Traceability Strategies for Managing Requirements with Use Cases

by Ian Spence, Rational U.K. and Leslee Probasco, Rational Canada,

Rational Software White Paper

Rational®
the e-development company™

Table of Contents

Abstract

In many commercial applications of use case modeling techniques, the use case model must be combined with more traditional requirements capture techniques to provide a requirements management process acceptable to all of the stakeholders involved in the project. This paper explores the traceability strategies available to organizations adopting use case modeling techniques as part of their requirements management strategy.

Introduction and Background

Traceability Items

A common point of confusion when discussing Requirements Management, especially when using a tool such as RequisitePro, is the overloading of the term "requirement". In addition to items commonly defined as "requirements", we need to capture and track the attributes of, and traceability between, many other kinds of item. These other traceability items include issues, assumptions, requests, glossary terms, Actors, Tests, etc.

Capturing and tracking these other kinds of traceability item helps us in effectively managing our project's requirements.

Definition: Traceability Item

> **Any textual, or model item, which needs to be explicitly traced from another textual, or model item, in order to keep track of the dependencies between them.**

With respect to RequisitePro this definition can be rephrased as:

> **Any textual or model item represented within RequisitePro by an instance of a RequisitePro requirement type.**

RequisitePro itself provides an excellent tool for defining, capturing and tracking the values, attributes and traceability links between the many kinds of traceability item involved in software development.

Implicit and Explicit Traceability

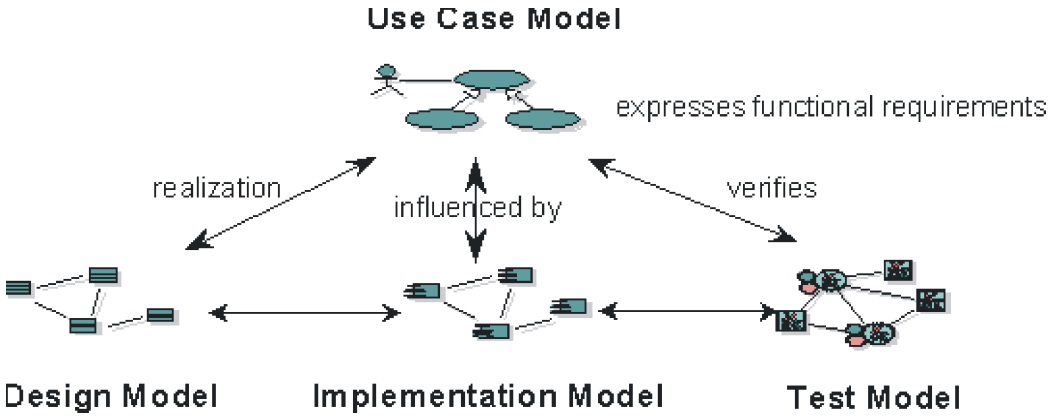
There is a certain amount of traceability implicit in any development process. This is usually supplied by the formal relationships between the artifacts in the process.

Examples of this sort of implicit traceability are:

- Naming conventions
  that is, the class in the design model called Fred is implemented by the class in the implementation model called Fred
- The construction of mappings between the models
  that is, the component view in Rose allows the packages and classes in the logical view in Rose to be explicitly mapped to packages in the implementation model. This mapping can contain re-naming between the models and the application of different packaging strategies
- Relationships between the model items themselves
  that is, in the Rational Unified Process the use case realizations in the design model are traced back to the use-cases they realize.
- The creation of different perspectives illustrating how the elements of one model satisfy the demands implicit in the elements of another model
  that is, the use case realizations in the design model demonstrate how the model elements of the design model collaborate to fulfill a use case. These provide a use case perspective onto the design model, which validates and supplements the static packaging of the classes and packages in the design model.

All of these examples provide a level of traceability and allow impact analysis to be undertaken using the information held in the development models.

As shown in the figure below, use case driven development involves a series of inter-related models.



The figure shows the models and the implicit relationships between these models. The relationship between the models provides a level of traceability that is implicit to the development process.

When undertaking a use case driven development some supporting artifacts are required to support the use-case model and enable the definition of a complete software requirements specification. In the Rational Unified Process these are the Supplementary Specifications and the Glossary. Also of interest are the Business Case and Vision documents, which will contain the definitions of the needs, goals and features for the project.

The relationships between the models do not involve these supporting artifacts and so they are not covered by the implicit traceability built into the development process.

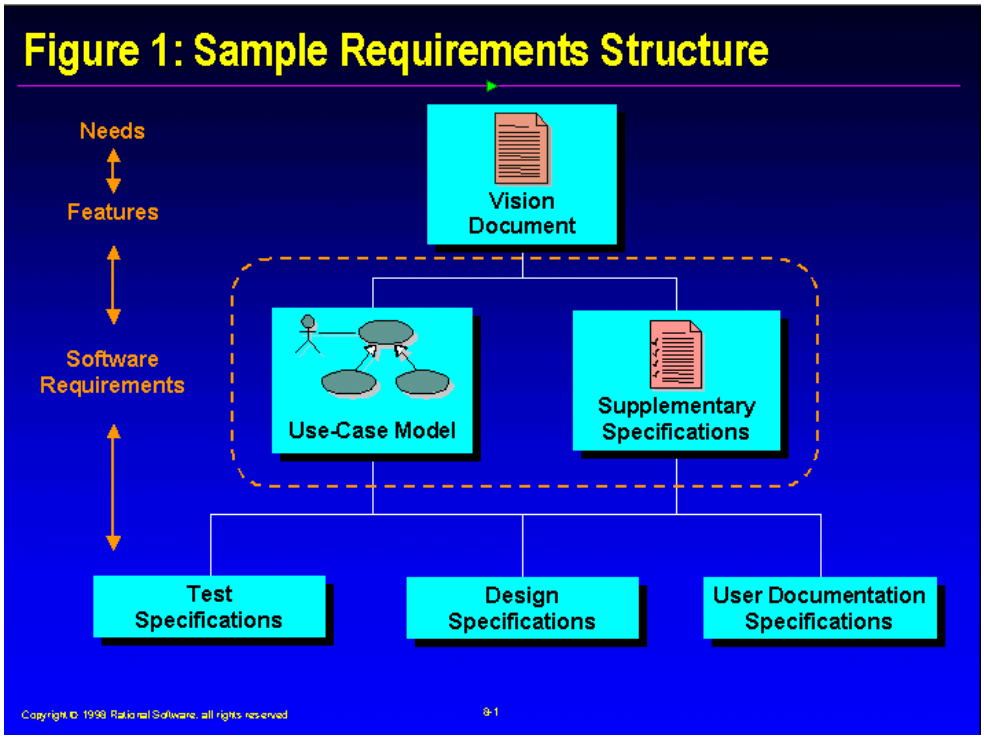
These implicit relationships are fundamental to the development process and benefit from being built as a natural part of the developer's work. These relationships are central to the modeling process and are constructed, and maintained, as the models are matured.

The implicit traceability is limited to the relationships available in our modeling notation.

So why would we want any additional explicit traceability?

Well if we are to adopt the principles of requirements traceability we would need to integrate the Requirements, model items and other traceability items into a traceability hierarchy. We may also want to add additional traceability relationships into our development process.

The following figure shows an example traceability hierarchy showing the relationship between "features" defined in a vision document and "software requirements" defined in a use case model and a supplementary specification. It also shows how the software requirements are traced into the Test Requirements, Design and User Documentation



If we look at the relationship between the supplementary requirements (defined in the Supplementary Specifications document) and the design, and implementation, models we will see that this is not covered by the implicit traceability between the models.

This is a good example of an additional level of explicit traceability that is often required on a project. Many requirements trace through the whole series of models due to their implicit relationship with the use-case model. The supplementary requirements, which are captured alongside the use-case model in the supplementary specification, are not directly related to any of the packages in the design model that need to consider them or to any of the components in the implementation model that need to fulfill them.

Other examples include the relationships between:

- the features of the system and the use case model
- the use case model and the user documentation
- the use case model and the test requirements.

One of the major decisions we need to make when setting up our requirements traceability process is the level of traceability we require and how much explicit traceability is required to meet this goal. We would like our approach to requirements, and traceability, management to facilitate the development process, not complicate and restrict it.

As we can see the addition of explicit traceability to our development artifacts could have a significant cost to our project. This is especially significant when we consider the long-term cost of populating and maintaining this additional information. What we need to do is ensure that we establish a suitable level of traceability for our project and that we will get a return on investment on any additional explicit traceability we decide to maintain. We want our developers to spend their time developing not tracing. To this end we need to establish and evaluate our traceability strategy before we add the cost of explicit traceability to our project.

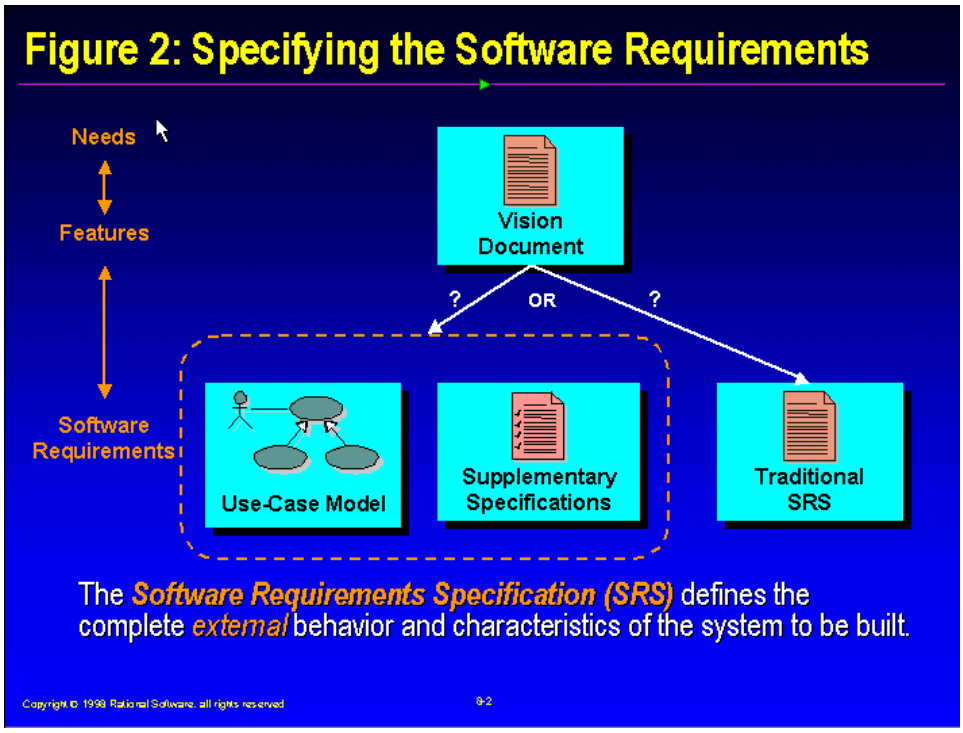
The traceability strategy will define the level of explicit traceability we wish to add to our software development process.

Managing the Supporting Artifacts

Figure 1 above shows the artifacts involved in requirements specification in the RUP.

The thing to note here is that the Use Case Model and the Supplementary Specification form our complete Software Requirements Specification (SRS). This means that there is no need for us to have a formal Software Requirements Specification document as required by traditional requirements management techniques.

Figure 2 below shows how a traditional SRS document is often related to the RUP artifacts. The traditional SRS is just an alternative way of documenting the Software Requirements. It is important to realize that both approaches can provide us with a Software Requirements Specification that defines the complete external behavior of the system to be built.



This relationship is often misinterpreted as implying that the two models of Requirements Management cannot co-exist. People often think that they must choose between traditional requirements management techniques, using a formal Software Requirement Specification document, and use case model based requirements management techniques using a use case model and a supplementary specification. In fact in certain circumstances it is necessary for the two forms of Software Requirements Specification to co-exist within the same project.

Possible Traceability Strategies

Is there more than one acceptable traceability strategy?

There are many traceability strategies that could be used to facilitate the requirements management process; even working within the framework of the Rational Unified Process many approaches are possible. Four of the most common, all of which the authors have seen in use, are:

- Use-Case Model Only.

  In this case the use-case model is the only statement of the systems requirements. Projects that choose this approach are characterized by close association and trust between customer and developer.

  The Use-Case Model and Glossary and Supplementary Specifications form the entire statement of the system's requirements – no additional definitions of Needs, Product Features or software requirements exist.

- Features Drive the Use Case Model

This is the default strategy recommended by the Rational Unified Process. The Use-Case Model and Supplementary Specifications form a complete software requirements specification. Features are documented in the Vision Document and are traced to use cases. If they are not reflected in the Use Case Model then they are traced to supplementary requirements in the Supplementary Specifications.

In this case the Use-Case Model acts as the main statement of the functional requirements. The Use Case Model and Supplementary Requirements are complemented with Needs and Product Features in addition to the Glossary and Supplementary Requirements.

- The Use-Case Model is an interpretation of the Software Requirements Specification

  In this case the Use Case Model is an interpretation of a formal, traditional Software Requirements Specification. This is most often used when a formal, traditional Software Requirement Specification is mandated due to regulatory, or internal, protocol and a use case model is required to enable the practice of use case driven development.

  The Features trace into a formal Software Requirement Specification document (as in traditional requirements management) but the software requirements are then traced, explicitly, into the Use-Case Model.

- The Use Case Model reconciles multiple sets of traditional software requirements

  The Use Case Model is the interpretation of a set of formal Software Requirement Specifications from multiple sources and provides the specification of a single common system.

  In this case each stakeholder has their own set of Product Features and Software Requirements, which are detailed within their own Vision and Software Requirements Specification documents. These multiple viewpoints, and possibly conflicting desires are then reconciled within a single Use-Case Model, which specifies what the system will do. This strategy can be very effective when dealing with a large set of independent stakeholders.

In all of the cases except option 1 we combine our use-case model with elements of the traditional requirements traceability process.

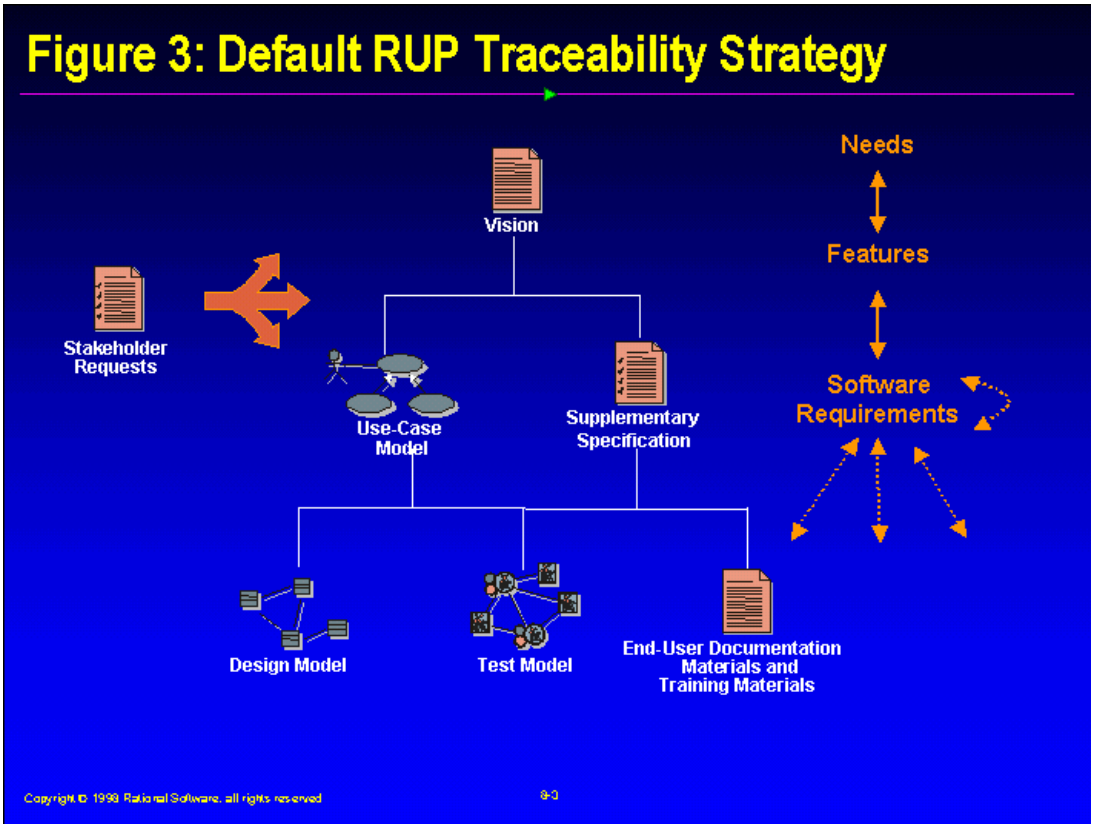
There are of course many other possible options one of which is to not have a Use-Case Model at all. We will call this option "No Use-Case Model".

These, and other approaches, are discussed in more detail in the Traceability Strategy Catalogue below.

Why Would We Want to Adopt One of the Hybrid Approaches?

As we can see the two extreme solutions (Use Case Model Only and No Use Case Model) above take a very purist view only allowing one form of requirement's capture. Both of them assume that a one-size-fits-all approach is applicable to all projects and all stakeholder communities. Both approaches have seen their fair share of success but have fallen into disrepute because of their inflexibility and inability to handle all of the complex situations, and sets of stakeholder relationships, that arise in "real world" projects.

The Rational Unified Process recommends the following traceability hierarchy:



This is the "Features Drive the Use Case Model" option above and is probably the most efficient traceability strategy but it should be noted that even when the Rational Unified Process has been adopted this approach is not always the most effective.

Examples where using Use-Case Modeling as the sole mechanism for functional software requirement specification can prove problematical include:

- Where there are many, contradictory requirement sources (i.e. many contradictory desires will need to be tracked and managed).
- Where the project is being undertaken within an organization that insists upon compliance with an existing traditional requirements capture process.
- Where there are problems in getting the stakeholders to attend modeling workshops to produce a single, consensus requirements model.
- Where Use-Case Modeling is to be used to enable Object Oriented software development within the constraints of an existing requirements capture process.
- Where the stakeholder community is unable, or unwilling, to express all of their functional software requirement level desires directly within the use-case model.
- Where the customer has defined the product to be delivered in the form of a set of traditional software requirements. This is a very common situation when a development is put out to tender – the traditional requirement statement then becomes part of the contract for delivery.

In our opinion the decision about which approach should be adopted must be made within the context of each project and development organization. There is not a one-size-fits-all solution to this problem and it is foolish to attempt to force all projects into a single approach to requirements management.

It should be remembered that the Rational Unified Process is a configurable process and can cope will all of the traceability strategies presented in this document except for the "No Use Case Model" approach (the use-case driven nature of the

Rational Unified Process precludes the adoption of this option). The decision about which approach to adopt is one of the decisions to be made during the production of the Rational Unified Process development case.

About the Traceability Strategy Catalogue

To be able to define our traceability strategies we need a mechanism to categories and define our traceability items:

Definition: Traceability Type

A categorization of traceability time (for example, need, product feature, use case, software requirement, test requirement, actor, glossary term, and so on) based on common characteristics and attributes.

Note: In RequisitePro, Traceability Types will be represented by Requirement Types.

Setting up a traceability strategy therefore involves three closely coupled activities:

- Identify the set of traceability types required to define our traceability items.
- Identify the valid traceability relationships between these traceability types.
- Identify the attributes required by the traceability items to enable effective requirements management for the project.

The Traceability Strategy Catalogue facilitates the first two of these steps by documenting known sets of traceability items and their traceability relationships. It does not cover the third activity as the definition of the appropriate attributes for the traceability types is, currently, outside the scope of this paper).

The traceability strategies described in the catalogue all make use of subsets of the same basic set of traceability types.

- Needs
- Product Features
- Software Requirements (both functional and non-functional)
- Glossary Items
- Use Cases
- Use Case Sections
- Actors

Note: Often when use case modeling the only software requirements will be the supplementary requirements defined by the supplementary specification.

Having the potential to trace between all of the traditional traceability types and the component parts of the use-case model opens up the number of traceability strategies available to the project.

There are two levels of traceability between our traceability items:

- Basic Traceability

  There is basic traceability that applies whichever traceability strategy is chosen. This traceability is implicit in the nature of the traceability types. This covers things like the relationship between Use Cases and Use Case Sections or between Use Cases and Actors.

  When reading the overview diagrams in the Traceability Strategies Section below this basic traceability is not repeated for each strategy but is included in the applicable traceability strategies by default.

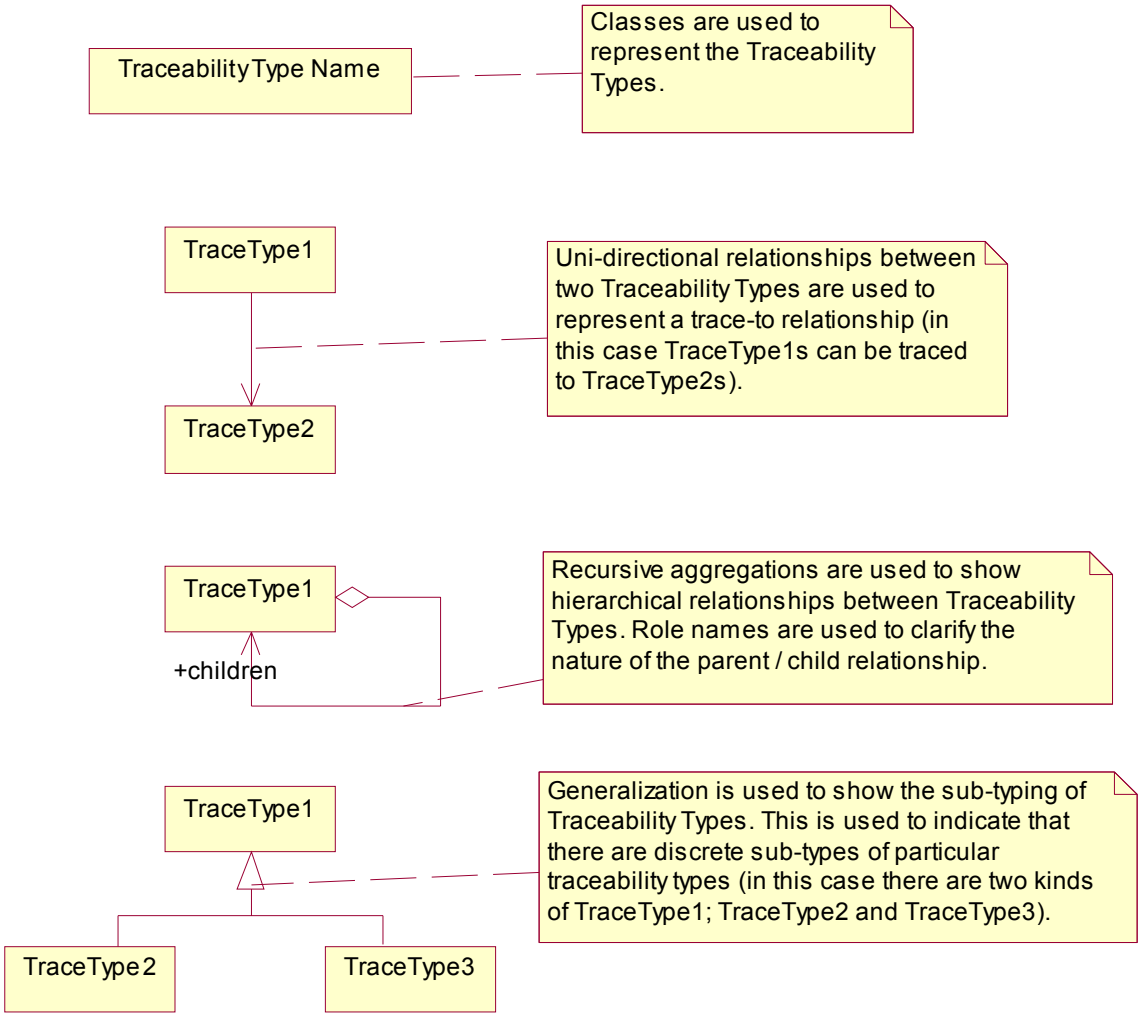
- Extended Traceability

  This is the traceability that is introduced to support one of the specific traceability strategies. This traceability is much more subjective and varies between the various traceability strategies.

Traceability Strategy Catalogue

Diagramming Notation

The Traceability Types and their traceability relationships are shown as Unified Modeling Language (UML) diagrams. The figure below shows how to interpret the usage of the UML in this context.



To fully understand the diagrams it is useful to know the implementation mapping used when implementing the definitions in RequisitePro. The table below explains how the diagramming notation can be mapped onto RequisitePro projects.

Diagramming Notation	RequisitePro Mapping
Class / Traceability Type	Requirement Type
Relationships	The RequisitePro "trace-to" relationships
Aggregations	Hierarchical Requirements
Generalization	Classification of the super Requirement Type by adding an additional "sub-classification" attribute.

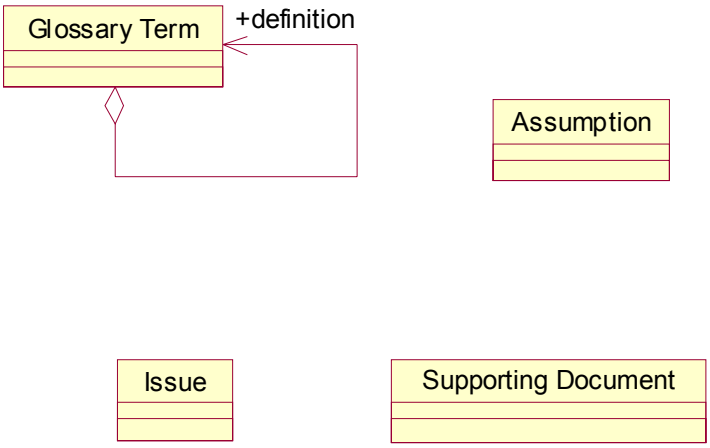
Note: RequisitePro allows any traceability item to be traced to any other item. What the traceability strategy defines is the meaningful traceability links that will be at the heart of the project's requirement management strategy.

Supporting Traceability Types

Description

In this section we define a set of supporting traceability types that can be used to support whichever traceability strategy is selected.

Overview



Traceability Types

Traceability Type	Description
Glossary Term	This Traceability Type defines the traceability items representing Glossary Terms and their definitions. This is included in the set of supporting traceability types, as a Glossary is required regardless of which traceability strategy you choose to adopt.
Issue	This Traceability Type allows you to add traceability items representing issues you want to track within RequisitePro. These issues can then be associated with whichever traceability items that they impact. An example of using the Issue traceability type would be to track issues associated with Glossary Items. If a definition is uncertain, or in dispute, issues could be raised and including in RequisitePro. This will ensure that the issue is not forgotten and allows a view to built reporting on all Glossary Items with outstanding issues. Another good use of this traceability type is to track issues raised when reviewing the use cases and other development artifacts.
Assumption	This Traceability Type allows you to track the assumptions that you have made. The assumptions can then be associated with whichever traceability items they affect.
Supporting Document	This Traceability Type allows you to add any documents that you like into the traceability hierarchy. This is particularly useful for including pre-existing examples or documentation that clarifies the meaning or

	purpose of another traceability item. The flexible traceability mechanisms of RequisitePro allow you to associate supporting documentation with any traceability item of any type. An example of using the Supporting Document type is to include the detailed EDI message specifications as supporting information for the Glossary, or as appendices to the use cases that will use the messages.

Basic Traceability

Traceability Link	Description
Glossary Term to Glossary Term	This relationship allows us to capture both the name of the Glossary Term and its definition using a single traceability type.
Supporting Traceability Type to any other Traceability Type	These supporting traceability types can be traced to any of the other traceability types involved in the chosen traceability strategy.

No Use Case Model

Description

In this case there is no use-case model. The Needs give rise to Product Features, which in turn give rise to Software Requirements documented in a formal Software Requirements Specification.

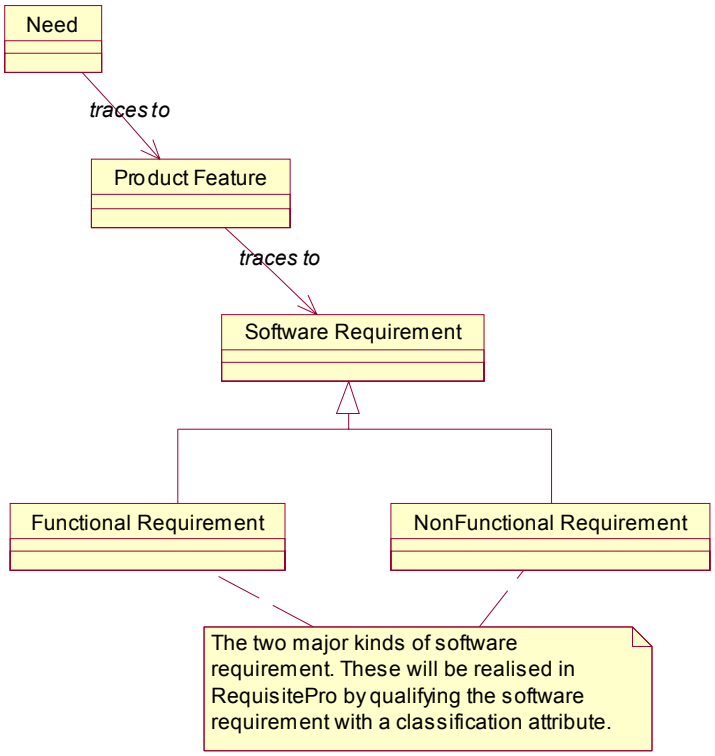
Typified by project managers who say, "I don't need no stinkin' Use-Case Model!"

Characteristics

Characteristic	Value	Comment
Explicit Traceability	High	Projects practicing requirements management techniques without the use of use cases typically maintain a high level of explicit traceability between the traceability types.
Trust	Low	
Accountability	High	
Formality	High	
Completeness	Low	It is very difficult to assess a set of traditional software requirements for completeness.
Document Set	Large	The document set is usually measured in feet not inches.
Focus	Contract	The focus of the requirements capture process is on establishing a legally enforceable contract between the customers and the developers rather than establishing a shared understanding of the problem to be solved and the proposed solution.
Understandability	Low	The requirements documentation is often inaccessible to both the user community and the developers. It usually consists of many individual line items grouped by type or functional area providing little context for reviewers.
Process	Typically Waterfall	Traditional requirements capture techniques are often practiced as part of a waterfall development process. The lack of context for, and the difficulty in assessing the completeness of, any subsets of the requirements do not facilitate the

		adoption of iterative and incremental development processes.
Development Style	Functional Decomposition	The grouping of requirements by type or functional area tends to lead to continued functional decomposition as the requirements are translated into a solution.

Traceability Overview



Traceability Types

Requirement Type	Description
Need	The business or operational problem (opportunity) that must be fulfilled in order to justify purchase or use. Also known as Goal or Objective.
Product Feature	A capability or characteristic of a system that directly fulfills a Need. Often thought of as the "advertised benefits" of the system.
Software Requirement	A condition or capability to which the software being built must conform.

Traceability Summary

Traceability Link	Description
Needs trace to Product Features	Each Need will be realized by a set of Features. This relationship allows the business benefit of each Feature to be tracked.
Product Features trace to Software Requirements.	Each Feature will be realized by a set of Software Requirements. This relationship allows the business benefit of each Software Requirement to be tracked and enables the scope management of the Software Requirements at the Product Feature level.

Benefits and Disadvantages

Pros:

- Well understood
- Is thought to be good for legal contracts (see the many on-going court cases related to delivered software's ability / inability to satisfy the requirements specified).
- Recommended by many standard processes.
- Enables detailed, low level, formal traceability.
- Does not upset the status quo by introducing "darned new-fangled" ideas

Cons:

- Hard to complete the requirements capture - it is very easy to get stuck in the requirements phase.
- Hard to understand requirements expressed in this form.
- Impact analysis of requirements change is hard to assess.
- Individual requirements have no context.
- High maintenance costs. The lack of any implicit traceability leaves projects with the cost of maintaining large amounts of explicit traceability relationships.
- The lack of context makes it difficult to identify meaningful sub-sets of the requirements. This in turn makes scope management, and the incremental delivery of the product, more difficult.

Examples

The no use case model approach to requirements traceability is used widely in many projects in many business areas. Many organizations require a formal, traditional Software Requirements Specification as the basis for formal contractual negotiation. This leads them to think that the traditional requirements management approach is the only approach appropriate for their projects.

Use Case Model Only

Description

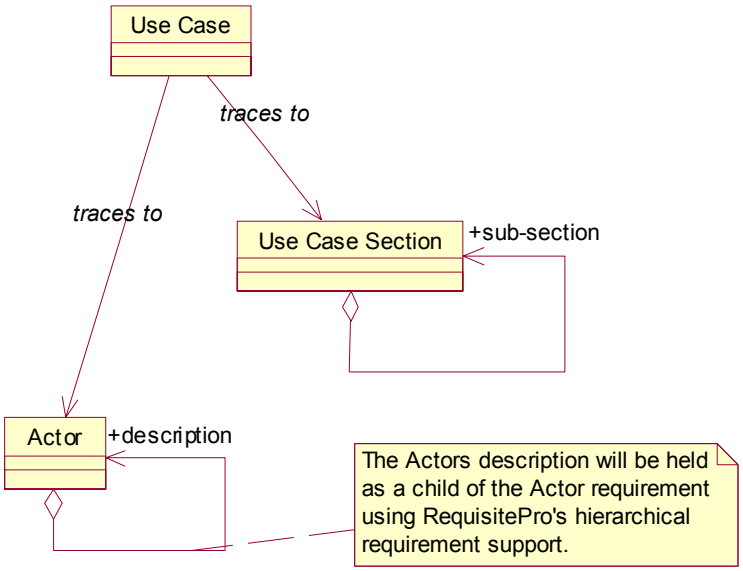
"The use-case model is my requirements." A close association, and high level of trust, between the customers and the developers usually characterize projects that adopt this approach. It is normally used for internal, low-accountability projects where the developers hope to demonstrate, or gain, a clear understanding of the requirements by developing the use-case model along with (or approved by) the customer(s).

In this case the use-case model is the only statement of the systems requirements. The Use-Case Model, Glossary and Supplementary Specifications form the entire statement of the system's requirements.

Characteristics

Characteristic	Value	Comment
Explicit Traceability	Low	No explicit traceability required. The implicit traceability that is provided by adopting a use case driven approach is considered sufficient. Probably no explicit traceability is maintained, so no Requirements Management tool is used.
Trust	High	The lack of any needs or feature level analysis means that the developers of the use case model are given a high level of trust by the stakeholders to deliver the right system.
Accountability	Low	
Formality	Low	
Completeness	Low	Although a use case model itself facilitates establishing the completeness of the software requirements specification the lack of any traceability back to the stakeholder needs often leads to the production of an incomplete, or over elaborate, system.
Document Set	Small	This approach involves the minimal document set.
Focus	User	The use case model has a user perspective.
Understandability	High	The use case model is easy to understand for all of the stakeholders in the project.
Process	Typically Iterative and Incremental	The use cases place the software requirements in context facilitating iterative and incremental development (the use case provides a good unit of delivery). Use cases can also be used with Waterfall development processes.
Development Style	Typically Object Oriented	Typically the use cases are used to drive object oriented software developments although use cases will work with any style of development. If an OO style is not adopted then a high level of explicit traceability will be required

Traceability Overview



Traceability Types

Traceability Type	Description
Use Case	This Traceability Type defines the traceability items representing the Use Cases.
Use Case Section	The Use Case Section enables us to include the sections of the Use Cases in our traceability hierarchy. This allows us to trace into the individual flows, and the other properties that comprise the use case. The presence of the hierarchical relationship to sub-sections allows us to capture the individual pieces of each section. For example this would enable us to identify the individual pre-conditions that make up the pre-conditions section. Note: In some cases it may be applicable to identify the individual software requirements within the flow of events of the use case (in this case very small sections) but this should not be attempted until the use case itself is stable.
Actor	This Traceability Type defines the traceability items representing the Actors.

Traceability Summary

Traceability Link	Description
Use Case to Use Case Section	Each use case is made up of a set of use case sections. This relationship allows us to track which use case sections make up which use case.
Use Case Section to Use Case Section	Some of the more complex use case sections are made up of many sub-sections. For example a flow of events may be made up of many sub-flows or the pre-condition section may be made up of many pre-conditions.
Use Case to Actor	This relationship allows us to see which Actors are involved in which use case.
Actor to Actor	This relationship allows us to capture both the name of the Actor and its brief description using a single traceability type.

Benefits and Disadvantages

Pros:

- Minimal document set
- Minimal effort involved in Requirements Management
- Good support for scope management, impact analysis and incremental development.
- Use cases are easy to understand.

Cons:

- There is no relationship back to the stakeholder needs. No real attempt is made to analyze the problem before starting on the definition of the solution.
- Some people consider it difficult to accept a contract based on just a use case model.
- Without undertaking any needs analysis it can be difficult to know when the use case model itself describes a suitable solution. It is easy to let your imagination run away with you when writing the use cases.
- If performing regular releases it can become difficult to product manage and continually manage the stakeholders expectations without any information at a higher level than the use cases themselves.

Examples

This approach is usually used for small scale, informal, internal projects where the developers and the users work very closely together.

The Use-Case Model Defines the Product Features

Description

In this case use-case modeling is used as the main requirement elicitation method and the Use-Case Model becomes the definition of the Product Features to be provided by the system as well as the statement of the software requirements.

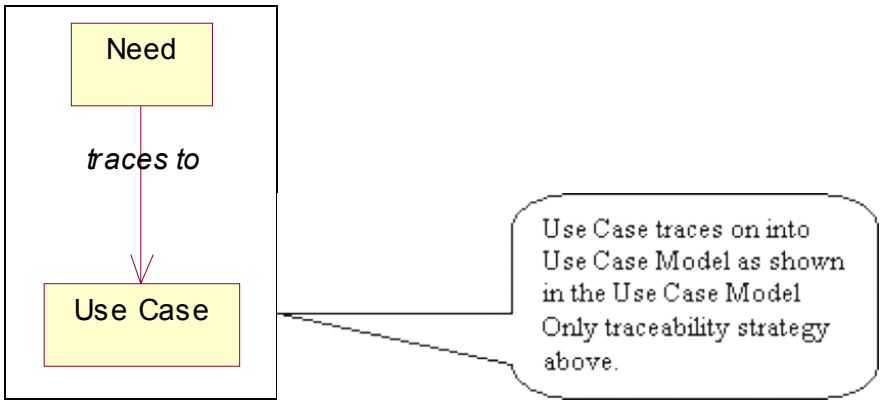
This option is only suitable for small developments, with short life cycles, as it does not scale. Even if each use case represents a feature of the system there will be more features than use cases and, in reality, many features may impact upon many use cases. As the system evolves the new features of each release are less and less likely to take the form of new use cases.

Characteristics

This is a variation on the previous "Use Case Model Only" approach. We have only noted the few differences when discussing this approach.

Characteristic	Value	Comment
Explicit Traceability	Low	As Use Case Model Only. There will be only a small set of needs and this small amount of additional traceability still results in small amounts of explicit traceability being required.
Trust	Medium / High	The addition of Needs to the Use Case Model makes this a slightly less trusting strategy than having a Use Case Model Only
Accountability	Low	
Formality	Low	
Completeness	Medium	The use case model itself facilitates establishing the completeness of the software requirements specification the additional traceability back to the stakeholder needs helps to validate the applicability of the use case model.
Document Set	Small	This approach involves a minimal document set. Use Case Model plus Vision Document containing the Needs.
Focus	User	The use case model and the needs both have user perspective.
Understandability	High	The needs and the use case model are both easy to understand for all of the stakeholders in the project.
Process	Typically Iterative and Incremental	As Use Case Model Only
Development Style	Typically Object Oriented	As Use Case Model Only

Traceability Overview



Traceability Types

Traceability Type	Description
Need	As defined for "No Use Case Model"
Use Case	As defined for "Use Case Model Only"

Traceability Summary

Traceability Link	Description
Need to Use Case	In this case the Needs trace directly to the use cases. The assumption being that the use cases can play the role of the product features when product and scope managing.

Benefits and Disadvantages

This approach is very similar to the "Use Case Model Only" strategy. The benefits and disadvantages are the same with the following additions and caveats.

Pros:

- In this case the use case model is related back to the stakeholder needs which helps to assess the suitability of the use case model.

Cons:

- The use cases may appear to define the features of the system in the early stages of the project but the two concepts will diverge as the project matures.
- Use cases are not features—what appears to be a time and labor saving strategy will quite quickly become an un-maintainable mess.

Examples

Although attempts to use this traceability strategy have been observed on small internal projects this approach is not recommended because of the scalability and long-term product evolution problems. It is recommended that the "Features Drive the Use Case Model" strategy be adopted if the use case model is to be supplemented with traceability back to the stakeholder needs.

Features Drive the Use-Case Model

Description

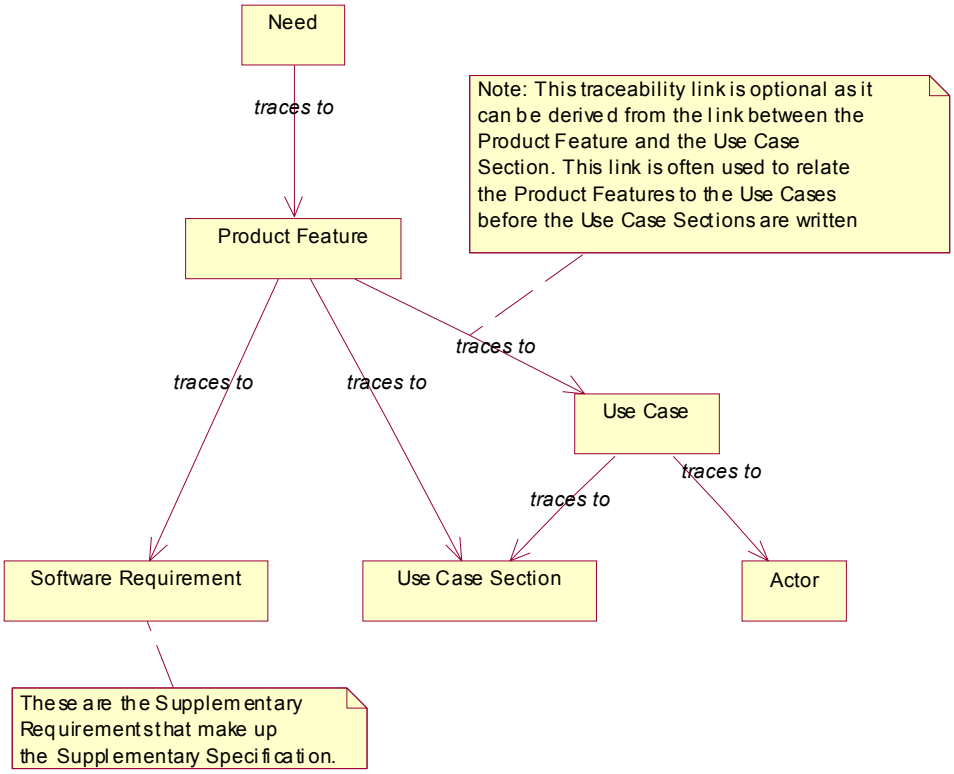
The Use-Case Model and Supplementary Specifications comprise my SRS." This is the strategy outlined and recommended by the RUP. The Needs and Product Features are documented in the Vision Document and are traced to use cases. If they are not reflected in the Use Case Model, then they are traced into the Supplementary Specifications.

In this case the Use-Case Model acts as the main statement of the software requirements. This is complemented by a supplementary specification containing the software requirements that cannot easily be expressed in the use cases themselves.

Categories

Characteristic	Value	Comment
Explicit Traceability	Medium	In addition to the use case model's implicit traceability, in this case, we have to explicitly maintain the traceability between the needs, features and the use case model.
Trust	Medium	
Accountability	High	
Formality	Medium	The addition of Needs and Product Features to the Use-Case Model gives rise to a more formal requirements management process than just maintaining a Use Case Model.
Completeness	High	Having both the Feature and the Use Case perspective on the Software Requirements makes it possible to achieve a high level of completeness with regards to capturing and prioritizing the Software Requirements.
Document Set	Medium	We now have a Vision document containing needs and features, a Use-Case Model and a Supplementary Specification.
Focus	Users, Stakeholders and Project Managers	The addition of needs and features to the use case model broadens the focus of the requirements activity to more actively encompass the product manager and all of the other stakeholders as well as the users. Features are a very powerful tool for managing stakeholder expectations and provide a good complement to the use case perspective of the software requirements.
Understandability	High	The definition of needs and features alongside a use case model with supplementary specifications provides a requirements model that is easily understandable by all of the stakeholders in the project.
Process	Typically Iterative and Incremental	As Use-Case Model only.
Development Style	Typically Object Oriented	As Use-Case Model only.

Traceability Overview



Traceability Types

Traceability Type	Description
Need	As defined for "No Use Case Model"
Product Feature	As defined for "No Use Case Model"
Use Case	As defined for "Use Case Model Only"
Software Requirement	Any software requirements that apply to the whole system or do not fit easily into a use case. Where a software requirement is a condition or capability to which the software being built must conform.
Use Case Section	As defined for "Use Case Model Only"
Actor	As defined for "Use Case Model Only"

Traceability Summary

Traceability Link	Description
Need to Product Feature	As defined for "No Use Case Model"
Product Feature to Use Case	Optional traceability link. The Product Feature may trace directly to Use Cases. This allows the Product Features to be assigned to Use Cases before the Use Case Sections have been written and allows you to do impact analysis on the Use-Case Model at the Product Feature level and vice versa.
Product Feature to Use Case Section*	The Product Features trace to the Use Case Sections. This allows the Use Case Model to be scope managed on a Feature basis and facilitates impact analysis between the Feature set and the Use Case Model at a level more applicable than the Use Case itself. All Product Features traced to a Use Case must also trace to one of the sections of the use case.
Product Feature to Software Requirement*	The Product Features also trace to the Software Requirements in the Supplementary Specification. All Product Features that do not trace into the use case model must trace to at least one of the software requirements in the Supplementary Specification.
Use Case to Actor	As defined for Use Case Model Only.
Use Case to Use Case Section	As defined for Use Case Model Only.

*Each Product Feature must trace to at least one Use Case Section or Supplementary Software Requirement. If it doesn't then it will not be included in the Software.

Benefits and Disadvantages

This approach maximizes the benefits provided by both the use case and traditional requirements management approaches whilst minimizing the disadvantages.

Pros:

- Well understood
- Recommended by the Rational Unified Process.
- Enables detailed, low level, formal traceability.
- Having both a Product Feature and Use Case perspective on the Software Requirements facilitates the completion of the requirements capture - this minimizes the chances of getting stuck in the requirements elicitation and capture activities.
- The software requirements are expressed in an easy to understand form.
- Impact analysis of requirements change is facilitated by this traceability strategy - the impact of not implementing a feature or a use case section can be clearly understood.
- The individual requirements have context supplied by the use cases and/or the product features. This makes it easy to identify meaningful sub-sets of the requirements. This in turn makes scope management, and the incremental delivery of the product, much easier.
- Minimal complete document set.
- Minimizes the effort involved in Requirements Management.

- This solution scales well. If performing regular releases the ability to scope manage at both the feature and use case level allows all of the stakeholders to track the progress of the project at the level of detail they find appropriate.
- In this case the use case model is related back to the stakeholder needs via the product features that help all the stakeholders assess the suitability of the use-case model.

Cons:

- Not acceptable in all organizations.
- Some people consider it difficult to write a contract based on Software Requirements expressed mainly as a use case model although many organizations have successfully achieved this.

Examples

This approach is applicable to all projects where use cases are accepted as a suitable format for the expression of the majority of the Software Requirements.

The Use-Case Model is an Interpretation of the Software Requirements Specification

Description

"The Use-Case Model is an interpretation of the formal SRS". This is most often used when a formal SRS is mandated due to regulatory or internal protocol.

A traditional SRS is often considered an essential part of reaching a contractual agreement when outsourcing or undertaking a fixed price development. This leads to two typical situations:

1. The development organization is supplied with a traditional SRS, by the customer, as the starting point for their system's development.
2. The SRS document is a mandatory or regulatory deliverable early in the project lifecycle. Every project must have a traditional, formal SRS document expressing the systems requirements in the same way as all of the other projects.

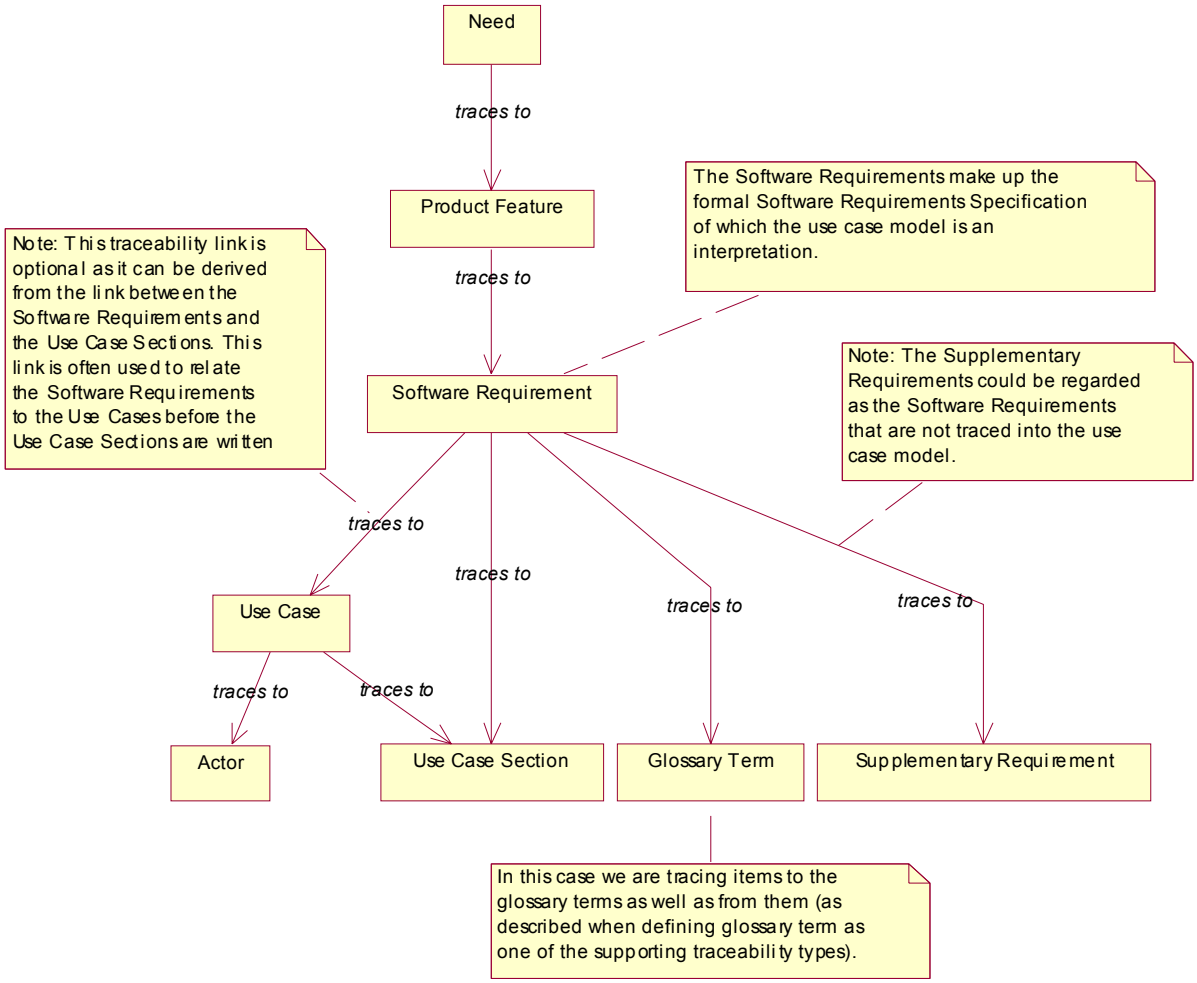
In these cases the use-case model is used to model and reinterpret all of the software requirements within the scope of the project. When this approach is adopted it is usual for the SRS to come first - there are other techniques available to render the information held by a use case model in a format that looks like a formal, traditional SRS (especially when the "Features Drive the Use Case Model" approach has been adopted) without creating a second Software Requirements definition.

Note: When adopting this approach there is no need for the set of "traditional" Software Requirements to be a complete statement of the functionality required—the Use-Case Model will provide or ensure completeness of the functional specification. The "traditional" Software Requirements may just be used to capture the Software Requirements directly identified or raised by the stakeholders.

Characteristics

Characteristic	Value	Comment
Explicit Traceability	Very High	All of the explicit traceability required by the "No Use Case Model" approach is required to maintain the formal SRS plus there is the additional overhead of tracing the traditional software requirements into the use case model.
Trust	Very Low	This is a very "belt and braces" approach implying a very low level of trust.
Accountability	High	
Formality	Very High	Again this is a very formal approach with two requirement management approaches being applied in parallel.
Completeness	Very High	The complementing of the traditional Software Requirements Specification with a Use Case Model makes this a very high completeness approach. Note: in this case it is assumed that it is the Use Case Model, which will ensure a complete specification of the systems functionality. The set of Software Requirement Specifications need not be to the same level of completeness.
Document Set	Very Large	In this situation we are basically specifying the system twice.
Focus	Contracts	The adoption of this approach is driven by the need to either fulfill an existing contract, expressed by a traditional SRS, or fit in with an existing development methodology requiring an SRS as the contract between the developers and the customers.
Understandability	Medium	The production of two Software Requirement definitions can be confusing at first but the use of the use case model as the master software requirement definition should lead to an easily understandable statement of the systems requirements.
Process	OPEN	There is enough stuff flying around to support almost any development process although the use case model is often added to a traditional SRS to enable the use of Iterative and Incremental techniques.
Development Style	OPEN	There is enough stuff flying around to support almost any development style although the use case model is often added to a traditional SRS to enable the use of Object Oriented techniques.

Traceability Overview



Traceability Types

Traceability Type	Description
Need	As defined for "No Use Case Model"
Product Feature	As defined for "No Use Case Model"
Software Requirement	As defined for "No Use Case Model"
Use Case	As defined for "Use Case Model Only"
Actor	As defined for "Use Case Model Only"
Use Case Section	As defined for "Use Case Model Only"
Glossary Term	As defined for "Use Case Model Only"
Supplementary Requirement	Any software requirements that apply to the whole system or do not fit easily into a use case. These may not need to be restated from the original set of software requirements but may just be the in scope software requirements that are not traced to the use case model.

Traceability Summary

Traceability Link	Description
Need to Product Feature	As defined for "No Use Case Model"
Product Feature to Software Requirement	As defined for "No Use Case Model"
Software Requirement to Use Case	Functional Software Requirements are traced to Use Cases. A sub-set of non-functional Software Requirements also trace to Use Cases. This relationship allows the high level scoping and assessment of the Use-Case Model in terms of the Use Case's requirements and business benefits. Note: All in scope functional requirements **must** trace to use cases or to the Glossary. If they are not reflected in this way then they will not be implemented.
Software Requirement to Use Case Section	The Software Requirements that are traced to a Use Case must also trace to one of the Use Case's Use-Case Sections. This relationship allows the verification of a Use Case's Use-Case Sections with regard to the requirements placed upon the Use Case. All of the Software Requirements traced to a Use Case must be fulfilled by one of the sections within the Use Case. The double traceability allows us to verify this and to allocate the Software Requirements to the Use Cases themselves before considering the Use-Case Sections required. It is possible that some sections will have no matching Software Requirements. Note: All functional requirements that trace to a use case **must** also trace to one of the use case's sections. If they are not reflected in this way then they will not be implemented.
Software Requirement to Glossary Term	Functional and non-functional requirements may trace to items in the Glossary. This is particularly true for "static requirements" that identify the attributes and relationships of the entities involved in the system. If a Glossary Term is traced from a Software Requirement then the Glossary Term must be used in one of the use cases or it is unlikely to be carried forward into the Design Model.
Use Case to Actor	As defined for "Use Case Model Only"
Use Case to Use Case Section	As defined for "Use Case Model Only"
Software Requirement to Supplementary Requirements	The Supplementary Requirements may restate the Software Requirements that apply to the whole system or do not fit easily fit into the use case model. An alternative approach is to regard all of the in scope Software Requirements that do not trace to the use case model as Supplementary Requirements - this would avoid restating them.

Benefits and Disadvantages

This approach is to be completely honest a little bit over the top. It is really only appropriate for projects that are presented with a traditional SRS and wish to use use-case modeling to attain an understanding of the supplied requirements and to facilitate a use case driven approach.

Pros:

- Enables very detailed, low level, formal traceability.
- The software requirements are expressed in an easy to understand form.
- Impact analysis of requirements change is facilitated by this traceability strategy - the impact of not implementing a feature, a software requirement or a use case section can be clearly understood.
- The individual requirements have context supplied by the use cases and / or the product features. The presence of the use case model makes it easy to identify meaningful sub-sets of the requirements. This in turn makes scope management, and the incremental delivery of the product, much easier.
- In this case the use case model is eventually related back to the stakeholder needs via the software requirements and the product features that help all the stakeholders to assess the suitability of the use case model.
- Acceptable (with caveats) in most organizations - this approach is all things to all people. This approach is often used on initial use case projects as a form of parallel requirements process (i.e. the project is running both the old way and the new way) or it can be adopted to hide the fact that the developers are using use cases.
- Good for minimizing the disruption to the organization when first adopting, or experimenting with, use cases. The outside world continues to see the traditional SRS, which allows the standard procedures and contracts to be used.

Cons:

- Not well understood - people will get confused by having both the traditional statements of requirements and the use case model.
- Having both traditional Software Requirement Specification and a Use Case Model gives you two places to get stuck in the requirements activities. It is easy to become confused over which should be the complete Software Requirement Specification
- A very large document set must be maintained.
- There is a lot of duplication that complicates the requirements management process. The traditional Software Requirements are likely to fall into disuse as the use cases are updated directly with requirement changes.
- This is a very high cost, high maintenance approach.

Examples

This approach is useful for development companies using use case driven development techniques that are given a traditional Software Requirements Specification as part of their contract. The introduction of use cases will enable the development company to demonstrate their understanding of the requirements and deliver the software in an iterative and incremental fashion.

It can also be a useful strategy when introducing use case techniques into a company that uses traditional requirements capture techniques and has a resistance to changing to a use case driven approach. In this case the intention is for the use cases to prove their value to the development organization and for the traditional Software Requirements Specification to be phased out as confidence in use cases grows. This could be the first step in moving towards the "Features Drive the Use Case Model" approach.

The Use Case Model Reconciles Multiple Sets of Traditional Software Requirements

Description

"The Use-Case Model is the interpretation of formal SRS's from multiple sources and provide the specification of a single common system."

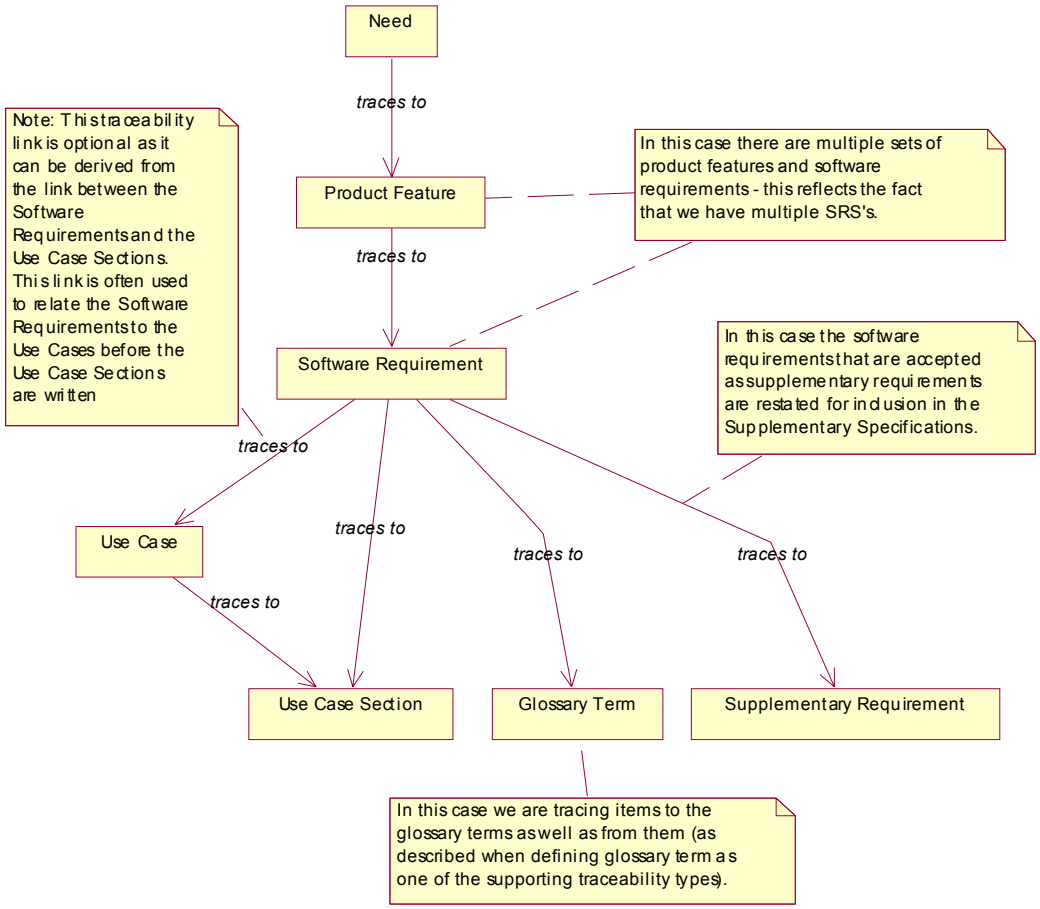
This is a variation on the "the Use Case Model is an interpretation of the Software Requirements Specification" except in this case there are multiple traditional SRS's supplied by multiple, independent sets of stakeholders. This situation often arises for software houses that are developing a single application to satisfy the requirements of many different, independent, un-connected customers. In this case the Use Case Model is the developers consolidated view of the system's requirements and the individual SRS's are the individual stakeholders view of their own requirements (with no integration or reflection of the other stakeholder's requirements). Tracing between the many, individual sets of requirements and the Use Case Model allows the developers to assess how well they are doing in assessing the needs of the various stakeholder.

Characteristics

This strategy is a variation on the previous "Use Case Model is an interpretation of the Software Requirements Specification" approach. We have only noted the few differences when discussing this approach.

Characteristic	Value	Comment
Explicit Traceability	Very High	As "Use Case Model is an interpretation of the Software Requirements Specification".
Trust	Very Low	As "Use Case Model is an interpretation of the Software Requirements Specification".
Accountability	Very High	In this case we preserve all of the individual customer perspectives in their own SRS's.
Formality	Very High	As "Use Case Model is an interpretation of the Software Requirements Specification".
Completeness	Very High	As "Use Case Model is an interpretation of the Software Requirements Specification".
Document Set	Very Large	In this case we have multiple specifications of the desired system, which are reconciled by the single Use Case Model.
Focus	Managing multiple independent customers.	The focus of this approach is in the management of multiple, independent, possibly contradictory requirements sources that are politically, geographically or organizationally unable to work closely together.
Understandability	Medium	As "Use Case Model is an interpretation of the Software Requirements Specification".
Process	Typically Iterative and Incremental.	In this case the developers use the Use Case Model as their SRS. See "Use Case Model Only"
Development Style	Typically Object Oriented	In this case the developers use the Use Case Model as their SRS to drive the software's development. See "Use Case Model Only"

Traceability Overview



Requirement Types

Requirement Type	Description
Need	As defined for "No Use Case Model"
Product Feature	As defined for "No Use Case Model"
Software Requirement	As defined for "No Use Case Model"
Use Case	As defined for "Use Case Model Only"
Use Case Section	As defined for "Use Case Model Only"
Glossary Term	As defined for "Use Case Model Only"
Supplementary Requirement	Any software requirements that apply to the whole system or do not easily fit into a use case.

Traceability Summary

Traceability Link	Description
Need to Product Feature	As defined for "No Use Case Model"
Product Feature to Software Requirement	As defined for "No Use Case Model"
Software Requirement to Use Case	As for Use Case Model is interpretation of SRS
Software Requirement to Use Case Section	As for Use Case Model is interpretation of SRS
Software Requirement to Glossary Term	As for Use Case Model is interpretation of SRS
Software Requirement to Supplementary Specification	In this case the software requirements must be restated in the Supplementary Specifications document that supports the use case model to allow the production of a single, consistent, compete SRS for the development that draws open information from the multiple, individual stakeholder SRS's.

Benefits and Disadvantages

This strategy is a variation on the previous "Use Case Model is an interpretation of the Software Requirements Specification" approach and has much the same benefits and disadvantages.

The benefit of this approach that differentiates it from the other strategies is its ability to deal with, and preserve the viewpoints of independent stakeholders in the form of their own formal, individual SRS's.

It also has the additional disadvantage of generating an even bigger document set to maintain and trace.

Examples

A software house in the UK was developing a system to support Insurance Brokers that would allow the Insurance companies to electronically distribute new products.

This project had in the region of 22 stakeholders of which approximately two thirds were Brokerage companies and one third was Insurance companies. Among these stakeholders were widely differing requirements; in many cases the insurers had completely contradictory requirements to the Brokers.

In this case it was decided to produce a Software Requirement Specification for each stakeholder company detailing their specific requirements and allowing their individual perspectives to be easily maintained. The use case model was used to present the consolidated vision of the system to all of the stakeholders. Traceability from their original SRS's into the use case model allowed the stakeholder to see exactly which of their requirements would be met by the system and to validate that the system was suitable for their needs. It also allowed the software house to track their progress against their target of fulfilling 80% of all stakeholder requirements for each stakeholder.

Rational®
the e-development company™

Corporate Headquarters
18880 Homestead Road
Cupertino, CA 95014
Toll-free: 800-728-1212
Tel: 408-863-9900
Fax: 408-863-4120
E-mail: info@rational.com
Web: www.rational.com

For International Offices: www.rational.com/worldwide

TP 166 /00. Subject to change without notice.