The Software Product Management Workbench: An Integrated Environment for Managing Product Releases in a Distributed Development Context

Richard Nieuwenhuis, Inge van de Weerd, Lex Bijlsma,
Sjaak Brinkkemper, Johan Versendaal

Department of Information and Computing Sciences
Utrecht University, The Netherlands

{ rnieuwen, i.vandeweerd, a.bijlsma, s.brinkkemper, j.versendaal }@cs.uu.nl

Abstract. Product management in software product companies is complex due to the many intertwined information flows on releases, products and requirements, and the high number of different stakeholders that are involved. We propose the Software Product Management Workbench for operational support of requirements management, release planning, product roadmapping, and portfolio management. In this paper, we describe the functional design, architectural design, and show the requirements similarity functionality in the prototype.

1 Introduction

In product software companies, product management is an important function. The product manager (PM) has a lot of responsibilities regarding the content of upcoming releases of the various products. Nonetheless, he does not have management authority over the development teams, and thus the project progress, which makes this job complex. Every day, the PM is confronted with several stakeholders in and outside the company, like customers, sales, development teams and services, who all have their own wishes. We developed a reference framework for software product management [3], in which all interactions of the functionalities and stakeholders are modeled. Parts of this framework are supported by tools, for example requirements organizing, or release planning. However, no tools exist that integrate all features dedicated to product management. To support the PM with all daily activities, we propose the Software Product Management Workbench (SPMW).

2 Functional Design

The SPMW is designed to support five types of users in a product software company:

1. The PM is the main user of the SPMW, and uses it in his everyday work for organizing requirements, planning product releases, scheduling teams, etc.

2. Developers use the system to acquire or add the latest development details on releases and requirements.
3. The Core Asset Developer is responsible for attaching core assets to releases, so that the PM and the developers have easy access to the latest core assets.
4. The customer is an optional user. When a company wishes to integrate its website with the SPMW, the customer can use it to submit requirements. The SPMW also provides self service look-ups to the clients, enabling the clients to view the status of their submitted requirement.
5. The System Administrator is responsible for giving access to new users, such as board members, marketing and sales representatives and selected customers and facilitating the product manager in starting up new products.

In Figure 1, we visualized the key users and the four main modules of the SPMW.

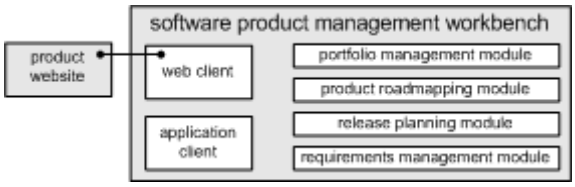

Fig. 1: Functional Design of the SPMW

In the *requirements management module* the PM can add, edit and organize requirements. The strength of this module is that it is capable of giving suggestions of requirements that may be referring to the same functionality automatically. The *release planning module* is used by the PM to develop new releases. This module is able to suggest requirements that should preferably be put in a next release based on parameters provided by the PM. Change and status management is also possible in this module. The *product roadmapping module* is used for monitoring the roadmap of a product. Finally, the *portfolio management module* is used by the PM and by the Core Asset Developer to manage the product portfolio and product lines.

3 Architectural Design

Building *Enterprise Applications* is a difficult and daunting task [4]. These types of applications need to be designed carefully to provide good performance. The J2EE platform [1] takes care of many of the difficulties, thus making the creation of it easier. At the heart of J2EE lie the so called *Enterprise Java Beans* (EJB). The architecture uses two different types of EJBs, namely: *Entity* and *Session* beans. One Entity bean represents one row, for example a requirement, in a database table. Session beans perform work for their clients (the users mentioned in Sec. 2), and are often used to provide coarse grained access to other beans, shielding the client from the complex business logic. The architecture of the SPMW uses J2EE design patterns [1], giving the SPMW the desirable performance and maintainability.

A high level overview of the system, is shown in Fig. 2, together with a small part of the architecture showing some of the key patterns used throughout the system. An *application client* runs on the client machine. This has as advantage that heavy calcu-

lations, for example the suggesting of requirements that refer to the same functionality, can be performed here without affecting the server. In general, only the PM uses the application client, since he is the only one who executes heavy calculations. All other users login through the web client with a browser from anywhere in the world, so that they are always up-to-date with the latest release and the current contents.

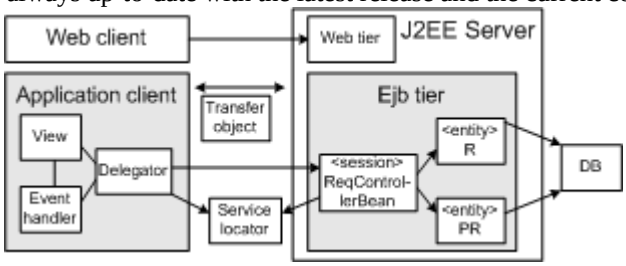

Fig. 2: High level architecture of the SPMW

Session Façades provide coarse-grained access to the entity beans, making the clients less dependent on the EJB tier. Calls from the application client and the web tier to the EJB tier are fairly slow. Data is therefore sent back and forward from the clients to the EJB tier using *Transfer Objects*, containing all data the client possibly wants to get a hold on, instead of having one remote call for every piece of data. Locating objects residing on another machine is done by performing lookups using the Java Naming and Directory Interface, which are relatively slow. Extracting all the lookup code into a singleton object not only removes duplicated lookup code, but when different objects need a reference to the same objects, caching mechanisms for object references can be used to improve system performance. The clients are designed following the Model View Controller pattern.

The SPMW architecture is designed such that new product management innovations, which are likely to be introduced in future releases, can easily be incorporated.

4 Prototype

Presently, we have realized the requirements and the release planning module of the SPMW. The function shown in fig. 3 refers to the selection of similar requirements based on the input of a given requirements using techniques of linguistic engineering [2]. This technique transforms a text by performing several pre-preprocessing steps on it and then calculates the similarity between requirements using the vector-space model. These steps are: f*lattening, tokenizing, stop word removal,* and *stemming.* Requirements are flattened by merging the label and the description of it. After this a requirement is tokenized into a sequence of tokens by removing capitals, punctuations etc. Common stop words are removed, followed by performing stemming on the words to remove affixes and other lexical components. After this the requirements are represented as vectors so that the “angle” between them can be calculated. This results in a number between 0-1 indicating how similar they are. When this process is complete, the SPMW displays the requirements ordered by their similarity. This proc-

ess is not bound to English requirements alone, but requirements in Dutch, French, and German can also be processed.

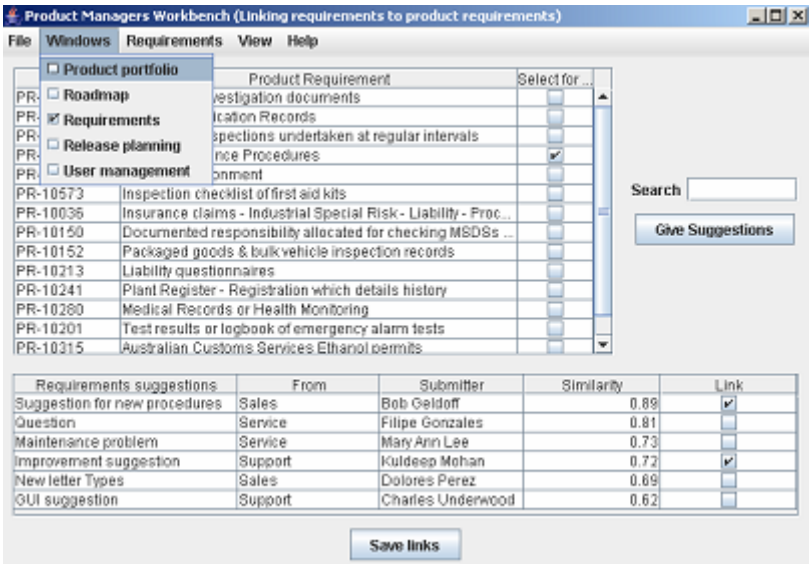

Fig. 3: Requirement similarity suggestion

5 Future Work

Building on the SPMW will continue, and studies on more efficient support of product management processes are performed, and if found useful, they will surely make their way as extensions into the support tool. For example, studies on improving the requirements linking process [2] are performed. The next step is extending the SPMW with the product roadmapping and portfolio management modules. The SPMW is therefore not only innovative at the present time, but is also built for the future.

6 References

[1] Armstrong, E., Ball, J., Bodoff, S., Carson, D.B., Evans, I., Green, D., Haase, K., Jendrock, E.: The J2EE 1.4 Tutorial, 2nd ed. Sun Microsystems and Addison-Wesley (2004)

[2] Natt och Dag, J., Regnell, B., Gervasi, V., Brinkkemper, S.: A linguistic-Engineering Approach to Large-Scale Requirements Management. IEEE Software, Vol. 22 (2005) 32-39

[3] Weerd, I, van de, Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., Bijlsma, L.: A Reference Framework for Software Product Management. Accepted for publication in the 14th International Requirements Engineering Conference, Minneapolis/St. Paul, Minnesota, USA (2006)

[4] Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley Boston, MA, USA (2003)