T. Gorschek and C. Wohlin, "Requirements Abstraction Model", Requirements Engineering Journal, Vol. 11, No. 1, pp. 79-101, 2006.

Requirements Abstraction Model

Tony Gorschek
tony.gorschek@bth.se

Claes Wohlin
claes.wohlin@bth.se

School of Engineering
Blekinge Institute of Technology
PO Box 520, S-372 25 Ronneby, Sweden
Phone: +46 457 385817
Fax: +46 457 27125

Abstract

Software requirements arrive in different shapes and forms to development organizations. This is particularly the case in market-driven requirements engineering, where the requirements are on products rather than directed towards projects. This result in challenges related to making different requirements comparable. In particular, this situation was identified in a collaborative effort between academia and industry. A model, with four abstraction levels, was developed as a response to the industrial need. The model allows for placement of requirements on different levels, and it supports abstraction or break down of requirements to make them comparable to each other. The model was successfully validated in several steps at a company. The results from the industrial validation point to the usefulness of the model. The model will allow companies to ensure comparability between requirements, and hence it generates important input to activities such as prioritization and packaging of requirements before launching a development project.

Key Words

Market Driven Product Centered Continuous Requirements Engineering, Requirements Abstraction, Product Management, Product Strategy

1. Introduction

Market driven incremental product development and delivery (release) is becoming increasingly commonplace in software industry [1, 2]. Incremental product development is planned and executed with the goal of delivering an optimal subset of requirements in a certain release (version of a product that is distributed to customers) [3]. The idea is to select *what* a release should contain (requirements), *when* it should be released (time), and at what *cost* (pertaining to the resources needed designing and implementing a requirement) this should be achieved. The decision about which customers get what features and quality at what point in time has to be taken, i.e. making these activities a major determinant of the success of a product [4].

All activities described above, i.e. establishing *what* should be included in a release, *when* it should be released and at what *cost*, are vitally dependent on the product requirements and that they are elicited/caught, analyzed, and specified before any planning and development activity can commence.

The situation is far more complex than the one presented by the classical bespoke [3] development situation where elicitation could be targeted and concentrated mainly to the customer organization and stakeholders identified there. The development activity was initiated by e.g. an order, which generally resulted in a project (maybe preceded by a pre-study) and then requirements engineering was initiated through this project. As illustrated in Figure 1, the situation in a market driven development situation is different since the requirements flow is not limited to a development instance (e.g. a project), but rather continuous in nature, and the requirements themselves should act as the catalyst for initiating development.

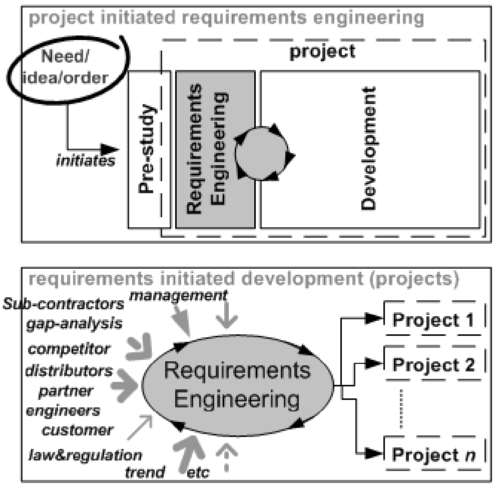


Figure 1. Requirement’s role in the development process.

In market driven development requirements are generally generated by multiple sources, both internal (e.g. engineers to management) and external (e.g. customers and partners). It can be everything from direct requests for added functionality from existing and/or potential customers to updates proposed by engineers working on the product.

In addition, *indirect requirements* need to be caught. This can be everything from idea-like requirements caught by the marketing department during a competitor analysis or a market survey, to information gathered during product support and conveyed internally.

As the sources of the requirements vary and the requirements themselves are both direct and indirect in nature it is not surprising that they come in different shapes and forms, at multiple levels of abstraction, and described on varying levels of refinement.

In Figure 1 this is depicted as *requirements initiated development*, where all these types of requirements are input to a requirements engineering process which has the capability of performing the needed refinement, analysis and negotiation, producing good-enough requirements to initiate and drive development activities (e.g. projects) in an continuous manner, i.e. development is initiated when the requirements warrant it.

The market driven product development situation described above was identified during a cooperative software process improvement (SPI) venture in industry, performed at Danaher Motion Särö AB (DHR). There was a need for adapting requirements engineering at DHR to a continuous process, moving from traditional *project initiated requirements engineering* to *requirements initiated development*. This involved not only creating a new way of working (e.g. how to specify requirements, what roles and responsibilities should be present etc), but also a new way of thinking (e.g. that requirements are the basis for product development).

The Product Managers (the ones charged with implementing the new way of working) were faced with the challenge of how to take care of the continuous incoming stream of requirements ranging from abstract to technically detailed. Based on this problem the *Requirements Abstraction Model* was developed. This paper presents the *Requirements Abstraction Model (RAM)*, how it was created in close cooperation with industry, and validated through feedback from professionals as well as how it was tested in a live project.

Although, the model was developed based on needs identified at DHR, the objective was for the model to be generally usable, i.e. the aim of RAM was to give professionals working with product planning/development (e.g. Product Managers) a requirements engineering model that help them in their work. To this end RAM is modeled towards a product perspective, supporting a continuous requirement engineering effort, aimed at taking requirements of multiple types (abstraction level) as input, and offer a structure for the *work-up* of these requirements, i.e. breaking down abstract requirements into detailed ones, and vice-versa (see Section 4.3).

The benefits of using RAM as a support in product centered continuous requirements engineering can be summarized in four bullets.

(I) All requirements are compared to the product strategies, offering an assurance that requirements do not violate the overall goals set by management. This offers the possibility to dismiss requirements early in the process, freeing up resources to work on/refine relevant requirements that are in line with the product strategies.

(II) All requirements are broken down to an abstraction level where they are good-enough for initiating a development effort (project). This assures that the projects (whose aim it is to realize the requirements) get good-enough requirements to base their development efforts on (e.g. testable and unambiguous).

(III) Work-up of requirements means that they are formulated on the *same* level of abstraction, and hence they can be compared and set against one another. The ability to compare requirements is a prerequisite to effective release planning and prioritization.

(IV) All requirements can be followed through several levels of abstraction giving a richer understanding of each requirement, and thus better decision support can be obtained for all professionals, from management to developers.

The paper is outlined as follows. Section 2 offers an overview of related work. In Section 3 the background and motivation is presented, along with how it relates to both DHR and the general industry case. Section 4 offers an introduction and exemplification of the Requirements Abstraction Model (RAM), and Section 5 presents how RAM was evaluated through static and dynamic validation. Section 6 presents the Conclusions.

2. Related Work

The idea of market driven incremental development is not new. An example of this is the IBM REQUEST technique presented in 1992 by Yeh in [5]. Here arguments are put forward for the necessity of market driven (i.e. listening to the customer) development and having mechanisms (prioritization) in place to select what requirements to develop. Recent research focuses on release planning from a perspective of what requirements to put forward taking issues of e.g. interdependencies between the requirements into account [4, 6, 7] when planning a release, as well as priority from the customer perspective [8]. Dependencies, allocation to a certain release, risk, effort and priority are all factors addressed by the *Evolve method* presented by Greer and Ruhe in [1, 9] .

In the same manner the realization that requirements are often on different levels of abstraction and in varying stages of refinement has been recognized in both industry and research [10, 11].

This is often used as a way of working in industry by having different requirements documents for different reasons, e.g. one aimed for market/management were requirements are abstract and are closer to visions than actual requirements (Market Requirements Specification - MRS). The MRS is later refined to product requirements (Product Requirements Specification - PRS) by engineers and/or managers that interpret the MRS to form actual requirements (that should be testable and unambiguous, although it may not always the case). The next step is to refine and add to the requirements by adding technical details and producing Technical Requirements Specifications (TRS) based on the PRS. This offers a refinement of requirements from vision to technical requirements, almost design, through a document oriented perspective where different people work on different documents interpreting requirement statements along the way. The Product Manager's role may span from actively participating in the work with high level (MRS), to the next stage lower level (PRS) requirements, and planning for what requirements should be allocated to what projects. Exactly what requirements are used as input to projects varies, as does the abstraction level within most documents (MRS, PRS, and TRS).

Requirements on different levels of abstraction and at varying levels of refinement are considered by some as crucial input to the development in order to get a better understanding of what should be developed and why [10]. The basic notion is that both the abstract (long-term overview) and the detailed (giving context and the short term-view) is important [12, 13], and the two used in combinations offers a better understanding.

The field of goal based requirements engineering (see e.g. [14-16]) focuses on the elicitation of goals that are to become requirements at some point, working from the top down.

Wiegers [17] and Lauesen [18] describe that requirements can be of different types pertaining to what they should be used for, not totally unlike the industry view of dividing requirements of different types into documents, from goal (abstract natural language formulations [19]) to technical design like specifications. The focus is on that requirements be specified on different abstraction levels depending on usage, e.g. project type.

The contribution of the Requirements Abstraction Model is set around taking advantage of the fact that continuous requirements engineering means that requirements are caught/elicited on different abstraction levels. Abstraction levels subsequently are used as a "motor" to facilitate work-up (not flattening, i.e. forcing all requirements to one level of abstraction) of requirements. Work-up is accomplished by breaking down abstract requirements into detailed ones, and vice-versa (see Section 4.3).

This work-up facilitates initial analysis and refinement of requirements to the degree of producing good-enough requirements for project initiation, as well as explicitly linking all requirements to product strategies as a means to offer decision support for e.g. management.

RAM does not assume a starting point (e.g. starting with goals), but rather takes what requirement is available as input and uses it as a base. Furthermore there is no choice of *one* abstraction level, i.e. flattening, all requirements depending on project type since the continuous requirements engineering is not project initiated rather product oriented in nature. In a product development situation there is a need for decision support on multiple levels before any release planning and development activity can be undertaken (e.g. in project form). Through work with RAM requirements on a high level of abstraction (comparable to product strategy), and requirements on a low level of abstraction (good-enough as input to a project) are available. In addition, as several levels of abstraction are offered, a richer understanding can be obtained as to the purpose of a requirement, its origin, and so on, by looking at requirements over the abstraction level boundaries.

The fact that requirements are not flattened (forced to the same level of abstraction), or put into one repository (regardless of abstraction level), means that requirements produced when using RAM offer the possibility for comparison of requirements against each other on one or several abstraction levels. This is a prerequisite for later planning and prioritization activities.

3. Research Context – Background and Motivation

The development of RAM was prompted by the increased pressure on product development organizations to handle an escalating load of requirements coming in to product management on varying levels of abstraction and detail. Moving away from project centered development (project initiated requirements engineering) towards product centered development (requirements initiated development) demands support for handling the incoming requirements. Giving product management a model for how to handle (and use) the requirements and their varying abstraction levels was the central motivation behind the development of RAM.

The need for a supporting model for handling and working with requirements in this context was also explicitly identified during a software process assessment activity performed at DanaherMotion Särö AB (DHR) [20]. The DHR case is used as an illustration of the problem and the needs described throughout this paper.

DHR develops and sells software and hardware equipment for navigation, control, fleet management and service for Automated Guided Vehicle (AGV) systems. More than 50 AGV system suppliers worldwide are using DHR technologies and expertise together with their own products in effective transport and logistic solutions to various markets worldwide. The headquarters and R & D Centre is located in Särö, south of

Gothenburg, Sweden. DHR has 85 employees. DHR is certified according to SS-EN ISO 9001:1994 (currently working on certification according to ISO 9001:2000), but there have not been any attempts towards CMM or CMMI certification.

DHR has a wide product portfolio, as the ability to offer partners and customers a wide selection of general variants of hardware and supporting software is regarded as important. Product Managers oversee development and new releases of products.

The initiative for an extensive process improvement program was initiated by DHR in recognition of the importance of optimizing their product development, and especially the area of requirements engineering was targeted. The first step of the improvement program was to perform an assessment activity to establish a baseline and identify possible improvement issues.

3.1. Process Assessment Results

During the process assessment conducted at DHR in total nine major improvement issues were identified and formulated (see [20] for detailed information). These nine issues were subsequently prioritized and packaged (according to dependencies and priority) into three separate improvement packages to be addressed in turn (see [21] for detailed information). The Requirements Abstraction Model was primarily built to address the first of these three improvement packages, but also to prepare for the subsequent two (see Section 7). Improvement issue package 1 consisted of three issues as can be seen in Table 1[1]. Below the issues are expanded upon and described to illustrate the state they were in at initiation of the process improvement activity that initiated the creation of RAM.

Table 1. Improvement issue package 1.

Improvement Issue Package 1
Issue-1: Abstraction level & Contents of requirements *Each requirement should be specified on a predefined level of abstraction with certain characteristics (attributes attached to it), enabling requirements to be comparable and specified to a certain predefined degree of detail.*
Issue-2: Roles and responsibilities - RE process *To avoid misunderstandings as well as avoiding certain tasks not being completed the roles and responsibilities of all project members should be clearly defined before project start.*
Issue-3: Requirements upkeep during & post project *In order to keep the requirements up to date during and post project the requirements have to be updated as they change.*

3.1.1. Abstraction Level & Contents of Requirements

Issue-1 speaks to the need of looking over and establishing how the requirements are specified regarding abstraction level and level of detail. During the process assessment, it was ascertained that requirements (obtained from multiple sources) were often specified on different levels of abstraction, and that some were detailed while other were not. This depended on several factors, e.g. who stated the requirement (source), who specified the requirement (experience and expertise), and to what extent the requirement was analyzed and refined subsequent to the initial draft. See Example 1 for an example.

Example 1. Abstraction level and level of detail.

This is an example of two requirements specified on different levels of abstraction and at different levels of detail (i.e. more information is given in the case of Req. 2).
Requirement 1:
TITLE: "Support standardized formats"
DESC: "The system should support standardized formats"
Requirement 2:
ID: "X-11B"
TITLE: "Save output to XML"
DESC: "A user should be able to save output to a file in xml format in order for the data to be exported to the ERP system. Requirement O-7C needs to be implemented before this requirement."
SOURCE: "Kevin Incognito"

Issue-1 and State-of-the-art: Looking at literature and previous studies conducted regarding the state of requirements engineering in industry the points described in Issue-1 are not exclusive to the DHR case in any way. An example is the findings in context of the REAIMS Esprit project [22, 23]. This is further supported by the findings in [10, 24-27], where certain issues were identified as general deficiencies in the requirements engineering process, i.e.

- Analysis and Negotiation (leading to good-enough specification),
- Interpretation and Structuring of requirements,
- Testability and Measurability (of the specified requirements),
- Reviews (of the requirements), and
- Varying abstraction level of specified requirements leading to problems in e.g. comparing requirements.

[1] The improvement issues "issue-id" has been altered from their original state for reasons of simplification. Title and description in Table 1 are unaltered.

3.1.2. Roles and Responsibilities – RE Process

Issue-2 is related to there being an unambiguous and clear description of the responsibilities needed to support the requirements engineering process, as well as an explicitly defined structure for what these responsibilities entailed.

The market driven product development at DHR made it necessary to elicit/catch requirements continuously, i.e. there was a need for a continuous requirement engineering process and roles to support the activities this involved, which partly lie outside of the traditional project initiated requirements engineering process.

Issue-2 and State-of-the-art: The importance of making roles and subsequent responsibilities clear is not an issue reserved for requirements engineering, but pertinent in any organization involved in product development [28, 29]. Roles and responsibilities needed to handle requirements and in essence manage products often lay outside the focus of traditional quality assurance frameworks like CMM [30] since it is not a direct constituent of the development, rather an initiator of it.

3.1.3. Requirements Upkeep During & Post Project

Issue-3 is about keeping requirements “alive” as long as they are relevant. Keeping requirements updated is largely connected to Issue-2 (i.e. who has the responsibility to update requirements). The reasoning was that the requirements should be used throughout the development cycle, i.e. initially for establishing what is to be done and why, later as a basis for validation (e.g. system test), and for purposes of possible reuse. Keeping requirements updated was deemed necessary in order for requirements to be usable over (and beyond) the entire development cycle.

Issue-3 and State-of-the-art: There are any number of reasons for keeping the requirements up-to-date during and post development. If changes are made to what is done and the requirements are not updated the requirements do not reflect the end-result of the development activity. This can be a problem both technically and legally since the requirements cannot be used as a basis for e.g. system test [31] or as a binding agreement (contract) between customer and developers [18].

In the case of DHR’s continuous product development the requirements sources are many (see Figure 1) and the customer is not generally identifiable as one external customer, but rather the role of customer is taken by e.g. the Product Managers. Thus, the requirements are the contract between management (which Product Managers are a part of) and the projects designing and implementing new products/product versions.

3.2. Motivation Summary

Two main factors motivated the creation and evolvement of the Requirements Abstraction Model, (i) a direct need identified in industry, (ii) and that a suitable model was not be found in literature, i.e. a model for continuous requirements engineering catching and handling requirements on multiple levels of abstraction.

Regarding the industry need (i) there was an expressed interest to make the way of working with requirements market driven and product centered [32]. Utilizing product management and the multitude of requirements gathered (from multiple sources, but more importantly on multiple levels of abstraction) by the organization to improve the product alignment towards the needs of the market. The actual need for this type of model has also become even more apparent after the development of RAM. A different organization has shown an interest in applying the model to their organization, due to that they experienced similar challenges as DHR.

The way of working with requirements was to reflect good practices identified in state-of-the-art, i.e. adopting appropriate good practices (see e.g. [22, 23, 33]) in RAM. Moreover, aspects like repeatability and structure were seen as important, but only to the extent of offering a clear support-framework without being cumbersome and overbearing. The plan for achieving a process improvement was to develop a way of working with requirements based on the needs and experiences of the management, Product Managers, and engineers.

Looking at state-of-the-art there is research being conducted in the area of continuous requirements engineering in a market driven development situation, although many problems remain (as described above in combination with the improvement issues), and there is a lack of an appropriate model (ii). Offering support for continuous product centered requirements engineering that not only considers abstraction levels, but also use them, prompted the explicit effort to develop RAM in a way suitable for organizations faced with certain issues, rather than tailoring the model towards one organization.

3.3. Evolvement of the Requirements Abstraction Model

RAM was developed in several stages as can be viewed in Figure 2.

Subsequent to the initial formulation RAM went through two major validations (Validation One and Two are seen as *static validation*) before it was deemed good-enough to be tested in a live industry setting (*dynamic validation*). The *static validation* steps involved brainstorming/interview sessions with product and Project Managers as well as discussions with representatives for development, system test, and upper management.

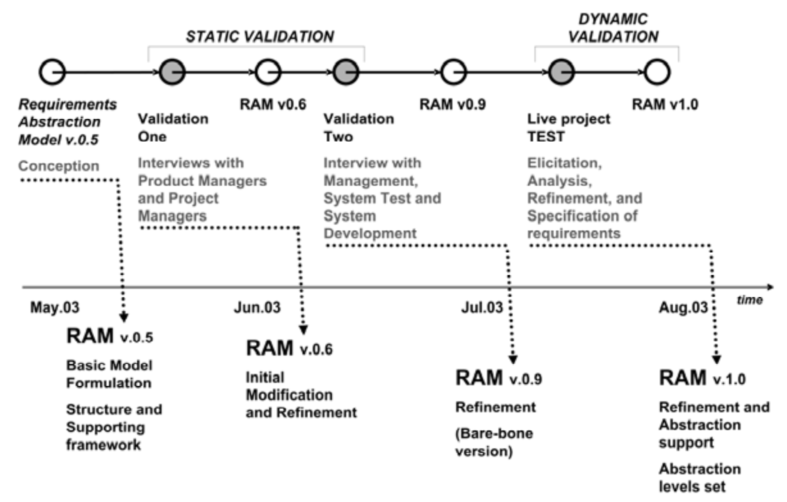

STATIC VALIDATION
DYNAMIC VALIDATION
Requirements Abstraction Model v.0.5
Conception
Validation One
Interviews with Product Managers and Project Managers
RAM v0.6
Validation Two
Interview with Management, System Test and System Development
RAM v0.9
Live project TEST
Elicitation, Analysis, Refinement, and Specification of requirements
RAM v1.0
time
May.03
Jun.03
Jul.03
Aug.03
RAM v.0.5
Basic Model Formulation
Structure and Supporting framework
RAM v.0.6
Initial Modification and Refinement
RAM v.0.9
Refinement
(Bare-bone version)
RAM v.1.0
Refinement and Abstraction support
Abstraction levels set

Figure 2. Evolvement time-line of RAM.

The *dynamic validation* consisted of using RAM for a real live requirements engineering effort. Both the static and dynamic validations had a fundamental impact on the model, and they are described in Section 5.

RAM version presented in this paper is version 1.0 (see Section 4), i.e. the model that evolved as a result of the validations. RAM v.1.0 was mainly aimed towards establishing support for the initial stages (specification, placement and work-up) of the continuous requirements engineering performed by Product Managers. This is illustrated in Figure 3 where two types of requirements engineering can be seen, continuous and dedicated. The continuous requirements engineering is considered a prerequisite for being able to deliver a sub-set of the total requirements to one or more development projects.

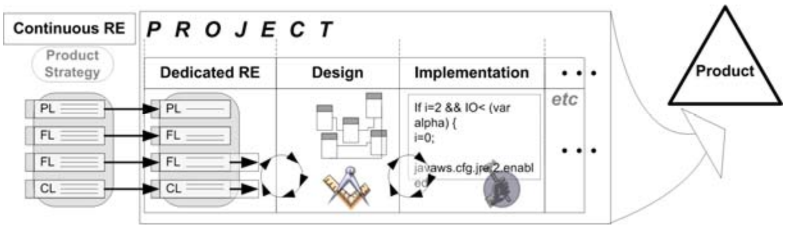


Figure 3. RAM overview – continuous vs. dedicated requirements engineering.

4. RAM Structure & Supporting Process – An Overview

This section describes the Requirements Abstraction Model's structure and the supporting process developed as a part of it. The gray example boxes in this section offer exemplification continuously throughout the text. Examples are important when working with RAM since the model is example-driven, i.e. relevant examples (regarding the requirements engineering of the product in question) are important as a means to support training and use of RAM for continuous requirements engineering. Developing relevant examples is a part of the Model Tailoring (see Section 4.5), as is ascertaining the number of abstraction levels appropriate, and attributes needed for each requirement. The version of RAM presented in this paper (Sections 4.1 through 4.3) is based on the one developed at DHR, and it is an example of what RAM can look like. The examples are not specific to any domain or organization, rather general in nature. This offers a general model exemplification and overview. The version of the model presented here is intended as a starting point for anyone wanting to tailor it for their specific organization and products.

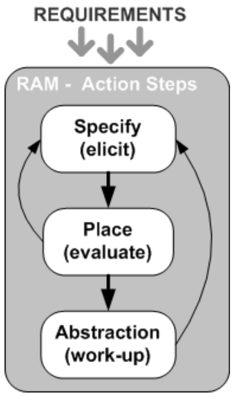


Figure 4. RAM action steps.

Requirements engineering using the model involves the following three basic steps, as can be seen in Figure 4. The first step (Specify) involves specifying the initial (raw) requirement and eliciting enough information about it to specify a number of attributes. The second step (Place) is centered around what abstraction level the now specified requirements resides on, and last (Abstraction) each requirement goes through a work-up. Each of these steps is described in further detail below.

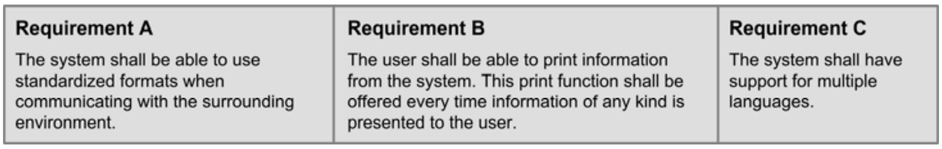
4.1. Action Step One – Specify (elicit)

In order to get a uniform way of specifying requirements, four main attributes are to be specified manually in this initial step. The goal of this step is to get an overview of the raw requirement to the extent of it being understood by the Product Manager performing the continuous requirements engineering.

The four attributes are:

1. **Description** The requirement description should not be more than about 5 sentences, and should in broad strokes describe the central essence of the requirement. Example 2 offers examples of this attribute.

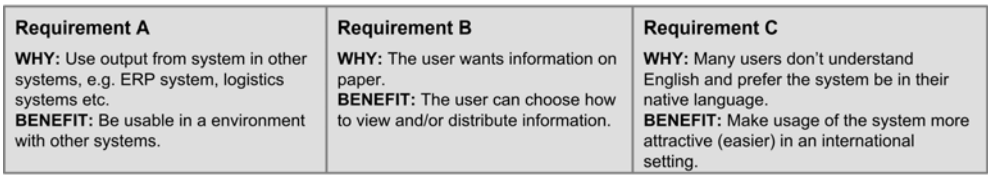
Example 2. Description.



Requirement A	Requirement B	Requirement C
The system shall be able to use standardized formats when communicating with the surrounding environment.	The user shall be able to print information from the system. This print function shall be offered every time information of any kind is presented to the user.	The system shall have support for multiple languages.

2. **Reason/Benefit/Rationale** This attribute consists of two parts; **WHY** the requirement is specified and **BENEFIT** of the specified requirement. "Benefit" should be seen in the context of the subject of the requirement being stated. If the subject is an *user* it should illustrate the benefit to him/her (see Example 3, Requirement B), if the subject is the product itself, e.g. in a non-functional requirement, the benefit should reflect the benefit from the requirements perspective (see Example 3, Requirement A and C).

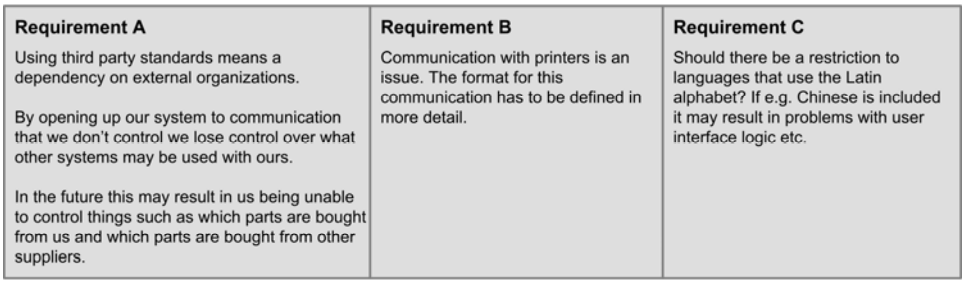
Example 3. Reason/Benefit /Rationale.



Requirement A	Requirement B	Requirement C
WHY: Use output from system in other systems, e.g. ERP system, logistics systems etc. **BENEFIT:** Be usable in a environment with other systems.	**WHY:** The user wants information on paper. **BENEFIT:** The user can choose how to view and/or distribute information.	**WHY:** Many users don't understand English and prefer the system be in their native language. **BENEFIT:** Make usage of the system more attractive (easier) in an international setting.

3. **Restrictions/Risks** This attribute describes the restrictions and/or risks with the requirement that is not obvious in the description. It is possible to say that this attribute constitutes the negotiation space in the requirement. Example 4 offers examples of this attribute.

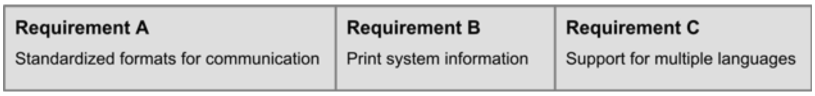
Example 4. Restrictions/Risks



Requirement A	Requirement B	Requirement C
Using third party standards means a dependency on external organizations. By opening up our system to communication that we don't control we lose control over what other systems may be used with ours. In the future this may result in us being unable to control things such as which parts are bought from us and which parts are bought from other suppliers.	Communication with printers is an issue. The format for this communication has to be defined in more detail.	Should there be a restriction to languages that use the Latin alphabet? If e.g. Chinese is included it may result in problems with user interface logic etc.

4. **Title** The title should reflect the contents of the requirement and should not be longer than five words. Example 5 offers examples of this attribute.

Example 5. Title.



Requirement A	Requirement B	Requirement C
Standardized formats for communication	Print system information	Support for multiple languages

As these four attributes have been specified, the next step is to ascertain the abstraction level of the requirement in question. Additional attributes to be specified are described in Section 4.4.

4.2. Action Step Two - Place

RAM consists of a number of abstraction levels (the driving motor of the model). This step involves analyzing what level a requirement is on, and placing it on this level.

Looking at Figure 5, four abstraction levels can be seen, i.e. *Product Level*, *Feature Level*, *Function Level*, and *Component Level*. (See Example 6 for exemplification of the placement of requirements on abstraction level described here)

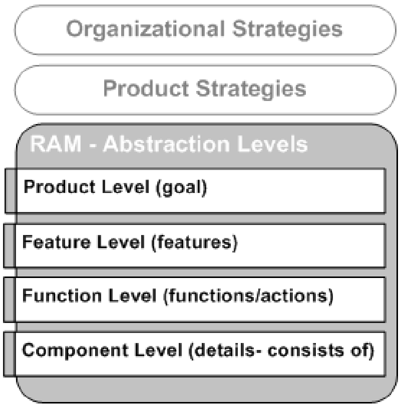

Organizational Strategies
Product Strategies
RAM - Abstraction Levels
Product Level (goal)
Feature Level (features)
Function Level (functions/actions)
Component Level (details- consists of)

Figure 5. RAM abstraction levels.

The *Product Level* is the most abstract level. Requirements on this level are goal-like in nature, i.e. not fitting the normal definition of a requirement (e.g. testable and unambiguous, [33]), thus the use of the term "requirement" can be viewed as somewhat questionable in this case. Nevertheless, "requirement" is used for statements on all the four levels of abstraction. This is motivated by the fact that requirements in the model do not exist in isolation, but are always broken down to a level that is in line with the traditional meaning of the word "requirement" (see Section 4.3), as a part of RAM work-up.

Example 6. Placing example requirements.

Looking at Requirements A, B and C (see Examples 2 through 5) the following is an exemplification of the placement procedure following the how-to guide displayed in Figure 9 in Appendix A.

As illustrated, the guide poses a series of questions to steer the initial placement of the requirement on a certain level. By following the guide step-by-step (grey dashed line with the arrows) an attempt is made to place the example-requirements ***A: Standardized formats for communication***, ***B: Print system information*** and ***C: Support for multiple languages***.

The first question is if the requirement is functional or not, or if the requirement in question described what (testable) characteristics a system should provide. This applies to ***B: Print system information***, but not to the other two requirements since none of them fall into the description of providing testable characteristics to the system.
The next question is if the requirement consists of specific suggestions of HOW things are solved. This does not apply to any of the example-requirements. However if ***B: Print system information*** had information in it that spoke to details for solutions, e.g. "support the post-script standard for printing" it would have been a candidate for the Component Level.
The next question is if the requirement is comparable to the product strategies. Both requirements ***A: Standardized formats for communication*** and ***C: Support for multiple languages*** are fairly abstract in nature. Requirement ***A*** basically means that the product should open up to communicating in a standardized way with the surrounding environment. This goes against a strategy saying, "to box in the customer to a certain standard (maybe specific to the product developing organization) and a certain range of products". Requirement ***A*** is however in-line with a strategy saying "to offer a customer the choice of other products by enabling them to communicate with ours". We place requirement **A** on the Product Level as it is directly comparable to product strategies.
If requirement ***C*** is compared to the product strategies it may be possible to deduct whether or not it complies, but probably only indirectly. It is probably not within the product's strategy "to support multiple languages in their products", however "to offer the product to an international market" may be the case. Thus, requirement ***C*** is not directly comparable to the product strategies, but close.
The next question posed is if the requirement describes "what the system should include/support". This fits requirement ***C*** since it speaks to that the system should have support for multiple languages, i.e. requirement ***C*** is placed on Feature Level.

Product Level requirements are considered abstract enough to be comparable directly to the product strategies, and indirectly to the organizational strategies. In the context of RAM, product strategies are e.g. rules, long and short-term goals, and visions pertaining to a product specified by management. Product strategies are in other words what govern what is to be done (direction of the product), and what is not, depending on e.g. targeted market segments, competitor situation, and so on.

The *Feature Level* is the next level in the model. The requirements on this level are features that the product supports. Feature Level requirements should not offer details as to what functions are needed in order for the product to support a feature; rather the requirements should be an abstract description of the feature itself.

The *Function Level* is as the name suggests a repository for functional requirements, i.e. what a user should be able to do (actions that are possible to perform), but also for non-functional requirements. The main criterion is that the requirement should be descriptive of what a user (or the system in the case of non-functional requirements) should be able to perform/do. In general, Function Level requirements are detailed and complete enough to be handed over to a system designer for further evolution and finally be a basis for the design. Functional Level requirements should strive to be testable and unambiguous.

The *Component Level* is the last level of abstraction in RAM. Component Level requirements are of a detailed nature depicting information that is closer to (or even examples of) *how* something should be solved, i.e. on the boundary of design information. The main reason for the existence of this level is twofold. Many requirements that come from internal sources (e.g. engineers) are on this level of abstraction. The Component Level can also act as a possibility to break down Function Level requirements in more detail and/or set limits to a Function Level requirement. This last point is elaborated upon in Section 4.3.

In order for this RAM action step to be completed (i.e. the initial placement of the requirement on appropriate abstraction level) a *how-to* guide was developed, as can be seen in Figure 9 in Appendix A. This guide operates on the principle of asking a series of questions and thus guiding the initial placement of the requirement. It should be noted that the how-to guide is only part of the support material offered to the professionals working with the requirements. Other support materials consist of e.g. a list of multiple examples (of requirements on all abstraction levels) relevant to the requirements engineering of the product in question. Example 6 offers exemplification of the placement of a requirement on a certain abstraction level.

It is important to realize that the initial placement of requirements as described in this section (and exemplified in Example 6) is not an absolute, but a balance, where the requirements have to be weighed against the examples and instructions in e.g. the how-to guide. It is imperative for the requirements engineers (Requirements Manager) working with RAM to be consistent, i.e. placing requirements of comparable abstraction levels on the same level, and enforcing consistency over time.

4.3. Action Step Three – Abstraction (Work-up)

Subsequent to the initial placement of the requirement on an appropriate abstraction level the work-up of the requirement can commence. This third step of RAM involves abstracting and/or breakdown of a requirement, depending on the initial placement of the *original* requirement. The work-up process involves creating new requirements (called *work-up requirements* hereafter) on adjacent abstraction levels or linking to already existing ones, depending on the situation.

This process takes place for several reasons. First, as mentioned before, one of the model's goals is for every requirement to be comparable with the product strategies. Thus, every requirement (on Feature Level or lower) has to be abstracted up to the Product Level (see Figure 5) in order for this to be possible. This creates the first work-up rule (R1):

R1: No requirement may exist without having a connection to the Product Level.

R1 can be met in one of two ways, one or more new *work-up requirements* are created, or the requirement in question is linked to already existing requirements on an adjacent upper level. In either case, the original requirement is abstracted upward and can be compared (indirectly) to the product strategies.

In addition to abstraction, there may also be reason for requirements to be broken down enough to act as a basis for design (good-enough for project initiation). For a requirement to be detailed enough and on the right level of abstraction for this to be possible every requirement (on Feature Level or higher) has to be broken down to Function Level (testable and unambiguous). This creates the second work-up rule (R2):

R2: All requirements have to be broken down to Function Level.

Like in the case of R1 this can mean creating one or more work-up requirements on adjacent levels, or linking the requirement in question to already existing requirements on lower adjacent levels. Either is acceptable as long as the requirement(s) on lower levels satisfy the upper level one.

Satisfy in this case pertains to the issue of breaking down a high-level requirements (from e.g. Feature Level) to Function Level, where the requirements on lower level together satisfy the original one to the extent of giving a foundation good-enough for the initiation of realization (design) of the requirement. The main reasoning behind this is that requirements are meaningless if they cannot be delivered to a development effort (i.e. left on a too abstract level). Typically, R2 involves the creation of several work-up requirements being created.

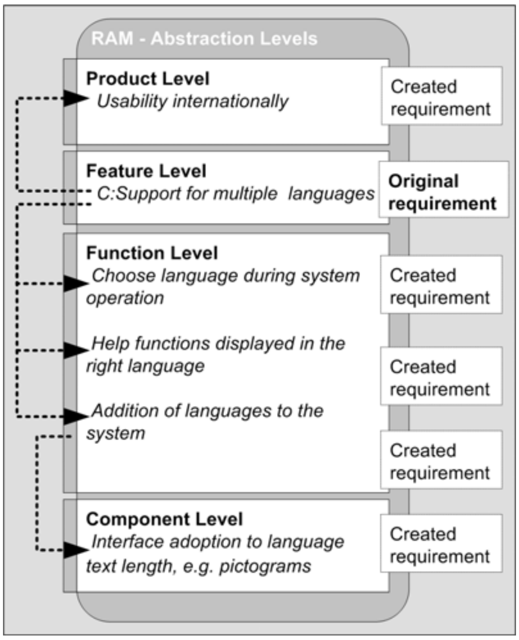
An example of R1 and R2 can be viewed in Example 7. Here the original requirement "*C:Support for multiple languages*" (placed on Feature Level) is abstracted to Product Level (R1) through the creation of a work-up requirement "*Usability internationally*", and broken down to Functional Level (R2) where three work-up requirements are created as a part of the breakdown.

The breakdown to Component Level offers further details to relatively low-level requirements and is often associated with suggestions of *how* to implement a requirement. This level is not mandatory in the process of breaking down requirements. However, it should be observed that it is not unusual that several of the incoming requirements are on the Component Level of abstraction, thus the level cannot be discarded. The need for the Component Level is illustrated in the validation part in Section 5. In addition, the Component Level can also act as clarification (refinement and/or limitation) to one or more Function Level requirements. An example of this can be viewed in Example 7, where a technical limitation pertaining to the user interface adds to (and limits the possible solutions by adding a design element) the requirement on Function Level.

As was the case in the previous step (initial placement of the requirement) a how-to guide was developed to aid in the work-up of requirements. This guide (see Figure 10 in Appendix A) is example-driven adhering to the two rules (R1 and R2) described above. It shows mandatory as well as optional work-up of the requirements.

It should be noted that none of RAM action steps is final, i.e. in some instances it is necessary to iterate (go back) and rethink the initial placement of a requirement (as the work-up offers an analysis that may change the perception of the original requirement). This can also involve eliciting additional information from the requirement's source if this is possible.

Example 7. Abstraction and breakdown of example-requirement *C: Support for multiple languages.*


RAM - Abstraction Levels
Product Level
Usability internationally
Created requirement
Feature Level
C:Support for multiple languages
Original requirement
Function Level
Choose language during system operation
Created requirement
Help functions displayed in the right language
Created requirement
Addition of languages to the system
Created requirement
Component Level
Interface adoption to language text length, e.g. pictograms
Created requirement

During the work-up, it is important to stay true to the original requirement, or rather the intention of it. The creation of new requirements as a part of the work-up should not stray too far from the initial intention of the original requirement, and thus give rise to totally new requirements that are related to but outside the scope of initial intention. It is inevitable to create new requirements (especially if the original requirement is on a high abstraction level) as the work-up is designed to create new relevant work-up requirements to the extent that they satisfy the original requirement. However, this is not the same as including new requirements based on "this might also be a good idea" philosophy, as this could give rise to a mass of new requirements. As the model (and the usage of it) is aimed at offering support to professionals, it is also very dependent on the same professionals pertaining to how well it works. A recommendation when performing work-up of original requirements is always to ask the question "is this new (work-up created) requirement really necessary in order to satisfy the original requirement"? If the answer is "yes" then there is no problem, but if there is uncertainty, the work-up should be stopped.

This does not mean that good ideas pertaining to new requirements should be discarded along the way in any case, but they should not be a part of the work-up, rather be specified as new original requirements on an appropriate level (and in turn get a work-up themselves).

4.3.1. Requirements Work-up - Discussion

Looking further into the process of work-up using RAM, several potential issues can be identified when applying the work-up rules.

As mentioned before, no requirement may exist without having a connection to the Product Level (R1). This can imply that new work-up requirements are created on upper levels using lower level original requirements as a base. A potential issue during this activity is that an original incoming low-level requirements can give rise to (or be linked to) several requirements on a more abstract level. Two rules of thumb apply here, based on what was learned during the validations described in Section **Fel! Hittar inte referenskälla.**. First, staying true to the intention of the original requirement is a priority. Inventing new requirements from scratch outside what is needed to work-up an original requirement can result in overwork of requirements at this early stage (product management). New requirements that are created as a part of the work-up process *are* separate requirements, but not *original* requirements. This distinction can seem somewhat trivial but it helps in keeping focus on expanding the meaning of the original requirements and the work-up of them instead of using the whole process as an excuse to create large amount of new independent requirements. Like all activities the requirements engineers, e.g. product managers, base this distinction on their expertise and judgment, and they must judge not only the impact on future development but also impact on product strategy. Secondly, a new requirement that cannot be directly attributed to the work-up process, but still invented as a result of an original requirement idea, should not be dismissed, but rather stated as a new original requirement (inventor is the source), and then subjected to work-up of its own. Work-up has to be seen as a part of the analysis, refinement and specification of requirements, as well as a way to compare incoming requirements to product strategies.

Once the work-up is completed and there are distinct links from the original requirement upwards and downwards (if applicable) a requirement in the "chain" cannot be removed without considering the whole structure of the requirements. For example, removing a requirement on Product Level would result in that all requirements linked under it must either be re-linked to another requirement on Product Level, or all requirements in the chain have to be deleted as well.

The same consideration has to be taken when removing requirements on lower levels, i.e. removing a requirement on Function Level would demand an explicit decision stating that the above linked requirement(s) can be satisfied after the removal (by other requirements linked to the Feature Level requirement in question), or the Feature Level requirement should also be removed. Looking at Example 7, this would mean that the removal of e.g. the Function Level requirement *"Addition of languages to the system"* would demand that an explicit decision is made that functionality for adding new 'language sets' to the system should not be incorporated as a part of the feature of supporting multiple languages in the system. This in turn has implications. How should languages be added? Are some set of them just added manually at development, and so on? The decision of when (in this case a Feature Level) a requirement is satisfied by lower level requirements is also a judgment call, but an explicit decision has to be made all the way up to Product Level regarding adequacy and satisfaction. Ultimately management (both business and technical) have to decide if a certain chain of requirements are complete and within the product strategies or not.

4.4. Additional Structure and Process – Roles, Attributes and Rules

As requirements are specified, placed and worked-up additional attributes are specified (in addition to those introduced in Section 4.1) for each requirement as applicable. Table 2 summarizes the additional attributes with a short description linked to each. Under the comment column there is indication whether or not the attribute has to be specified (mandatory) or not (optional), as well as if it has to be specified manually or if it is auto generated (assuming tool use). The subsequent subsections (0 and 4.4.2) elaborate regarding the attributes and their rationale.

Table 2. RAM requirement's attributes. (For attribute 1 to 4, see Section 4.1).

Attribute Title	Description	Comment
5. Requirement Source	This is a link to the source of the requirement. This can be a physical person, document, group, or meeting. The exactness depends on the information available.	Mandatory (Manual)
6. Requirement Owner	A link to the person who "owns" the requirement and is responsible for the follow-up of the requirement. This person acts as the advocate of the requirement. This role is always held by a internal person in the product development organization.	Mandatory (Manual)
7. Requirements manager	An identification of the Product Manager responsible for the specification, placement, and work-up of the requirement.	Mandatory (Auto generated)
8. Relation/Dependency	One or several links to other requirements on the same level of abstraction. This attribute's aim is to record important relations/interdependencies of different types between requirements. Every link can be augmented by a explanation in free-text.	Optional (Manual)
9. State	A requirement in RAM can have different states giving information of the status of the requirement, see Figure 6 for details.	Mandatory (Manual)
10. Reject Reason	If a requirements state is set to "Rejected Requirement", i.e. it is deemed out of scope in relation to the product strategies, then this attribute records the rationale of the decision.	Mandatory if requirement is rejected. (Manual)
11. Due Date	This attribute's purpose is to ascertain that requirements are not forgotten, e.g. put as draft indefinitely. The manual usage of the attribute can be in case of e.g. customer deadline etc.	Mandatory, auto set to 30 days if nothing else is specified.
12. Version	Records version on requirements level rather than document level. Enables requirements' history to be viewed.	Mandatory (Auto generated)
13. Date of Creation	Records the creation date of the requirement.	Mandatory (Auto generated)
14. Last Changed	Indicates when the last change was performed.	Mandatory (Auto generated)

4.4.1. Traceability and Role Attributes

Attributes 5, 6 and 7 are linked to traceability issues, enabling (roles) people to be linked to a certain requirement, ensuring that responsibilities are clear not open to interpretation, on a requirement level rather than a document level, to avoid issues like certain requirements being neglected or even overlooked entirely.

As requirements are caught/elicited from multiple sources everything from a person, to a market survey or a competitor analysis, can be the official "source" of the requirement (in the latter two the sources are the reports/documents produced). This makes the role of Requirement Owner important as this person is charged with the responsibility of seeing that the requirement is followed through on. A Requirement Owner is the representative of the Requirement Source when this role is silent, e.g. when the source is a survey or an external party like a group of customers. In some instances the Requirement Source and the Requirement Owner can be the same person (e.g. in the case of an internal engineer formulating the original requirement).

The Requirements Manager role is typically represented by the Product Manager charged with the responsibility of actually working with the requirement throughout its lifecycle. During RAM action steps (see Figure 4) the Requirements Manager can utilize resources needed, e.g. asking system experts and/or domain specialists for input during the work-up of the requirement. The cooperation between the Requirements Manager , Owner and Source (when applicable) is especially important as they respectively possess different perspectives and knowledge pertaining the requirement in question.

If a requirement is created as a part of the work-up (a work-up requirement not an original one), the attributes of Requirement Source/Owner/Manager are set to the Requirements Manager responsible for the work-up.

4.4.2. Process (Attributes)

State reflects how the requirement is handled in the product development organization and how it is set to different states reflecting the status. Figure 6 offers an overview of the different states. Looking at Figure 6 states *A*, *B* and *C* are a part of RAM action steps (see Figure 4) and the continuous (project independent) requirements engineering and is basically about drafting the requirement. The work done associated with the three states is specified in further detail in Table 3 where each separate step and sub-step is described. Observe that states *D (dependent on prioritization)* and *F (dependent on packaging into release packages)* are addressed in future work (see Section 7).

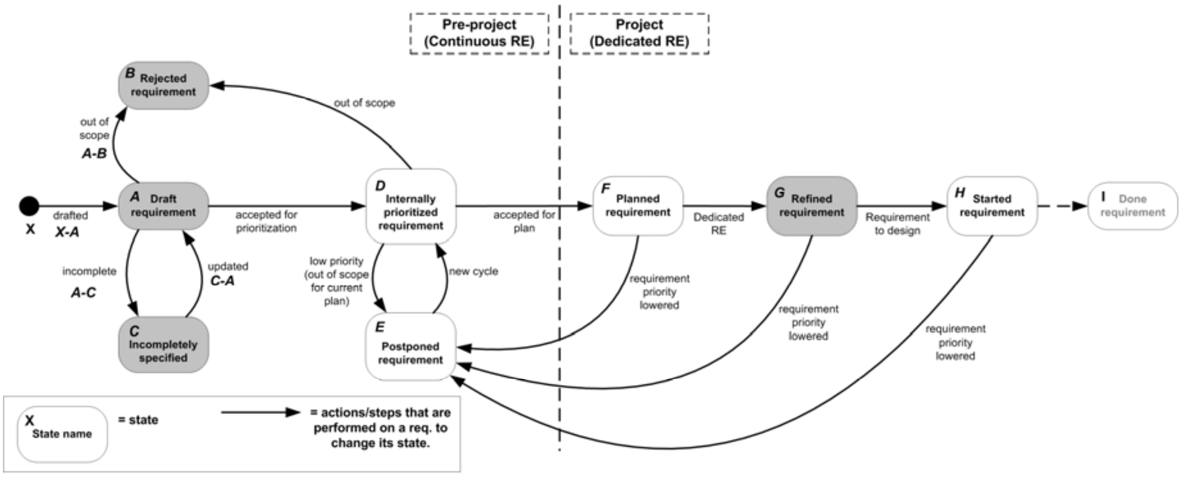


Figure 6. Requirement's states in RAM.

In RAM v.1.0 the possible states a requirement can exist in are *A: Draft requirement*, *B: Rejected requirement*, *C: Incompletely specified*, and *G: Refined requirement*. A requirement can reach states *A*, *B*, and *C* during the continuous requirements engineering, i.e. the work done as a part of RAM action steps. State *G* is however reached as the requirement in question is subjected to further refinement and validation during the *dedicated* requirements engineering.

Dedicated requirements engineering is performed on a chosen subset of requirements after project initiation, and involves refinement by the development project and system test departments to assure testability and unambiguity as well as completeness of the requirements.

It should be noted that state *B* is dependent on the requirement being out of scope, i.e. that it is not in line with the product strategies. Generally the out of scope requirement is rejected off hand (and attribute *10: Reject reason* is specified), but in some instances an alternate decision can be taken. If a requirement is considered out of scope but is important for any reason (e.g. an important customer is the source) an exception can be made. However, this exception is an explicit action and has to be approved by both the Requirements Manager and the Requirement Owner, as well as checked against upper management. The general rule is that all requirements not rejected should be in line with the strategies formulated for a product. Exceptions to this rule should be kept at a minimum in order to use the full potential of RAM.

Table 3. RAM state steps (see Figure 6).

Step ID	Step Description	Sub-Steps
X-A	The requirement (req.) is caught/elicited and drafted by the Requirements Manager . The work is done by the Requirements Manager , that utilizes relevant resources if the need arises (e.g. Requirement Owner, experts and so on are important parties in this process).	**X-A-1:** Specify attribute 1 to 4 (Specify). **X-A-2:** Determine on which level the original requirement is on (Place). **X-A-3:** Specify all attributes on relevant level. **X-A-4:** Abstract and/or Breakdown the original requirement according to work-up rule **R1** and **R2** (work-up). **X-A-5:** Validate requirement against Requirement Owner and Requirement Source.
A-B	During (or rather, as the requirement is abstracted) work-up the requirement is checked against product strategy. Requirements deemed as not in line with strategies are deemed out of scope and thus rejected. As a requirement is rejected, directly related requirements (work-up requirements) on adjacent abstraction levels either are removed as well or re-linked (enforcing **R1** and **R2**).	**A-B-1:** Compare requirement (directly or indirectly) to product strategies. If the req. is not in line with strategy it is rejected. **A-B-2:** If a requirement is rejected for any reason created work-up req. have to be evaluated if they should also be removed. If all are removed, nothing further is needed. However if a work-up req. is left on any level it has to be re-linked to requirements above and/or beyond for the work-up rules **R1** and **R2** to be enforced.
A-C	During validation against the Requirement Source/Owner (**X-A-5**) the req. can be deemed incomplete. This can be a result of e.g. incorrect initial placement, unsatisfactory work-up and so on.	

4.5. Model Tailoring

The basic elements of the Requirements Abstraction Model revolve around using abstraction levels, instead of flattening them (or dividing them into separate documents), to work with requirements. RAM action steps described in Sections 4.1, 4.2, and 4.3 follow this ambition, and the use of attributes (Section 4.4) enables a requirements oriented (not document oriented) way of working.

This does not mean that RAM was developed to be prescriptive in nature. One model fitting all types of products is not a realistic aim, not to mention differences between organizations. To this end, the model is intended to be a framework on which continuous requirements engineering can be based. Several things need to be addressed prior to the model being set into operation in an organization. This can be seen as a tailoring of RAM to fit a specific product (organization), as well as giving the adopting organization’s representatives (e.g. Product Managers) a chance to acquaint themselves with RAM.

Below an overview of the tailoring aspects is offered.

- **Abstraction Levels** (and usage of them) are a fundamental part of RAM. However, the *number of abstraction levels* and the *abstraction degree* of each level is not to be considered absolute.

  The number of abstraction levels needed depends on the product and organization. To facilitate abstraction (to a level comparable to product strategy) and breakdown (to a level good-enough for project initiation) different organizations may have different needs. Three levels may suffice in some cases where requirements are generally only caught/elicited on three levels, other organizations may have the need for five abstraction levels, and so on.

  The same reasoning applies to the abstraction degree of each level. It is dependent on the number of abstraction levels used, i.e. the jumps between levels are greater if few levels are used and vice versa.

  As a rule, two things determine the number of abstraction levels and the abstraction degree of these levels. First, the requirements caught/elicited can be used as an indicator (if requirements are typically caught on four levels this amount may be appropriate). Second, the need for work-up of the requirements, i.e. does e.g. four levels (instead of three) help in making the work-up of the requirements easier? This mainly pertains to the distance between the abstraction levels.

- **Example-driven** is a term used in the description of RAM. As different organizations (products) are dependent on different vocabularies, domains, technical terminology, traditions and needs, in terms of e.g. the number of abstraction levels and abstraction degree, relevant examples have to be developed. The examples are to illustrate most “common” situations encountered by the Product Manager charged with the job of performing the continuous requirements engineering.

- **Attributes** need to be reviewed and adapted to the needs of the organization. It is important to realize that perfect attributes are meaningless if they are not specified. A balance between benefit and cost has to be reached, and the benefit of specifying a certain attribute should be well anchored in the organization.

- **Roles** present differ from organization to organization. In some cases new roles may be needed, e.g. if no Product Manager or equivalent role exists charged with the continuous requirements engineering, in other cases already present roles can be redefined. This is not primarily to fit the model, but rather to fit the nature of requirements initiated development (market driven and continuous).
- **Naming** of abstraction levels, attributes, roles and so on should be adapted to fit the vocabulary of the organization. The main concern is that all parties have the same understanding of what is meant and a naming standard is used.
- **Product Focus** means that an organization with homogenous products (pertaining to the development) can work with the same version of RAM (tailored to their needs). If an organization has heterogeneous products (e.g. large organization with different types of products/product lines) several tailored versions of RAM may be beneficial, e.g. one for each “group/type” of product.
- **Support Material**, e.g. how-to guides is a result of the other tailoring points described above. The number of abstraction levels, abstraction degree, relevant examples, and so on all govern how the guides are developed and what contents they have.

Details pertaining to the tailoring of RAM are considered outside the scope of this paper. However, it should be noted that a tailoring effort of RAM is underway at present, which is a prelude to the test and implementation of RAM in a second organization faced with a situation similar to the one identified at DHR.

4.6. Requirement Abstraction Model – Summary of the Action Steps

Product Managers are offered a model to handle requirements on multiple levels of abstraction in a continuous requirements engineering environment, and a structured framework for how to work-up these requirements. The main steps of RAM are summarized below:

1. **Specify** initial attributes (attributes 1-4) to form a basic understanding of the requirement and ascertain its abstraction level. This is generally done by the Requirements Manager (Product Manager) in cooperation with the Requirement Owner (and the Requirement Source when applicable) (see Section 4.1 and 4.4).
2. **Place** the requirement (see Section 4.2) on an appropriate abstraction level (using product relevant examples, see Section 4.5, and the how-to guide in Figure 9 in Appendix A).
3. **Work-up** (see Section 4.3) the requirement by specifying additional requirements (called work-up requirements) on adjacent abstraction levels until work-up rule R1 and R2 are satisfied (using product relevant examples, see Section 4.5, and the how-to guide in Figure 10 in Appendix A).
   During work-up attributes 1-4 are specified for the new work-up requirements. As the work-up is being completed additional attributes need to be specified (see Section 4.4).

During work-up it may be necessary to rethink the initial placement of the original requirement and reinitiate work-up.

5. RAM - Validation

This section offers a presentation of the evolutionary development of RAM as it was validated in industry against an organization faced with the improvement issues described in Section 3.

The validation is divided into two main parts, static and dynamic validation (see Figure 2). The static validation consists of reviews and walkthroughs of RAM, and the dynamic validation consists of a live industry requirements engineering effort where the model was put through its phases.

5.1. Static Validation

As RAM was developed it was considered important to get input from all stakeholders involved with product management and requirements engineering, as well as the ones using requirements as input to their work.

The first step was to identify relevant general roles at DHR, and their relation to the requirements engineering process. The roles identified were project centered, as the organization largely was centered around development projects (i.e. project initiated requirements engineering) at the time of the model’s development. This also meant that the roles were not directly compatible with the ones described by RAM (see Section 4.4).

Below the identified roles are listed. The roles were to some extent influenced by the roles identified during the process assessment activity performed earlier at DHR [20] (see Section 3.1).

A. The *Product Manager* role was new to the DHR organization (<1 year). The role's responsibilities were centered on product coordination and product (release) planning, as well as development project coordination. The Product Manager was considered responsible for a product and answered to upper management.

B. The *Orderer* had the task of being the internal owner of a certain project, i.e. having the customer role and if applicable the official contact with an external customer and/or partner. This role is responsible for the official signing-off when it comes to the requirements, i.e. he/she places an order.

C. The *Project Manager* has the traditional role of managing the project, resources, planning, and follow-up. As far as requirements are concerned, the Project Manager is responsible for that the requirements engineering is performed, the requirements specification is written and signed off by the System Engineer.

D. The *System Engineer* is the technical responsible for a project. It is also important to recognize that the System Engineer has the official responsibility for the requirements specification in a project.

E. The *Developer* is a representative for the developers (e.g. programmers) in a project, the ones actually implementing the requirements. The developers use the requirements specification.

F. The System Test role can be described as the traditional role of system test. This role is officially present during initial project meetings and is a part of the verification of the requirements specification.

G. *Upper Management* was identified as a crucial stakeholder since executive power to a large extent resided here. This role also represented the process owners, as well as the ones actively participating in the creation of product strategies.

Static Validation One involved eliciting feedback/input regarding RAM from the following roles:

- A. Product Manager (2 persons interviewed)
- B. Orderer (1 person interviewed)
- C. Project Manager (2 persons interviewed)
- D. System Engineer (1 person interviewed)

Static Validation Two involved eliciting feedback/input regarding RAM from the following roles:

- E. Developer (1 person interviewed)
- F. System Test (1 person interviewed)
- G. Upper Management (1 person interviewed)

The division of roles between the two static validations was deliberate. Static Validation One was centered on the initial formulation of the model and cooperation with roles working with requirements (elicitation, analysis, management, and so on) were considered as crucial. Static Validation Two was centered on the input to and output from the model (what the model needed, e.g. product strategies, and what the model produced, e.g. requirements good-enough for project initiation). Developers and System Test were in the position to give input as to e.g. when a requirement was good-enough for project imitation, testability, and so on. Upper Management was directly involved with product strategies, as well as the process as a whole.

Both validations were carried out in the form of unstructured interviews [34] in the offices of the interview subjects and lasted between one and two hours each.

In addition to the two official static validations described above it should be noted that the development of RAM was conducted on-site and that unofficial discussions and feedback were commonplace from multiple sources not limited to, but sometimes including, the roles and interview subjects that participated in the official validations.

5.1.1. Static Validation Impact and Lessons Learned

Static Validation One mainly influenced RAM pertaining to *size* and *complexity*.

The initial designs of RAM were considered too ambitious in terms of how requirements were specified (e.g. more attributes pertaining to relations and dependencies). It led to a larger model demanding further detail in the specification. There was a view amongst the Product Managers that too rigorous demands would result in either that the model would not be used (at least not as intended), or that too much time would go into specifying details that were never used (even if potentially beneficial). It was realized that by simplifying the model pertaining to what was specified in relation to a requirement also dulled the requirement in question. This trade-off was considered and the general view was that it was better to have a model (requirements) which was good-enough and used, than a larger more complete model that was considered too cumbersome.

In addition, the usability of the model was debated and the consensus was that the example-driven nature of the model was important as far as usability was concerned. Product centered relevant examples of specification, placement and work-up were considered necessary, and in combination with abbreviated how-to guides (see Appendix A) enhanced usability of the model substantially, especially in comparison to just offering e.g. a list of rules describing RAM.

The structure and rules of RAM were considered important to get a repeatable and comparable continuous requirements engineering process that produced requirements on a level of detail that was predetermined, and not ad-hoc. The ability to compare (directly or indirectly through work-up requirements) requirements to product strategies was considered very beneficial as it theoretically would be possible to dismiss some requirements off-hand if they were considered to be out of scope. The main risk identified in this scenario was that product strategies had to be present and explicit in order for this to work.

Static Validation Two mainly influenced RAM pertaining to *process* related issues.

This validation can be divided into two main parts: *RAM produced requirements* and *RAM general process*. RAM produced requirements was the part discussed at length with the Development and System Test roles, and involved how to ascertain that RAM produced requirements (on Function/Component Level) passed to development (input to initial design) were refined enough, unambiguous and testable. These aspects were used as input to the validation with upper management regarding RAM *general process* and gave rise to a division of RAM requirements engineering into two main parts: a *continuous* (pre-project) part and *dedicated* (during project) part. Figure 3 illustrates this.

To ensure testability and to get a pre-design review of all requirements a *testability review* was proposed. It meant creating test cases based on the requirements and thus get an indirect review of both testability and (to large extent) completeness and refinement. The creation of test cases based on the requirements was not adding any work effort to the development as this was to be done in any case, but usually at a much later stage (i.e. as implementation is close to completion).

The testability review was not deemed possible to perform during the continuous part, but was placed in the dedicated part of the process. The main reason for this was not to waste effort on requirements not yet planned for development, i.e. only the Function/Component Level requirements allocated to a project would be subjected to the review.

This meant that testability could not be assured for all Function/Component Level requirements in RAM requirements "repository", but rather only for the ones prioritized enough to be allocated to a project. As a general rule was set that, a requirement on Function/Component Level should strive to be testable, but no formal review would be conducted until project allocation (i.e. performed during the dedicated part).

In addition to the division of the requirements engineering into two parts, upper management identified product strategies as an important factor in the context of performing requirements engineering with RAM. The general view was that the creation of explicitly defined product strategies would allow better control over product development, as well as tie development efforts (projects) closer to the goals for the product (and indirectly the organization).

5.2. Dynamic Validation

As a candidate model (RAM v.0.9, see Figure 2) was completed, it was set to be tested in a live development situation (referred to as *dynamic validation* hereafter). The idea was to use the model for actual requirements engineering in order validate its components (e.g. attributes) and if the model was scalable to a real development situation. This involved validating several aspects of RAM:

1. Abstraction levels
    a. On what abstraction levels did requirements actually come in, and what sources produced what requirements (pertaining to abstraction level)?

    b. Did the work-up create any problems with e.g. too many new requirements (i.e. work-up requirements) created as a result of work-up? This aspect was largely linked to the scalability of RAM.
2. Was it possible to use requirements directly from the model for the intended purposes, i.e.
    a. Were Product Level requirements comparable to product strategies?
    b. Did the Function Level (together with corresponding Component Level) requirements offer enough material for a development project to perform dedicated requirements engineering?
3. Usability, i.e. was RAM easy and intuitive enough to use practically?

The main factor limiting the effectiveness of the dynamic validation were the difficulties of actual performing continuous requirements engineering over a long time-period, as the dynamic validation was limited in time and scope. This meant that the "continuous" part was not really tested, i.e. only one development project was to be initiated as a result of the requirements engineering performed during the dynamic validation. Following a continuous product centered requirements engineering view (as intended when using RAM and described in e.g. Figure 1) the dynamic validation would have to be performed over a much longer period eliciting/catching requirements continuously, and initiation several development projects as needed (based on the requirements).

However, the consensus at DHR was that a limited, but nevertheless *real*, test of RAM would produce a clear indication as to the applicability of the model to solve the problems described in Section 5, as well as answer the questions posed above (see questions 1-3).

Looking at the process of the dynamic validation the Product Manager role (at DHR) corresponded to the Requirement Manager role in RAM (see Section 4.4). Two persons in cooperation conducted the actual requirements engineering work performed with RAM, i.e. the normal Requirements Manager role of RAM was divided during the validation between a DHR Product Manager (driving the work) and the researcher (collection/support). In reality, the researcher's role was not only collection, but also cooperation and collaboration to some extent. The main idea was for a professional in industry to use RAM in a requirements engineering situation, and by doing so involve other parts of the organization described by RAM, e.g. Requirement Owner and Requirement Source, but also e.g. system experts and developers as needed to complete the tasks. The guarantee for scalability of RAM resided in this fact, i.e. that seasoned professionals used the model for a real requirements engineering effort, and could estimate scalability based on their experience. The researcher's role was to support and help in the work.

The dynamic validation (requirements engineering effort) was centered on the development of a new small product needed at DHR. It was set up to be as close to a real market driven continuous requirements engineering effort as possible (but focused in length). Figure 7 illustrates the set-up. Requirements were caught /elicited from multiple sources both internal and external. Each requirement was specified, placed, and worked-up in turn. The stop condition for the dynamic validation was when there were adequate requirements in RAM repository for one development project to be initiated. In a continuous requirements situation the requirements engineering with RAM would never stop, but continue initiating project after project over time as the requirements came in.

During the dynamic validation the dedicated part of the requirements engineering (performed in the project) was largely omitted as the aim was to validate RAM v.0.9 (supporting the Product Manager during the continuous part), however a number of testability reviews were performed. The motivation for this was to check Function/Component Level requirements (as they were delivered to the development project) for testability, completeness and ambiguity since the formal review ensuring these aspects was moved to the dedicated requirements engineering (see Section 5.1.1, Static Validation Two).

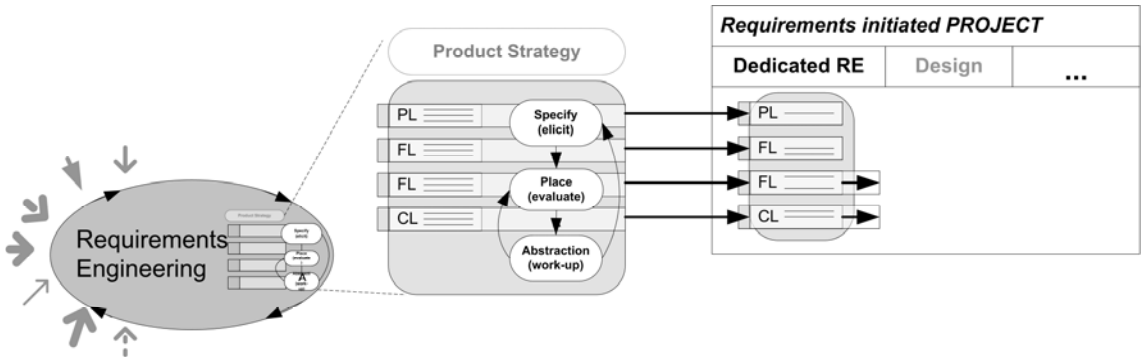


Figure 7. Requirements engineering during dynamic validation.

The role of Requirement Source was represented by a diverse group of different internal stakeholders with varying agendas. Some of these internal stakeholders (primarily marketing and sales personnel) represented external parties during the dynamic validation (as they often are pertaining to the "normal" case in everyday work).

5.2.1. Dynamic Validation - Lessons Learned

Below each of the questions posed in Section 5.2 (i.e. question 1 to 3 with sub-questions) is discussed based on the experiences from the dynamic validation of RAM.

Looking at the requirements engineering performed, 78 original requirements were elicited/caught before Product Management considered a development effort possible.

1.a. As the original requirements came in, the distribution over abstraction levels was diverse, as can be seen in Figure 8. In the center of Figure 8 the incoming requirements' distribution over abstraction levels can be viewed as percentages of the total amount of requirements. To the left in the figure the sources (grouped) of the requirements can be seen along with a percentage indicating their relative contribution of the total amount of requirements on each abstraction level. E.g. 6% of the requirements came in (were placed) on Product Level, and the two sources (Customer and Management) had an equal (50%-50%) share of these requirements.

As the dynamic validation was a limited test-run of RAM, and the organization had limited infrastructure to register requirements sources, some grouping was necessary. The grouping is based on general source, e.g. the group "Developers" is populated by all requirements elicited/caught from the development department, regardless of e.g. who the developer was. In the same way, the "Management" group consists of requirements from all management sources (whether it was upper or middle management). The Partner group was elicited indirectly through the marketing department, as direct access was limited, as was the case in most Customer group cases.

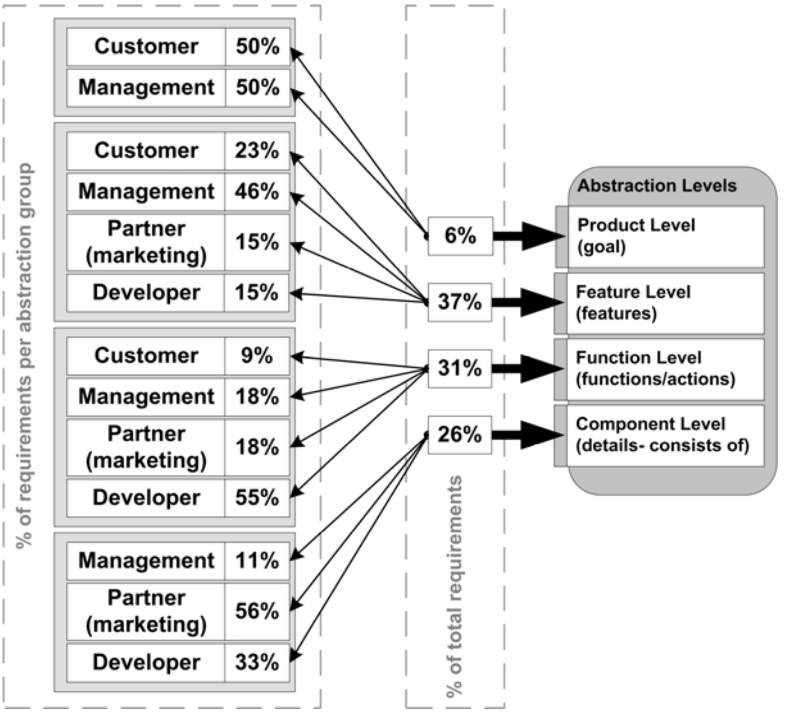


Figure 8. Requirements distribution over sources and abstraction levels.

The distribution over the abstraction levels was relatively even except for the Product Level. This was not surprising as the Product Level only holds relatively abstract requirements, and not statements such as "the product should support" which was the most common type (as indicated by 37% at Feature Level). The abstract nature of requirements on the Product Level means that in reality the requirements will result in many more requirements on lower levels when the requirements are broken down. On the one hand, this means that the percentage figures are not comparable. On the other hand, the inability to compare between different levels of abstraction is one of the main motivations for the development of the model.

Most of the requirements came in on Feature Level, not testable or broken down enough for development, but feature focused, as were the Customer and Management groups that dominated this abstraction level. There was a limited introduction of requirements from both the Partner and Developer groups on Feature Level as well.

The Developer group stated most of the requirements on Function Level, and had a fair share on Component Level as well. This is not surprising, as developers generally are perceived as more hands-on and not prone to abstract non-verifiable statements. The Partner group consisted of requirements were the source could be derived to industry partners which used DHR's products as components in their own products. The partners' high level of technical expertise and insight into DHR product's technical detail probably led to their high representation on Component Level.

It should be noted is that the Management group has relatively high representation on the low abstraction levels. This might be a result of that many management representatives are experienced engineers, well versed in technical aspects, and have been with the organization for a large number of years.

In regards to the distribution, it was seen as beneficial that RAM supported varying levels of abstraction for initial placement of requirements since incoming requirements varied a great deal regarding abstraction level. The alternative was to put all requirements into one single repository, regardless of abstraction level, as was largely done prior to RAM.

1.b. The total number of requirements after work-up was about twice as many as the original requirements, i.e. for every original requirement in average one work-up requirement was created. The original requirements that came in on Product Level (6%) were relatively few in comparison, but they resulted in the creation of several work-up requirements, more so than original requirements coming in on lower levels of abstraction. I.e. the product impact of the requirements on high abstraction levels (Product Level) was substantial even if the amount of original requirements coming in on this level was relatively low.

The main issue with creating work-up requirements (new requirements as a result of work-up) was knowing when to stop. Satisfying the work-up rules (R1 and R2) were not the real problem. The main challenge was to avoid inventing new and not completely relevant requirements. Relevance in this case was staying true to the original requirement and only perform the work-up that was necessary to satisfy it. The Product Manager felt that it could be easy to lose focus and create additional requirements that were semi-relevant, e.g. adding functionality that could be "good to have", but was not sought after when looking at the original requirement and the Requirement Source.

The main experiences from the work-up was that staying true to the original requirements was very important, and probably a matter of experience in using RAM. The doubling of the total amount of requirements as a result of work-up was not considered a problem since the benefits outweighed the costs. The benefits were not only comparison to strategies and producing good-enough requirements for development, but also that the work-up process itself gave a better overview of the requirements, and allowed for a better holistic view. In addition, the increase in effort, despite the creation of work-up requirements, was considered moderate in comparison to the requirements engineering performed before the use of RAM. In conclusion, the work-up (i.e. abstraction and breakdown) of requirements was considered beneficial in terms of that it implied analysis, refinement, and completion of requirements in a structured way.

2.a. As the requirements were abstracted upwards, it was possible to compare most of the requirements to the product strategies. It should be noted that some of the product strategies were not in place prior to the requirements engineering (as the product was new to the organization this was rather expected). However, the benefit was that explicit decisions had to be made during the requirements engineering as to what requirements were to be dismissed and what were not. This activity can be seen as a "reverse engineering" of product strategies, but it can also be seen as the creation (or rather explicit specification) of product strategies that are lacking, using requirements as a road-map to what is needed. Either way, as the strategic decisions were made they offered the possibility to dismiss requirements that were out of scope at an early stage, and keeping resources spent on irrelevant requirements at a minimum.

The main concern was for a need for management to establish routines for creating and re-creating (e.g. adding to/reformulating) product strategies continuously, as needed, to support the continuous requirements engineering. A lack of these routines was not a direct problem during the dynamic validation as the scope of the requirements engineering effort was limited, and did not influence critical core products.

2.b. As the requirements engineering come close to its stop condition (adequate requirements for project initiation) a requirements review was performed in order to assess if the requirements on Feature Level and Component Level were good-enough for use in a project and dedicated requirements engineering. In total ten requirements were reviewed on Feature Level (chosen by random). Three of these were considered as testable and unambiguous right away. The other seven were in need of some additional work before testability could be assured.

During the review, all ten requirements were considered good-enough as input to the dedicated requirements engineering. There was adequate material present (looking at the entire abstraction chain) to continue the work of refinement and completion before the design stage.

The input to the project planning effort was also improved as all requirements were broken down to a level were greater accuracy regarding resource cost could be achieved, especially in comparison to having a mix of abstract and detailed technical requirements as a basis for project planning and initiation.

It should be noted that some communication with the Product Manager and minor refinement (reformulation) of requirements would have been necessary if the requirements should have been good-enough in terms of being unambiguous and testable. This was deemed as acceptable and considered a substantial improvement over previous input to projects.

3. The overall impression of using RAM was positive. The action steps of specification, placement, and work-up were performed without much trouble, and if supported by relevant examples rather intuitive. As the dynamic validation was the first attempt to actually use RAM there were of course some problems, mainly pertaining to coming up with relevant examples and figuring out strategy related issues. However, the model helped in raising number of issues early instead of leaving more uncertainties to development. Thus, the model is also indirectly a risk mitigation tool.

The material produced by using RAM was considered more than adequate, as well as offering it on an even level, i.e. as the requirements were specified and worked-up in the same way, most requirements were equally refined and comparable (on the same level of abstraction).

The potential problems with the model were not directly associated with RAM, but rather the infrastructure needed to support RAM. This included issues mentioned before, e.g. product strategies, but also issues such as adequate tool support and organizational infrastructure (traceability to original requirement source), and so on.

Looking at the dynamic validation as a first live run of RAM, the overall usability of the model was considered more than adequate by the Product Manager using RAM, as were the results produced by RAM.

6. Conclusions

The Requirements Abstraction Model was developed in response to direct needs identified in industry, and the lack of an appropriate model to address these needs. The goal was to offer Product Managers a model supporting the ability to handle and work with requirements on multiple levels of abstraction in a continuous product centered requirements engineering effort. This meant going from e.g. one repository where all requirements were kept, regardless of abstraction level, to a structure mirroring the reality of the requirements coming in.

RAM enforces work-up rules that abstracts and breaks down requirements as needed, offering requirements on several levels of abstraction reflecting the needs of a development organization. Product Level requirements can be compared to product strategies, in an attempt to dismiss out of scope requirements at an early stage. Function and Component Level requirements (needed for project initiation) effectively assure that developers are not burdened with requirements too abstract for development. In addition, working with the model gave a homogenous refinement and analysis of requirements, and the effect that several issues, normally left to projects, were caught early on. This mitigating the risk of some problems creeping into development, and thus caught only after the development effort (e.g. project) was planned and initiated.

As requirements engineering is an integrated part of development the effective use of RAM is dependent on certain infrastructure being in place, this pertains mainly to explicitly formulated product strategies, which existence cannot be taken for granted. As parts of the benefit is dependent on this fact it could be considered a drawback, however it should also be noted that the abstraction of requirements can trigger explicit questions, challenging management to refine product goals and strategies to reflect the needs of the development organization.

All parties working with development (management in particular) can compare requirements, as they are homogenous regarding abstraction level, and are not forced to decide between an abstract requirement and a detailed one as e.g. planning and prioritization activities are performed.

As a part of its development, RAM was validated in industry through both static validations, giving feedback from professionals working with requirements engineering and product development, and through dynamic validation, giving a real live test-run of the model. The validations were performed to assure that the model complied with the needs of industry.

During the validations it was ascertained that requirements did come in on different levels of abstraction, and that specification, placement, and work-up were feasible in a real live requirements engineering situation. The usability of the model was premiered in its development, and was partly assured during the static validation, and tested during the dynamic validation.

As RAM is not prescriptive in nature, but rather adaptable and example-driven, tailoring towards a product (or organization) may be required prior to use in order to develop support materials like guides and relevant examples. The main stakeholder pertaining to the model is the Requirements Manager (e.g. a Product Manager), and as the model should be tailored to support the work performed, the work performed must adhere to work-up rules, but also the consistency regarding abstraction levels is critical and how requirements are handled in relation to these rules. The main point is not having a certain requirement of a certain abstraction on a particular level, but rather having all requirements of a certain abstraction on the *same* level. This (i.e. adequate usage of the model) is obtained through supporting users with guides, relevant examples, and explicitly formulated product strategies, but also through training in model usage prior and during model implementation in an organization. Issues such as the number of abstraction levels needed is up to the organization in question and their particular case, and is a part of the tailoring.

Requirements specified using RAM were based on attributes validated against industry professionals, and were considered adequate in amount and detail pertaining to fulfilling the functions needed. The usability was premiered in the models development, but not at the expense of substantially lowering the quality of the requirements produced. This was assured through the validity reviews performed.

As stated earlier, RAM presented in this paper was designed with Product Managers in mind, supporting them in their requirements engineering effort producing requirements good-enough for planning activities, giving detailed abstraction as well as a big picture view. However, it was also developed with engineers (developers) in mind, controlling the abstraction level and refinement of requirements handed over to projects. The motivation for RAM was to address needs identified at DHR, these same needs that were later also confirmed by a second (independent) development organization, and their explicit interest in tailoring RAM to their products.

The non-prescriptive, adaptable nature of the model is based on the assumption that a perfect model and way of working cannot be specified, not even for one organization with homogenous products, since the products and needs of the organization may change over time. RAM can be tailored to satisfy the needs of different organizations and products. In the same way, it can be modified over time to reflect the current situation of a development organization, supporting the collection of requirements over time and product life cycle.

7. Future Work – RAM Validation and Evolution

RAM v.1.0, as described in this paper, was aimed at assuring that requirements be specified on multiple abstraction levels in a repeatable and homogenous way, enabling e.g. product managers in early requirements engineering efforts. This was considered a prerequisite for subsequent activities, i.e. requirements estimation, risk analysis, prioritization and packaging of requirements taking e.g. coupling between requirements into consideration. The next phase that will be undertaken in the validation and evolution of RAM can be described as a two-step process. First, a controlled empirical evaluation in a lab environment will be performed. This evaluation will be designed to further test the concepts behind RAM in an experimental setting, prior to industry trials. The goal for this evaluation will be to give important feedback regarding the usability and usefulness of RAM without spending valuable industry resources, but nonetheless enabling further improvements on the model's concepts, identification of potential weaknesses, and collection metrics regarding the usage of RAM. All information gathered during this lab evaluation will be used as input to the next step, the tailoring and large scale piloting of RAM in industry. The present plan is for RAM to be tailored to, and subsequently piloted in, two organizations, DanaherMotion Särö AB and ABB Automation Technology Products. Data, both qualitative and quantitative, gathered during the pilots will act as decision support material for further model improvements, but also as input to the organizations when considering the full scale adoption of a tailored version of RAM in their product planning and development process.

In addition, the natural evolution of RAM will continue, incorporating explicit support for requirements estimation, risk analysis, prioritization and packaging of requirements into development instances. The work name for this endeavor is RAM v.2.0. Requirements engineering closer to and within projects will be incorporated. The goal will be to create one usable and flexible lightweight model that cover requirements engineering activities performed during product planning and management activities, but also within and post projects, thus effectively addressing most of the issues identified in industry (see e.g. Improvement Package Two and Three in [21]).

8. Acknowledgements

The work and results presented in this paper were done and produced in close and fruitful cooperation between research and industry. The authors would like to thank everyone involved during the SPI work at DHR, and especially the members of the SPI work team for their commitment and tireless work.

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project "Blekinge - Engineering Software Qualities (BESQ)" (http://www.ipd.bth.se/besq).

9. References

1. Ruhe G, Greer D (2003) Quantitative Studies in Software Release Planning under Risk and Resource Constraints. In Proceedings of International Symposium on Empirical Software Engineering (ISESE), IEEE, Los Alamitos CA, pp. 262-271
2. Butscher SA, Laker M (2000) Market-Driven Product Development. Marketing Management 9:48-53.
3. Sommerville I (2001) Software Engineering. Addison-Wesley, Essex, UK
4. Carlshamre P (2002) Release Planning in Market-Driven Software Product Development: Provoking an Understanding. Requirements Engineering 7:139-151.
5. Yeh AC (1992) Requirements Engineering Support Technique (Request): A Market Driven Requirements Management Process. In Proceedings of the Second Symposium on Assessment of Quality Software Development Tools, pp. 211-223
6. Carlshamre P, Sandahl K, Lindvall M, Regnell B, Natt och Dag J (2001) An Industrial Survey of Requirements Interdependencies in Software Product Release Planning. In Proceedings Fifth IEEE International Symposium on Requirements Engineering, IEEE, Los Alamitos CA, pp. 84-92
7. Dahlstedt Å, Persson A (2003) Requirements Interdependencies - Moulding the State of Research into a Research Agenda. In Ninth International Workshop on Requirements Engineering (REFSQ '03), Klagenfurt/Velden, Austria, pp. 71-80
8. Regnell B, Host M, Natt och Dag JBP, Hjelm T (2001) An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software. Requirements Engineering 6:51-62.
9. Greer D, Ruhe G (2004) Software Release Planning: An Evolutionary and Iterative Approach. Information and Software Technology 46:243-253.
10. Weber M, Weisbrod J (2003) Requirements Engineering in Automotive Development: Experiences and Challenges. IEEE Software 20:16-24.
11. Higgins SA, de Laat M, Gieles PMC (2003) Managing Requirements for Medical It Products. IEEE Software 20:26-34.
12. Potts C (1997) Requirements Models in Context. In Proceedings of the Third IEEE International Symposium on Requirements Engineering, IEEE, Los Alamitos CA, pp. 102-104
13. Potts C, Hsi I (1997) Abstraction and Context in Requirements Engineering: Toward a Synthesis. Annals of Software Engineering 3:23-61.
14. Anton AI (1996) Goal-Based Requirements Analysis. In Proceedings of the Second International Conference on Requirements Engineering, IEEE, Los Alamitos CA, pp. 136-144
15. Kavakli E, Loucopoulos P, Filippidou D (1996) Using Scenarios to Systematically Support Goal-Directed Elaboration for Information System Requirements. In Proceedings of IEEE Symposium and Workshop on Engineering of Computer-Based Systems, IEEE, Los Alamitos CA, pp. 308-314
16. Castro J, Kolp M, Mylopoulos J (2002) Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems 27:365-389.
17. Wiegers KE (1999) Software Requirements. Microsoft Press, Redmon, Wash.
18. Lauesen S (2000) Software Requirements : Styles and Techniques. Samfundslitteratur, Fredriksberg
19. Robertson S, Robertson J (1999) Mastering the Requirements Process. Addison-Wesley, Harlow
20. Gorschek T, Wohlin C (2003) Identification of Improvement Issues Using a Lightweight Triangulation Approach (Eurospi'03). In European Software Process Improvement Conference, Verlag der Technischen Universität, Graz, Austria, pp. VI.1-VI.14
21. Gorschek T, Wohlin C (2004) Packaging Software Process Improvement Issues - a Method and a Case Study. Software - Practice & Experience in press
22. Sawyer P, Sommerville I, Viller S (1999) Capturing the Benefits of Requirements Engineering. IEEE Software 16:78-85.
23. Sommerville I, Sawyer P (1999) Requirements Engineering : A Good Practice Guide. Wiley, Chichester, Eng. ; New York
24. Pfleeger SL (1998) Software Engineering : Theory and Practice. Prentice-Hall International, London

25. **Martin S, Aurum A, Jeffery R, Barbara P (2002) Requirements Engineering Process Models in Practice. In Seventh Australian Workshop on Requirements Engineering (AWRE'02), Melbourne, Australia, pp. 141-155**
26. **Juristo N, Moreno AM, Silva A (2002) Is the European Industry Moving toward Solving Requirements Engineering Problems? IEEE Software 19:70-78.**
27. **Gorschek T, Svahnberg M, Tejle K (2003) Introduction and Application of a Lightweight Requirements Engineering Process Evaluation Method. In Requirements Engineering: Foundation for Software Quality (REFSQ'03), Velden, Austria, pp. 101-112**
28. **Kivisto K (1999) Roles of Developers as Part of a Software Process Model. In Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences, IEEE Computer Soc., pp. 1-19**
29. **Humphrey WS (2000) Introduction to the Team Software Process. Addison-Wesley, Reading, Mass.**
30. **Fritzhanns T, Kudorfer F (2002) Product Management Assessment - a New Approach to Optimize the Early Phases. In Proceedings of IEEE Joint International Conference on Requirements Engineering, IEEE, Los Alamitos CA, pp. 124-134**
31. **Rakitin SR (2001) Software Verification and Validation for Practitioners and Managers. Artech House, Boston, Mass. ; London**
32. **Lehmann DR, Winer RS (2002) Product Management. McGraw-Hill, Boston, Mass.**
33. **Kotonya G, Sommerville I (1998) Requirements Engineering : Processes and Techniques. John Wiley, New York**
34. **Robson C (2002) Real World Research : A Resource for Social Scientists and Practitioner-Researchers. Blackwell Publishers, Oxford, UK ; Madden, Mass.**
35. **Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in Software Engineering : An Introduction. Kluwer Academic, Boston**

Appendix A

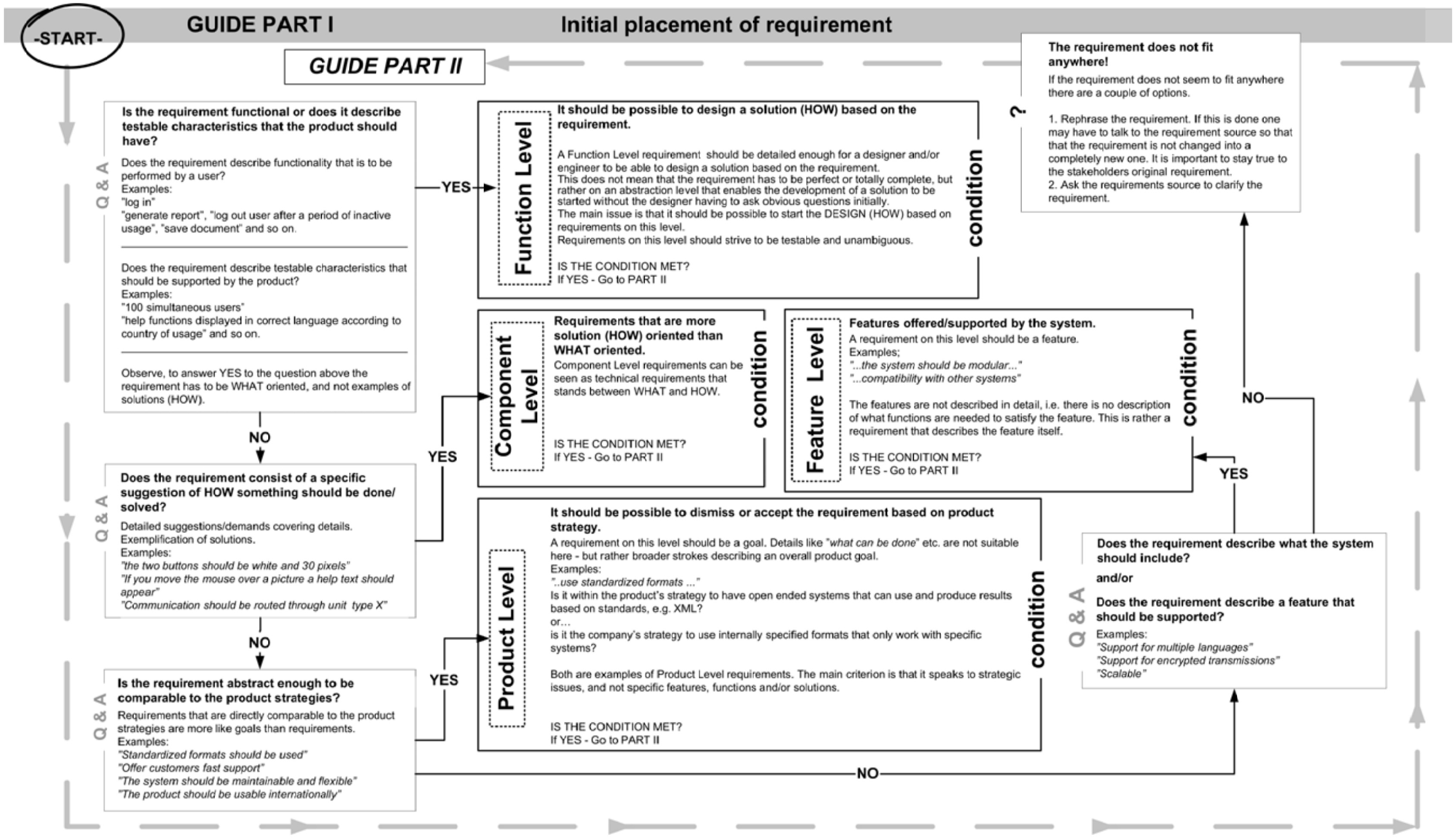
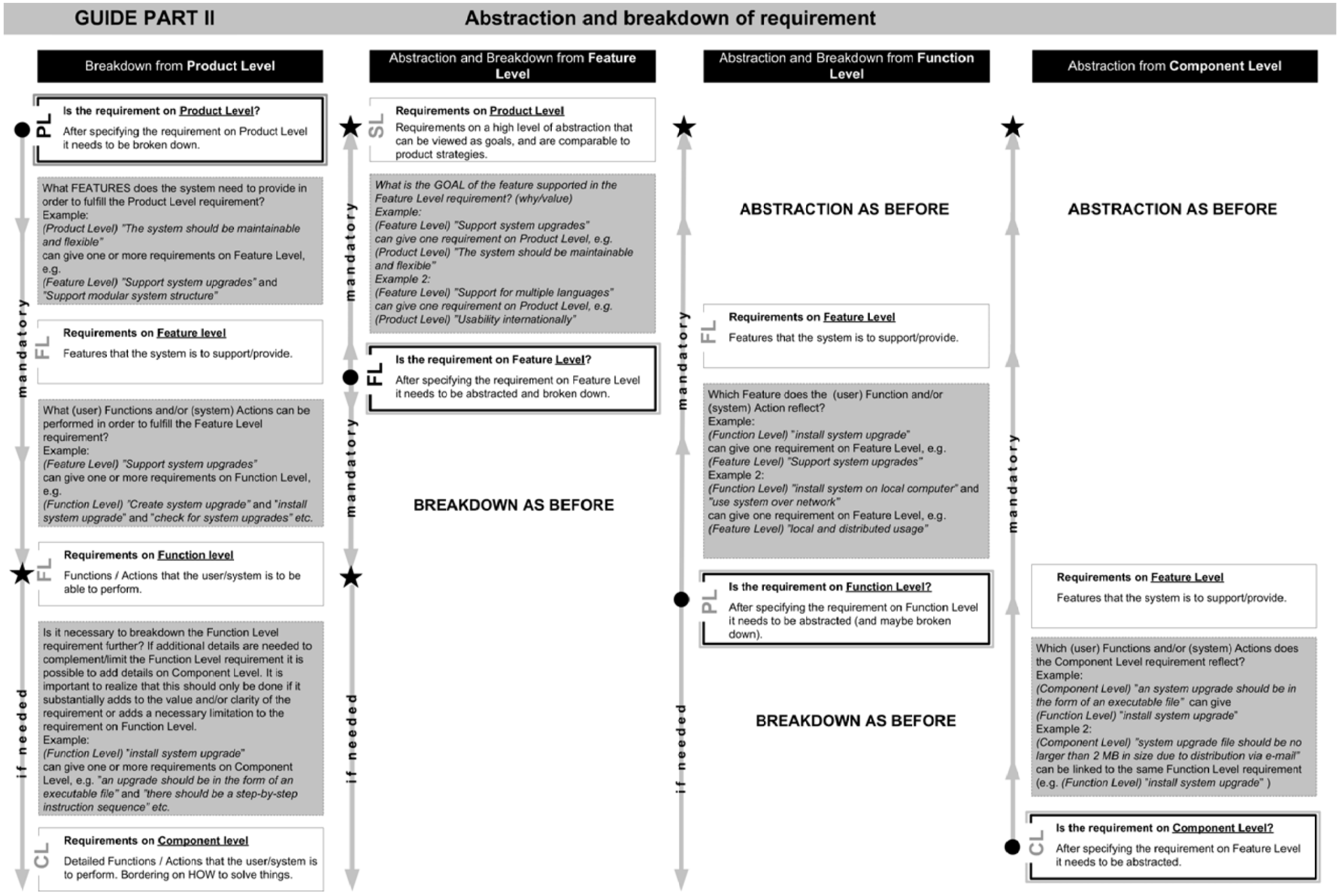

-START-
GUIDE PART I
Initial placement of requirement
GUIDE PART II
Is the requirement functional or does it describe testable characteristics that the product should have?
Q & A
Does the requirement describe functionality that is to be performed by a user?
Examples:
"log in"
"generate report", "log out user after a period of inactive usage", "save document" and so on.
Does the requirement describe testable characteristics that should be supported by the product?
Examples:
"100 simultaneous users"
"help functions displayed in correct language according to country of usage" and so on.
Observe, to answer YES to the question above the requirement has to be WHAT oriented, and not examples of solutions (HOW).
YES
NO
Does the requirement consist of a specific suggestion of HOW something should be done/ solved?
Q & A
Detailed suggestions/demands covering details.
Exemplification of solutions.
Examples:
"the two buttons should be white and 30 pixels"
"If you move the mouse over a picture a help text should appear"
"Communication should be routed through unit type X"
YES
NO
Is the requirement abstract enough to be comparable to the product strategies?
Q & A
Requirements that are directly comparable to the product strategies are more like goals than requirements.
Examples:
"Standardized formats should be used"
"Offer customers fast support"
"The system should be maintainable and flexible"
"The product should be usable internationally"
YES
NO
Function Level
It should be possible to design a solution (HOW) based on the requirement.
A Function Level requirement should be detailed enough for a designer and/or engineer to be able to design a solution based on the requirement.
This does not mean that the requirement has to be perfect or totally complete, but rather on an abstraction level that enables the development of a solution to be started without the designer having to ask obvious questions initially.
The main issue is that it should be possible to start the DESIGN (HOW) based on requirements on this level.
Requirements on this level should strive to be testable and unambiguous.
IS THE CONDITION MET?
If YES - Go to PART II
condition
Component Level
Requirements that are more solution (HOW) oriented than WHAT oriented.
Component Level requirements can be seen as technical requirements that stands between WHAT and HOW.
IS THE CONDITION MET?
If YES - Go to PART II
condition
Feature Level
Features offered/supported by the system.
A requirement on this level should be a feature.
Examples;
"...the system should be modular..."
"...compatibility with other systems"
The features are not described in detail, i.e. there is no description of what functions are needed to satisfy the feature. This is rather a requirement that describes the feature itself.
IS THE CONDITION MET?
If YES - Go to PART II
condition
Product Level
It should be possible to dismiss or accept the requirement based on product strategy.
A requirement on this level should be a goal. Details like "what can be done" etc. are not suitable here - but rather broader strokes describing an overall product goal.
Examples:
"..use standardized formats ..."
Is it within the product's strategy to have open ended systems that can use and produce results based on standards, e.g. XML?
or…
is it the company's strategy to use internally specified formats that only work with specific systems?
Both are examples of Product Level requirements. The main criterion is that it speaks to strategic issues, and not specific features, functions and/or solutions.
IS THE CONDITION MET?
If YES - Go to PART II
condition
Does the requirement describe what the system should include?
and/or
Does the requirement describe a feature that should be supported?
Q & A
Examples:
"Support for multiple languages"
"Support for encrypted transmissions"
"Scalable"
YES
NO
NO
?
The requirement does not fit anywhere!
If the requirement does not seem to fit anywhere there are a couple of options.
1. Rephrase the requirement. If this is done one may have to talk to the requirement source so that that the requirement is not changed into a completely new one. It is important to stay true to the stakeholders original requirement.
2. Ask the requirements source to clarify the requirement.

Figure 9. RAM how-to guide part I - initial placement of requirements.



GUIDE PART II

Abstraction and breakdown of requirement

Breakdown from Product Level

PL **Is the requirement on Product Level?**
After specifying the requirement on Product Level it needs to be broken down.

What FEATURES does the system need to provide in order to fulfill the Product Level requirement?
Example:
(Product Level) "The system should be maintainable and flexible"
can give one or more requirements on Feature Level, e.g.
(Feature Level) "Support system upgrades" and *"Support modular system structure"*

FL **Requirements on Feature level**
Features that the system is to support/provide.

What (user) Functions and/or (system) Actions can be performed in order to fulfill the Feature Level requirement?
Example:
(Feature Level) "Support system upgrades"
can give one or more requirements on Function Level, e.g.
(Function Level) "Create system upgrade" and *"install system upgrade"* and *"check for system upgrades" etc.*

FL **Requirements on Function level**
Functions / Actions that the user/system is to be able to perform.

Is it necessary to breakdown the Function Level requirement further? If additional details are needed to complement/limit the Function Level requirement it is possible to add details on Component Level. It is important to realize that this should only be done if it substantially adds to the value and/or clarity of the requirement or adds a necessary limitation to the requirement on Function Level.
Example:
(Function Level) "install system upgrade"
can give one or more requirements on Component Level, e.g. *"an upgrade should be in the form of an executable file"* and *"there should be a step-by-step instruction sequence" etc.*

CL **Requirements on Component level**
Detailed Functions / Actions that the user/system is to perform. Bordering on HOW to solve things.

Abstraction and Breakdown from Feature Level

SL **Requirements on Product Level**
Requirements on a high level of abstraction that can be viewed as goals, and are comparable to product strategies.

What is the GOAL of the feature supported in the Feature Level requirement? (why/value)
Example:
(Feature Level) "Support system upgrades"
can give one requirement on Product Level, e.g.
(Product Level) "The system should be maintainable and flexible"
Example 2:
(Feature Level) "Support for multiple languages"
can give one requirement on Product Level, e.g.
(Product Level) "Usability internationally"

FL **Is the requirement on Feature Level?**
After specifying the requirement on Feature Level it needs to be abstracted and broken down.

BREAKDOWN AS BEFORE

Abstraction and Breakdown from Function Level

ABSTRACTION AS BEFORE

FL **Requirements on Feature Level**
Features that the system is to support/provide.

Which Feature does the (user) Function and/or (system) Action reflect?
Example:
(Function Level) "install system upgrade"
can give one requirement on Feature Level, e.g.
(Feature Level) "Support system upgrades"
Example 2:
(Function Level) "install system on local computer" and *"use system over network"*
can give one requirement on Feature Level, e.g.
(Feature Level) "local and distributed usage"

PL **Is the requirement on Function Level?**
After specifying the requirement on Function Level it needs to be abstracted (and maybe broken down).

BREAKDOWN AS BEFORE

Abstraction from Component Level

ABSTRACTION AS BEFORE

Requirements on Feature Level
Features that the system is to support/provide.

Which (user) Functions and/or (system) Actions does the Component Level requirement reflect?
Example:
(Component Level) "an system upgrade should be in the form of an executable file" can give
(Function Level) "install system upgrade"
Example 2:
(Component Level) "system upgrade file should be no larger than 2 MB in size due to distribution via e-mail"
can be linked to the same Function Level requirement (e.g. *(Function Level) "install system upgrade"*)

CL **Is the requirement on Component Level?**
After specifying the requirement on Feature Level it needs to be abstracted.

Figure 10. RAM how-to guide part II – Abstraction and breakdown of requirement.