Requirement modeling: The process

P-RMP-2004-05-1-0

In the scope of systems and software engineering, requirement modeling is increasingly recognized as a separate activity. Its importance grows with the size and complexity of the intended system.

To carry out requirement modeling, a number of different approaches have been developed, many of which are supported by dedicated CASE tools (to name but a few, Caliber RM, Rational Requisite Pro, Catalyze, etc.). This article outlines the requirement modeling approach based on experience accumulated by Cybernetic Intelligence GmbH in a number of medium-to-large scale software projects, as well as on the results of the internal research into the requirement modeling. We have also created a dedicated requirement management tool – EasyRM (see http://www.easy-rm.com), which is specifically suited to our requirement modeling approach.

Requirement modeling vs. requirement analysis

The term "requirement analysis" is often used in the systems and software engineering to refer to a process of building up a specification of the system to be built.

Typically, this term refers to a process of building such specification of the system that:

1. Satisfies the customer's demands with respect to the system in question.
2. Provides sufficient information to build the system.

The requirement analysis often employs formal or semi-formal modeling techniques and notations, such as business process modeling, use case modeling, class or data modeling, etc. This, generally, means that the results of requirement analysis are not directly presentable to a customer, who is usually an expert in his domain area but has little understanding of these notations.

The term "requirement modeling" is somewhat similar – it refers to a process of building up a specification of the system which has the same two properties as listed above, but, in addition, is centered about the third, most important property:

3. The resulting specification must be understood in the same manner by both customer (who wants the system to be built and pays for the development) and developer (who is responsible for actually building it).

The approach described here makes a clear distinction between requirement modeling and requirement analysis:

- Requirement modeling results in a requirement specification, the main purpose of which is to allow the customer and the developer to agree on what is being developed.
- The main goal of analysis (specifically, requirement analysis) is to provide a formal or semi-formal description of the system being built (in general, the main goal of analysis is to provide a formal or semi-formal description of the problem to be solved, while design provides a formal or semi-formal description of the intended solution).

Requirement modeling overview

The general process flow of the requirement modeling is shown on the Figure 1. This represents the general case; depending on the specifics of a system being developed some activities may become trivial or may be bypassed altogether.

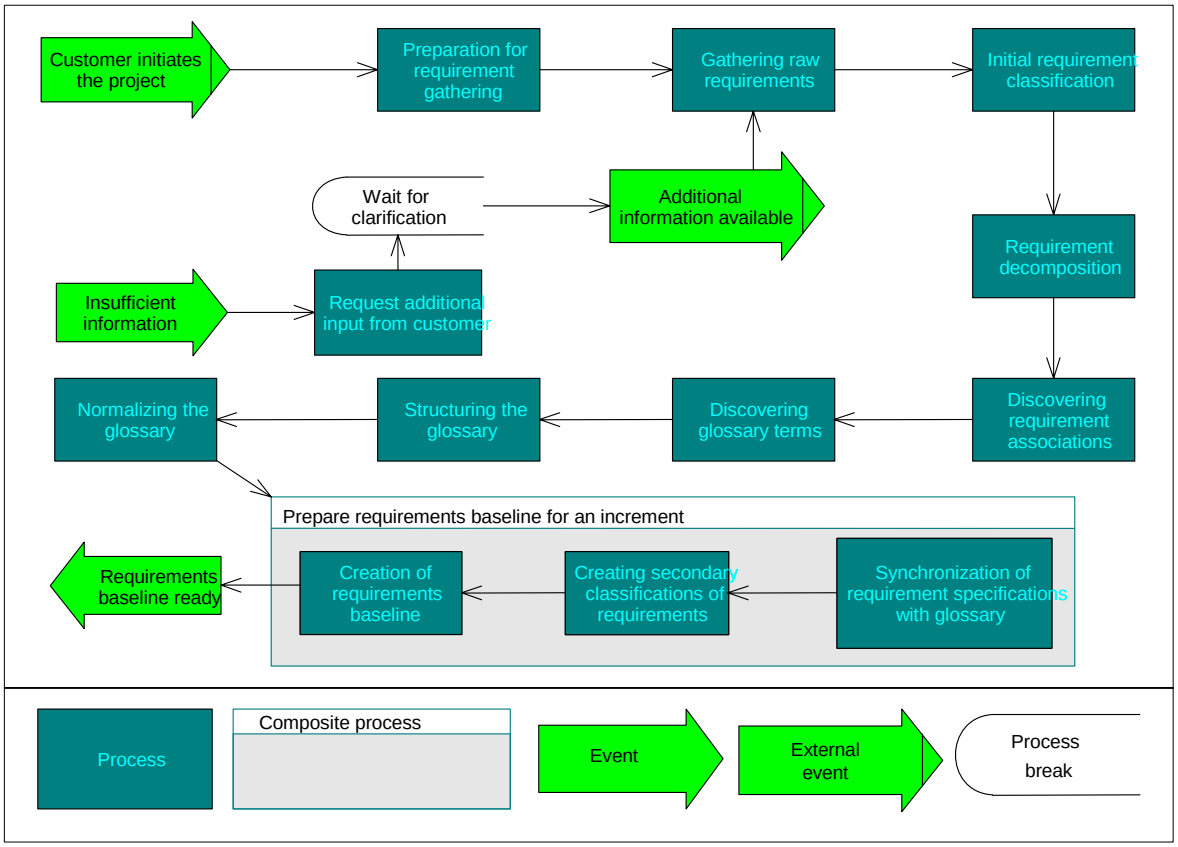

Figure 1: The requirement modeling process

Depending on the situation, the requirement modeling approach described here has been successfully used for “straightforward” software engineering, software reverse engineering, change request management, and many other tasks.

Individual elements of the requirement modeling process are examined in more detail below.

Participants

In the scope of our approach, we distinguish several different parties (also known as actors or roles) involved into the requirement modeling:

1. Customer – wants the system in question to be developed and pays for the development.
2. Developer – a person or a team who will do the actual development and deliver the system to the customer.
3. Requirement modeler – a person or a team who facilitates the communication between the customer and the developer. An important part of this communication is constituted by requirement specification.

Of course, in real projects the same person can play more than one of the above roles. This, however, does not affect the general requirement modeling process.

Initiating requirement modeling

The process of requirement gathering is always initiated by the customer. The customer's decision "it's time to start building a solution" is usually accompanied by some initial information about what the customer wants. Of course, this information rarely provides enough to build the system in question. Usually the customer will later volunteer more demands (or change existing ones) and provide clarifications to the gray areas when a developer needs it.

The preparation for requirement gathering largely depends on what tools are being used for requirement modeling. The most important part of this preparation is capturing and categorizing all information provided by the customer in its raw form - this is what requirement modeling will work with.

In the scope of our approach we use the term *project document library* to refer to a collection of raw information provided by the customer for the specific project. The individual documents in this library are project visions, relevant standards, customer interview logs, etc. The main idea here is that a project document library will contain all raw information relevant for the project and nothing else.

Gathering raw requirements

A single requirement is, basically, a constraint on the system being developed. Each of these constraints typically addresses a single specific aspect of the system and can be positive ("the system must do X"), negative ("the system must not do X") or some shade in between ("It would be nice for the system to do X, but we can live without it just as well").

The process of gathering raw requirements basically involves going through the project document library, finding such constraints and creating a separate requirement for each constraint found. At this point we don't care about things like classifying these requirements or whether they conflict with each other. All we need to do is to collect the requirements and tag each one with the importance / urgency (Must, Should, Could, Must not, etc.).

As the requirement modeling diagram (Figure 1) suggests, the process of gathering raw requirements is not usually performed in one go. Each time customer provides more information (either voluntary or answering a request from a requirement modeler or developer) this new information must be scanned for additional requirements.

Initial classification of requirements

Once requirements are identified, it's a good idea to classify them. This breaks up a large set of requirements into logically related subsets (requirement classes), which greatly simplifies working with requirements. Note, that requirement classification is usually not visible to the customer, but is introduced to make life easier for the developer.

How to classify requirement is largely project- and developer- specific. The traditional separation of requirements into functional and non-functional is well known and can be further refined if necessary. For example, non-functional requirements may be further subdivided into requirements dealing with performance, requirements dealing with security, etc. It is also often

beneficial to have several independent classifications of the same set of requirements, thus providing different views of the same requirement set for different purposes.

In our work, we often prefer to have requirements classified into “to do” (dynamic, describing what the system being developed must or must not do) and “to be” (static, describing what various elements within the system are) classes. We find that static requirements (such as "in our bank, the account number is a 12-digit number") nearly always make their way into the glossary.

Requirement decomposition

Requirement decomposition establishes the parent-child relationship between requirements. It handles two quite different situations:

1. Sometimes, the requirements as formulated by customer deal with more than one aspect of the system. It's usually a good idea to break such *composite requirements* into simpler ones, which, when taken together, describe the same constraint on the system as the original requirement. After breaking up, the original requirement is no longer necessary. We refer to this process as *split decomposition*.
2. Quite frequently, one requirement is a clarification of another. For example, a general requirement "the system should be fast enough to operate in real time" will be further clarified by requirements specifying the required response time, processing time, throughput, etc. In this case the general requirement is made a "parent" of the corresponding specific requirements. This is known as *requirement qualification*.

In some cases, it is possible that a child requirement participates in the split decomposition or qualification of more than one parent requirement. This most often happens when a requirement modeling is underway, but may also make it into the requirement baseline.

The Figure 2 illustrates both cases of the requirement decomposition.

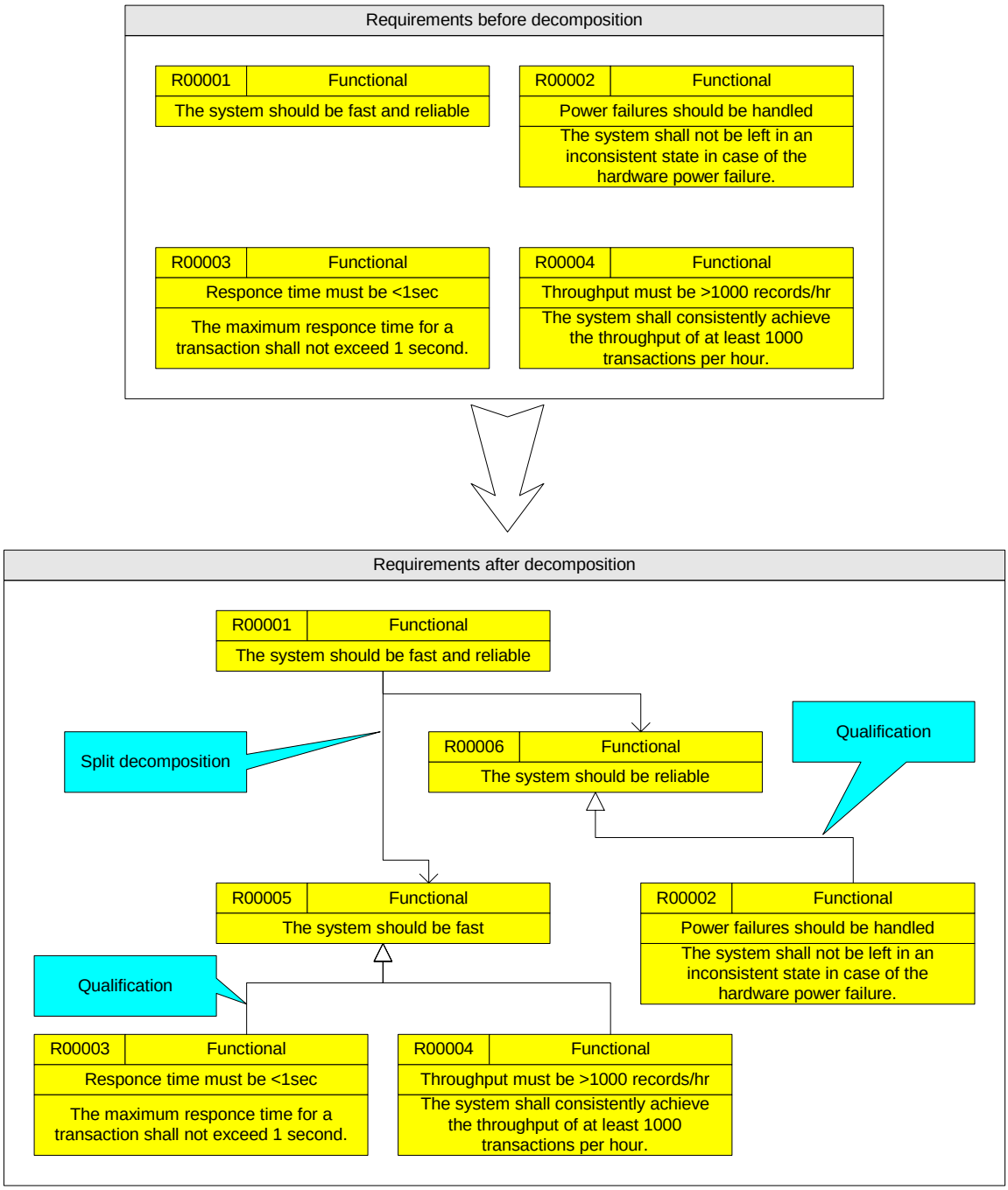

Figure 2: Requirement decomposition

Discovering requirement associations

Requirements are not independent of each other. One example of the association between requirements is the requirement qualification described above, where the parent requirement provides the general context and its child requirements address more specific issues within that context.

In general, it is up to the requirement modeler to decide what associations between requirements are relevant and shall be tracked. Specific examples of such associations include:

- Requirement dependency: if a requirement A depends upon the requirement B, then it is impossible to implement A without implementing B first. The information about requirement dependency is important for both change impact analysis and deciding what requirements to include in a specific requirement baseline.
- Requirement conflict: it is not uncommon for the customer to make contradictory demands, especially when these demands come from more than one source. When these demands become requirements, it is essential that the information about their incompatibility be captured. Conflicting requirements can never be implemented in the same system and, as a result, always require clarification from the customer.
- Requirement equivalence: two equivalent requirements describe the same constraint on the system in different terms. Clearly, only one of these should be retained so, again, a clarification from the customer is needed.
- Requirement correlation: it is often the case that a number of requirements must be treated as a group, i.e. if one of these requirements makes it into the release, then the entire group must do the same. For example, when a composite requirement is split into simple requirements, the resulting simple requirements are correlated, because, to the customer, they are still one requirement (the original composite one; decomposition is just a way to make life easier for the developer).

The project glossary

At this point the project requirements have been identified, classified and relevant information about their associations have been captured. It is now time to build a project glossary.

A *project glossary* specifies the common language both the customer a developer shall use to speak about the system. Traditionally, a project glossary is a list of terms and phrases which have a specific meaning within the scope of the project, including:

- Terms and phrases which have several different meanings in the natural language in which requirements are written.
- Terms and phrases which, within the scope of the project, have a meaning different from that normally assumed.
- Terms and phrases specific for the customer's industrial domain or enterprise.
- Acronyms and abbreviations.

The terms of a project glossary normally represent important concepts in the scope of the project. Most of these concepts will later make it into the analysis and design models.

The detailed description of the glossary modeling is in itself a large subject. Glossary modeling is described in detail in the next article, "Requirement modeling: Advanced issues". In here, it is sufficient to say that the project glossary, when complete, contains:

- All terms and phrases relevant for the project, all in their proper contexts.
- A unique unambiguous definition for each glossary term.
- All relevant relationships between glossary terms, such as equivalence, dependency, etc.

Synchronization of requirements with the glossary

By the time this activity starts, the common language both the customer a developer shall use to speak about the system has been consolidated in the form of a project glossary. Synchronization of requirements with the project glossary ensures that the project requirements are actually written using this language.

To synchronize the requirements with the glossary, all occurrences of glossary terms within requirements must be analyzed. Specifically, it shall be verified that the definition of a glossary term makes sense in the scope of all requirements where this term is used. If this is not the case for some requirement, a conflict has been detected, which may be resolved in different ways:

- A requirement in question may need to be reformulated to avoid the usage of glossary terms that do not really belong there.
- A glossary term in question may need to be renamed.
- A definition of the glossary term in question may need to be changed so that the usage of the term in the 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 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.

Creating secondary requirement classifications

For the convenience of the developer, additional classifications of the requirement set can be established if necessary. Examples of such secondary requirement classifications include:

- Classification by increment: for a system developed incrementally, it is often convenient to classify requirements by the increment in which they will be implemented.
- Classification by components: for a component-bases software system, it is extremely useful to classify requirements by the component to which they apply. A more complex component structure (i.e. components which have their own subcomponents) may be established if necessary.
- Classification by responsibility: if a system is developed by a team or several teams, it is convenient to classify requirements by the person or team responsible for their implementation.

Creating requirement baselines

A requirement baseline is a set of requirements, together with the relevant glossary and reference documentation, which can be approved by customer and used as a basis for further development steps, resource estimations, acceptance tests, etc.

There are important considerations for the contents of a requirement baseline, such as:

- A requirement baseline shall include requirements for one or more complete useful business function(s), which customer can employ.

- A requirements baseline shall be consistent, i.e. it shall not contain any conflicting requirements, a group of correlated requirements cannot be included into the baseline only partially, etc.
- A requirements baseline shall be simple. That means that equivalent requirements shall not be included into the baseline (i.e. when creating a baseline, 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.
- 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 modeling stages, any consistent and complete subset (including the entire requirement model, if it is itself consistent and complete) may be chosen as a baseline.

The decision on when to create a requirement baseline 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 analyzed and stabilized. This approach means that requirements for an incrementally built product can be analyzed in incremental groups instead of all together. However, when performing requirement analysis for incremental groups of requirements, it is possible that analyzing 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 analyzed 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 analyzed 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), 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 modeling steps, and, once these steps are completed, the prototype requirement model is converted into a baseline.

Conclusion

The requirement modeling approach described here has, as mentioned already, been successfully used in a number of medium-to-large software projects, mainly in the financial / banking domain. While, certainly, not being the final word in the requirement modeling, this approach addresses many issues not found in other requirement modeling methods. We at Cybernetic Intelligence GmbH are currently continuing our research into software development process in general and requirement modeling in particular.

The next article, "Requirement modeling: Advanced issues" will deal with advanced techniques employed in the process of requirement modeling. Specifically, it will discuss:

- Glossary modeling.
- Glossary and requirement sharing.
- Probabilistic requirement modeling.