If you study this stored data, you will learn enough to be able to do a function point count.

It does not matter how the data is stored now or how it will be stored in the future. To do your function point count, you need to do a rough business data model. If you do not know how to do this, then co-opt a data modeler to help you. In Figure 9, you see a data model for the work of managing library loans; these are the classes of data you would expect to find.

The "borrower" has "loans," and each one is for a "book." The books are supplied by the "publisher," and each book has an "author" and a "payment" that needs to be made to the publisher. Inside each of the classes, you will find attributes that belong to that class. For example, the class "book" would have attributes like:

- Book title
- Book ISBN number
- Book category
- Book published date
- And more

The stored data contributes to the functionality of the work you are studying, and you can use it in conjunction with the BUCs to come up with the function point count.

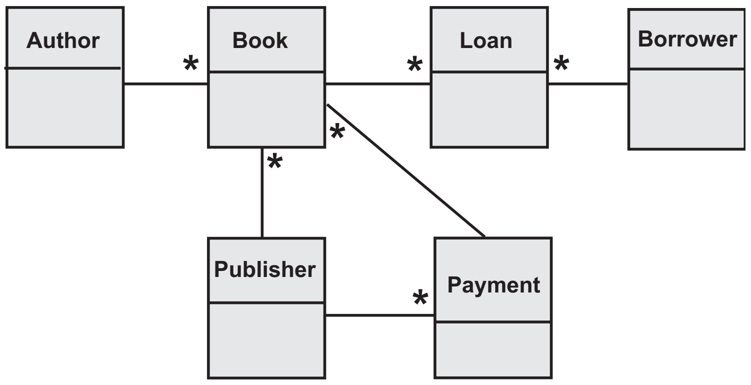

Figure 9 — This data model shows the classes of business data that need to be stored within the work. The asterisk (*) indicates cardinality, for example, an author has potentially many (*) books.

***Counting the BUCs***

Earlier in this report, we discussed how you can use business events to partition the work into functionally related pieces. Each of these pieces is a business use case. Referring back to Table 1, you see that each event has a BUC that contains some processing and some data stored and/or retrieved by its processing.

Looking through Table 1, you see that there are two different types of events: those that originate outside the work (1. Borrower chooses book) and those that are triggered by time (5. Time to pay for new books). This difference is important to function point counting as this process counts the resulting BUCs slightly differently depending on whether they are an input, an output, or an inquiry.

**Counting Output BUCs**

A BUC whose primary purpose is to deliver output is referred to in function point terms as an output — or, to be absolutely precise, an external output. An example of this is the BUC for Event 2 (borrower wants to extend loan). The main purpose of this BUC is to give the borrower a response to his or her request to extend the loan for the book. Along with one or more significant output flows, this sort of BUC will also contain processing that makes some calculations and/or decisions, updates stored data, or both. As this is classified as an output BUC, we start by counting the attributes of the two output flows — loan extension approval and refused loan extension — as follows:

- **Loan extension approval**
  - — Book title
  - — Loan extension date
  - — Loan extension conditions
- **Refused loan extension**
  - — Book title
  - — Extension refusal reason

The analysis of these two output flows gives a total of five data elements. Just remember this for a moment: at this stage, it is not vital for you to know what the data elements are; the important issue is to know how many there are. This is all happening very early in the project, and you might not know enough details to be precise about the number of elements; in that case, you can make an informed guess. Later you will see why that is acceptable.

To continue with the count for this BUC, we now need to look at the stored data. Look back at the data model in Figure 9 and ask which of the business classes are referenced by this BUC. The BUC needs to reference the borrower so that it can find all his or her loans; also, it needs to reference the book for each of the loans. So this means it needs to reference three classes of data: borrower, loan, book.

These counts are converted to function points by referencing the box in Table 3. The five data elements of the output flows put us in the first column, and the three data classes referenced means we are in the middle row. The intersection gives us a total of four function points.

You can see now why it is permissible to guess a little about the number of data elements in the data flow. Table 3 shows ranges for the number of data elements. Providing you can guess within these ranges, it is not necessary at this stage to do the detailed analysis required to find the precise number of elements.

Add the function point count for this BUC to the accumulated function point count for the work, and then count the next BUC. Continue with this process until you have counted all the BUCs. Events 1 and 3 (in Table 1) are also external output events, so you would count them using the table in Table 3. But Event 4 is classified as an external input and will be counted slightly differently. We also have to discuss how to count Event 5, which is a temporal event.

### Counting Input BUCs

Business Event 4 is “publisher delivers new book.” This qualifies as an external input because the primary intention of its BUC is to update some internally stored data. Output flows from this kind of BUC, if there are any, are trivial. In this example, the new book delivery causes changes to some of the classes of data within the work. We start by estimating the

**Table 3 — Counting External Output Events**

| Data Classes Referenced | Data Elements: 1-5 | Data Elements: 6-19 | Data Elements: 20+ |
|---|---|---|---|
| <2 | 4 | 4 | 5 |
| 2-3 | 4 | 5 | 7 |
| >3 | 5 | 5 | 7 |

This table shows function point counts for BUCs where the primary intention is to provide an output. The correct terminology for these is external outputs.

number of data elements in the input flow. Suppose we say that the elements in new book delivery are something like: publisher name, publisher address, book title, book ISBN, author name, book category, book published date, and payment amount due. This gives a total of eight data elements.

The next thing to do is to count the number of classes of stored data that are referenced by this BUC. Looking back at the data model for some help, it looks as if this BUC is concerned with creating and/or updating instances of the data classes of author, book, publisher, and payment. So that gives us four classes.

All that remains is to reference the table and find the intersection between our eight data elements and our four classes. The result is that the function point count for BUC Event 4 is six, as seen in Table 4.

These function points are added to the aggregation, and you continue to count the function points for the rest of the BUCs.

**Counting Temporal or Time-Triggered BUCs**

In any piece of work that you study, there will be events that are triggered by time. It might be that it is time to report on weekly sales, time to renew a subscription, or time to pay a bill. There is one of these temporal business events in the work of managing library loans: Event 5 (time to pay for new books).

The name that function point counters give to this type of BUC is inquiry. This is because the work of the BUC is to inquire about some of the stored data within the work. Note that if the processing is more than just the retrieval of stored data — suppose it is also concerned with making nontrivial calculations and updating the data — then the BUC should be treated as an output BUC to reflect the greater complexity.

Let's suppose that the work of BUC Event 5 is to review the payments due at the end of the month and to produce a new book payment that is sent to the publisher. The question now is how many data elements there are in the flow that goes to the publisher. It makes sense to include the publisher name, publisher address, book title, book receipt date, payment amount, and payment date; this gives us an estimate of six data elements.

The next step is to look at the data model to identify how many of the business classes are referenced by this BUC. The BUC certainly involves book, publisher, and payment, so we have three classes. Table 5 gives us the function point counts for inquiries, and the intersection of six data elements and three classes of data results in four function points. As before, you add the function points for this BUC to the accumulated total.

***Counting the Stored Data***

In addition to counting the function points for each BUC, you also need to consider the stored data. The data needs to be maintained, and this requires an amount of functionality that you can estimate by measuring the amount and

**Table 4 — Counting External Input Events**

| Data Classes Referenced | Data Elements: 1-4 | 5-15 | 16+ |
|---|---|---|---|
| <2 | 3 | 3 | 4 |
| 2 | 3 | 4 | 6 |
| >2 | 4 | 6 | 6 |

This shows the function point counts for an input BUC. Note that the input must come from one of the adjacent systems on the work context diagram. The correct function point counting terminology for these is external inputs.

complexity of the data. You can use the data model as input to doing this count. This time, for each class of data, estimate the number of data elements that it contains. Take the data class "book" as an example. Earlier we came up with book title, book ISBN number, book category, and book published date — a total of four data elements — that belong to that class of data. Table 6 tells us how many function points that gives us.

However, there's something a little bit different this time. We have estimated that the data class contains four data elements, so that leads us to the first column (1-19) in Table 6. The next count is record elements. This refers to subclasses of the data. For example, suppose the class "book" has two subclasses: fiction book and nonfiction book. In function point language, we would say that book has two record elements, which along with the four data elements gives a function point count of seven for the data class book.

So you add five function points to your accumulated count and then count the remaining data classes.

If you have any externally stored data — data that is used by your work but maintained by another system — then you also count that external stored data and look up the function points using Table 7. The library loan management example does not have any external stored data.

**Table 5 — Counting Temporal Events**

| | Data Elements | | |
|---|---|---|---|
| Data Classes Referenced | *1-5* | *6-19* | *20+* |
| *1* | **3** | **3** | **4** |
| *2-3* | **3** | **4** | **6** |
| *>3* | **4** | **6** | **6** |

This table shows the function point counts for inquiries.

**Table 6 — Counting Internal Stored Data**

| | Data Elements or Attributes | | |
|---|---|---|---|
| Record Elements | *1-19* | *20-50* | *51+* |
| *<2* | **7** | **7** | **10** |
| *2-5* | **7** | **10** | **15** |
| *>5* | **10** | **15** | **15** |

These are the function point counts for internal stored data, or internal logical files as they are called in function point terminology.

**Table 7 — Counting External Stored Data**

| | Data Elements or Attributes | | |
|---|---|---|---|
| Record Elements | *1-19* | *20-50* | *51+* |
| *<2* | **5** | **5** | **7** |
| *2-5* | **5** | **7** | **10** |
| *>5* | **7** | **10** | **10** |

These are the function point counts for data that is used by the work but stored by another system. The function point terminology is external interface files.

***Using the Counts***

If you did a function point count for all of the BUCs and all of the stored data in the library loans work, you would have a result that looks like Table 8.

You have used your high-level requirements knowledge, and you have counted 67 function points within the scope of the work. Now you need to convert the function point count into effort needed to gather the requirements by multiplying the number of function points by the time or effort needed by your organization to complete the analysis and requirements for one function point. If you do not know this number for your organization, then either derive it from previous projects or derive it by doing a quick simulation to see how much effort is necessary to discover the requirements for one of the BUCs in your project.

If you are outsourcing your development, then the traceability between your levels of requirements together with your function point count will help you to communicate with suppliers. And, most importantly, you have the ability to manage change. If your work context produces a function point count of 55, then adding extra interfaces or making changes will alter your function point count. This provides you with an objective way of dealing with changes.

This discussion on how to count function points early in the project has illustrated how high-level requirements knowledge — assuming you have a consistent way of communicating it — provides you with a valuable project management tool. The next section discusses how you can take advantage of a consistent way of talking about requirements to decide how to profitably spend your effort.

**Table 8 — Total Function Point Count**

| | Function Points |
|---|---|
| BUC1 | 7 |
| BUC2 | 4 |
| BUC3 | 4 |
| BUC4 | 6 |
| BUC5 | 4 |
| Data Class Author | 7 |
| Data Class Book | 7 |
| Data Class Borrower | 7 |
| Data Class Loan | 7 |
| Data Class Payment | 7 |
| Data Class Publisher | 7 |
| **TOTAL ESTIMATED** | 67 |

## POTENTIAL FOR AGILITY

An agile requirements strategy is one where there is *no wasted effort*. All the effort you spend on requirements (meeting, interviewing, modeling, reviewing, prototyping, documenting, testing — everything) brings you closer to being able to meet your project's goals. Think back to the requirements knowledge model discussed in the first part of this report. Remember that the model represents a filing system for keeping track of the classes of knowledge you need to understand and the connections between them. The projects with the greatest potential for agility would be ones that could incrementally discover and share all that knowledge, and make changes to it, without having to formalize how it is discovered and how it is represented.

But not all projects have the same potential for agility. Large numbers of stakeholders, scattered development teams, varying levels of experience, and other factors that make it difficult to get answers and make decisions all influence your potential for agility. To help make your requirements strategy as agile as it can possibly be it is useful to consider the agility potential for your project.

***Rabbit Projects***

Rabbit projects, like their namesake, are small and fast. If your project has the characteristics of a rabbit, then you have the highest potential for agility. Rabbit projects typically occur where close stakeholder participation is possible. The developers and the domain experts are either physically located in the same place or have developed a way of working where distance does not impede the ability to share ideas and make decisions. Rabbit projects are iterative. They gather requirements in small units (typically one BUC at a time) and then implement the solution piece by piece, using the implementation to get feedback from the stakeholders. Rabbit projects are not focused on a process that delivers a requirements specification; instead, they have a process that discovers and communicates requirements one logical chunk at a time. Rabbit projects benefit from having a sketch of a requirements knowledge model on their whiteboard so that stakeholders have some consistent way of talking to each other. However, these projects will not produce formal deliverables for each one of the classes of knowledge. If these teams sketch a work context and come up with a list of business events, then they might choose to do BUC scenarios for the most complex ones and keep a list of naming conventions so that everyone can see them.

Rabbit projects benefit from paying attention to all the classes of requirements knowledge, but the amount of time and effort that the teams spend in representing the knowledge is minimized because they share their understanding by talking to each other.

### *Horse Projects*

Horse projects have less potential for agility. They are larger than rabbit projects and hence more constrained by the size of the project and the organization. There is more need to have a formal process for representing classes of knowledge. Horse projects are the most common corporate projects. There is a need to formalize the documentation for some of the classes of requirements knowledge because it is likely that requirements must be handed from one department to another. Another factor is that these projects usually involve more than a few stakeholders, often in a number of locations.

Horse projects are working from the same knowledge model as rabbit projects, but they need a more formal process for how each of the classes of knowledge is discovered, who is responsible for it, and how it must be represented both in terms of notation and documents. This extra bureaucracy is necessary in order to exploit the potential for agility by making communication of understanding easier. But there is a trap here and that is that horse projects often start to work by rote and stop questioning whether a particular activity or deliverable is necessary in all cases. To keep a horse galloping, you need to keep questioning whether everything in the saddlebags is still necessary.

### *Elephant Projects*

Elephant projects have the least potential for agility. These projects have a long duration and hence like elephants need a long memory. Sometimes they are so long that none of the people involved at the start of the project are still there at the end. They involve many stakeholders in many locations at many levels of authority and interest. Their technical infrastructure is diverse, and there are many developers involved. These projects often outsource part of their development, often to another country. Owing to this huge diversity, the elephant project has a need for a very formal and consistent representation of the requirements knowledge — one that is not open to interpretation. That representation is normally in the form of a requirements specification document.

When elephant projects decide how to represent their requirements knowledge, the notation and format for how each class and relationship will be represented is often mandated by organizational or industry standards.

The truth is that even in the most extreme of elephant projects there is still potential for agility. You can exploit this potential if you have a consistent way of partitioning the elephant and keeping track of the connections between the pieces.

Within the large project, you can discover a number of linked smaller projects, and some of these pieces — especially the ones with colocated stakeholders — can be more agile than others.

## CONCLUSION

This report discusses how managers can use consistent and understandable requirements knowledge as input to making decisions and steering a project down its most agile path. Here's a short checklist for putting these ideas into practice:

- Agree with your team on the requirements knowledge you intend to manage and how you are going to talk about it. The first section of this report gives an example of how to build a requirements knowledge model. You can use this example as a starting point, or you can build your own. It does not matter which you do provided it is consistent for your project. Keep your model visible, and change anything that does not work for you.
- Use the early requirements knowledge as input to making estimates that are traceable to the requirements knowledge. This report provides a primer on using early requirements knowledge to count function points.
- Use your requirements knowledge as the vehicle for identifying releases and iterations and for allocating tasks. For example, "We've agreed that BUCs 3 and 10 are the highest value so we'll concentrate on delivering them in the first iteration."
- Use your requirements knowledge to monitor progress. For example, "How many of the PUCs in release 1 have all their requirements defined?"
- Have your team analyze the additional/changed requirements knowledge for each change and then adjust the function point count and the estimate to reflect the change.

## REFERENCES

1. Robertson, Suzanne, and James Robertson. *Mastering the Requirements Process, Second Edition*. Addison-Wesley Professional, 2006.

2. Robertson, Suzanne, and James Robertson. *Requirements-Led Project Management: Discovering David's Slingshot*. Addison-Wesley Professional, 2005.

3. Volere. "Volere Requirements Specification Template, Edition 11" (www.volere.co.uk/template.htm).

## RECOMMENDED READING

Alexander, Ian, Neil Maiden, et al. *Scenarios, Stories, Use Cases Through the Systems Development Life-Cycle*. John Wiley, 2004.

Davis, Alan M. *Just Enough Requirements Management*. Dorset House, 2005.

DeMarco, Tom, and Timothy Lister. *Waltzing with Bears: Managing Risks on Software Projects*. Dorset House, 2003.

Garmus, David, and David Herron. *Function Point Analysis: Measurement Practices for Successful Software Projects*. Addison-Wesley Professional, 2001.

Gottesdiener, Ellen. *Requirements by Collaboration: Workshops for Defining Needs*. Addison-Wesley Professional, 2002.

Lauesen, Soren. *Software Requirements: Styles and Techniques*. Addison-Wesley Professional, 2002.

Weinberg, Gerald M. *Quality Software Management, Volume 2: First-Order Measurement*. Dorset House, 1993.

Wiegers, Karl E. *Software Requirements*. Microsoft Press, 2003.

## ABOUT THE AUTHOR

Suzanne Robertson is a Senior Consultant with Cutter Consortium's IT Management and Agile Product & Project Management practices, a contributor to the Agile Product & Project Management advisory service, and is a principal and founder of the Atlantic Systems Guild. She is coauthor of *Requirements-Led Project Management*, a book that provides project managers with guidance on how to use the work done by requirements analysts as input to steering projects. The book defines the relationships between effective requirements practices and project success. This management book connects to *Mastering the Requirements Process*, a guide for practitioners on finding requirements and writing them so that all stakeholders can understand them.

Current work includes research and consulting on the management, sociological, and technological aspects of requirements. The product of this research is Volere, a complete requirements process and template for assessing requirements quality and for specifying requirements.

Ms. Robertson is the author of many papers on systems engineering. She also speaks at numerous conferences and universities. She is a member of IEEE and on the board of the British Computer Society's Requirements Groups. She was the founding editor of the requirements column in *IEEE Software*. Other interests include a passion for the opera, cooking, skiing, and finding out about curious things. She can be reached at srobertson@cutter.com.

# Agile Product & Project Management Practice

Cutter Consortium's Agile Product & Project Management practice provides you with information and guidance — from experienced, hands-on Senior Consultants — to help you transition (or make the decision to transition) to agile methods. Led by Practice Director Jim Highsmith, Cutter's team of experts focuses on agile principles and traits — delivering customer value, embracing change, reflection, adaptation, etc. — to help you shorten your product development schedules and increase the quality of your resultant products. Cutting-edge ideas on collaboration, governance, and measurement/metrics are united with agile practices, such as iterative development, test-first design, project chartering, team collocation, onsite customers, sustainable work schedules, and others, to help your organization innovate and ultimately deliver high return on investment.

Through the subscription-based publications and the consulting, mentoring, and training the Agile Product & Project Management Practice offers, clients get insight into Agile methodologies, including Adaptive Software Development, Extreme Programming, Dynamic Systems Development Method, and Lean Development; the peopleware issues of managing high-profile projects; advice on how to elicit adequate requirements and managing changing requirements; productivity benchmarking; the conflict that inevitably arises within high-visibility initiatives; issues associated with globally disbursed software teams; and more.

**Products and Services Available from the Agile Product & Project Management Practice**

- The Agile Product & Project Management Advisory Service
- Consulting
- Inhouse Workshops
- Mentoring
- Research Reports

**Other Cutter Consortium Practices**

Cutter Consortium aligns its products and services into the nine practice areas below. Each of these practices includes a subscription-based periodical service, plus consulting and training services.

- Agile Product & Project Management
- Business Intelligence
- Business-IT Strategies
- Business Technology Trends & Impacts
- Enterprise Architecture
- Innovation & Enterprise Agility
- IT Management
- Measurement & Benchmarking Strategies
- Enterprise Risk Management & Governance
- Sourcing & Vendor Relationships

## Senior Consultant Team

The Cutter Consortium Agile Product & Project Management Senior Consultant Team includes many of the trailblazers in the project management/peopleware field, from those who've written the textbooks that continue to crystallize the issues of hiring, retaining, and motivating software professionals, to those who've developed today's hottest Agile methodologies. You'll get sound advice and cutting-edge tips, as well as case studies and data analysis from best-in-class experts. This brain trust includes:

- Jim Highsmith, Director
- Scott W. Ambler
- Christopher M. Avery
- Paul G. Bassett
- Sam Bayer
- Kent Beck
- E.M. Bennatan
- Tom Bragg
- David R. Caruso
- Robert N. Charette
- Alistair Cockburn
- Jens Coldewey
- Ken Collier
- Ward Cunningham
- Rachel Davies
- Doug DeCarlo
- Tom DeMarco
- Lance Dublin
- Khaled El Emam
- Kerry F. Gentry
- Sid Henkin
- David Hussman
- Ron Jeffries
- Joshua Kerievsky
- Bartosz Kiepuszewski
- Brian Lawrence
- Tim Lister
- Michael C. Mah
- Lynne Nix
- Siobhán O'Mahony
- Ken Orr
- Patricia Patrick
- Mary Poppendieck
- Roger Pressman
- James Robertson
- Suzanne Robertson
- Alexandre Rodrigues
- Johanna Rothman
- David Spann
- Rob Thomsett
- John Tibbetts
- Colin Tully
- Jim Watson
- Robert K. Wysocki
- Richard Zultner