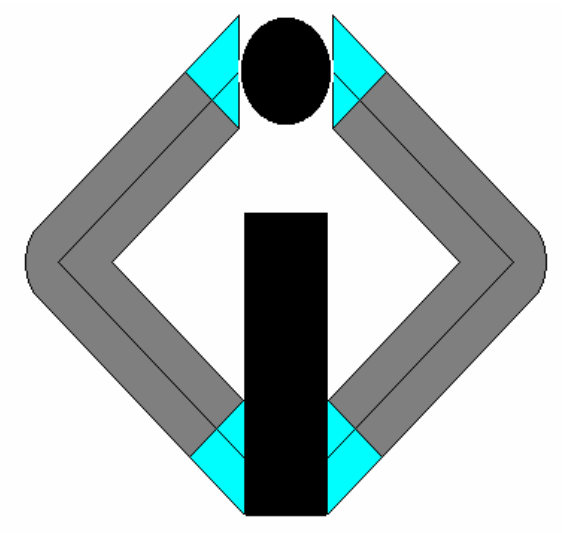


Authors: Andrei Kapustin,
Vadim Chirikov,
Marina Farr
Cybernetic Intelligence GmbH,
Zug, Switzerland

Requirement analysis: Methodology

P-RAM-2002-10-1-0

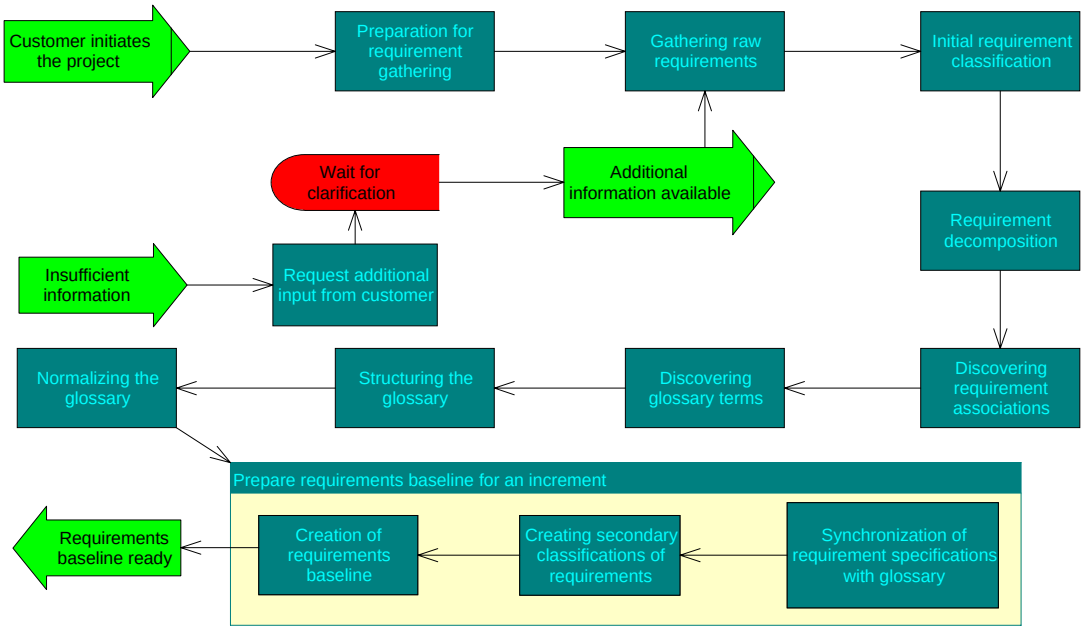
September 10, 2002

Contents

This document outlines main steps and techniques used during requirement analysis in software system development. It is assumed that requirement analysis is assisted by EasyRM.

1 Overview

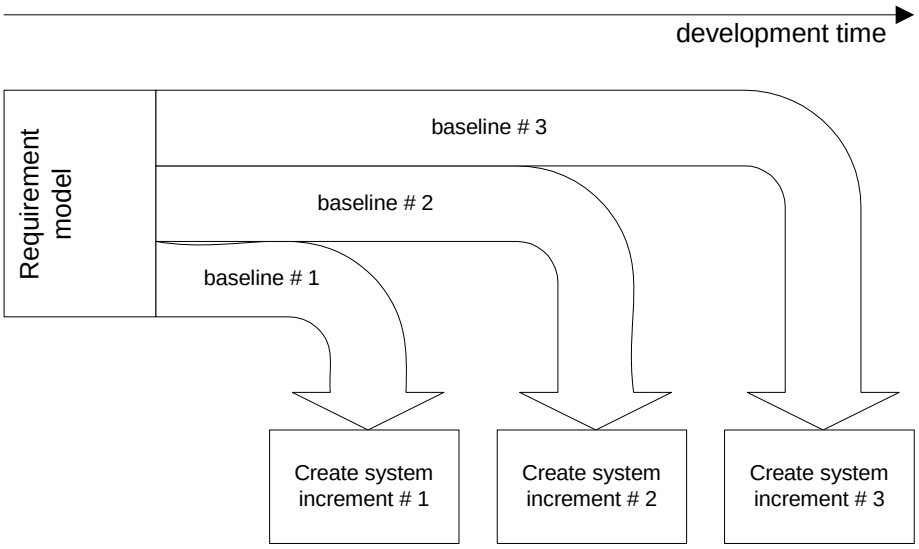
The diagram below presents the overall view of the requirement modelling process.



2 Preparation for requirement gathering

Before we start gathering and analysing project requirements, we must prepare a place where these requirements will be gathered and analysed.

Generally, requirements for a specific project will be gathered and analysed in a single EasyRM database. As requirement analysis is carried out, one or more requirement baselines will emerge from the project requirement set to be used as a basis for further development activities.



There are three basic steps involved in preparation for requirement analysis:

- Creation of project-specific requirement repository
- Injection of generally relevant information into the requirement repository
- Injection of project-specific information into the requirement repository

2.1 Creation of a new requirement model

Creating a new requirements model in EasyRM is simple enough, but this is not the end of story. The newly created EasyRM database does not have anything in it. The first step that must be performed next is an introduction of standard requirement model template.

A standard requirement model template defines:

- A basic document library structure, which allows categorizing project-related information into shared vs. project-specific, and further sub-categorization of shared project-related information into industry-wide and corporate standards.
- A basic requirement classification by the aspect of the system requirements describe. This basic classification separates static (“to be”) from dynamic (“to do”) requirements, and then further separates business process – related dynamic requirements from technical dynamic requirements.

The standard requirement model template is described in Appendix A to this document.

2.2 Introduction of shared knowledge

Shared knowledge represents the context in which a project is developed and/or will eventually operate. This shared knowledge can come in several forms, such as:

- Industrial and/or corporate standards which shall be adhered to in the course of project development

- A product within a product family (or a component within a component-based product) will inherit some glossary terms and/or requirements from the product family (or a component-based product).
- A product re-using a pre-made solution may inherit glossary terms and/or requirements from that pre-made solution's requirement model.

Usually, shared knowledge will be copied from another (shared) requirement model and, once copied to a project-specific requirement model, will be placed in proper context.

2.3 Introduction of project-specific knowledge

Project-specific knowledge is introduced into the document library area of the project's requirement model in the form of documents, such as:

- Project vision and mission statements
- Logs of customer interviews
- Evaluations by subject matter experts and elaborations on the subject

2.4 Iterative nature of knowledge introduction

Rarely is all project-relevant information available at the beginning of the requirement analysis. Typical examples of this situation are:

- A need to clarify the meaning of term or requirement during later stages of requirement analysis. This may cause additional standards, expert knowledge or, indeed, explanations in any form to be registered as project-related documents, either shared or project-specific.
- A need by the customer to change his demands on the system being developed. The logs of newly introduced customer demands need then to be registered as project-specific information sources.

3 Gathering raw requirements

Once a set of project-related information sources is established, these are processed to extract the initial (raw) set of project requirements.

3.1 Creating raw requirements

All project-related information source documents can be roughly subdivided into two categories:

- Documents that describe what is expected of the solution (usually project-specific), and
- Documents that provide the frame of reference for the project (usually corporate and industrial standards).

All documents of the 1st category and some documents of the 2nd category must be processed for raw requirements. The rule of thumb here is that an industrial and corporate standard is used for "See also" references unless it specifies business processed and/or rules relevant in the context of the project.

Extracting raw requirements from project source documents is mainly a manual process, which involves carefully reading through the documents and explicitly creating requirements discovered on the way.

Typical patterns to search for when looking for raw requirements include:

- "The program must…" – such sentence becomes a requirement.
- "The following rules must be followed …" – each rule becomes a requirement.
- "A [some subject] has the following properties …" – this sentence becomes a requirement.

Requirements mentioned in project source documents usually come with some explanations. For example, a demand that "each account shall have a unique number" is usually accompanied by the description of what an account number looks like. Similarly, a demand that the form for entering customer' details must have a specific look-and-feel is usually followed by a rough drawing of the desired form layout. In such cases, detailed explanations are left out of the main requirement text – anyone who needs these explanations can look them up.

3.2 Requirement traceability

All requirements shall be, from the beginning, traceable to their sources and other relevant information. To ensure this:

- When a new [raw] requirement is created, it shall be immediately associated with the project source document in which it has been discovered as having originated from that document.
- When a description of the requirement [in some project source document] contains "See also" references to other information sources, the raw requirement, once created, must be explicitly associated with these "See also" information sources as utilizing them.

During initial requirement gathering, no attempt is made to rephrase obscurely formulated requirements, resolve conflicts between different requirements, etc. – all these steps will be taken later on.

4 Initial requirement classification

It is, usually, not possible to collect all raw project requirements in one go. A good point to stop gathering raw requirements and proceed with further requirement analysis is when reading any project-specific document does not help the reader to easily come up with more requirements.

The initial classification of requirements takes place next. During this step raw requirements are assigned to classes pre-defined by the standard requirement model template.

4.1 Static requirements ("to be")

Static requirements define static properties of the system. Typical static requirements include:

- Specification of attributes (structure) of various business objects (e.g. "For each customer a full name, phone number, postal address, etc. must be stored").
- Specification of constraints on relationships between business objects e.g. "A customer is either a personal or corporate customer" or "Each account can only have one owner").

Later on during requirement analysis, many static requirements will migrate to the glossary, in form of glossary terms and descriptions of their properties.

4.2 Dynamic requirements ("to do")

Dynamic requirements specify the intended behaviour of the system. During the initial classification of requirements, business dynamic requirements are already separated from technical dynamic requirements.

4.2.1 Business dynamic requirements

Broadly speaking, business dynamic requirements describe how the business process [which is to be supported by the application being developed] shall proceed. These requirements specify common business actors, threads, events, rules, exceptions, etc. which all take place in the course of the business process regardless of whether it has been computerized or not.

4.2.2 Technical dynamic requirements

As opposed to business dynamic requirements, technical dynamic requirements specify that behaviour which results from the fact that a business process will be computerized. This may include (among other things) specifications of:

- What computer platform(s) the intended solution shall operate on.
- What temporal constraints must be respected (if the system under development is a real-time system).
- What technical and business resources shall be required and/or sufficient for maintenance and trouble-free operation of the solution.
- What level of competence will be required from the intended users of the solution, and how these users will attain the said level of competence (guides, training, help systems, etc.).

5 Requirement decomposition

The next step of requirement analysis is requirement decomposition. This process involves going through the raw requirements [extracted from the project-related information supplied by customer], locating the requirements specifying more than one constraint on the system and breaking them into atomic requirements.

5.1 Why decompose?

There are several reasons why atomic requirements are of importance during software development and must be explicitly discovered. Some of these reasons are:

- Raw requirements specified by customers may mention several aspects of the system in one requirement. This makes clean classification of these requirements into static vs. dynamic and business vs. technical difficult.
- Raw requirements specified by customers may have repetitions or rephrasing of the same constraint on the system in several raw requirements. This duplication makes the further project development difficult.
- Raw requirements specified by customers may partially contradict each other. Decomposition of such requirements into simpler ones helps to clearly understand which parts of requirements conflict.
- Each atomic requirement represents a system function point – a software metric used in several software measurements. Identifying system function points early on helps to estimate system complexity and resources required for development.

The decomposition of requirements is the first step in the attempt to create a clear, unambiguous and minimal set of requirements to be used as a basis for further development work.

5.2 Completeness during decomposition

As mentioned, requirement decomposition is a manual process. However, it is governed by several guidelines, which have to be strictly adhered to in order to avoid introducing distortion into the requirement model.

One of these guidelines insists on a completeness of requirement decomposition:

> *If a requirement is discovered to be complex and is then decomposed into several simpler requirements, these simpler requirements, when taken together, must cover the original complex requirement completely. In other words, a complex requirement shall be decomposed into simpler subrequirements in such a way that the system satisfies the original requirement if and only if it satisfies all its simpler subrequirements.*

5.3 Decomposition of related requirements

Informally, two or more requirements are considered related if they, fully or partially, specify the same constraint on the system.

During decomposition of such requirements, each will become several simpler (and, eventually, atomic) subrequirements. Some of these subrequirements will, though, say roughly the same thing.

These “roughly saying the same thing” requirements are left “as is” for now. During the following steps of requirement modelling, these requirements will be explicitly checked to see how “same” they are in the context of a project and, maybe, some of them will be later merged. This may eventually result in some atomic requirements participating in decomposition of more than one original [raw] requirement.

5.4 Keeping requirement traceability up-to-date

When decomposing raw requirements, new [simpler] requirements are introduced into the system. For each such new requirement, its source must be explicitly traceable to.

The rule of thumb is that a subrequirement shall be initially marked as originating from the same source as its superrequirement (i.e. a requirement whose decomposition has yielded a subrequirement). Later on, when some common atomic subrequirements are merged, their sources must be merged as well.

6 Discovering requirement associations

Requirements defining the needed system functionality do not normally stand each on their own, but have a complex pattern of associations. One such association – decomposition relation between complex requirements and simpler requirements obtained during its decomposition – has already been investigated during the previous stage of requirement modelling.

6.1 Discovering dependencies between requirements

Dependencies between requirements are essential for impact analysis of requirement models. A requirement X is dependent upon requirement Y in the following situations:

- If, in order to understand the requirement X, a system designer or implementer must first read and understand requirement Y.
- If, in order to implement the requirement X, the requirement Y must be implemented first.

Requirement dependencies may form an arbitrary acyclic graph. In other words, a single requirement may depend upon any number of other requirements. Similarly, any number of other requirements may depend upon the same requirements. Paths along the requirement dependency graph roughly correspond to various orders in which requirements can be implemented.

6.2 Discovering conflicts between requirements

Conflicting requirements are most commonly introduced when project-related information (and, subsequently, requirements) come from more than one source.

Initially, we have to manually discover conflicting requirements and mark them as such. Note, that two requirements shall be always either “entirely conflicting” (i.e. exclusive) or not – if two requirements are “partially conflicting” then their decomposition [performed earlier on] shall have yielded mutually exclusive subrequirements by now.

Later on, during requirement baselining, information about conflicts between requirements will be used [among with other available information] to ensure consistency of requirement baselines.

6.3 Discovering equivalent requirements

Informally, equivalent requirements are requirements that say "the same thing" in the context of the project. Implementing any one of the equivalent requirements is enough to implement them all.

Later on, during requirement baselining, information about equivalence between requirements will be used [among with other available information] to ensure that requirement baselines do not include irrelevant requirements.

6.4 Discovering correlations between requirements

There is no formal or universally agreed upon definition of what the phrase "two requirements are correlated" means. In our approach, we recommend marking as correlated those requirements that, for some business or technical reason, shall be implemented together.

Later on, during requirement baselining, information about correlations between requirements will be used [among with other available information] to ensure consistency of requirement baselines.

7 Discovering glossary terms

The next step of requirements analysis establishes the language in which requirements are specified.

7.1 Scanning requirement specifications for glossary terms

During this step, specifications of requirements are read and searched for:

- Words and/or phrases that do not have a single, universally accepted, meaning.
- Words and/or phrases whose universally accepted meaning does not correspond to what they mean in the context of the project.
- Acronyms.
- Words and phrases for which explicit definitions are given in the requirements (usually these will be static "to be" requirements).

All these things are potential glossary terms. The criterion defining which of these shall become "true" glossary terms is the question "Is this word or phrase relevant for this project". The simpler version of the same question is "Do we need to use this word or phrase when talking about this project".

It shall be explicitly said that all glossary terms are project-specific. This means that, although there can be, in glossaries for different projects, terms with the same names and definitions, the set of terms included into the glossary is entirely dependent upon the project that glossary belongs to.

7.2 Writing definitions for glossary terms

Once project's glossary terms are identified, they must be defined. Initial definitions of glossary terms come from two sources:

- Glossary terms explicitly defined in static requirements receive their definition according to *how* the said static requirements actually define them.
- For glossary terms mentioned in the requirements but for which there are no requirements explicitly containing term definitions, appropriate definitions must be found in the project source documentation. This may include the need to extend the set of project source documentation by logs of interviews with customers (the subject of said interviews being the interpretation of unknown terms) and industrial or corporate standards (which often include glossaries of industrial or corporate terms).

When writing initial definitions for glossary terms, one shall not worry about repetitions or inconsistencies – these will be taken into account later.

7.3 Traceability of glossary terms

All glossary terms shall be, from the beginning, traceable to their sources and other relevant information. To ensure this:

- When a definition of the term is created from a requirement, the document(s) from which this requirement has originated shall be immediately associated with the term as term origin(s).
- When a definition of the term is created from a document (user interview log, industrial or corporate standard glossary, etc.), the document(s) where the definition of term was taken from shall be immediately associated with the term as term origin(s).
- When a definition of the term contains “See also” references to other information sources, this term, once created and defined, must be explicitly associated with these “See also” information sources as utilizing them.

8 Structuring the glossary

Having a glossary in the form of a list of glossary terms together with their definitions is often insufficient for one or both of the following reasons:

- Large flat lists of glossary terms are hard to navigate and search.
- The same word or phrase may have different meanings in different contexts.

Structuring the glossary solves both problems.

8.1 Creating the domain structure

The structure that can be imposed upon the glossary is a hierarchy of chapters, each containing some number of terms.

While chapters can be created according to any criteria, we strongly recommend that the structure of glossary chapters and their subchapters shall reflect the structure of real-world domains and their sub-domains (also called subject areas).

In all cases, each chapter shall have a description, which gives a short insight into what criteria shall be used to determine which terms belong to this chapter.

8.2 Placing terms in context

Once the glossary terms are identified and described and the structure of glossary chapters is established, glossary terms must be assigned to proper chapters.

Usually, knowing the definition of a glossary term and the description of a glossary chapter is enough to decide whether the term belongs to this chapter or not. However, various special cases can arise, such as:

- Several glossary terms with the same name but different definitions (most probably, one term was given different definitions by different sources) end up in the same glossary chapter. We let the matter be for now, but these terms will have to be refined (resolved) during glossary normalization.
- A term may appear to belong to several different chapters. This usually happens when a term is used in the real world in more than one domain. EasyRM allows placing a term into more than one glossary chapter.
- A term may appear to belong to all sub-chapters of some chapter. In this case, the term shall be explicitly moved into the super-chapter. This relies upon the fact that, if a term has the same meaning in all sub-domains of a given domain, it can be considered a domain-wide term, implicitly inherited by sub-domains.

9 Normalizing the glossary

After structuring the glossary, the next step is to normalize it. This step ensures that:

- Terms are given proper and complete definitions.
- Relationships of terms to each other are specified.
- Possible inconsistencies and redundancies are identified.

9.1 Explaining the terms

This step ensures that term definitions are consistent and complete.

To do this, we employ much the same procedure as we did when extracting glossary terms from requirements – we read through definitions of glossary terms, finding out:

- Which other terms are used in the definition of a given term.
- Are these other terms used properly (i.e. do their meanings really correspond to the context in which these terms appear).
- Are there unknown or unclear words or expressions in the definition of a term? These may need to be introduced as additional glossary terms and further explained, or a definition of the original term may need to be reformulated in clearer language.

It may be that, in the definition of some glossary term, some other term is used improperly (i.e. out of context). In this case the original term may need to be reformulated to avoid erroneous use of the referred term.

When we are satisfied with a definition of a given term (i.e. we have checked all other terms mentioned in this definition and they really make sense in this context), we must explicitly mark these other terms mentioned in the definition of the original term

as its dependencies. This conveys to the subsequent reader the fact that the term definition has been examined and found to be proper in its use of the language defined in the glossary.

9.2 Discovering semantic dependencies

We may have already identified some dependencies between glossary terms during the previous step. During this step, we look at the set of glossary terms in order to decide whether some of them represent generalizations or specializations of others.

The typical case when it is so is when there is a term that introduces a general concept and some other term that describes a specific version of the general concept. Usually, when a term has specialization [in form of some other term] the special term is located in the same glossary chapter as the general one, or in some sub-chapter thereof.

Note, that these specializations are actually a form of dependency, since, in order to understand a specific concept, reader must understand the more general concept first.

9.3 Discovering synonyms

Synonyms are terms that have the same meaning in the context of the project. From this definition it is clear that the fact that two terms are synonyms is project-specific, not term specific, and the same two terms may be synonyms in the scope of one project but quite different in the scope of another project.

All glossary terms shall be examined in order to find out which can be considered synonyms for the project at hand. The terms found to actually be synonyms shall be explicitly marked as such. Later on, when designing and implementing business concepts, this information will be used to ensure that each business concept is designed and implemented only once, regardless of how many disguises it comes under.

9.4 Resolving conflicting terms

Here we find out if there are glossary chapters that contain several glossary terms with the same name but different definitions.

This situation usually arises when several different information sources attempt to provide definitions for the same term. Then, one of the two situations can arise:

- If these definitions were found [during the previous step] to be synonymous, they shall be merged into one term (i.e. all but one of these terms with the same name and different but synonymous definitions shall be deleted from the glossary, and all associations these deleted terms has to other terms, requirements and/or documents shall be established for the remaining term)
- If these definitions were found to actually mean different things, additional consultations must be held with customers and/or domain experts in order to find which, if any, of these different definitions of a term is actually applicable within the project. Once this conflict is resolved, only the agreed upon correct definition will remain; all other terms with the same name and different definitions will be removed from the glossary.

10 Synchronization of requirement specifications with glossary

At this point, the language in which to talk about the project (and, specifically, about its requirements) has been consolidated (in the form of the project's glossary). Now it is time to ensure that requirements are actually formulated using this language.

10.1 Validating the use of terms in requirement specifications

During this step we read through specifications (descriptions) of all requirements in the requirement model, looking at what glossary terms are actually used in each such description (EasyRM makes that easy by actually highlighting all terms used in a specification of a requirement). For each glossary term encountered in a specification of a requirement a decision must be made about:

- Whether this term shall be actually used in the specification of this requirement.
- Does the definition of the glossary term actually agree with the context in which the term is being used (such context being, of course, a specification of a requirement).

If the answer to both questions is "yes", the glossary term is used properly and intentionally in the specification of a requirement. In this case, an explicit association between the term and requirement must be created, signifying the fact.

10.2 Resolving invalid terms usage

It may also be the case that the term does not really belong in the specification of the requirement. In this case, the use of term in the specification of the requirement is accidental and erroneous, and shall be resolved.

Typical ways to resolve the erroneous use of a glossary term in a specification of requirement include:

- Reformulating the requirement so that it does not use the term.
- Renaming the glossary term, so that an occurrence of its old name in the specification of a requirement does not actually constitute a usage of the term.
- Rewriting the definition of a term, so that the term usage in the specification of a requirement becomes valid.

It shall be noted, though, that the two last approaches are potentially dangerous and shall be used rarely and with great care, since they may cause the project's glossary (and, hence, *the language which we use to speak about the entire project, not only its requirements*) to change. Rephrasing requirements, on the other hand, usually has only limited and local effects on the project as a whole.

11 Creating secondary classifications of requirements

So far, we have only one classification of requirements, established in the standard requirements template. This main classification separated static from dynamic requirements and business from technical requirements.

What secondary classifications shall be created? There is no single “right” answer to this question. Classifications of requirements are, put simply, different ways to group requirements to help in managing them. Classifications of requirements are created to help the project team, not the customer.

Still, there are some standard situations when specific secondary classifications of requirements are useful. Several of these situations are described below.

11.1 Secondary classification for incremental development

If a product is scheduled for incremental development, it is extremely useful to classify the requirements by the increment in which they will be implemented.

For this secondary classification of requirements, a new requirement classification profile named “By increment” (or similar) shall be created, with one subclassifier per each planned increment (these subclassifiers can be named simple “Increment 1”, “Increment 2”, etc.). Each requirement shall, then, be associated with the increment during which it will be designed and implemented.

The secondary requirement classification by product increment covers the entire requirement set, since all requirements will be implemented in some increment.

11.2 Secondary classification for component-based development

If a decision is made to build a component-based product, it is extremely useful to classify the requirements by the component that will be implementing them.

For this secondary classification of requirements, a new requirement classification profile named “By component” (or similar) shall be created, with further requirement subclasses corresponding to individual components. A more complex component-based architecture (i.e. components which have subcomponents, etc.) may be directly represented by means of further subclassification. Each requirement shall, then, be associated with the component where it will be designed and implemented.

The secondary requirement classification by product components covers the entire requirement set, since all requirements will be implemented by some component.

11.3 Secondary classification for parallel development

If a decision is made to build a product by several teams working in parallel, it is extremely useful to classify the requirements by the team that will be implementing them.

For this secondary classification of requirements, a new requirement classification profile named “By team” (or “By responsibility” or similar) shall be created, with further requirement subclasses corresponding to individual development teams. Each requirement shall, then, be associated with the team that will be designing and implementing it.

The secondary requirement classification by development team covers the entire requirement set, since all requirements will be implemented by some team.

12 Creation of requirement baseline

A requirement baseline is an end product of the requirement analysis. It contains consistent set of system requirements, sufficiently complete to be used as a basis for further development activities.

12.1 What to baseline

It is intuitively clear that the requirements baseline must contain requirements – after all these will be used as a basic for further development.

However, requirements are not enough – designers and/or implementers will need additional information in order to understand these requirements. This means that the requirements baseline must, beside requirements, contain glossary terms used in specifications of these requirements and the relevant reference documentation.

In addition, there are other important considerations, such as:

- A requirements baseline shall be consistent, i.e. it shall not contain any conflicting requirements.
- A requirements baseline shall be simple. That means that equivalent requirements shall not be included into the baseline (i.e. when baselining, exactly one of equivalent requirements shall be selected for inclusion), that documents and terms not relevant *for these requirements included into the baseline* shall not be in the baseline, etc.
- The use of synonym glossary terms in a baseline, although permitted, increases the demands on the designer.
- A requirements baseline shall be complete, i.e. it shall contain sufficient information to perform design and implementation of the software it describes. This is the most difficult of all baselining goals to achieve, since no formal or automated assistance can be offered in any form to ensure completeness.
- A requirements baseline shall be non-modifiable, thereby providing a clear non-changing goal for further development work.

In short, the rule of thumb is that, for a given requirement model that went through all the requirement analysis stages described in this document, any consistent and complete subset (including the entire requirement model, if it is itself consistent and complete) may be chosen as a baseline. The list of consistency criteria for requirement models is given in an Appendix B to this document.

12.2 When to baseline

That largely depends on the nature of the product being developed and on the development mode selected. Some typical guidelines on when to baseline requirements are:

- When a product is scheduled for incremental development, it is possible to baseline requirements for an increment once all requirements related to a specific increment *and all earlier increments* have been analysed and stabilized. This approach means that requirements for an incrementally built product can be analysed in incremental groups instead of all together. However, when performing requirement analysis for incremental groups of

requirements, it is possible that analysing requirements for the next increment may affect requirements and/or terms used in earlier increments. The suggestion is, therefore, to perform initial processing of all requirements for an incremental product together, up to and including the glossary creation and normalization, and then proceed with synchronizing, reclassifying and baselining of requirements in incremental blocks, one per product increment.

- When a product is scheduled for the component-based development, it is possible to baseline requirements for a component once all requirements related to some component *and all other component upon which this component depends* have been analysed and stabilized. Just as in the case of incremental development, it makes sense to perform initial processing of all requirements, set up and normalize a glossary common for all components, and then proceed with synchronizing, reclassifying and baselining of requirements in incremental blocks, one per component.
- When a product is scheduled for parallel development by several teams, it is possible to baseline requirements for a specific team once all requirements to be implemented by this team *and all their dependencies* have been analysed and stabilized. Again, the initial processing of all requirements shall be performed, a project-wide glossary shall be set up and normalized, and then requirements for a given development team can be synchronized, reclassified as necessary, baselined and given to the team to proceed with.
- When a customer explicitly asks for a "quick-and-dirty" prototype based on some specific subset of requirements (usually, it is possible to agree with the customer which requirements out of the complete project requirement set shall be included into the prototype). In this case a new requirement model must be created, containing these prototyped requirements *plus all related information but nothing extra*. This, essentially, is the requirement model for the prototype project, and it is usually much smaller than the main project requirement set. The prototype requirement model must then be taken through all usual requirement analysis steps, and, once these steps are completed, the prototype requirement model must be converted into a baseline by applying consistency criteria to it.

12.3 How to baseline

Creating a requirements baseline consists of several steps:

- Deciding which requirements to include into the requirements baseline.
- Making sure that all requirements chosen for the inclusion into the requirements baseline have passed through all mandatory requirement analysis steps (as specified in this document).
- Creating a new requirements model that will represent the requirements baseline.
- Copying to this model the following artefacts from the main project requirements model:
    - All requirements that have been selected for baselining and all their direct or indirect dependencies, including their classification.
    - All glossary terms used in specifications of requirements included into the requirements baseline and all their dependencies, including the glossary chapters these terms belong to.

    - All documents related to requirements and/or glossary terms included into the requirements baseline, including the library folders these documents reside in.
- The resulting requirement model (which is, by now, a candidate baseline) must be checked against all requirement model consistency criteria. If some consistency criteria is violated, the main project requirement model must be updated to correct that violation and the candidate baseline must then be re-created (and then re-checked against all requirement model consistency criteria, this sequence of steps repeating until a candidate baseline becomes consistent).
- The consistent candidate baseline must be checked for completeness. If it is decided that the consistent candidate requirements baseline is incomplete, the choice of what requirements to include into the requirements baseline must be augmented, and the baseline creation must re-commence normally.

It is recommended that requirements baselines be given consistent names. The following naming scheme is proposed:

- A project requirement set is named "<project name>_<project version>", where <project name> is a short [internal] identifier of the project and <project version> is a project version number.
- A requirement set for a sub-project of a project (e.g. a requirement set for a single component of a component-based project, etc.) is named "<project name>_<project version>_<subproject name>", where <project name> and <project version> are used as specified above, and <subproject name> is a short [internal] identifier of the sub-project (or component) within a project.
- A requirements baseline is always created from some live requirement set. The requirement baseline shall be named after the original [live] requirement set, with the suffix "_baseline_<baseline identifier>" attached. The <baseline identifier> may be chosen arbitrarily; we recommend using either sequential numbering of baselines or ISO8601 date strings as baseline identifiers.

Appendix A: Standard requirement model template

The ZIP archive embedded below contains the EasyRM database with a standard template for requirement modelling.


Template.zip

Appendix B: Requirement model consistency criteria

1. All elements of the requirement model must have mandatory names and descriptions. This covers documents, library folders, glossary terms and chapters, requirements and requirement classifiers.
2. All glossary terms and requirements must be traceable to the sources (documents) where their definitions come from. A requirement or glossary term not traceable to any origin is an inconsistency.
3. Documents that do not trace to any glossary terms and/or requirements (as either originating or utilized documents) shall not exist in a consistent requirement model.
4. All remote documents must be accessible (i.e. user shall be able to open them from EasyRM Document Manager by double-click).
5. All glossary terms used in the description of requirement (i.e. underlined in EasyRM Requirement Manager tool) must be explicitly associated with this requirement as intentionally used terms.
6. All glossary terms explicitly associated with a requirement as intentionally used terms must actually appear in the description of this requirement.
7. All glossary terms used in the definition of a glossary term (i.e. underlined in EasyRM Glossary tool) must either be explicitly associated with this term as its dependencies or be its direct or indirect generalizations.
8. Glossary terms that are not used in (i.e. mentioned in the text of) the description of some requirement(s) and/or other term(s) shall not exist in a consistent requirement model.
9. Requirements explicitly marked as mutually exclusive cannot have any other associations between them.
10. Each atomic requirement must be classified exactly once in each existing classification profile.
11. A requirement classifier without any requirements associated with it shall not exist in a consistent requirement model.
12. A requirement cannot be equivalent to its direct or indirect decomposition.
13. A requirement cannot be equivalent to its direct or indirect dependency.
14. A glossary chapter cannot contain two or more terms with the same name.
15. A glossary chapter without any terms in it shall not exist in a consistent requirement model.
16. A glossary term cannot be a synonym of its direct or indirect specialization.
17. Requirements baselines shall not contain equivalent or exclusive requirements.

Appendix C: A case study

As an example, we took a simple case study through the requirement analysis process.

The project around which the case study is built is to add an extra component to the EasyRM suite. This extra component shall check the requirements model currently open in EasyRM for consistency using a number of criteria and produce a consistency report, which will indicate all places where the requirement model needs further work.

We took this project through all requirement analysis steps indicated in this document. The results of each step are included below.

Step	Result
Preparation for requirement gathering	 Example1.zip
Gathering raw requirements	 Example2.zip
Initial requirement classification	 Example3.zip
Requirement decomposition	 Example4.zip
Discovering requirement associations	 Example5.zip
Discovering glossary terms	 Example6.zip
Structuring the glossary	 Example7.zip
Normalizing the glossary	 Example8.zip
Synchronizing requirement specifications with glossary	 Example9.zip
Creating secondary classifications of requirements	 example10.zip
Creation of requirements baseline	The same as above

To view the results of a specific step, follow the following procedure:

1. Create the empty directory "`C:\RequirementAnalysisCaseStudy`".

2. UnZIP the included archive with the results of the requirement analysis step you'd like to view into "`C:\RequirementAnalysisCaseStudy`".
3. Open the "`.srm`" file in "`C:\RequirementAnalysisCaseStudy`" directory with EasyRM.