A linguistic-engineering approach to large-scale requirements management

Natt och Dag, Johan; Regnell, Björn; Gervasi, V; Brinkkemper, S

Published in:
IEEE Software

DOI:
10.1109/MS.2005.1

2005

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
A Linguistic-Engineering Approach to Large-Scale Requirements Management

Johan Natt och Dag and Björn Regnell, Lund University
Vincenzo Gervasi, University of Pisa
Sjaak Brinkkemper, Utrecht University

For large software companies, the sheer number of textual requirements presents specific challenges. To find market opportunities, organizations must continuously elicit new requirements and reevaluate old ones as market needs evolve. Even so, you can implement only a subset of these requirements for the next release. Using linguistic-engineering techniques early on in the requirements management process could make this ongoing process less of a hindrance.

**Market-driven requirements management**

In a product company, requirements management bridges market interaction (existing customers, prospects, and analysts) with product development planning (content, resource planning, and timing). We can therefore distinguish between two major groups of requirements: customer wishes and product requirements.

Customer wishes are expressions of the perceived market need. These are naturally subject to frequent change—as the product evolves, the market need changes accordingly. Customer wishes make up a vital and valuable information source for decision-making. Also, because the wishes are written using the customer’s own perspective, they enable better communication with each customer.

Product requirements are those that the developing company finds worthwhile to pursue (stated from the developing company’s perspective). Companies also use these as a basis for product release planning as well as for conducting feasibility studies and, if selected for implementation, starting actual software development.

Customer wishes and product requirements often emerge independently of one another, and, for several reasons, it’s essential to keep them separated. For example, customers might ex-

---

Developing large, complex software products aimed at broad markets involves identifying and maintaining the link between product requirements and the massive inflow of customers’ wishes. Automating this support through linguistic engineering could save considerable time and improve software quality.
press their wishes slightly differently from one
another and without referring to the software or
business architecture. However, a product re-
quirement addressing all the differently stated
wishes might include additional information
that’s required for decision-making and develop-
ment but that shouldn’t be communicated
back to the customers (for example, references
to potential technical solutions, either the com-
pany’s own or from competitive analysis).

Naturally, the two requirements groups
share multiple associations. Organizations
must find and maintain these links, because
they constitute a significant piece of informa-
tion for requirements prioritization and re-
lease planning.

Unfortunately, the linkage process is cumber-
some. Each time a new customer wish arrives,
the task of determining whether it’s related to
the wide variety of product requirements is time
consuming. Organizations often accomplish
this task using simple search facilities, which
takes effort and can result in missing links. A
hierarchical requirements organization that’s
well thought out might help (for example,
based on the software architecture), but as the
product gets more complex, requirements
won’t always fit nicely into such a structure.
Moreover, as the architecture, product, and
company focus evolve, the requirements hier-
archy deteriorates.

A linguistic-engineering approach

Industrial experience shows the need for
automated support in the requirements man-
agement area.1,2 Modeling several thousand
requirements, even incrementally, to be able to
efficiently select only a small subset for imple-
mentation is simply not financially beneficial.
Any automated support must rely purely on
the original form of requirements, or unre-
fined natural language.

Approaches using natural language pro-
cessing (NLP) techniques to model, validate,
and help understand requirements are avail-
able but aren’t directly applicable here.3 These
approaches present interesting opportunities
but can’t effectively cope with the large
amount of requirements we’re considering.4,5
We have to choose a more pragmatic angle.

A link between a customer wish and a
product requirement indicates that they refer
to the same software functionality. Two re-
quirements should be linked if they have the
same meaning, even if expressed in a different
style and vocabulary. Unfortunately, there’s
still no method for representing meaning in a
way that automated systems can use success-
fully. We therefore choose to recast the chal-
lenge into suggesting semantic similarity on
the basis of lexical features. We assume that
customer wishes and product requirements re-
fer to the same functionality if they use the
same terminology. In a requirements-engineer-
ing context this is a reasonable assumption,
because the language tends to be more precise
than in literary text, and, moreover, both cus-
tomer wishes and product requirements refer
to the same domain.

When we submit the requirements to an au-
tomated process for establishing proper links,
an imagined support system first performs sev-
eral internal preprocessing steps (see the “Pre-
processing” sidebar).

The system will then internally represent
each requirement using a vector of terms, ac-
cording to the vector-space model (see the
“The Vector-Space Model and the Cosine
Measure” sidebar). From the vectors, the sys-
tem can derive how many terms the require-
ments have in common; we can use this as an
intuitive starting point for a similarity measure.

However, better measures exist that con-
sider both the requirements’ length and the
number of times the shared terms occur. One
common measure the literature suggests is the
Cosine measure, which calculates the angle be-
tween the vectors in the high-dimensional
space (see the “The Vector-Space Model and
the Cosine Measure” sidebar).

Once we define the similarity measure, sug-
gest potential links for an incoming require-
ment is a matter of sorting preexisting require-
ments according to their similarity to the new
one and offering the most similar requirements
to the user as candidates for establishing links.
Experiment: The Baan requirements set

From 1998 through 2002, Brinkkemper introduced a new requirements management process at Baan (now part of SSA Global). As Figure 1 shows, requirements management is part of the overall release development process, which also consists of development...
management to create the new releases and delivery management to control the software component delivery to customers (not shown in the figure).

The concepts this process introduces are

- **Market requirement (MR):** A customer wish related to future products, defined in the customer’s perspective and context.
- **Business requirement (BR):** A product requirement to be covered by Baan products, described in Baan’s perspective and context.
- **Release initiation (RI):** A formal document that triggers a release project in Baan (containing criteria for selecting BRs).
- **Version definition (VD):** A document listing the new release’s BRs with the needed personnel resources.
- **Conceptual solution (CS):** A document explaining the business solution preferably for one BR.

Continuously and as soon as possible after their receipt or creation, the new MRs and BRs are inserted into the Baan Requirements Database (BRD). Only after company management decides to start a new release project, an RI document triggers the writing of the corresponding VD and CS. These are then input for the development processes, which include writing design documents and actual coding.

Copying MRs into the BRD occurs without altering the original text as specified by the customer. So, if several customers suggest the same functional extensions, these are each recorded in separate MRs. Providing timely feedback helps maintain a good relationship with the customer, who receives an informative message after input review and also after the release is complete.

Product managers responsible for (a part of) the product create BRs, which should reflect a coherent, well-defined extension of the product. It might be that an MR is very large and therefore linked to many different BRs. A noncoherent MR dealing with dispersed functional areas is also linked to different BRs. Hence, the MR–BR relationship is of many-to-many cardinality, making proper MR-to-BR linking an essential process.

Linking MRs to BRs, and vice versa, is a daily routine for the product managers. Each time a new MR arrives into the BRD, they first check it by searching to find out whether one or more BRs already include the specified functionality. This process is quite time consuming, as current tools allow only text search in the requirement description.

Similarly, when a new BR is created, the corresponding MRs have to be found in the BRD, since the objective is to satisfy as many customers as possible. Finding all MRs that the BR at hand covers is virtually impossible because of the numerous MRs and the time-consuming process of trying to understand MR content.

For an idea of the high volume and complexity in the requirements management process, consider these statistics. By the end of 2000, the Baan software framework consisted of 250 modules and 10,000 components, comprising about 4.5 millions of lines of code. From 1996 through 2002, Baan elicited 3,800 business and 8,300 market requirements. Each month, 100 new market requirements arrive, of which 20 are handled for the coming release. Over the years, 2,400 market requirements have been linked to 1,100 business requirements.

**Example requirements**

Table 1 features representative examples of an MR and a linked BR. In the label and description fields, we find the principal information that constitutes the requirement. The contents in these fields are written in English, Baan’s
corporate language, and might very well contain spelling errors (italicized for clarity in the examples), acronyms, code snippets, and so on.

Currently, product managers would look for candidate MRs for the BR in Table 1 by searching for specific terms. For example, the term *container* gives hits in the label field of 37 requirements and in the description field of 318 requirements. In our case, five MRs were linked by experts, of which four were found through the label field. The last link was found by searching for *statistics* (giving 40 label and 99 description hits).

### Evaluation results

For a particular requirement of a given type, we’re interested in the other type’s list of candidate requirements. We therefore construct a top list for each market requirement by sorting the business requirements by similarity.

To evaluate how well the approach performs when it comes to presenting the correct links, we use the product manager’s manually identified links as the “presumably correct” answer. We can then calculate the recall rate as a function of the top list size (the ratio between the number of correct links found in the top list and the total number of correct links).

A top list size of 1 isn’t necessary, or wanted. For example, a top-10 list lets us quickly spot one or more correctly related requirements, while taking into account that we aren’t able to reach 100 percent recall.

The results from our evaluation show the recall curve for the top lists of suggested BRs for each MR (see Figure 2). The solid line represents the recall curve for calculating similarity using $1 + \log_2(\text{term frequency})$, and the dashed line for calculating it using just the term frequency.

As you can see, we never reach 100 percent recall. This is because some links couldn’t be identified at all—that is, some related requirements have no terms in common. We couldn’t identify 204 links, which resulted in a maximum recall of approximately 94 percent—that is, $(3,259 - 204)/3,259$—but to reach the maximum recall, we would require a top list of 3,000 requirements, which is quite unreasonable.

For a reasonable top list size of 10, as the figure shows, we reach a recall of 51 percent. This is good considering the pragmatic approach we took and the impact on the time that could be saved in industry.

### Saving time

To indicate the amount of time the process could save, we make a rough estimate on the basis of these statistics and another measure reflecting how many requirements could be

---

### Table 1

<table>
<thead>
<tr>
<th>Field</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>MR1000739 [Market requirement]</td>
</tr>
<tr>
<td>Customer</td>
<td>[Customer company. Proprietary information.]</td>
</tr>
<tr>
<td>Date</td>
<td>1996-05-29</td>
</tr>
<tr>
<td>Label</td>
<td>Pricing and containerization</td>
</tr>
<tr>
<td>Description</td>
<td>Specifically what I am interested in is containerization and pricing. For a prospect I am working with (pretty much a distributor of electronic components) I need pricing by type of package by customer type (wholesale or retail). I think pricing by container solves this problem, but I understand to use this feature the item must be a process item and I don’t know if this is good or bad. If I must use process what do I gain or lose, like do I have to run a separate MRP etc. Do I have to have one process company and one non-process company. They have mainly an assembly operation with no process involved. If process would be to cumbersome how difficult a mod would it be to disconnect containerization from process.</td>
</tr>
<tr>
<td>Keywords</td>
<td>Pricing, order planning</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Type</td>
<td>Functionality</td>
</tr>
<tr>
<td>Status</td>
<td>Closed/completed</td>
</tr>
<tr>
<td>User name</td>
<td>[Product manager. Proprietary information.]</td>
</tr>
<tr>
<td>Comments</td>
<td>020699: functionality is available in BaanERP in the pricing module</td>
</tr>
<tr>
<td>Agreement</td>
<td>None</td>
</tr>
</tbody>
</table>

| Id           | BR1000025 [Business requirement] |
| Customer     | [Customer company. Proprietary information.] |
| Date         | 1998-01-27 |
| Label        | Statistics and containers |
| Description  | 1. Container (end item) in statistics. Purchase and sales statistics used to be maintained only at main item level. But now it has also become possible to build statistics at container level. There are two aspects: printing statistics in the number of containers for a main item selecting and/or printing statistics at container level. 2. Displays in statistics. Displays are compositions of end items (for example, an attractive display of different types of cake). The statistics will be updated at both the levels of display item and container (which is part of the display). Prevention of duplicate counting, and correct pricing must be arranged in a procedural manner. |
| Keywords     | Process industries |
| Type         | Usability |
| Status       | Assigned |
| User name    | [Product manager. Proprietary information.] |
| Comments     | Warehousing only |
completely linked just by browsing a top-10 list. We found that for 690 of the BRs, the recall rate would be 100 percent using a top list size of 10. This means that every related MR for each of the BRs would be found within a top-10 list. The 690 BRs link to 1,279 MRs, giving an average of 1.85 MRs per BR. But to not exaggerate the gain, we assume that, in the manual case, one search term is enough to find all the links for one requirement.

We further assume that a manual search would return approximately 30 hits (based on the previous search examples). Thus, the worst-case scenario would be to browse 30 requirements. With a top list size of 10, the worst-case scenario with automated support would be to browse 10 requirements. Consequently, the process could save up to 66-percent effort.

If we assume that it takes about 30 seconds to accept or reject a requirement as a link, we find that the gain is \((690 \cdot 30 \cdot 0.5) – (690 \cdot 10 \cdot 0.5)\) minutes, which is 6,900 minutes, or 115 hours.

The critical reader might say that in a real setting it’s impossible to know the stop criterion, or how to know if a presented top-10 list comprises all the possible links for an arbitrary requirement. Although that’s true, the same applies to the manual case: searching for more keywords could yield more links. Nevertheless, the calculations just given show that a similar coverage level can be reached more efficiently (that is, with less effort) by applying lexical similarity compared to keywords search. If so desired, the time saved can be spent in increasing the level of coverage, by examining more candidates, or devoted to other RE activities if the coverage attained is deemed acceptable.

**The ReqSimile tool**

Considering the need for automated support in the described linkage process and also to demonstrate our approach, we’ve implemented an open source tool in Java called ReqSimile (see Figure 3; http://reqsimile.sourceforge.net). The tool operates on arbitrary requirements sources, which are accessed through a standard interface (Java Database Connectivity). ReqSimile can therefore integrate well with existing requirements databases (provided a database driver is present). All the involved database tables and fields can be specified from within the tool.

The left side in the top pane of the window presents a brief list of requirements. Selecting a requirement makes the requirement’s details show on the right. Double-clicking a requirement makes ReqSimile calculate the requirement’s similarity to all the requirements in the other set.

A list of candidate requirements, sorted by similarity, then shows in the bottom pane. For flexibility, the user can affect the ranking by adding more search terms in a separate text box.

Already linked requirements are highlighted, and each requirement’s details are shown on the right when the requirement is selected. Through the buttons next to each requirement, the user can remove or add links between the selected requirements, and the program will update the requirements database accordingly.

For research purposes, the program also calculates and reports different measures, such as statistics on the requirements sets and recall rates based on the currently linked set.

For practical use, a company can integrate the technique we describe into its existing requirements management solution, or it can use ReqSimile as an external tool supporting the linking step. In the latter case, the company obtains integration with the existing requirements management practices by configuring
ReqSimile to interface with the organization’s requirements database.

Further work

The ReqSimile tool and the underlying techniques do a good job of supporting the linkage process. Our approach’s statistical nature makes it resilient to unintentional errors in the text (for example, spelling errors). It’s not worth trying to correct such errors automatically as part of the linking process. In fact, automatic correction is likely to introduce more errors. Furthermore, occasional typos have negligible impact on the recall. However, we’ve identified other issues that might be interesting to pursue (previous work provides a more elaborate review of potential improvement⁴).⁶

One worthwhile approach would be to incorporate and aggregate several different similarity calculation techniques⁶. Different linguistic models might together contribute to better precision.

Another promising approach for improving the precision in future suggestion lists would be to reuse the information in previously linked requirements. A support tool could use the information to improve similarity measures, to leave out already linked requirements, or as a learning set to add relevant terms not originally in the requirement.

A third issue is to incorporate semantics to catch more distant similarities. We expect the handling of synonyms, hypernyms (more general terms, such as vehicle⁷), and hyponyms (more specific terms, such as bike⁷) to provide marginal improvement over the results reported.

Implementing all the extensions we’ve mentioned would likely bring further improvements over our results, making the approach even more effective.
Software engineers have yet to fully exploit linguistic-engineering techniques to support large-scale software product development. One reason for this is researchers’ limited access to industrial requirements collections. These information sources, together with the requirements activities currently performed in industry, will reveal new opportunities for applicable linguistic-engineering research. The challenge is to consider all the criteria to yield acceptance: usability, cost-benefit, flexibility, robustness, and efficiency, to mention a few. The approach we present is a promising step toward well-engineered systems to aid large-scale requirements management in companies that rely on communication in natural language.

Acknowledgments

We thank Pierre Breuls and Wim van Rijswijk at Baan in Barneveld for kindly providing the requirements database. Thanks to Per Runeson and Lena Karlsson for critical reviews. Thanks to Ernhold Lundstrom Foundation for covering travel expenses.

References


For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.