Managing Requirements for Medical IT Products

Stewart A. Higgins, Maurice de Laat, Paul M.C. Gieles, and Emilienne M. Geurts, Philips Medical Systems

Philips Medical Systems produces products for diagnosis and treatment of patients by healthcare professionals. Our Mimit (Medical Imaging Information Technology) group is responsible for IT products within this portfolio. Examples include products for viewing, analyzing, printing, archiving, and communicating medical images. We also produce systems that control workflow and administration of patient data.

In the past, our team wrote many requirements documents at the start of a release project. This approach sometimes led to a bottleneck because the stakeholders who provided the content of product requirements were also needed for testing at the end of the previous release project. Our current approach recognizes that there is a continuous stream of new ideas for requirements from the outside world.1-3 We have built a process, supported by tools, in which we capture and refine requirements as they come in. In this way, we pool requirements ready for allocation to release projects. The requirements valid for the next release project are published in a document for that release. This process involves requirements analysts working closely with requirements stakeholders.

The environment

Our new process had to fit into the existing environment. Because we produce medical products, particular attention had to be paid to communicating the domain terminology and required functionality to software developers. We had stakeholders from multiple disciplines and requirements from many sources. The process also had to enable regular product releases.

Product families for medical professionals

Our products are for a specific group of medical professionals—for instance, those working in radiology and cardiology. Our application specialists (who often had worked in radiology or cardiology departments) bring medical domain knowledge with them. Requirements analysts increasingly play a bridging role between these clinical and technical specialists.

One of the largest products in our product families4,5 contains approximately 27 software applications running on a common
Each application supports general viewing or specific clinical tasks. In practice, application functionalities, such as letting users magnify or print images, can appear in more than one application. Commonalities like these allow considerable scope for reusing requirements.

Requirements from multiple disciplines and sources

Many stakeholders with different viewpoints bring requirements to the table. Internal stakeholders include:

- Product managers (commercial)
- Application specialists (medical)
- Service engineers (customer support)
- Architects and software engineers (technical)

Logistics and manufacturing, infoware engineers (user documentation specialists), testers, and imposed standards are also important sources of both requirements and design constraints. The issues that can arise from dealing with requirements from multiple perspectives have been the subject of considerable research. Handling inconsistencies is particularly important.6

Requirements also come from many other sources, such as:

- Demonstrations of existing products to potential customers
- User evaluation of prototypes
- Enhancement requests from daily users of existing products
- Radiology and cardiology trade shows
- Observations during hospital visits
- Imposed standards such as legal constraints
- Clinical literature
- Suppliers
- Knowledge of competitor products

Regular product releases

Some literature on the subject of requirements analysis contains the assumption that a completely new product will be made for a specific customer.7 But we work in an environment of ongoing product development where new versions are brought to market via regular product releases. This means that we have to respect the historical evolution of the products and the processes that produced them while improving the requirements management process.

The baseline

When we started our improvements, the process in place used a document-centered approach and no requirements management tools.

The internal stakeholders link us to markets and users—directly or indirectly—via our sales and service divisions. Therefore, part of their job is to make the organization aware of new requirements from those sources. At the start of a new project (during the feasibility phase), all disciplines traditionally wrote requirements documents from their own perspective.

Product managers wrote a commercial requirement specification that contained not only business information but also the commercial view on which functions should be included. The CRS also included requirements from other disciplines such as customer support, logistics, and manufacturing. Normally product managers wrote only one CRS for the product family per release.

Application specialists wrote application requirements specifications, which contained clinical information and the application view on which functionality should be included. The specialists usually wrote one ARS document for each package in this product family.

The requirements in the CRS and ARS documents were written in a “free” style. That is, requirements, their rationale, and other contextual information tended to be mixed together. Also, several requirements might be combined into one sentence or paragraph. Because no explicit requirements tagging was in place, it was not always clear to readers which text contained the requirements and which was just additional or background information.

These CRS and ARS requirements documents comprised the product requirements. Due to resource and time constraints, however, the responsible stakeholders accepted only a subset of the product requirements for a particular release. Some valid requirements remained in these documents for future releases. When we began a new release while still testing the previous one, we had to pull requirements from the documents and decide whether they were still valid.
Figure 1 shows the resource bottleneck that can occur when a new project starts and the previous one slips.

**Numerous requirements documents**

In response to the CRS and ARS documents, the development part of the organization wrote a *system requirements specification* (the contract between the business and development disciplines) and detailed *functional requirements specifications* grouped per application. The CRS was the global request document from the business disciplines and the SRS the global answer document from the development organization. The multiple ARS documents were the detailed request documents, and the FRS documented the detailed answer.

Figure 2 illustrates the requirements documents that were in place and the cross-referencing among them. A release project in one of our larger families contained one CRS, 16 ARS, one SRS, and 28 FRS documents—a total of 46.

**Our approach**

We began a step-by-step process to introduce change in an evolutionary fashion that allowed for fine-tuning our approach as we went.

**Step 1: Manage the requirements in existing documents**

We focused on requirements from the business perspective. We tried to improve the process for reaching the initial CRS and ARS product requirements as well as tracing them to the SRS. Once the SRS is written, the business disciplines want to know which CRS requirements the SRS has answered and which remain and need to be further discussed or preserved for the next release. To support traceability between the CRS and SRS, we put both sets of requirements into a commercial *requirements management tool*. This tool not only let us view the requirements in their original documents but also allowed database views, including attribute matrices, traceability tables, and traceability trees. We could then introduce the idea of such a tool without changing existing documents.

The tool let stakeholders immediately see the links between what was asked and what was answered. It also aided discussion about unanswered requirements. It clarified which additional (usually more technical) requirements had been added directly to the SRS without a corresponding request in the CRS. This capability increased stakeholder confidence in the approach.

**Step 2: Write more easily managed requirements**

The requirements management tool highlighted the importance of writing requirements so they are individual units of information. It also provided a way to more easily separate the requirements from contextual information. For example, when we were prototyping a new version of a product, a requirements analyst facilitated the requirements-gathering process by assuming a more specific role. The analyst worked closely with product managers and applications specialists to act as a link between product requirements and the developers. Our team then wrote the requirements so that we could more easily manage them, using two categories:

- Product requirements (PRQs)
- Product use cases (PUCs)

The PRQs capture static information and are written in natural language in the form “The product shall ...” The requirements analyst checks regularly to ensure adherence to this style and that the requirements exhibit good characteristics such as clarity and conciseness. The PUCs capture dynamic information about how the user interacts with the product. They are written by the analyst with input from other team members and then checked by the team.

In our experience, the PRQs and PUCs should be developed in parallel using an iterative approach. The use cases not only...
help to explain the requirements to developers but also can serve as input to user manuals and testing activities. For example, a PRQ at the highest level might state “The product shall allow the user to annotate and make measurements on images.” Related to this, another, more detailed PRQ might specify “The product shall support a range of fonts.” And an even more detailed PRQ might state which fonts are supported.

In this project, we kept all product requirements on the same level—that is, we did not store the requirements in a parent-child hierarchy. We kept related requirements together in document sections and used a category attribute to filter them in the database. The category attribute’s values corresponded to the document sections. The PRQ example just given would have a document section and a category “Annotation and Measurement.”

In parallel, we developed a PUC describing the user-product interactions for annotation and measurement. As we developed the PUC, we discovered more PRQs. Consequently, we repeated this process until we were sure that we had covered all static and dynamic information of interest to internal stakeholders.

We also carried out most of the requirements work before the new release project started and used simulation and prototyping to visualize the requirements and use cases. By simulation we mean building a mockup using a GUI builder, and by prototyping we mean building parts of the functionality on the target platform in an evolutionary way. In turn, the simulation and prototyping generated new PRQs and PUCs, becoming another iterative process. Among other advantages, having the requirements in the database let us track which requirements had been prototyped by adding a “prototyped” field to the PRQs.

Working in this way, we were ready with the PRQs and PUCs at the beginning of the release project, which helped to resolve the bottleneck shown in Figure 1.

Step 3: Collect a richer set of information

Now that we had structured the requirements so that they could be better managed, our next step was to start collecting richer information about the requirements. For example, we added a rationale for each requirement to more clearly separate the “whats” from the “whys.” The stakeholders found this helpful because it reduced discussion about why a requirement is included. Often the rationale comes from a clinical perspective (why the clinical user needs a certain function), but sometimes it is from a commercial perspective (for example, some hospital IT departments will buy only products that run on a certain operating system).

Another attribute that we find useful, particularly for nonfunctional requirements, is the acceptance criterion, similar to the “Fit Criterion.”

The following PRQ has both a rationale and an acceptance criterion:

- **Requirement:** The product shall automatically display a case as it was last viewed.
- **Rationale:** Workflow improvement: Optimal view need not be reproduced repeatedly.
- **Acceptance criterion:** Manipulate images on screen (zoom/pan/window as well as geometrical manipulations). Exit the case (saving not needed). Retrieve the same case. This case will be displayed as last seen.

Step 4: Manage the requirements in a database and publish documents

To capture various attributes that let us categorize and prioritize product requirements, we started using a *product requirements template*. This was particularly useful in customer workshops, where the template forced us to clearly separate a requirement from its rationale and acceptance criterion. Stakeholders could also use the template to submit their latest requirements ideas.

The first approach we used had a distinct disadvantage. Writing requirements first in a document and then linking to the requirements management database meant that the requirements had to remain in their original documents. This approach did not let us reuse requirements in specification documents for other products. Additionally, we did not want...
to have to wait until someone wrote a full document to include new ideas. Therefore, we now add requirements directly to the database. We also add the attributes from the template. This allows us much more flexibility when we produce the product requirements documents, which we publish from the database. We can also vary the richness of the information printed in the document: In some cases, we might wish to publish only the requirements, and in others we might want to show the rationale and acceptance criterion as well.

Another advantage of putting the requirements directly into the database is that we do not have to decide initially which release the requirements will be allocated to. The product team can decide this when the product requirements are known. Our release-independent approach to building product requirements reflects the fact that requirements are continually coming from the internal stakeholders through their contact with the market and end users. We feel it is better to invest the requirements effort when the requirements are “hot” rather than expecting people to store them in their heads, emails, or trip reports and then recalling them at the start of a project. Thus, we continuously gather requirements and then allocate them to release projects (see Figure 3). Using an attribute in the database, we can publish all the requirements for a particular release in a product requirements specification.

**Step 5: Institutionalize our experiences**

Because of the positive experiences gained from our process improvement, other projects in our Mimit group began to adopt some practices from our pilot projects and started to use the template and the requirements management tool. Of course, each project adapted the practices to suit its needs. To capture best practice and develop a process suitable for all stakeholders in our group, we formed a quality improvement project called Charm (Changing to Active Requirements Management). One objective was to design a template that everyone could use (Table 1 describes the template’s fields). As we gain more experience with it, we expect to fine-

**Table 1**

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The product requirement statement</td>
<td>To improve the requirement's quality</td>
</tr>
<tr>
<td>Rationale</td>
<td>The reason behind the requirement</td>
<td>To provide history and responsibility (owner is assigned by product team)</td>
</tr>
<tr>
<td>Acceptance criterion</td>
<td>A description of how to measure whether the requirement is met</td>
<td>To help categorize the requirement</td>
</tr>
<tr>
<td>Submitter</td>
<td>The person who suggested the requirement</td>
<td></td>
</tr>
<tr>
<td>Submission date</td>
<td>The date on which the requirement was suggested</td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td>Where the requirement came from</td>
<td></td>
</tr>
<tr>
<td>Owner</td>
<td>The person who takes responsibility for the requirement</td>
<td></td>
</tr>
<tr>
<td>Application areas</td>
<td>The clinical areas in which the requirement is needed</td>
<td>To help prioritize the requirement (the product team assigns values of High, Medium, or Low)</td>
</tr>
<tr>
<td>Intended users</td>
<td>Who needs the functionality in their work</td>
<td></td>
</tr>
<tr>
<td>Modalities</td>
<td>Which scanner groups have an interest in the requirement</td>
<td></td>
</tr>
<tr>
<td>Packages</td>
<td>Which parts of the product family would use this requirement</td>
<td></td>
</tr>
<tr>
<td>Business value</td>
<td>Value from the product management perspective</td>
<td></td>
</tr>
<tr>
<td>Clinical value</td>
<td>Value from the application specialist perspective</td>
<td></td>
</tr>
<tr>
<td>Service value</td>
<td>Value from the customer support perspective</td>
<td></td>
</tr>
</tbody>
</table>
tune the set of parameters. This template has proved to be an important tool for collecting stakeholder input in the product requirements process.

We also wanted to develop a process for handling requirements from the time the stakeholder first suggests them until the product team accepts or rejects them. Figure 4 illustrates a small part of this process. First, the submitter enters a suggestion into the template. The requirements analyst then screens the requirement:

- If the analyst decides that it is a problem report on an existing functionality, he refers it to the problem-reporting process for handling.
- If the suggestion is a duplicate of an existing requirement, the product team might approve updating that requirement’s attributes. The requirements management tool keeps a history of changes.
- If the suggestion is not a completely new requirement, it might trigger the analyst to modify an existing requirement after the product team has authorized it.
- If the suggestion is indeed a completely new requirement, it is stored, and the product team must then approve or reject it. (Allocating an approved product requirement to a release project is another part of the process.)

The Charm team also developed a way to measure process improvement. They defined a Requirements Management Maturity model with three levels. Each level has a number of statements related to the requirements themselves and the requirements management process. The first level contains statements corresponding to an initial level of maturity. The second- and third-level statements correspond to successively greater maturity. For example, one of the eight statements for the first level is “Each requirement has a unique identifier.” One of the seven statements for the second level is “Requirements are testable.” One of the five statements for the third level is “Requirements can be reused over projects.” Each statement is scored according to the degree to which it applies:

- 0 (if never)
- 1 (if to a few requirements or it occasionally occurs in the process)
- 2 (if to most requirements or it usually occurs in the process)
- 3 (if to all requirements or it always occurs in the process)

The team calculates, for a given project, the average score over all statements in the model, as well as a score per level for more detailed insight.

The Charm team measures projects quarterly. Figure 5 shows some measurement results—the average score over all statements in the model for each of six projects from our group. The results to the right are more recent than those to the left and reflect that
best practices have been adopted. Note the gradual improvement of the quality of the requirements and the requirements management process.

By improving the product requirements process, we also hoped to influence the time-to-market. Time-to-market is influenced by the project lead time, which consists of two main parts: the time needed to reach an accepted specification (time-to-spec) and the implementation and testing time. To see the impact of our improved process, we measured the percentage slip in time-to-spec by calculating the percentage of difference between the estimated and actual time-to-spec. We call this time-to-spec slip.

Predicting time-to-spec, although difficult, is improved by the availability of higher-quality product requirements at the project’s start. Figure 6 shows how the time-to-spec slip has improved over a period of seven months. We believe that the improved quality of the requirements management process and of the resulting requirements is a significant factor. The time-to-spec slip is a “moving annual average”—it averages the time-to-spec slip over all Mimit projects in the previous year.

As we progressed, we found that hierarchical (parent-child) relationships between the related product requirements became another increasingly useful factor, so we added these relationships. This let us group all related requirements on a subject. If we must remove these requirements from a release, we remove only the parent requirement—all others are automatically removed. This structure also helps with detailed allocation to release projects. Higher-level requirements are more abstract and therefore less subject to change between releases. Lower-level requirements, especially at the leaves of the hierarchical tree, are more detailed. If needs change for the next release, we can retain the original requirement, allocated to the previous releases, and add a new requirement for future releases on the same level.

Lessons learned

In carrying out this work, we have learned several lessons:

- An evolutionary approach is the best way to build stakeholder confidence in new requirements management processes and tools. New elements should be applied initially to suitable projects on a step-by-step basis and in the right order. When these have been tried and tested, they can then be spread to other projects.
- Capturing a richer set of information with a requirement (such as rationale and acceptance criterion) helps stakeholders from multiple disciplines better understand the requirement.
- The requirements analyst’s role should be made explicit in the process. Establishing the roles and responsibilities of the submitters, analysts, and product team contributes to a smoother running process.
- Tools are essential for handling requirements individually. Just a few benefits include automatic allocation of unique IDs, a history for each requirement, attributes for each requirement, and traceability between requirements. Additionally, all requirements information is available in one place and can be made accessible to all stakeholders (for example, via a Web interface).
- Introducing a requirements management tool without a process is ineffective, and introducing a process without tools is difficult. Both must be introduced in parallel, but the tool should always support the process.
- Continuously managing requirements contributes to better-quality requirements and a more predictable time-to-slip at the start of a release project.

We found that our approach improved the quality of the requirements and contributed to a reduction in the time-to-spec slip. It linked the product requirements management process to
some business goals—namely, reducing the time for overall product creation and effectively using human resources. Another outcome is that we have created requirements components to be used in similar products, which benefits our product families.

We also see numerous challenges for the future. We would like to apply our approach to all our development projects for medical IT products. This might mean working with suppliers who have different work cultures and are located in different time zones. We would also like to find the best methods for keeping all stakeholders continuously engaged in the process. For example, although stakeholders can now look directly into the tool to see the status of their requirements, they should probably be kept informed by other means. Achieving the right balance without information overload is the issue. We would also like to achieve the right balance of having enough but not too many attributes per requirement type, since they all need to be maintained.

We would like to collect data to show the relationship between an improved requirements management process and achieving reduced time-to-market, using resources more effectively, and improving product quality.

We also want to determine the optimum scope of the databases. On the one hand, it is possible to have one database per product or product family. On the other hand, it would be useful to store the requirements common to several families only once and then share them among all relevant products.

Finally, we hope to achieve smooth integration of requirements management tools with other tools such as test management tools.

Acknowledgments
We acknowledge Rob Vader’s contribution to setting up the measurement model for the product requirements management process and the Charm team for helping to improve the overall process.

References