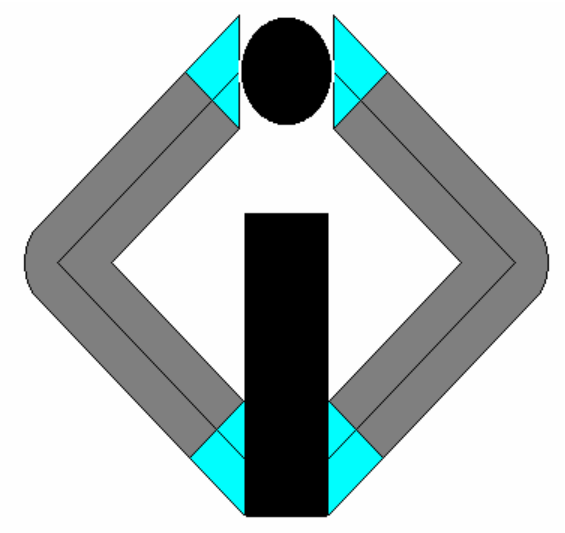


By Andrei Kapustin,
Cybernetic Intelligence GmbH,
Luzern, Switzerland

Requirements Modeling

Process Description

P-RMP-2000-08-1-0

August 1, 2000

Contents

Introduction

This report is dedicated to the requirements analysis for software projects. In particular, it addresses the following issues:

- The place of the requirements analysis in the project development cycle.
- The information flow and processing during the requirements analysis.
- The notational conventions used in the process of requirements analysis and to represent its results.
- The issues related to requirements analysis in the real world, including reuse of requirements analysis results and dealing with uncertain information.

Software project life span

Traditionally, the life span of a software system under development is subdivided into three well-known phases:

- Analysis – creating the formal (or semi-formal) description of the problem.
- Design – creating the formal (or semi-formal) description of the solution to the problem defined by an analysis stage.
- Implementation – increasing the formality of the solution supplied by design stage and lowering the utilized level of abstraction to match some real or virtual computer.

Each stage (in the simplest case) feeds its results as an input to the subsequent stage. The results of the implementation stage, clearly, are used as an input to the real/virtual computer.

However, the analysis stage cannot start in vacuum. The input to the analysis stage is the description of the problem performed by using a variety of diverse means. In general case, any information can be used as an input to the analysis stage, regardless of how formal the said information is. This information, taken together for a given project, is traditionally referred to as the project requirements. Just as traditionally, the level of formality achieved in project requirements depends on the project domain. In some fields (notably scientific calculations) the formality of project requirements hits the 100%. Other fields (notably business support software) are known for informal, incomplete and contradictory requirements for projects. The rest of this document is dedicated to the software projects in such low-formality requirements domains.

For small projects, it may well be possible to perform a competent analysis using the original (raw, as passed on from the customer) project requirements as an input. However, as the project gets larger, the informality of the project requirements starts to play increasingly destructive role. The importance of this problem hits larger projects more severely than smaller once.

Requirements analysis

The solution, when we come to think of it, is fairly obvious: before we start the analysis phase (which will, eventually, result in formalizing the *problem*), we perform requirements analysis, which formalizes the *description* of the problem.

The requirements analysis consists of three steps:

- Collecting requirements from the customer.
- Synchronizing requirements, i.e. making sure the set of project requirements has no repetitions and/or contradictions between its elements.
- Formalizing requirements, i.e. defining explicitly what exactly is stated by each requirement.

The following sections briefly describe the ways to formalize the requirements.

Establishing a common language

The most common form in which project requirements come is the textual description written in some natural language. On the positive side, the natural language is the most expressive means of communication we have so far. On the negative side, we can think of several shortcomings of natural languages:

- The semantics of a natural language is non-functional (i.e. a single word or phrase can have more than one meaning, and in some cases the exact meaning of the word or phrase cannot be determined from the context).
- The semantics of a natural language is person-specific (i.e. different persons assume the same word or phrase to have different meanings).
- The semantics of a natural language is not static (i.e. the same word or phrase may be assumed to have different semantics at different times).

The first and last point always contribute to the knowledge distortion, regardless of the number of people involved in the project. We can also suggest that the knowledge distortion originating from the last point (dynamic nature of natural language semantics) may increase with the number of people involved.

The second point (semantics of a natural language being person-specific) hits teams proportionally to their size. However, even a one-person project is not entirely free from this trouble – because there's still at least one other person involved – the customer who states the requirements.

It's not generally possible to eliminate the use of the natural language in the project requirements. However, it is possible to reduce the damage. If we look at the list above, we can use the negation of assertions listed therein as the first approximation of what to do:

- Define explicitly the full functional semantics for the natural language, which is used to write requirements.
- Ensure that the all project participants are using same language.
- Disallow (or, at least, minimize) the mutation of the semantics over time.

This list, of course, represents the situation that can never be achieved:

- A definition of full functional semantics of a natural language is theoretically impossible (due to the fact that natural languages mutate).
- Forcing all project participants to use the same language is practically impossible (there is little chance to force customer to do so).
- Stopping the mutation of semantics over time is possible – but usually not achievable in real projects.

Therefore, we must remember that the semantics of the language used to describe project requirements can only achieve a certain degree of being fully functional and static (the larger – the better for the project).

The following sections elaborate on the points made above.

Defining the full functional semantics

The words and phrases of the natural language can be roughly subdivided into 4 categories:

1. Those, which are not related to the project domain.
2. Those, which are related to the project domain but have a single universally accepted semantics in the natural language of choice.
3. Those, which are related to the project domain and have multiple universally accepted semantics in the natural language of choice.
4. Those, which are related to the project domain and have no universally accepted semantics in the natural language of choice.

The first two categories are, of course, no problem, since then do not violate the functionality of semantics with respect to a given project. The latter two categories, however, do.

The 3rd category requires qualification of semantics – i.e. the choice of one of alternative meanings. The choice is, clearly, project- (or domain-) specific.

The 4th category consists of special terms, acronyms, jargon, etc. Each word or phrase in this category must be explicitly assigned a single semantics. Again, the assignment is project- (or domain-) specific.

Any of the words and/or phrases in categories 2 through 4 may require the explicit definition of semantics given in the context of the project.

Sharing the language

It’s not sufficient just to define a project-specific subset of a natural language – the said subset must also be consistently used by all participants in the project. The term “all participants” refers, of course to the project team – but also includes the customers who write the project requirements.

From this observation, it is clear that the semantics of a language in which requirements are written is oriented to the customer, not to the project team. The latter usually has no choice but to accept the language spoken by the customer.

Immutability

It clearly does no good to have a common language with full functional semantics if we allow the semantics to mutate with time. Such mutation leads to the same word or phrase being used with different semantics in different project-related documents. To the person reading such documents at some later time, the semantics of the language used to write the documents appears non-functional.

Traceability of semantics

Often it's not enough to just arbitrarily select some semantics for a word or phrase in a natural language. What is needed is not only the information about *what* semantics has been selected but also *why* it (and no other) was selected. This information about "why" will be referred to hereafter as the *traceability of semantics*.

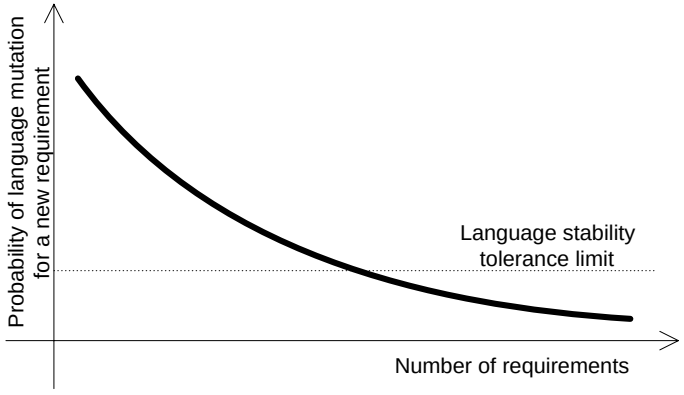
The latter piece of information offers most significant benefits to the immutability of the language semantics – each time the semantics of the language can change it becomes possible to weight the reasons for the change against the reasons to disallow it.

Processing the requirements

The most common (and about the most difficult to make any sense of) form in which project requirements are passed from customer to developer is a flat list of requirements, mostly written in a natural language (i.e. "we want A, B, C and D from your software").

Writing requirements in a common language

Of course, the first step is to ensure that all such requirements are written in the same language – the common language described in the previous section. Here it is important to realize that, given a set of requirements $R=\{r_1, r_2, \ldots, r_n\}$, all written in a common language L, the arrival of a new requirement r' from the customer may result not only in the mutation of the set R, but also in the mutation of the language L. This most often happens when r' uses some word or phrase which has not been used in any of the requirements $r_1 \ldots r_n$. Of course, with the arrival of more requirements the rate of mutation of the common language falls, as shown on the graph below:



(Of course, the graph above applies only to the situation when all requirements are collected in a single repository and are written in the same language).

The language in which requirements are written is considered “static enough” when the mutation probability falls below a project- (or domain-) specific limit. For some projects/domains reaching the full language stability is actually possible (for example, if we have an established international standard for project- (or domain-) specific jargon.

Structure and classification of requirements

Provided all requirements are written in a common language with full functional semantics, there are still things that can be done to make problem description more precise (which is, after all, the entire point of the requirements analysis).

Apart from their use of natural language, requirements usually come from the customer as a flat list (i.e. “we want A, B, C and D from your software”). The imposition of an appropriate structure on those requirements may, depending on the project, offer significant benefits, which increase with the number of requirements (and, hence, project size).

The most obvious (and, historically, the most widely known) way to structure requirements is to subdivide them into classes. The “Business”/”Technical” subdivision (sometimes referred to as “Functional”/”Non-functional”) of requirements into two disjoint sets is often used. However, two modifications to this scheme may result in significant benefits to projects:

- Providing hierarchical classification of requirements instead of flat classification. For example, we can further subdivide Technical requirements into something like Performance, User Interface, Scalability, etc.
- Allowing the requirement classes to intersect. This feature is, no doubt, not often needed. However, it cannot be proven that there are no possible projects that will benefit from this.

Associating requirements with requirement classes is not the only thing that can be done. Often two or more requirements are related to each other. Such association can, for example, be:

- The whole/part relationship, e.g. the requirement “The data system must be scaleable” can have two sub-requirements “The system should allow large data sets to be processed” and “The processing time should not increase more than logarithmically with the size of the data set”.
- The synonym relationship, e.g. two requirements “The system should be highly interactive” and “The system response time should not exceed 1 second for any query”, which come from two different customers, are (in the simple case) synonyms.
- Indeed, any n-ary relationship, e.g. we can have three requirements “The system should be delivered on time”, “The system should be reliable” and “The system should be created within a budget”. The well-known rule of

thumb says than at most two requirements out of the three above can be achieved.

In general, there appears to be no set of relationships between requirements that will be feasible “as is” for any conceivable software project.

Traceability of requirements

The requirement traceability is quite similar to the traceability of “common language” semantics. The only significant difference is following: for language semantics we need to trace *why* the semantics has been chosen, while for requirements the most important thing to know is *where* they come from.

Note that, although the semantics of traceability differs somewhat between “common language” semantics and requirements, both can be traceable to the same source. For example, both the definition of some business term (language semantics) and the constraints associated with the term (requirements) may come from the same document.

Requirements modeling results

The output from the requirements analysis stage consists of (at least) the following knowledge repositories:

- Glossary (also sometimes called Dictionary).
- Requirements repository, which must be in a consistent state.
- References to the sources of information upon which the requirements modeling was performed.

These knowledge repositories are describes briefly in the following sections.

Glossary

The glossary is used as the repository of knowledge about the “common language” used for the project. The glossary consists of those words and phrases of the natural language that do not have exactly one meaning in the said language (usually referred to as glossary terms). For each such word/phrase, the glossary contains:

- The unique unambiguous definition of the semantics of the word/phrase.
- The relationships (if any) with other words/phrases in the glossary (e.g. relationships such as “A is a synonym of B” or “A is used in the definition of B’s semantics”, etc.).
- Traceability links (if necessary), which record the reason why the glossary term is defined the way it is.

The project glossary is used throughout the entire project lifetime:

- Requirements written in a natural language utilize glossary terms where necessary.
- At the analysis/design stages elements of the model (e.g. classes, attributes, etc.) should be traceable to the glossary terms that represent business concepts.

- Glossary terms specific to a business domain (not to a particular projects) can be reused for multiple projects in the said domain.

Requirements repository

This knowledge repository stores all project requirements together with their structure (classification and relationships of requirements to each other). Just like glossary, the requirements repository, once created, is used throughout the latter stages of the project development:

- Requirements can be used for resource estimation in a uniform manner.
- At the analysis/design stages constraints on the model (e.g. business rules, business exceptions, etc.) should be traceable to requirements.
- Requirements specific to a business domain (not to a particular projects) can be reused for multiple projects in the said domain.
- Technical (non-functional) requirements are an essential input to the design and, later, implementation (coding) stage.

References

The traceability of glossary terms and requirements to their source requires references from the glossary/requirements repository to the information available outside of the project' scope. The exact nature of this information can vary widely (e.g. laws, business practices, corporate & worldwide standards, etc.)

Glossary modeling

The glossary is a semantic net where:

- Nodes represent glossary terms.
- Links represent relationships between different glossary terms.

Glossary terms

In its simplest, a glossary terms establishes semantics for some words or phrase in a natural language. Therefore, a structure of the glossary term is similar to the structure of a single entry in traditional (printed) dictionaries – the glossary term consists of key (the word/phrase being defined) and the definition.

This, however, presents only the static view of glossary terms. Two other points must be taken into consideration:

- The glossary is not a dead repository of knowledge, but a live data structure, which can change as the project is developed.
- Since the mainstream software development uses CASE, the glossary terms should be navigable to from where they are used.

Logging

In many situations, it is important to be able to track the evolution of a glossary term (e.g. the refinement of a definition, etc.)

This calls for some form of logging, i.e. the modification history associated with each glossary term.

Multiple definitions

It is not uncommon in real projects that the same word/phrase of the natural language allows several different definitions coming from several different sources.

Of course, the requirement for the full functional semantics means that, eventually, one of these definitions will be chosen (or a new definition will be combined out of the existing ones) but, while the choice is being made, all (conflicting) definitions must be in the glossary.

Navigation

If we do modeling in a CASE tool, we have many elements of the model described in a natural language (e.g. business processes, use cases, etc.). The project glossary is the “glue” which holds all these descriptions together, ensuring that the same word has the same meaning in every such description.

The navigation to the glossary is a sort of “context help” for the reader of these descriptions. The reader must be able to demand the definition of any unfamiliar word

or phrase he encounters in such description. If the definition exists in the glossary, the reader should have it.

In addition, the navigation by definition can be used if it can be implemented consistently. This covers the case when a user selects a definition of a concept and wants to find out if there is a glossary item that matches the definition.

This calls for some sort of hyperlinking, either explicit or context-based.

Relationships between glossary terms

It is not clear if the general n-ary associations between glossary terms are of much use. However, there are at least three specific cases:

- Synonym – an undirected n-ary association between glossary terms, which divides the glossary into non-intersecting subsets of terms. The relationship "synonym(A_1,A_2,…,A_n)" represents the fact that all glossary items A_1,A_2…A_n define concepts with the same semantics in a context of the project. We will refer to the transitive closure of this relationship as a synonym cluster.
- Dependency – a directed binary acyclic association between glossary terms. The relation "A depends-on B" means that to understand the meaning of A the reader must first understand the meaning of B.
- Qualification – a strong form of the dependency. The relation "A qualifies B" means that the semantics of the term A is the same as the semantics of the term B with some extra constraints imposed.

Traceability links

Generally, the glossary term should be traceable to its source (where the definition have come from). The support for the multiple definitions requires separate traceability for each alternative definition of the glossary term (definitions are different precisely *because* they come from different sources).

The more detailed elaboration of glossary term traceability links is given in the corresponding chapter of this report. Here we will only note that glossary terms are usually traceable to some information existing outside of the project scope (and, therefore, outside of project repository).

Glossary structure

Putting all glossary terms in one huge flat list will, no doubts, result in severe difficulties with glossary search and navigation.

To help matters, we should once again consider that project glossaries are (roughly) equivalent to traditional (printed) dictionaries. The standard technique to organize large dictionaries as sets of chapters, each chapter containing items on one particular subject, can be easily emulated for the CASE glossary.

In the simplest form, we organize the glossary not as one huge flat list, but as a hierarchy of subject-oriented chapters. The glossary term can then appear under the

appropriate chapter. Whether it should be allowed to have the same glossary terms in more than one chapter or not is unclear, but the more general solution (allowing glossary term to be listed in several chapters) is just as easy as the restricted one (requiring only one chapter for the glossary term) both conceptually and from the implementation point of view.

Glossary diagrams

Having all items stored in the single glossary is not enough – we have to view and update the glossary.

Traditionally, CASE tools provide a tree-like view of the dictionary. However, for most analysis/design domains (i.e. class domain, use case domain, etc.) diagrammatic representations of the subset of dictionary are available. This offers benefits such as:

- Each diagram can concentrate on only a small subset of dictionary, unified by some logical concepts (e.g. “all classes in the same package”, “all actors and use cases involved in the same business process”, etc.). “Small” always means “easier to read, understand and modify”.
- Diagrammatic notation specifies the dictionary items and their relations to each other more explicitly that a tree-like view of the dictionary does.

Therefore, while the glossary will be, undoubtedly, stored as a part of CASE tool dictionary, there is a need for the diagrammatic notation as well. At present, no CASE tool or modeling standard provides such notation (indeed, virtually no CASE tools/standards recognize the requirements modeling as a software development activity worth modeling).

An Appendix B to this document describes the notation used for glossary diagrams.

Requirements modeling

The requirements repository is a semantic net containing:

- Nodes representing requirements.
- Nodes representing requirement classifiers.
- Links between requirement classifier nodes, which represent the classification topology.
- Links between requirement nodes and requirement classifier nodes, which represent the classification of requirements.
- Links between requirements nodes, which represent relationships between different requirements.

Requirements

A requirement is a constraint imposed on the project.

As such, the requirement encapsulates at least one attribute – the constraint itself, expressed in some language. Natural language is most widely used, although the requirement can be expressed in a graphical (diagrams), mathematical (formulas) or any other language, as well as a combination of different notations.

Theoretically, requirements can emerge at any stage of the project development. Practically, the only requirements, which matter in a real project, are those that come from the customer.

Logging

In many situations, it is important to be able to track the evolution of a requirement (e.g. refinement, rephrasing, etc.)

This calls for some form of logging, i.e. the modification history associated with each requirement.

Glossary links

A project requirement expressed in a natural language is more likely than not to utilize some terms from the project glossary. Therefore, to write (and understand) requirements a person needs the glossary.

Requirement classifiers

A requirement classifier is a logical concept, which defines a subset of the available requirements.

Typical requirements classifiers are “Functional” and “Non-functional” (alternatively called “Business” and “Technical”), which are often used to separate project requirements into two (normally) disjoint sets.

However, the general definition of a requirement classifier as a logical concept allows other requirement classifiers to be built arbitrarily. Clearly, “all requirements for a given project” is one of such logical concepts, which can be applied to any project. Alternatively, requirements can be classified functionally (e.g. “performance requirements”, “security requirements”, etc.).

Classification topology

When we allow for the introduction of arbitrary classifiers on a project-specific basis, the “flat” topology of requirements classifiers becomes too inflexible.

Instead, we can allow requirement classifiers to be organized into a tree, where each node is a requirement classifier and the parent/child relationship between two classifiers denotes the set/subset relationship between requirements assigned to the classifier. The root of the tree is, of course, exactly the general requirement classifier “all requirements for a given project”.

Finally, the classification of requirements is, clearly, dependent on the purpose of classification. The classification by stereotypes (e.g. “Business”, ”Technical”, etc.) is helpful for the analyst/designer, while the functional classification (e.g. “performance requirements”, “security requirements”, etc.) may be much more convenient for the designed/implementer of individual parts of the project.

This, in general case, means that more than one classification can be used on the same set of requirements. Since each classification is represented by a hierarchy of requirement classifiers (a tree), multiple classifications will be represented by a set of disjoint hierarchies (a set of trees). We well refer to this entire set as the *requirements classification topology*, and to the individual classification (a single hierarchy of requirement classifiers) within the set as the *requirements classification profile*.

Given the semantic net representing the requirements repository, the requirements classification topology is represented by directed binary relationship “parent/child” between requirement classifier nodes. The relationship has the following properties:

- A requirement classifier node may not have more than one parent. However, it can have any number of children.
- Every requirement classifier node with no parent represents the separate requirements profile.

Classification of requirements

The classification of requirements is an assignment of requirements to sets defined by requirement classifiers. In the semantic net representing the requirements repository, the classification of requirements is represented by a binary relationship between the set of requirements and the set of requirement classifiers.

The assignment of a requirement to a requirement class may not coincide in time with the introduction of the requirement. Therefore, the relationship “classification of requirements” is optional.

On the other hand, each requirements classification profile covers (by definition) an entire set of project requirements. Therefore, a given requirement may be assigned to not more than one requirement classifier per profile.

Relationships between requirements

It is not clear if the general n-ary associations between requirements are of much use. However, there are at least five specific cases:

- Correlation – an undirected n-ary association between requirements, which divides the set of requirements into non-intersecting subsets of requirements. The relationship "correlate(A_1,A_2,…,A_n)" represents the fact that implementing one of the requirements A_1,A_2…A_n will imply the need to implement all other requirements in the same set. We will refer to the transitive closure of this relationship as a correlation cluster.
- Dependency – a directed binary acyclic association between requirements. The relation "A depends-on B" means that to satisfy the requirement A the system (being constructed) must first satisfy the requirement B. This semantics implies that the constraint imposed by requirement A is stronger than that imposed by requirement B.
- Decomposition – a strong form of the dependency. The set of relations "A decomposed-as B_1,B_2,…,B_n" means that the system satisfies the requirement A if and only if it satisfies all requirements B_1,B_2,…,B_n.
- Equivalence – an undirected n-ary association between requirements, which divides the set of requirements into non-intersecting subsets of requirements. The relationship "equivalent(A_1,A_2,…,A_n)" represents the fact that all requirements A_1,A_2…A_n define the same constraint on the project. We will refer to the transitive closure of this relationship as an equivalence cluster.
- Exclusivity – an undirected n-ary association between requirements, which divides the set of requirements into (possibly intersecting) subsets of requirements. The relationship "exclusive(A_1,A_2,…,A_n)" represents the fact that of all requirements A_1,A_2…A_n only and exactly one should apply to the project.

Traceability links

Generally, the requirement should be traceable to its source (where the requirement have come from).

The more detailed elaboration of requirement traceability links is given in the corresponding chapter of this report. Here we will only note that requirements are usually traceable to some information existing outside of the project scope (and, therefore, outside of CASE tool repository).

Requirements diagrams

Just as in case of glossary, project requirements can be presented in many different forms. The graphical (diagrammatic) representation of the requirements (based on the corresponding semantic net) offers advantages such as:

- A diagram can concentrate on a small subset of requirements, thus assisting in understanding requirements and their relationships to each other.
- A requirements diagram is more expressive and, so, is easier to understand and maintain than (nearly) any other representation.

In a manner similar to the glossary modeling, the requirements modeling calls for a specialized graphical notation. At present time, CASE tools do not offer this feature (or, indeed, any nontrivial support for requirements modeling at all).

An Appendix C to this document describes the notation used for various kinds of requirements diagrams.

Reference documentation

The input to the requirements modeling always comes from the customer.

This, in turn, means that the form in which it comes is heterogeneous and largely project domain – specific. In low-formality requirements domains (like business) this usually means that:

- A significant part of the information is expressed in some natural language.
- The said information is (nearly always) bound to have omissions and/or contradictions.

To achieve generality, the only thing we can (relatively) safely assume about the input information is that it consists of some number of project source documents – atomic (from the point of view of the customer!) knowledge repositories.

Obviously, it makes little to separate the software project from its source documents, since these documents are needed for many tasks throughout the project:

- Project source documents are the only form of input for the requirements analysis.
- Each business domain – related decision in the scope of the project (e.g. a presence of a requirement, a definition of a concept, a content of a process or interaction model, etc.) must be traceable to its source.

In previous chapters, it has been said that glossary terms and requirements must be traceable to their sources. This, conceptually, means that glossary terms (and requirements) have reference(s) to specific location(s) within the project source document(s) associated with them. When the requirements analysis is supported with the CASE tool, these references should be directly navigable.

The exact nature of reference depends on the nature of the project source document referred to. For CASE tool – supported requirements analysis, all project source documents can be divided into three categories:

- Local documents – project source documents that are stored in the CASE tool repository.
- Remote documents – project source documents that are available in electronic form outside of the CASE tool repository.
- Virtual documents – project source documents that are not available in electronic form and, therefore, cannot be directly referred to by CASE tools.

These three types of source documents roughly correspond to project-specific source documents (local), domain- or organization- specific standards (remote) and legal/international standards (virtual).

Project context

In the real world, project requirements rarely come from only one source.

Information reuse

More often then not some parts of the project glossary, requirements repository and/or source documents can be reused over several projects:

- The same glossary terms, requirements and/or source documents can apply to several sequential versions of the project.
- The organization-wide standards of glossary terms, requirements and/or source documents can apply to several separate projects developed by the organization.
- The business domain-specific standards of glossary terms, requirements and/or source documents can apply to several separate projects developed by different organizations.
- International standards of glossary terms, requirements and/or source documents can apply to any project being developed worldwide.

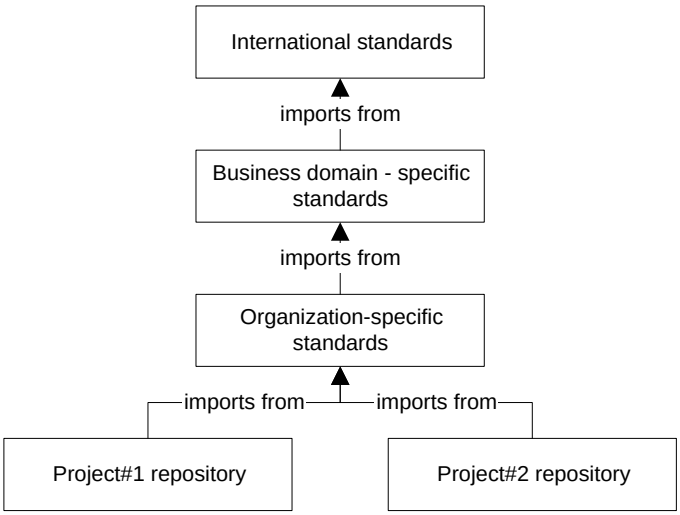
Information reuse topology

If we think of the complete set of information available on the project as the project repository, then the static snapshot of the entire software industry can be (in most cases) viewed as a DAG (Directed Acyclic Graph) where:

- Nodes represent various project repositories, which correspond to projects.
- Arcs from node A to node B indicate that the project A borrows (reuses) some information from the project repository of the project B.

Of course, both wide variety of largely incompatible means to organize & store the project repositories makes such importing possible only when it has been planned by both projects participating in the import (both client and supplier). Similarly, the visibility of projects is usually strictly limited (to one company, or even to one department within a company).

This allows to simplify the most common case of inter-project information reuse from DAG to the tree, as shown on the following diagram:



If we look at the reuse tree shown on the diagram above, we will see that information on higher levels offers the general covering of wide areas of knowledge, which lower levels offer more detailed information on narrower subjects. At the lowest level (single project), the information is related to only one software project and contains enough details to implement a solution.

CASE support

Of course, the most significant benefits are attained when:

- The project development is assisted with a CASE tool.
- The CASE tool directly supports the concept of a project repository.
- The CASE tool directly supports the notion of inter-project reuse of information.

If all three assertions are valid, then:

- The organization developing software can use the uniform technology for all its projects.
- High level of reuse can be achieved throughout the project lifetime, including the reuse of glossary terms, requirements and source documents during the requirements analysis.

The following sections briefly describe several most commonly used types of information reuse.

Linking

Normally, when an information item (a glossary term, a requirement, a class, etc.) is created in a project repository, it is local to this project repository. Linking allows to create an information item which is a ‘shadow’ of some item in another project repository. Additional semantics of linking may include:

- Access control – which defines whether modifications to the shadow item are allowed and, if yes, whether they inflict simultaneous modifications on the item being linked to.

- Freezing – which defines whether the modifications made to the original item should be reflected in its shadow. A frozen shadow remains immutable even if the original item changes.
- Filtering – which allows the shadow to possess only a subset of properties of an original item.

Subscription

A subscription to an item in a project repository causes the subscription client to receive notifications whenever the state of the item subscribed to changes. This, in effect, is the reuse of an item' mutation history.

Externalization

The externalization of an item (or a set of items) from the project repository is a process of writing a minimal consistent subset of a project repository that contains all externalized items to the data storage external to the CASE tool. The opposite process, called internalization, is the process of reading data from the data storage external to the CASE tool and incorporating the appropriate information into the project repository. Alternatively, the terms "export" and "import" are used in many existing CASE tools.

Externalization has two broad areas of application:

- Information interchange between different CASE tools, or between different repositories powered by the same CASE tool.
- Creation of backup copies of project repositories, in order to minimize information loss in case of an accident or deliberate mischief.

Probabilistic modeling

In low-formality requirements domains (including business) it is a common situation when any given person does not have the complete and/or accurate knowledge of the domain. Since it is these persons (domain experts) who provide the source documents for software projects, we can assume that, in general case, no source document can be safely considered 100% reliable.

To incorporate the measure of information reliability into the project development we have to start right at the beginning of the project life cycle – with requirements analysis, since this is the stage which deals with unreliable information more than any other stage of the project development.

Reliability ratings

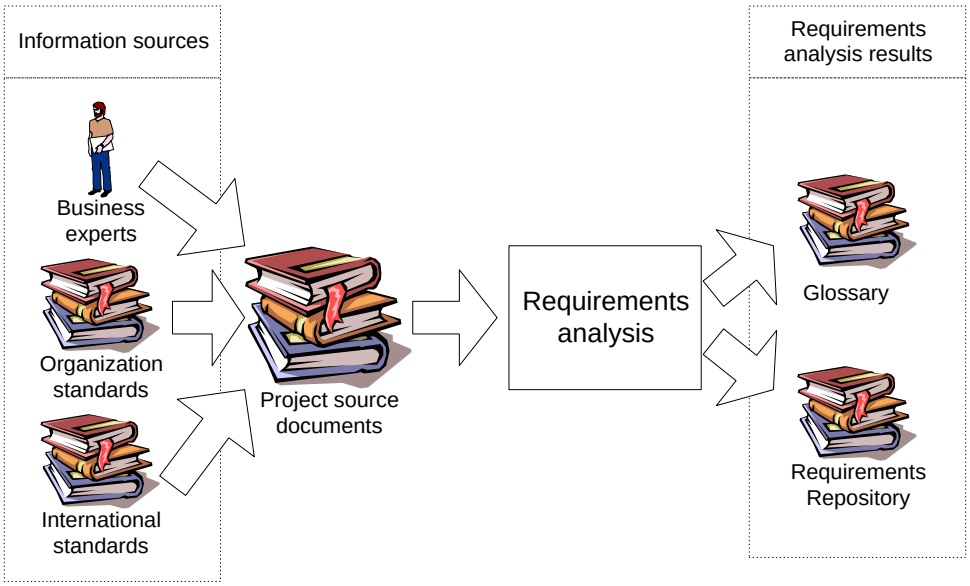
The reliability of information given as an input for the project depends on two measures:

1. The reliability of the information source. This roughly corresponds to how well the person giving the information knows the problem domain. This value is measured as the probability that the information source supplies the valid information and ranges from 0 (information source utterly unreliable) to 1 (information source utterly reliable).
2. The reliability of the information as judged by the person giving this information (e.g. a business expert may say things like “I’m nearly sure that terms ‘Client’ and ‘Customer’ mean the same thing for this project”). This value is measured as the probability that the information represents the truth as seen by the person supplying the information and ranges from 0 (a person is absolutely sure he supplies false information) to 1 (a person is absolutely sure he supplies the truth).

Both measures must be taken into account.

Reliability ratings during requirements modeling

The (simplified) information flow during the initial stages of the project development is shown on the following diagram:



To track information reliability throughout the requirements analysis, we need to:

- Associate the reliability rating with each information source providing input for the project. We will refer to this rating as the credibility of an information source.
- Make sure each project source document is traceable to its source. The credibility rating of a document is the same as the credibility rating of the document's source.
- Associate the reliability rating with project source document, according to how reliable the document's source believes the document to be. We will refer to this rating as the document certainty.

Based on these reliability ratings, various elements of the requirements model are assigned (manually or automatically, depending on available tool support) their own reliability ratings:

- A reliability of a glossary term (a definition of a project-related term) is the probability that the definition is true in the context of the project.
- A reliability of an association between two glossary terms is the probability that the association is both essential and true in the context of the project.
- A reliability of a requirement is the probability that the system being developed must obey the constraint imposed by the requirement. This measure should not be confused with the requirements priority (which roughly corresponds to the eventual number of intended users of the project who will want a requirement to be implemented).
- A reliability of an association between two requirements is the probability that the association is both essential and true in the context of the project.

There is no point in assigning reliability ratings to association between requirement classifiers, or to associations between requirements and classifiers they belong to. This is due to the fact that the requirements classification is created by the project team and, as such, has no direct relation to the customer's view of the project. In other

words, we assume that, although information coming from the customer may be unreliable, the project team always does the right thing.

CASE support for probabilistic modeling

If the requirements modeling is assisted by a CASE tool, then all (or, at least, as much as possible) project-related information is stored in a CASE tool repository. In the ideal case, that means that:

- All information sources are registered in the CASE tool repository, together with their credibility ratings.
- Project source documents are stored in the CASE tool repository as local, remote or virtual documents. Regardless of the document class, the document certainty rating is stored with every document.
- Each project source document is traceable to one (or more) information source(s) registered in the CASE tool repository.
- Various elements of the requirements model (glossary terms, requirements, associations, etc.) are stored in the CASE tool repository together with their assigned reliability ratings. These ratings can (in some cases) be assigned automatically, based on the credibility and certainty of the related project source documents.

Appendix A: Glossary

Certainty rating

A reliability rating of a project source document, as given by the project information source responsible for the document creation. The document certainty rating is a measure of the probability that the document represents the truth performed by the document's creator.

Credibility rating

A reliability rating of a project information source, measuring the probability that the information coming from the information source is true.

Equivalence

An n-ary relationship between requirements, which represent the fact that all requirements participating in the same instance of a relationship have the equivalent semantics in a context of a given project.

Equivalence cluster

An n-ary relationship between requirements, which is a transitive closure of the equivalence relationship.

Exclusivity

An n-ary relationship between requirements, which represent the fact that of all requirements participating in the same instance of a relationship only and exactly one should be taken into account.

Externalization

A process of writing a minimal consistent subset of a project repository that contains all externalized items to the data storage external to the CASE tool.

Glossary

A repository of knowledge, which defines the semantics of words/phrases of the natural language in a context of a given project.

Glossary term

A definition of semantics (meaning) of the word or phrase of the natural language given in a context of a given project.

Glossary term qualification

A strong form of the glossary term dependency. A glossary term A is a qualification of a glossary term B if and only if the semantics of A is the same as the semantics of B with a non-empty set of additional constraints imposed on it.

Glossary term dependency

A glossary term A is dependent on the glossary term B if and only if in order to understand the semantics of A the semantics of B must first be understood.

Information reuse between project repositories

The process of granting the project repository controlled access rights to entities stored in other project repositories.

Internalization

The process of reading data from the data storage external to the CASE tool and incorporating the appropriate information into the project repository.

Local document

A project source document, which is stored in the same CASE tool repository in which other analysis and/or design activities in the scope of the project take place.

Project information source

An entity, external to the project development team, which provides the information about the project' intended goal and/or constraints.

Project repository

A centralized container of information about the project, storing both individual items of information and the relation of these items to each other.

Project requirement

A constraint on the software project, typically given by the customer.

Project source document

A document, which is supplied by a customer and is used as a repository of knowledge about the project.

Reliability

A reliability of information is a measure of probability that the information represents the truth in a context of a given project.

Remote document

A project source document, which is available in an electronic form outside the CASE tool repository in which other analysis and/or design activities in the scope of the project take place.

Requirement classification profile

A hierarchy of requirement classifiers with a single root. Child classifiers in a hierarchy break the set of requirements selected by their immediate parent classifier into non-intersecting subsets of requirements.

Requirement classification topology

A set of requirements classification profiles, which describes the classification of project requirements from different points of view.

Requirement classifier

A logical concept, which defines a subset on the set of all existing project requirements, as appropriate for a given classification criteria in the scope of a project.

Requirement decomposition

A requirement A is decomposed into a set of requirements $B_1, B_2, \ldots, B_n$ if and only if to satisfy the constraint imposed by requirement A the system must satisfy all constraint imposed by requirements $B_1, B_2, \ldots, B_n$.

Requirement dependency

A requirement A is dependent on a requirement B if and only if to satisfy constraint imposed by requirement A the system must first satisfy the constraint imposed by requirement B.

Requirement exclusivity

Two requirements A and B are exclusive if and only if the system must satisfy exactly one of the constraints A or B, but not both or neither.

Requirements modeling

A process of collection, synchronization and formalization of available information about the project' intended goal and constraints, performed in order to pass knowledge from the customer to the project team.

Synonym

An n-ary relationship between glossary terms, which represent the fact that all glossary terms participating in the same instance of a relationship have the same semantics in a context of a given project.

Synonym cluster

An n-ary relationship between glossary terms, which is a transitive closure of the synonym relationship.

Virtual document

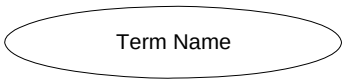
A project source document, which is not available in an electronic form.

Appendix B: Glossary diagrams

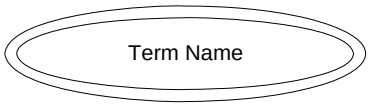
This appendix describes the graphical notation used for glossary diagrams.

Glossary terms

Glossary terms are drawn as elliptical shapes, containing the name of an item. If a glossary term has more than one definition, it is drawn with a double border.



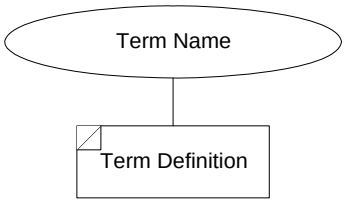
The glossary term with a single definition.



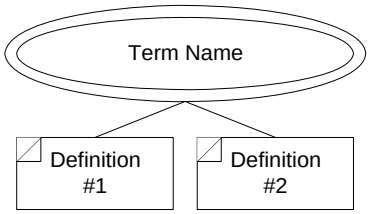
The glossary term with multiple definitions.

Definitions of glossary terms

A definition of a glossary term is represented by a rectangular box containing the definition text. The box is linked to the appropriate glossary term.



The glossary term with a single definition.

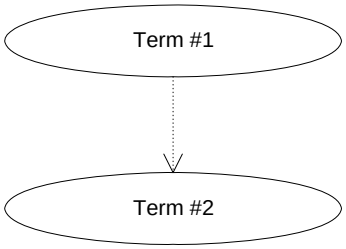


The glossary term with several definitions definition.

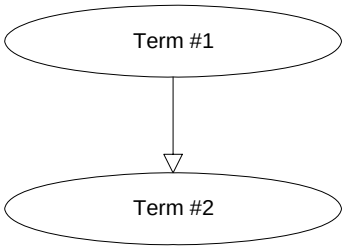
Whether all, some or none of glossary term definitions are shown on the particular glossary diagram is dependent on the purpose of the glossary diagram.

Dependencies and qualifications

The dependencies and qualifications between glossary terms are drawn as, respectively, dashed and solid arrows from the dependent term to an independent one.



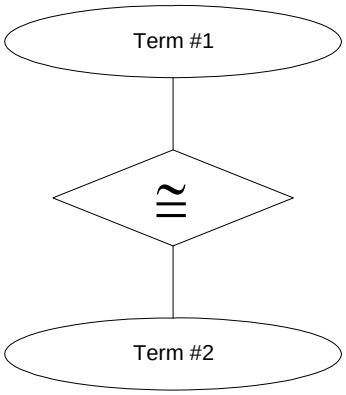
The glossary term #1 is dependent on the glossary term #2.



The glossary term #1 is a qualification of the glossary term #2.

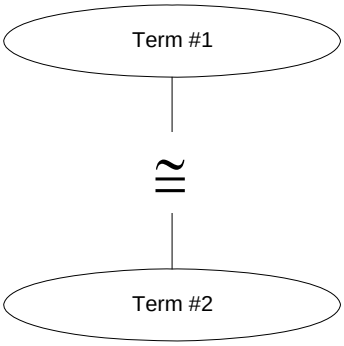
Synonyms

An n-ary synonym relationship is drawn as a rhombic shape labeled with the relationship stereotype ”≅“, which is connected to all participating glossary terms with solid lines.



The glossary terms #1 and #2 are synonyms.

If only two items participate in a synonym relationship, the rhombic shape in the middle of the line can be omitted, but the relationship stereotype ”≅“ must still be present.

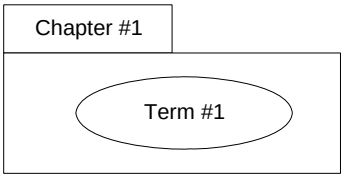


The glossary terms #1 and #2 are synonyms.

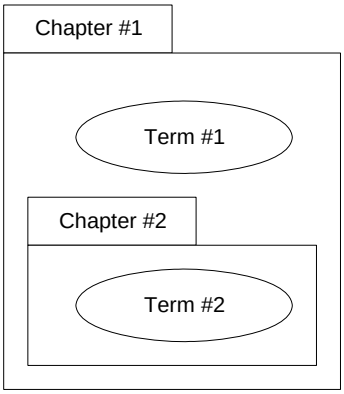
Glossary chapters

Is a glossary consists of several chapters, the folder shape is used to show the chapter boundaries on the glossary diagram. All glossary terms (and other chapters) drawn inside the chapter area belong to the chapter, unless the fully qualified name of the glossary term (or chapter) is given. This provides the notation similar to that of packages on class diagrams.

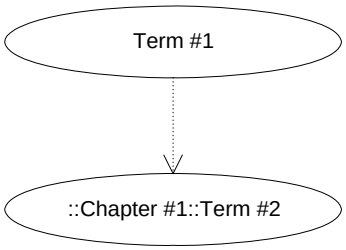
The fully qualified name consists of the full path in the chapter hierarchy from the glossary root to the glossary term (or chapter). Each element of the path, including the first one, is preceded by a double colon "::". The single name preceded with a double colon indicated the global glossary term or chapter.



The glossary term #1 is defined within a particular chapter.



The glossary chapter #1 contains a glossary term #1 and a sub-chapter #2. The latter, in turn, contains the glossary term #2.



The glossary term #1 is dependent on the glossary term #2 defined in the Chapter #1.

Arbitrary n-ary relationships

If there is a need to specify the arbitrary n-ary relationship between 2 or more glossary items, the notation similar to that used for synonym relationship should be used, with the relationship stereotype ”≅“ replaced with a custom stereotype.

In this case, the glossary is considered inconsistent unless it contains glossary terms for each custom relationship stereotype, and each term has a single definition.

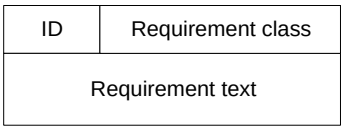
Appendix C: Requirements diagrams

This appendix describes the graphical notation used for requirements diagrams.

Requirements

A single requirement is drawn as a rectangular shape divided into three compartments, which correspond to the three attributes always essential for requirements:

- Project-wide requirements ID.
- Requirement class.
- Requirement text.



The requirement with a given ID, class and textually described semantics.

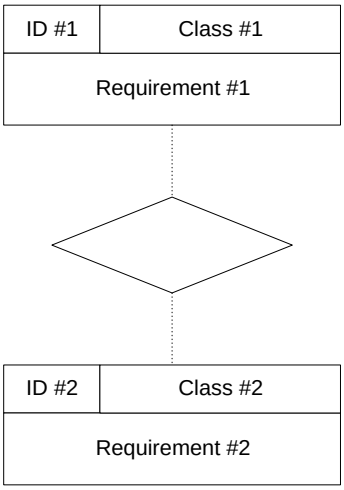
The requirement class is represented by the fully qualified name of the requirement classifier. This fully qualified name consists of the full path from the root of the requirement classification profile to the appropriate requirement classifier. Each element of the path is preceded by a double colon "::".

The topology of the requirement classification (i.e. a set of classifier trees representing requirement classification profiles) does not have the graphical notation. Therefore, requirement classifiers and their relationships to each other are never explicitly shown on requirements diagrams.

If a requirement is assigned to several classifiers from different profiles, these classifiers are written in the "Requirement class" compartment of the requirement shape one under another. Only those classifiers that are essential for the given requirements diagram need to be present on it.

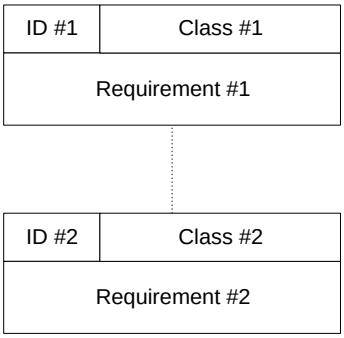
Correlation between requirements

A correlation of two or more requirement is drawn as a diamond shape from which a dashed line is drawn to each correlated requirement.



The requirements #1 and #2 are correlated.

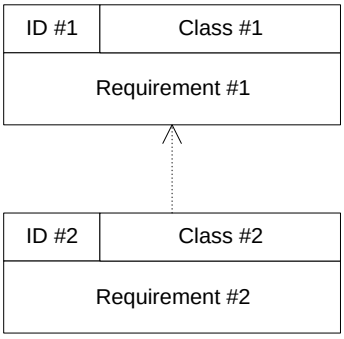
If the correlation is binary, the diamond shape is unnecessary and can be omitted.



The requirements #1 and #2 are correlated.

Dependencies between requirements

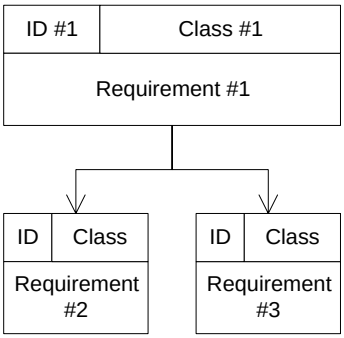
A dependency of one requirement upon another is drawn as a dashed arrow line from the dependent (stronger constraint) to the independent (weaker constraint) item.



The requirement #2 depends on the requirement #1. Effectively, this means that the requirement #2 imposes a stronger constraint on the system than the requirement #1.

Requirement decomposition

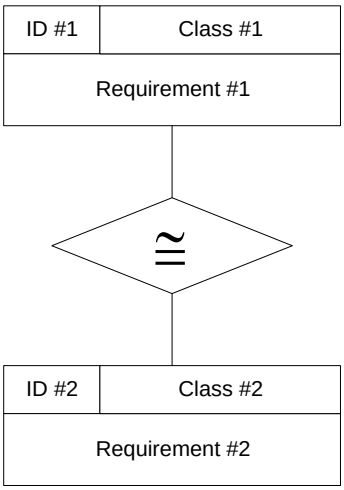
A decomposition of one requirement into a set of more specific requirements is drawn with a solid arrow line from the parent (full requirement) to the child (sub-requirement) symbols.



The requirement #1 is decomposed into two requirements #2 and #3. Each of the requirements #2 and #3 is weaker than the original requirement #1, because only when applied together do the requirements #2 and #3 match the semantics of the requirement #1.

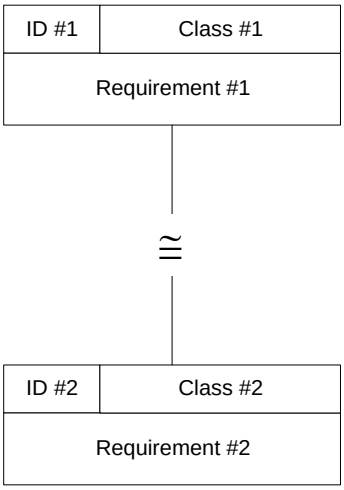
Equivalence

An equivalence relationship is drawn as a rhombic shape labeled with the relationship stereotype ”≅“, which is connected to all participating requirements with lines.



The requirements #1 and #2 are equivalent.

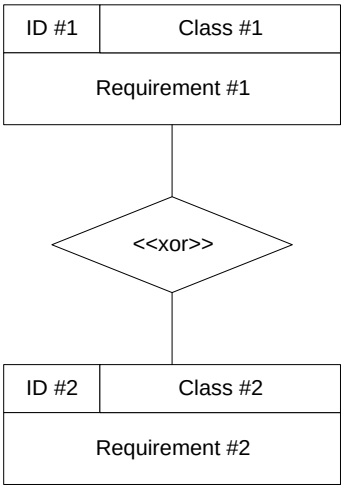
If only two items participate in an equivalence relationship, the rhombic shape in the middle of the line can be omitted, but the relationship stereotype ”≅“ must still be present.



The requirements #1 and #2 are equivalent.

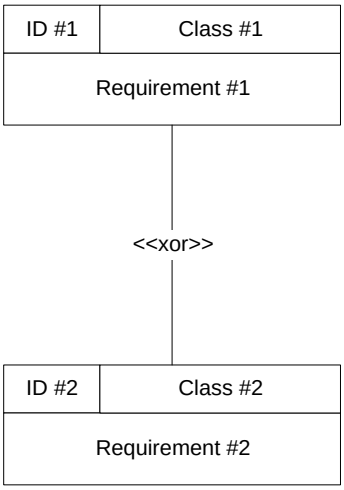
Exclusivity

An exclusivity relationship is drawn as a rhombic shape labeled with the relationship stereotype ”<<xor>>“, which is connected to all participating requirements with lines.



The requirements #1 and #2 are mutually exclusive.

If only two items participate in an equivalence relationship, the rhombic shape in the middle of the line can be omitted, but the relationship stereotype ”<<xor>>“ must still be present.



The requirements #1 and #2 are mutually exclusive.

Arbitrary n-ary relationships

If there is a need to specify the arbitrary n-ary relationship between 2 or more requirements, the notation similar to that used for equivalence/exclusivity relationships should be used, with the relationship stereotype ”≅“ or “<<xor>>” replaced with a custom stereotype.

In this case, the project glossary is considered inconsistent unless it contains glossary terms for each custom relationship stereotype, and each term has a single definition.