**UNIVERSITY OF VAASA**

**FACULTY OF TECHNOLOGY**

**SOFTWARE ENGINEERING**

Oskar Norrback

# A SOFTWARE IMPLEMENTATION OF THE REQUIREMENTS ABSTRACTION MODEL

Master´s thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, 24th February, 2014.

Supervisor Prof. Jouni Lampinen

**TABLE OF CONTENTS**

**page**

**VAASAN YLIOPISTO**
**Teknillinen tiedekunta**
**Tekijä:** Oskar Norrback
**Diplomityön nimi:** Ohjelmistototeutus vaatimusten abstrahointimallista
**Valvojan nimi:** Prof. Jouni Lampinen
**Tutkinto:** Diplomi-insinööri
**Koulutusohjelma:** Tietotekniikan koulutusohjelma
**Suunta:** Ohjelmistotekniikka
**Opintojen aloitusvuosi:** 2007
**Diplomityön valmistumisvuosi:** 2014 **Sivumäärä:** 51

**TIIVISTELMÄ:**

Puutteelliset vaatimusmäärittelyt luetellaan usein syynä epäonnistuneissa ohjelmiskehitysprojekteissa. Tässä työssä me tarkastelemme Tony Gorschekin ja Claes Wohlinin esittelemää RAM-nimistä vaatimusten abstrahointimallia. RAM kehitettiin vastaukseksi teollisuuden tarpeelle käsitellä vaatimuksia markkinalähtöisessä tuotekehityksessä. Mallissa vaatimukset asetetaan jollekin neljästä eri abstraktiotasosta ja niistä tehdään vertailukelpoisia luomalla uusia niihin liittyviä enemmän tai vähemmän abstrakteja vaatimuksia tarpeen mukaan.

Työn ensimmäinen tavoite on löytää toimiva malli eritasoisten vaatimusten hallitsemiseen. Mallin tulisi olla helppo ottaa käyttöön ilman syventävää koulutusta eikä se saisi tehdä vaatimustenhallinnan prosessista liian raskasta

Työn toinen tavoite on soveltaa valittua mallia uuden vaatimusten hallintatyökalun suunnittelussa. Toiveena on, että sopiva malli ja siihen perustuva työkalu voivat auttaa tuotepäälliköitä tuottamaan laadukkaampia vaatimusmäärittelyjä ja sen kautta onnistuneempia ohjelmistokehitysprojekteja.

RAM-mallia verrataan muihin menetelmiin joita voidaan käyttää vaatimusten hallinnassa, kuten Quality Function Deployment ja Agile Requirements Refinery. RAM valittiin lopulta sen yksinkertaisuuden, skaalautuvuuden, sen tuomien etujen ja sitä tukevien empiiristen tutkimustulosten perusteella.

RAM-mallin pääperiaatteita sovelletaan uuden RAMP-nimisen työkalun arkkitehtuurissa, oliomallinnuksessa sekä käyttötapausten ja käyttöliittymien suunnittelussa. RAMP-työkalun suunnittelu esitellään ja sen perusteella toteutetaan prototyyppi käyttöliittymästä.

**AVAINSANAT:** vaatimusten abstrahointimalli, työkalu vaatimusmäärittelyyn, ohjelmistojen vaatimusmäärittely, ohjelmistotuotanto

**UNIVERSITY OF VAASA**

**Faculty of technology**

| | |
|---|---|
| **Author:** | Oskar Norrback |
| **Topic of the Thesis:** | A Software Implementation of the Requirements Abstraction Model |
| **Supervisor:** | Prof. Jouni Lampinen |
| **Degree:** | Master of Science in Technology |
| **Degree Programme:** | Degree Programme in Information Technology |
| **Major of Subject:** | Software engineering |
| **Year of Entering the University:** | 2007 |
| **Year of Completing the Thesis:** | 2014 **Pages:** 51 |


**ABSTRACT:**

Requirements are often referred to as one of the main reasons for failed software projects. In this thesis we review the Requirement Abstraction Model (RAM) as outlined by Tony Gorschek and Claes Wohlin. RAM was developed as a response to an industrial need to handle requirements in market driven product development. The model places requirements on four different abstraction levels and it makes requirements comparable to each other through abstraction or break down.

The first objective of this thesis is to find a working model for managing an incoming stream of requirements with varying levels of abstraction. The model should not cause an unreasonable burden on the requirement engineering process and it should be easy to adopt, e.g. it should not be so complex that it requires excessive training.

The second objective of the thesis is to apply the chosen model to the design of a new requirement management tool. The hope is that the right model and a supporting tool in combination will help product managers create higher quality requirements and as a result more successful software projects.

RAM is compared to other methodologies that can be used in requirements engineering such as Quality Function Deployment and the Agile Requirements Refinery. RAM is ultimately chosen as the main model based on its simplicity, scalability, potential benefits and supporting empirical evidence.

The main principles of RAM are applied into the architecture, domain model, use case and user interface design of a new requirement management tool, RAMP. The design of RAMP is presented and a proof-of-concept prototype is also implemented.

**KEYWORDS:** Requirements Abstraction Model, requirement management tool, requirement engineering, software engineering

# 1. INTRODUCTION

This initial chapter outlines the research problem along with some background information and the motivation for choosing this particular topic. The structure of this thesis is also explained.

## 1.1. Background and motivation

Requirements engineering is a challenging and critical part of software engineering projects. Brooks (1987) describes it as follows:

> The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

According to the 1995 CHAOS report that was published by the Standish Group only 16.2 % of software projects are completed on-time and on-budget. 52.7 % of projects finish challenged, e.g. over-budget and with fewer features than planned. The remaining 31.1 % of projects are canceled (impaired) at some point. Incomplete and changing requirements were ranked as some of the main factors for challenged and canceled projects. (The Standish Group International, Inc. 1995.)

**Table 1.** Project challenged factors. (The Standish Group International, Inc. 1995.)

| Project Challenged Factors | % of Responses |
|---|---|
| 1. Lack of User Input | 12.8% |
| 2. Incomplete Requirements & Specifications | 12.3% |
| 3. Changing Requirements & Specifications | 11.8% |
| 4. Lack of Executive Support | 7.5% |
| 5. Technology Incompetence | 7.0% |
| 6. Lack of Resources | 6.4% |
| 7. Unrealistic Expectations | 5.9% |
| 8. Unclear Objectives | 5.3% |
| 9. Unrealistic Time Frames | 4.3% |
| 10. New Technology | 3.7% |
| Other | 23.0% |

**Table 2.** Project impaired factors. (The Standish Group International, Inc. 1995.)

| Project Impaired Factors | % of Responses |
|---|---|
| 1. Incomplete Requirements | 13.1% |
| 2. Lack of User Involvement | 12.4% |
| 3. Lack of Resources | 10.6% |
| 4. Unrealistic Expectations | 9.9% |
| 5. Lack of Executive Support | 9.3% |
| 6. Changing Requirements & Specifications | 8.7% |
| 7. Lack of Planning | 8.1% |
| 8. Didn't Need It Any Longer | 7.5% |
| 9. Lack of IT Management | 6.2% |
| 10. Technology Illiteracy | 4.3% |
| Other | 9.9% |

Although the problems in requirements engineering are numerous (Christel & Kang 1992) this thesis will focus on the issue of managing requirements of different abstraction levels and normalizing requirements of the same level to be comparable.

Gorschek & Wohlin (2006) present the Requirements Abstraction Model (RAM) as one solution to this problem, especially in the context of Market Driven Requirements Engineering (MDRE). Carlshamre & Regnell (2000) define MDRE as "continuous management of new and changed requirements in a way that ensures competitiveness on the market place". RAM is an empirically validated model and process for handling requirements of varying abstraction levels (Gorschek, Garre, Larsson & Wohlin 2007). Although similar tools exist there doesn't seem to be any dedicated tool support for RAM (Vähäniitty & Rautiainen 2008; Gorschek et. al 2007).

## 1.2. Objective

The first objective of this thesis is to find a practical model for managing an incoming stream of requirements with varying levels of abstraction. The model should be reasonably lightweight, e.g. it should not cause an unreasonable burden on the requirement engineering process and it should be easy to adopt, e.g. it should not be so complex that it requires excessive training.

The second objective of the thesis is to apply the chosen model to the design of a new requirement management tool. The hope is that the right model and a supporting tool in combination will help product managers create higher quality requirements and as a result more successful software projects.

## 1.3. Structure of the thesis

The thesis starts with an introduction to the research problem and a review of the Requirement Abstraction Model. It is then followed by a review of other related literature such as the agile requirements refinery and quality function deployment.

Then the architecture, domain model, use cases and user interface design of the new requirement management tool, RAMP, are presented. Additionally the prototype implementation of the RAMP application is presented. Finally the thesis is ended with conclusions, discussion and some ideas for future research.

# 2. REQUIREMENT ABSTRACTION MODEL

In this chapter we review the Requirements Abstraction Model in detail and we discuss the potential benefits of applying this model.

## 2.1. Overview

The 1.0 version of the requirement abstraction model (RAM) is presented by Gorschek and Wohlin (2006). It was developed to meet an industry need for a system to manage requirements in market-driven requirements engineering. MDRE can be challenging since requirements are continuously arriving from a wide variety of internal (e.g. engineers, management) and external (e.g. customers) sources. The requirements also vary in format between direct requirements (feature requests) and indirect requirements (ideas, goals). RAM v1.0 focuses on the *continuous requirements engineering* effort where product managers must process a steady stream of incoming requirements and normalize them for further analysis. At a later stage a development project is initiated with a subset of the initially processed requirements as the scope. *Dedicated requirements engineering* is done as part of the project where the initially processed draft requirements are refined to ensure that the requirements are unambiguous and testable.

## 2.2. Abstraction Levels

To bring structure to this wide variety of requirements RAM introduces a concept of abstraction levels. A RAM example implementation using four abstraction levels is presented. It is worth noting that the amount of abstraction levels is not fixed and should be tailored to suit the target organization. (Gorschek & Wohlin 2006.)

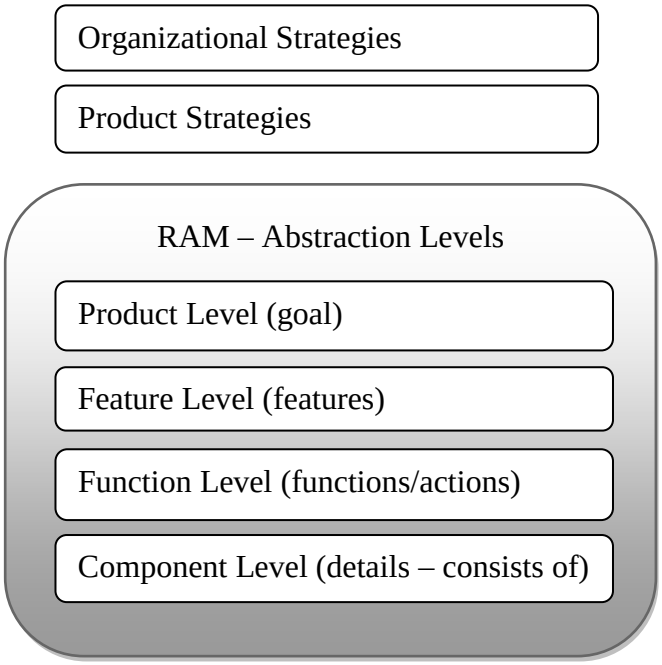


**Figure 3.** The four abstraction levels of the RAM example implementation. (Gorschek & Wohlin 2006.)

Abstract, goal-like requirements are placed on the *Product Level*. They might be based directly or indirectly on the products and parent organizations strategies and they typically don't fit the traditional definition of a requirement of being testable and

unambiguous. This is resolved later by the RAM work-up phase that breaks all high level requirements into more concrete requirements to all the levels below it. The *Feature Level* contains requirements that are actually abstract descriptions of features that the product supports. The *Function Level* contains functional and non-functional requirements about what actions the user or the system should be capable of doing. Function Level requirements should be detailed and complete enough to be testable and unambiguous. *Component Level* requirements are low-level, detailed requirements that typically come from internal sources (engineering). Component Level requirements can be used to break down complex Function Level requirements to add more detail or to set limitations. (Gorschek & Wohlin 2006.)

## 2.3. Action Steps

A supporting process example consisting of three action steps is also defined. The *Speficy* step involves specifying the initial raw requirement along with its basic attributes such as title, description, reason/benefit and restrictions/risks. The *Place* step involves choosing the correct abstraction level for the initial requirement. This step is usually aided by some reference material with example requirements and their abstraction levels. On the *Abstraction* step each initial requirement goes through a work-up process where new requirements are created on the adjacent abstraction levels or links are created to existing ones. (Gorschek & Wohlin 2006.)

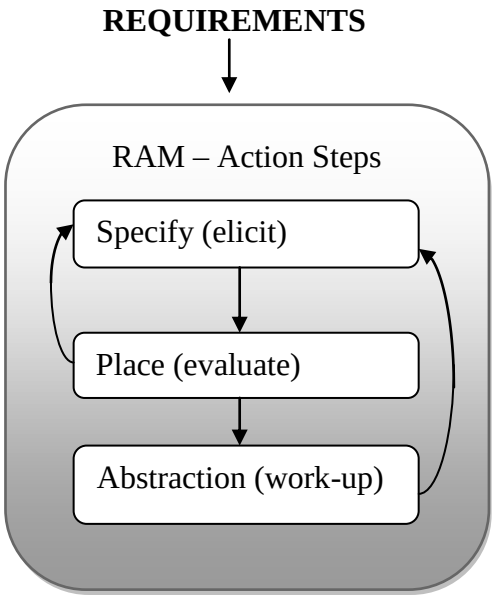


**Figure 4.** RAM action steps. (Gorschek & Wohlin 2006.)

The work-up process follows two rules: *(1) No requirement may exist without having a connection to the Product Level,* and *(2) All requirements have to be broken down to Function level*. The first rule can be met by either creating one or more new work-up requirements on the levels above the original requirements or by linking to existing requirements. The second rule can be met in similar ways by either creating new work-up requirements or linking to existing ones. The reasoning behind the second rule is that an abstract high level goal cannot be used to start a development effort until it has been specified (broken down) in enough detail to be used as an input for the design phase. It is also worth nothing that the break down is not mandatory all the way down to the Component Level since this level is seen as an optional level that suggests *how* a Function Level requirement should be implemented. (Gorschek & Wohlin 2006.)

The authors note that it is important for the RAM users to stay consistent with the abstraction levels they choose in the *Place* step. In the *Abstraction* step it is important to

only create new work-up requirements that are really required by the original requirement. Any side tracks along the lines of "this might also be a good idea" should be handled separately and trigger a new instance of the action steps process. (Gorschek & Wohlin 2006.)

## 2.4. Attributes

In addition to the four basic attributes mentioned earlier in 2.3 additional attributes should be specified during the action steps. The traceability and role attributes such as Requirement Source, Requirement Owner and Requirements manager are there to prevent traceability issues and to clarify responsibilities. Process attributes such as State reflect the requirements status in the product development organization. Additionally there are attributes for revision control such as Version, Date of Creation and Last Changed. (Gorschek & Wohlin 2006.)

## 2.5. Roles

RAM defines various roles how people can be involved in a requirement. The *requirement source* can be a person, document or an organization for example. The *requirement owner* is an internal person whose responsibility it is to ensure that the requirement is followed up on. The requirement owner represents the source in the cases where the source is silent (eg. an external party or a document). The *requirements manager*, typically the product manager for the product, is responsible for working with the requirement throughout its lifespan. It is important that the source, owner and

manager collaborate to combine their knowledge and perspective on the subject. (Gorschek & Wohlin 2006.)

## 2.6. Requirement states and lifecycle

A total of nine different states and the transitions between them are defined for RAM requirements. Please see Figure 5 below for details.

| | |
|---|---|
| Draft requirement | A requirement reaches this state when the requirements manager has collected the needed initial information about the requirement (*specify* action step) and it has been placed on the correct abstraction level (*place* action step). Work-up is also done to abstract the requirement onto all abstraction levels (*abstraction* action step).The requirement is then validated against the requirement owner and source. |
| Rejected requirement | A draft requirement is usually rejected if it is found to not be in line with product strategies. Related work-up requirements should also be removed or connected to other requirements if they are still valid. A rejection reason or a reason for the exemption should be recorded. |
| Incompletely specified | During the validation of a draft requirement the requirement can be deemed incompletely specified. The reasons for this can be e.g. wrong initial abstraction level or poor work-up. (Gorschek & Wohlin 2006.) |

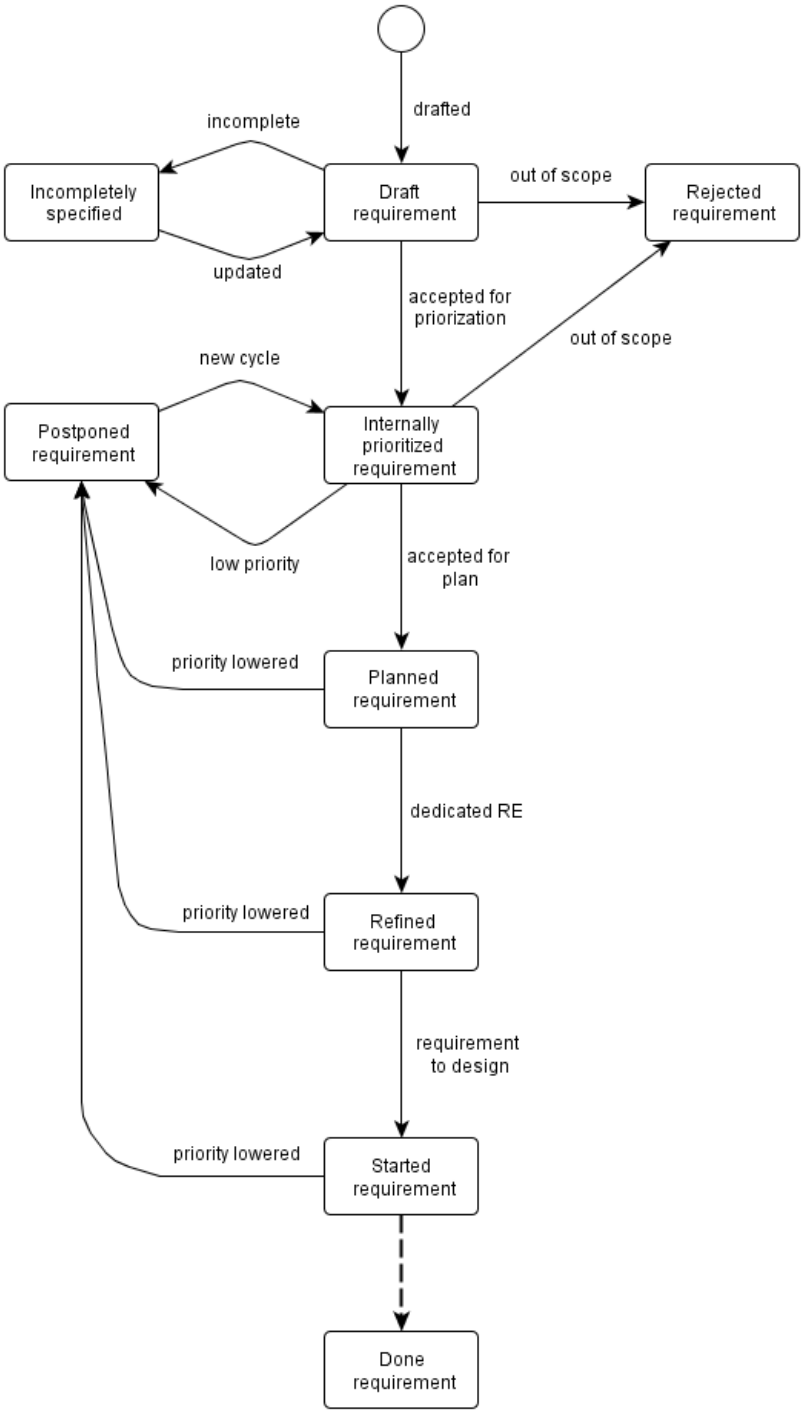


**Figure 5.** Requirement states in RAM. (Gorschek & Wohlin 2006.)

## 2.7. Benefits and discussion

Some clear potential benefits from applying RAM correctly can be observed. All requirements are validated early on against product strategies and discarded if needed. Additionally requirements are broken down to a detailed enough level to be used as a starting point for the dedicated requirements engineering followed by the design and eventual implementation of the requirement. The systematic requirements work-up also guarantees that requirements on the same abstraction level are comparable when doing release planning. The requirements hierarchy with connections through different abstraction levels can be used to get a better understanding of each individual requirement. (Gorschek & Wohlin 2006.)

RAM seems like a reasonably lightweight and easy to adopt requirement engineering process. There is also some empirical evidence to suggest that it does improve requirement quality in exchange for some extra effort (Gorschek et al. 2007). The clearly defined constructs defined by RAM (abstraction levels, attributes, roles, states) seem like they can be quite easily translated into the design of a new requirements engineering tool. Hence RAM is chosen to be the basis for the design of RAMP, the tool to be designed as part of this thesis.

# 3. RELATED WORK

In this chapter we review two other models that were considered to be used as the basis of RAMP: Agile Requirements Refinery and Quality Function Deployment.

## 3.1. Agile Requirements Refinery

The Agile Requirements Refinery is presented by Vlaanderen, Jansen, Brinkkemper & Jaspers (2011). They propose an agile Software Product Management (SPM) development method based on the popular SCRUM development method. Agile SPM runs in sprints just like regular SCRUM development but rather than developers producing working software the end result is product managers producing clearly specified requirements. The agile SPM process can be run in parallel with a traditional SCRUM development process and its purpose is to continuously feed the development process complete and detailed requirements to implement. (Vlaanderen et al. 2011.)

Typically the input to the agile SPM process is a *vision*, a high level idea or a wish for a new functionality. The vision is then iteratively broken down into one or more *themes*, a more detailed description of a vision. After review themes are further broken down into *concepts* that consist of solution stories that can be used later to construct a detailed *requirement definition*. The break-down is not necessarily always done top-down, it can also be done bottom-up when the input requirement is less abstract. This seems rather similar to RAMs abstraction levels and work-up process and the authors also acknowledge this. (Vlaanderen et al. 2011.)

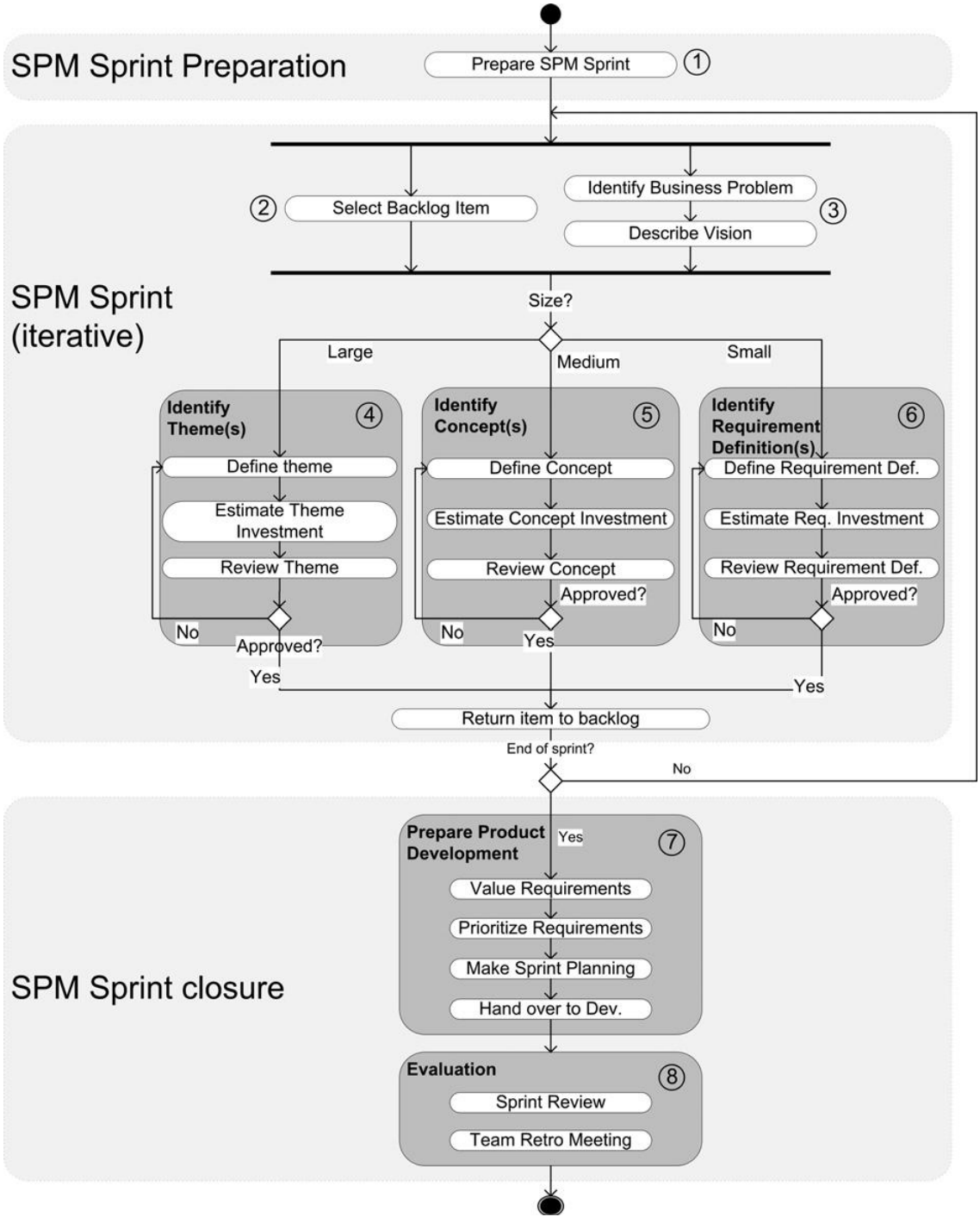


**Figure 6.** SCRUM Software Product Management process. (Vlaanderen et al. 2011.)

## 3.2. Quality Function Deployment

Quality Function Deployment (QFD) was introduced by Dr. Yoji Akao (Akao 1997). It was initially applied in Japanese heavy industry companies but has later been expanded to many other fields including software engineering. QFD focuses on the initial planning phase rather than having to make many changes later in the process where it is expected to be more expensive and slower. QFD can be considered a process for translating customer requirements (voice of the customer) into product design. During the process the organization must be able to communicate with the customer and acquire enough knowledge to developer products which will satisfy the customer. (Koski 2003.)

The QFD process starts by communicating with the customer and building a customer information table that consists of customer requirements typically combined with importance ratings and competitive evaluations against competitors. In the next stage the organization should work on creating a technical requirements table based on the customer requirements. (Koski 2003.)

Finally both the tables can be merged into a QFD matrix that is sometimes also called the house of quality. In the QFD matrix the rows are used to represent customer requirements and columns are used to list corresponding technical requirements. The cells contain symbols or numerical values defining the strength of the relationship between the customer requirement and the technical requirement. Competitive evaluations can also be included adjacent to the matrix so that each requirement row gets extra columns rating the company's product against the competitors. An example of a QFD matrix can be found in Figure 7 below. (Koski 2003.)

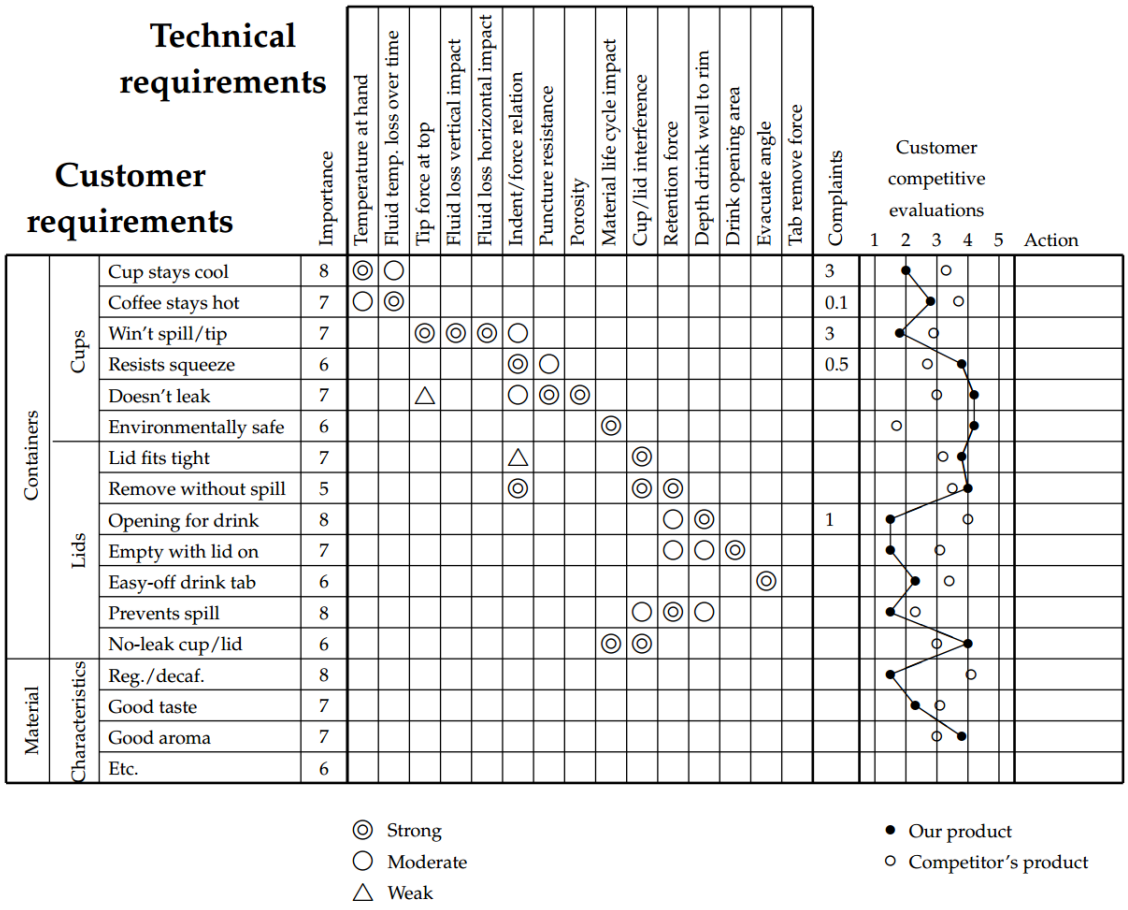


**Figure 7.** Example of a QFD matrix. (Koski 2003.)

There are several success stories about the introduction of QFD into organizations and projects. Haag, Raja & Schkade (1996) report positive results from utilizing Software QFD (SQFD) in software development projects in 16 organizations. Koski (2003) also reports positive results from four case studies where QFD was used in projects where software was involved but not in the main role.

On the other hand Poel (2007) discusses several methodological issues with QFD and suggests that "the core idea of the QFD approach is methodologically problematic". According to Karlsson (1997) it is recommended to keep the QFD matrix smaller than

30 by 30 relationships which puts constraints on the level of abstraction and might cause re-working of the existing requirements if the matrix starts to grow too large. Karlsson also observes that QFD seems to have no straightforward way of expressing temporal relationships between requirements.

# 4. DESIGN AND ARCHITECTURE

In this chapter we present the architecture, domain model, use case and user interface design of the new requirement management tool, RAMP. The parallels between RAM and the design of RAMP are also explained.

## 4.1. Architecture

RAMP architecture follows the three-tier model as described by Microsoft Patterns & Practices Team (2009). The client, a desktop application, is responsible for the presentation tier. It can be used to view and modify the application data locally on the end users computer. The server implements the logic and data tiers. All data is read and stored back to the server so that it becomes visible to other users. The server uses a relational database to persist the application state.

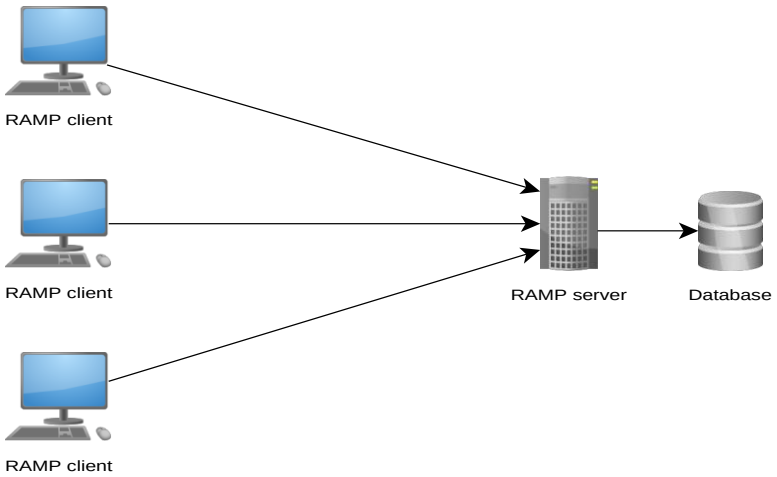


**Figure 8.** RAMP high level architecture.

The client desktop application is written in the C# programming language and utilizing the Microsoft .NET framework. The Windows Presentation Foundation (WPF) framework is used to implement the client user interfaces.

The Model-View-ViewModel (MVVM) design pattern is applied within the client application. MVVM encourages separation between the user interface (View) and the data model. The view model is responsible to exposing the data model to the view in a convenient way. (Gossman 2005.)

The RAMP server application is also written in C# and utilizing .NET and it is hosted on the Microsoft Internet Information Services (IIS) web server. A Windows Communication Foundation (WCF) web service acts as a communication endpoint for the RAMP clients to connect to. Persistence for a relational database is handled using the NHibernate object-relational-mapping (ORM) library.

## 4.2. Domain Model

The domain model of RAMP can be split into two logical parts: the *core model* and the *authorization model*. The core model describes the classes and relationships between the core application data, eg. requirements. The authorization model describes the relationships between users, roles and products so that the application can determine what actions and functions each user is authorized to access.

### 4.2.1. Core Model

The core domain model centers on the Requirement class. A Requirement instance has a reference to the AbstractionLevel it has been placed on. It also has a reference to a parent Requirement on the next AbstractionLevel when available. A Requirement also has a one-to-many relationship with the AttributeValue class. This corresponds to the core concepts of requirements, abstraction levels and attributes in RAM.

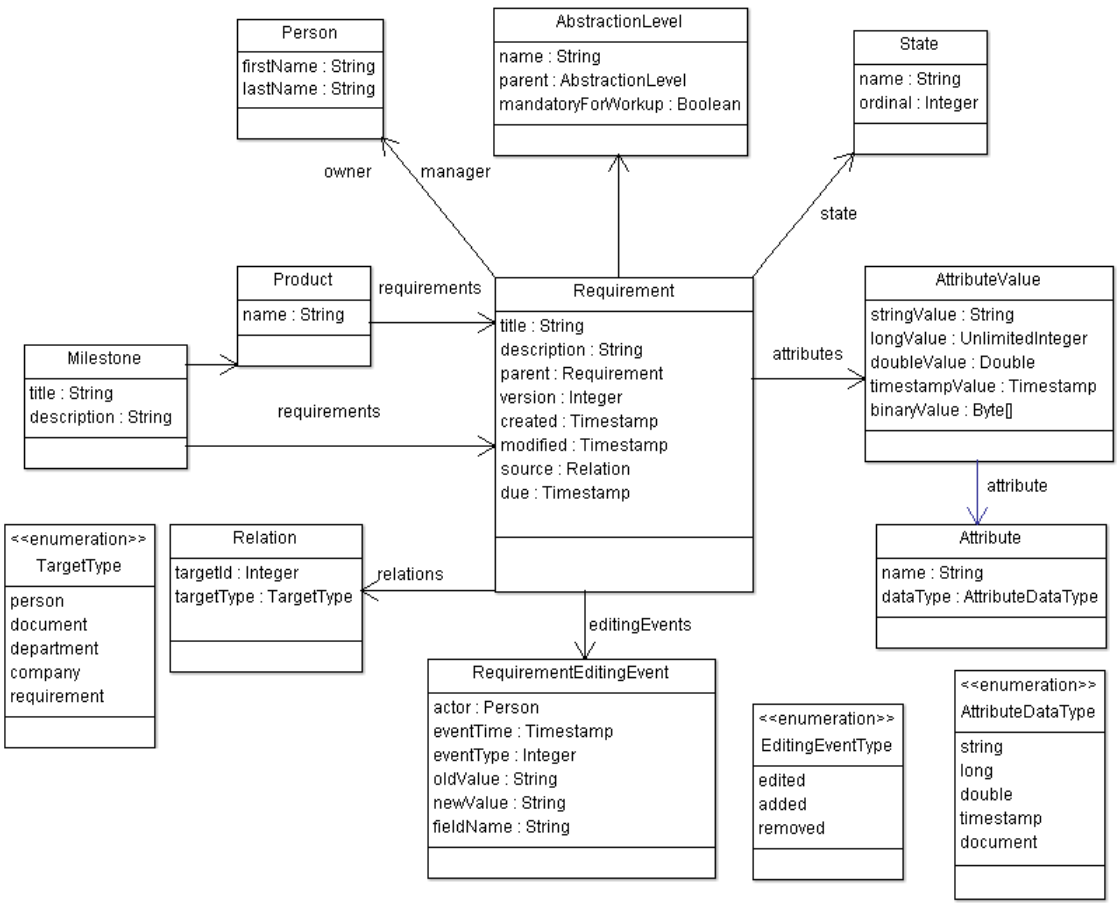


**Figure 9.** UML class diagram of the RAMP core domain model

Requirements additionally belong to a parent Product entity and they also target a specific Milestone. Milestones can be used to do release planning and packaging of requirements. For traceability reasons a collection of RequirementEditingEvents is maintained, containing information about each modification of a Requirement.

Gorschek et al (2006) mention that RAM is not a one-size-fits-all solution, e.g. some tailoring is needed for each organization. Attributes should be customizable to fit the needs of the target organization. Additionally in their empirical evaluation of RAM in two different companies they ended up using different abstraction levels, attributes and states for each company. Consequently the RAMP domain model also allows freely configurable AbstractionLevels, Attributes and States.

The relationships between Requirement, Attribute and AttributeValue are essentially an application of the entity-attribute-value (EAV) design. EAV is a model that allows the users to extend the system without requiring changes in the database schema or data access code. EAV is frequently used in the medical field to store data from clinical trials. In its simplest form this is accomplished by storing attribute values as property-value rows rather than fixed columns in a table. This flexibility in schema design comes with a performance trade-off. The trade-off is most noticeable when running ad-hoc queries such as “return all Requirements that have a value 3 for customAttribute1 and a value xyz for customAttribute2”. (Dinu & Nadkarni 2007.)

It was concluded that the flexibility of custom attributes was worth the potential performance issues and attribute metada complexity as outlined by Dinu et al (2007). It was also estimated that the amount of EAV modeled data in a RAMP installation would not become significant enough to affect the system performance, e.g. Wang et. al (2004) were still getting sub-second response times on a EAV dataset of roughly 100 megabytes.

A simple model for managing tasks was also designed. Tasks are automatically created by the system for example when a requirement needs workup. The users can also create tasks manually. The purpose of the tasks is to act as a simple to-do list reminding the user rather than being a full-blown task management or issue handling system.

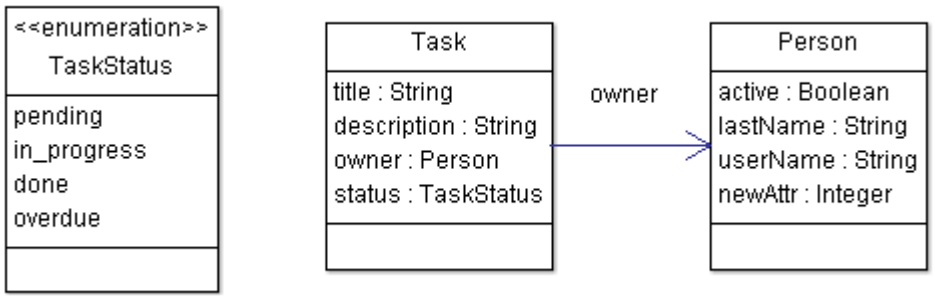


**Figure 10.** UML class diagram of the task model.

### 4.2.2. Authorization Model

The authorization model defines what Products the user has access to when he logs into the application. The model also provides a way to give a user one or more global application administrator roles. The RAMP authorization model is loosely based on Role-Based Access Controls as presented by Ferraiolo & Kuhn (1992).

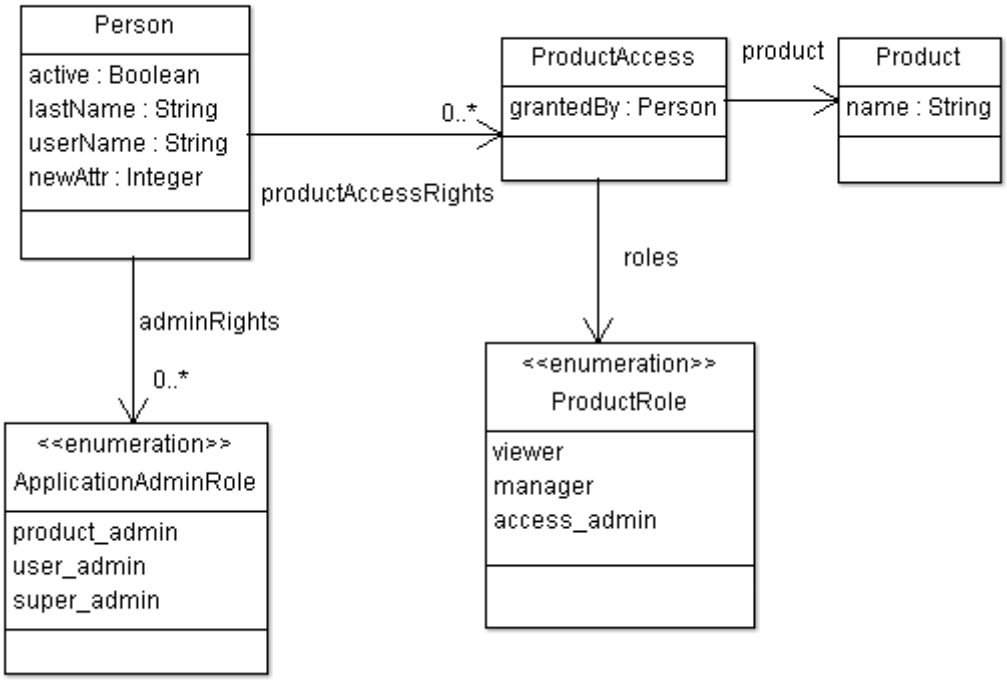


**Figure 11.** UML class diagram of the authorization domain model.

For each Product the user has access to there is a ProductAccess instance. ProductAccess defines which product there user gets access to and what roles the user has within that product. The product roles are defined as follows:

*Viewer* only has read-only access to the Products requirement and hence cannot modify or create new requirements.

*Manager* allows the user to create, modify and delete requirements for the target product.

*Access admin* allows the user to give other users access to the given product.

A user can also optionally have one or more of the following ApplicationAdminRoles:

*Product admin* is allowed to create, remove and edit products within the entire RAMP installation. This includes product specific settings such as the abstraction levels and requirement states.

*User admin* is allowed to give new users access to the system and products.

*Super admin* gives unrestricted administrator access to the entire RAMP installation and all of its data. Only a super admin can appoint other administrators.

## 4.3. Use Cases

Two UML use case diagrams were constructed, one for general usage of the application and one for application administration. In the general usage diagram the roles of the actors were based on the different roles in RAM (requirement owner, source and manager). The requirement owner was excluded since it was assumed that only the source and manager are guaranteed to be internal persons that have access to the system. The use cases were based on the action steps in the RAM process. The administration use cases and roles were based on the authorization domain model. The use case diagrams can be found in Figure 12 and Figure 13

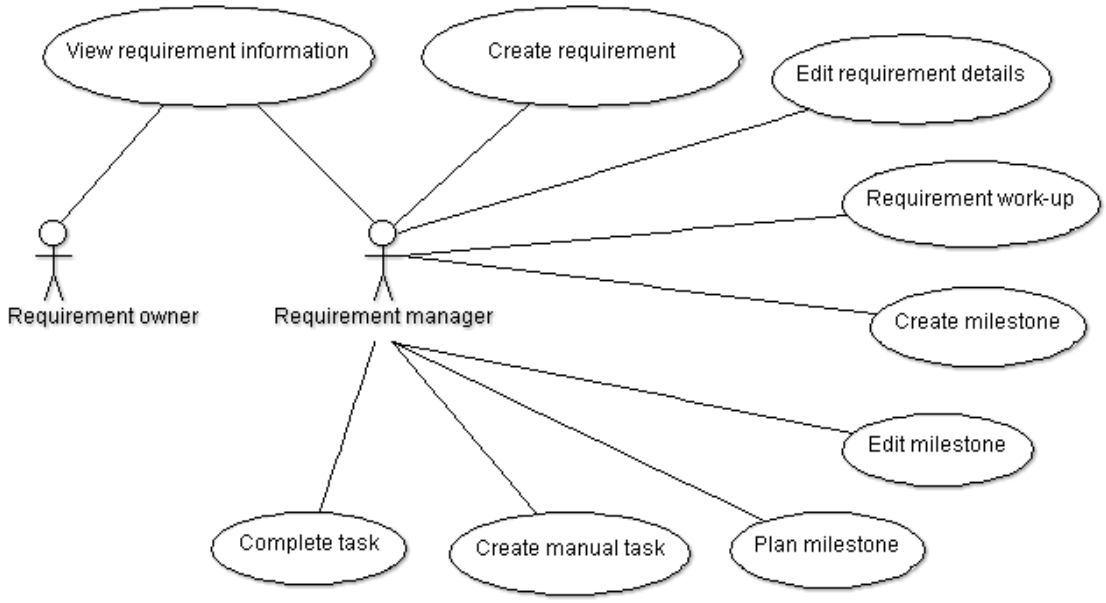


**Figure 12.** Use case diagram for general RAMP usage.

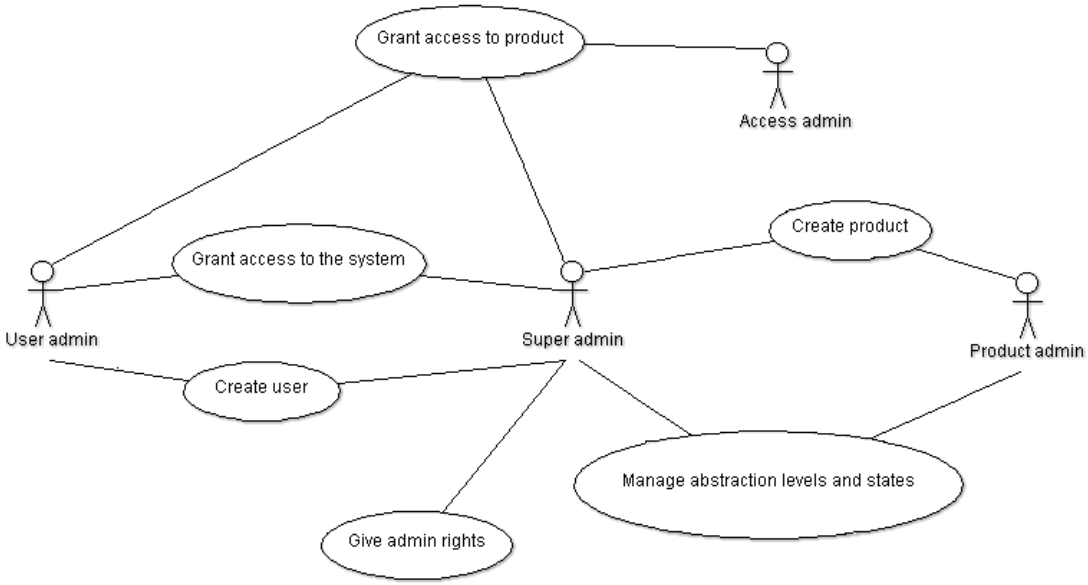


**Figure 13.** UML use case diagram for RAMP administration.

## 4.4. User Interface Design

The user interfaces for the desktop client were designed. It was assumed that the server would have few if any user interfaces. Additionally priority was given to user interfaces that were expected to be used frequently by regular users (requirement owners and managers), e.g. various administration user interfaces were excluded.

First a container application or *main window* of the desktop client was designed. The fixed parts of the main window are a ribbon control at the top of the window and a product tree and a task list on the left edge of the window.

The ribbon is a tabbed toolbar control introduced by Microsoft in their redesigned Microsoft Office Fluent user interfaces of the Microsoft Office 2007 applications. (Microsoft 2013b.)

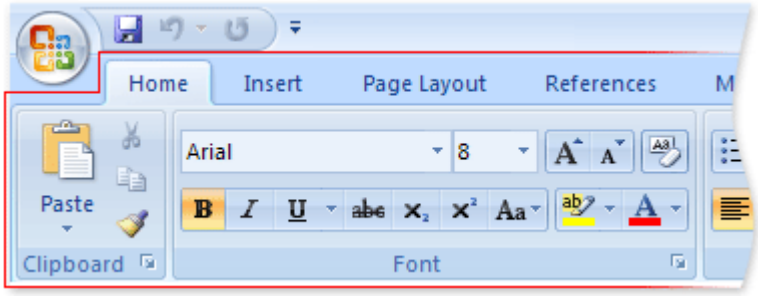


**Figure 14.** The Ribbon as it appears in Microsoft Office Word 2007. (Microsoft 2013.)

The product tree can be used to view and navigate the requirements and milestones of the current product. The tree has a dropdown on the top to group the requirements either by milestone or by abstraction level. Double clicking on a milestone in the tree can be used to open the milestone planning view for the selected milestone. Double clicking on a requirement in the tree can be used to open the requirement view for the selected requirement.

The task list shows the current pending tasks for the user. Tasks can be automatically generated by the system (e.g. when a requirement needs work-up) or manually by the user for his own task tracking purposes.

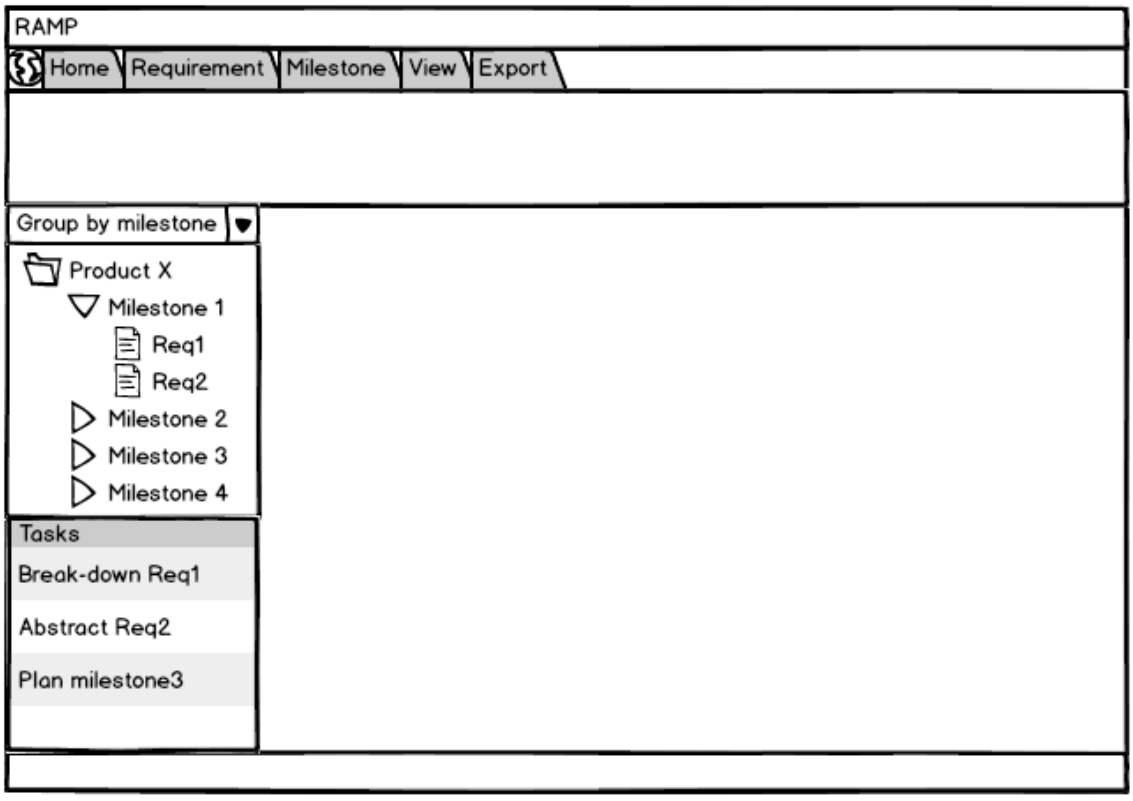


**Figure 15.** Mockup of the RAMP desktop client main window showing the ribbon, product tree and task list.

The ribbon would contain the following elements:

RAMP button — Button with the RAMP logo that would open a dropdown menu with options to open and save projects. The menu is similar to the “File” menu in traditional Windows applications.

Home tab — Tab containing buttons to create a new requirement or a milestone. Additionally a search box would be available to search for requirements by their title or description. Toggle buttons would be available to show or hide each abstraction level.


**Figure 16.** Home tab mockup.

Requirement tab — Context sensitive tab that becomes active automatically when a requirement is selected. Contains a button to open a dialog to edit the requirements details and buttons to abstract or break down the requirement.

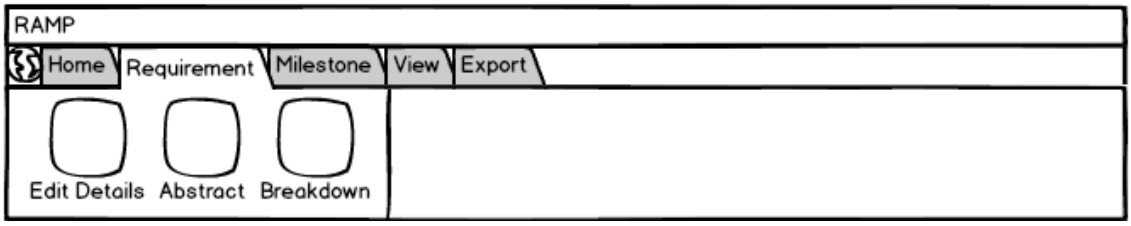


**Figure 17.** Requirement tab mockup.

| | |
|---|---|
| Milestone tab | Context sensitive tab that becomes active automatically when a milestone is selected. Contains buttons to edit the milestone details, freeze the scope of the milestone and to mark the milestone as released. Freezing the scope prevents requirements from being added or removed from the milestone unless someone reopens the milestone for editing. Marking a milestone released also makes it read-only and released milestones are hidden or moved towards the bottom when listing milestones. |


**Figure 18.** Mockup of milestone tab.

View tab — Tab containing toggle buttons to select what view is visible in the available space in the main window below the ribbon. A tree and a grid view are available for requirements and a planning view for milestones.

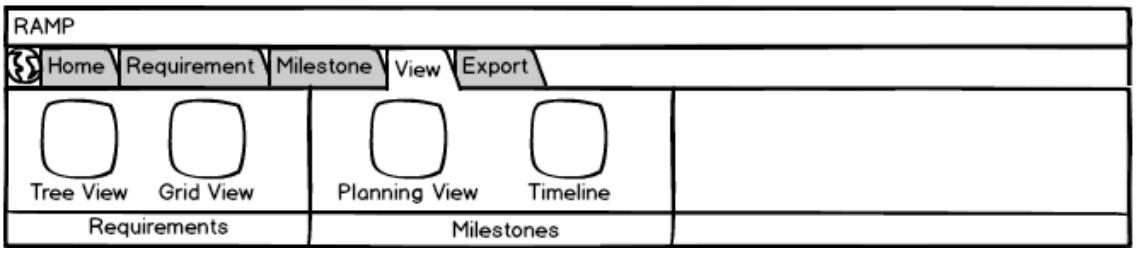


**Figure 19.** View tab mockup.

Export tab — Contains buttons to export requirements or milestone definitions into Microsoft Word documents.

*The requirements tree view* shows a group of related requirements as a tree. The requirements are grouped onto different tree levels based on their abstraction level. Connection lines are drawn to show parent/child relationships between requirements.

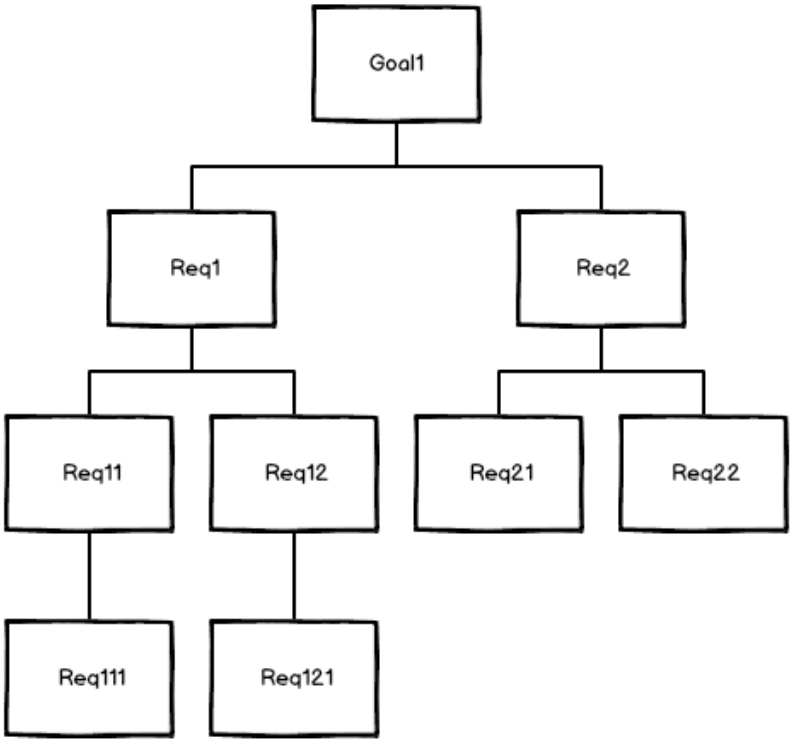


**Figure 20.** Requirements tree view mockup.

*The requirements grid view* shows requirements as a grid with one row per requirement. Since requirements can have freely configurable attributes the grid view should also support dynamically generate columns based on attributes and their data types. The requirement rows are grouped into parent rows based on their abstraction level.

| | Title | Milestone |
|---|---|---|
| Goal level | | |
| | Req1 | Milestone1 |
| | Req2 | Milestone2 |
| Feature level | | |
| | Req11 | Milestone1 |
| | Req12 | Milestone1 |

**Figure 21.** Requirements grid view mockup.

*The milestone planning view* shows one or more milestones as wide horizontal boxes where requirements can be dropped from the product requirement tree to allocate the requirement to that milestone. Requirements can also be reallocated by dragging them from an existing milestone box to another. If a higher level requirement is allocated to a milestone all the break-down requirements below it will also be included since it would be impossible to meet e.g. a goal without implementing all the features and functions related to it first. Dependant requirements are grouped together into a grey box and only the whole group can be moved around. Please see Figure 22 below for an illustration.

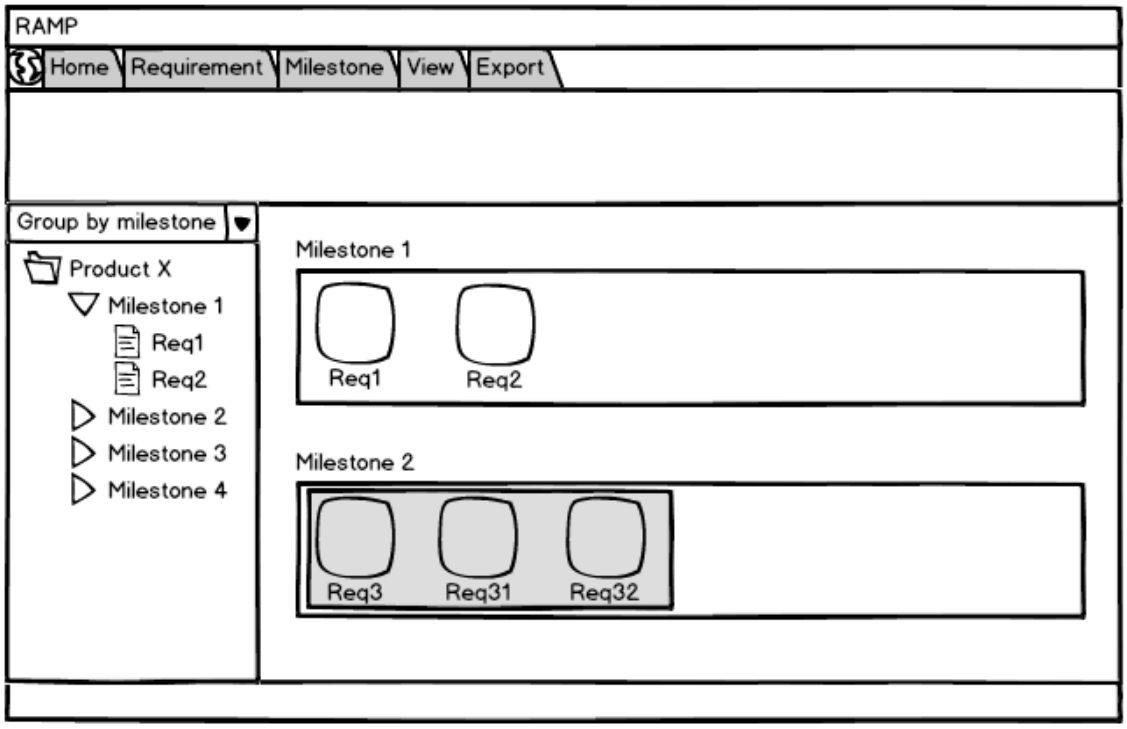


**Figure 22.** Milestone planning view mockup.

Based on the action steps in the RAM process a need for the following user interfaces was identified:

*A requirement editor* that allows entering information for a new requirement or editing an existing one. The information includes the requirements abstraction level. This corresponds to the *specify* and *place* steps in the RAM process. The requirement editor should also have a dynamically generated attributes section for any custom attributes that have been added to the system. The type of the attribute editing field (e.g. text field, file upload, drop-down menu) should be based on the attributes data type (e.g. string, document, enumeration).

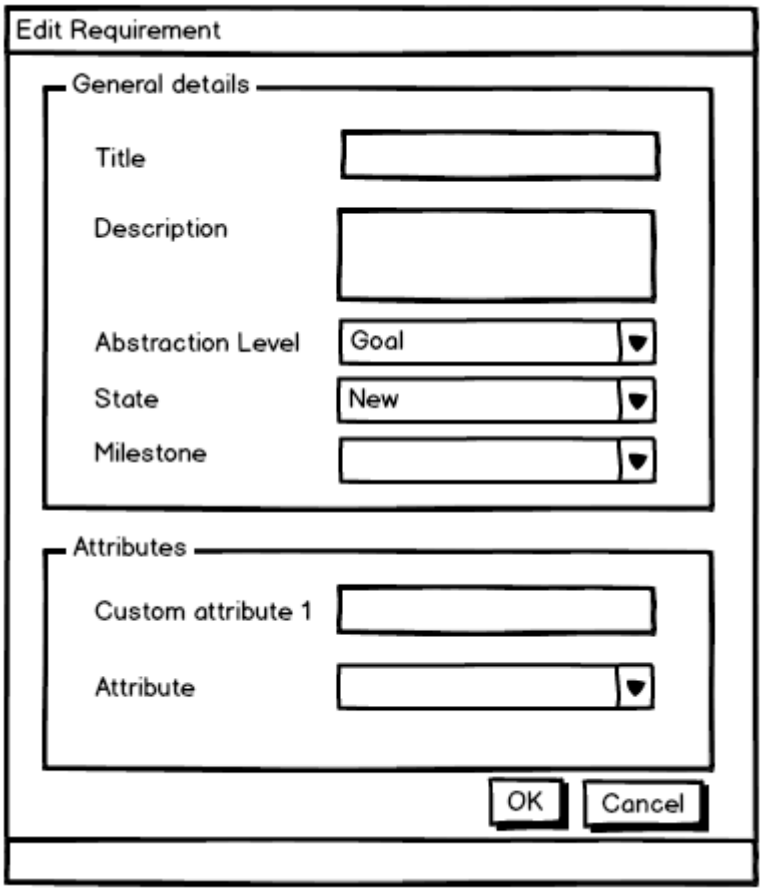


**Figure 23.** Requirement editor dialog mockup.

Corresponding to the *workup* step in the RAM process workup validation is done programmatically to ensure that the workup rules of RAM are met. The original rules in the example RAM implementation, eg. *(1) No requirement may exist without having a connection to the Product Level*, and *(2) All requirements have to be broken down to Function level* are slightly reformulated to a more generic version: *(1) No requirement may exist without having a connection to the topmost abstraction level*, and *(2) All requirements have to be broken down to the lowest abstraction level that has the*

*“mandatory for workup” boolean flag set to true*. The output of the validation is tasks for the user, eg. tasks to create more abstract or more specific workup requirements or to connect the new requirement to existing requirements to ensure that the workup rules are met. The user can then choose to complete the workup during the same session or continue it later; the tasks will still be there when he or she returns.

# 5. IMPLEMENTATION

In this chapter we report on the implementation of the RAMP proof-of-concept prototype. Some preliminary discussion about the usability of the prototype is also included.

## 5.1. Domain Model

The core domain model was implemented as a C# class library so that it can be shared between the desktop client and server solutions. An interface called IDataService was created to define methods that the clients can remotely invoke to fetch and store data on the server. All the prototype code targeted the 4.0 version of the .NET framework to allow using recent .NET features such as Language Integrated Query (LINQ) and the Task Parallel Library (TPL).

## 5.2. Desktop Client

A proof-of-concept prototype of the desktop client user interface was also implemented using WPF. A mock implementation of IDataService called DesignDataService was created to provide data to test the UI mockup without needing to implement the server side. A series of view model classes were also implemented according to the chosen

MVVM architecture to act as wrappers or converters between the domain model and the user interface views. The view models also implemented the observer pattern to make data binding from the user interface convenient. The MVVM Light Toolkit from Laurent Bugnion (Bugnion 2013) was used when implementing the views and view models. MVVM Light provides Visual Studio templates and various utilities such as a message broadcasting mechanism to make implementing MVVM applications more convenient.

The layout of the various user interfaces was defined in a markup language called XAML. XAML is supported by the WPF framework out of the box and it is the preferred way to create WPF user interfaces. WPF also includes some XAML extensions to make user interface implementation more convenient such as two-way data binding between the view and the view model. (Microsoft 2013c)

For example to create a text field that is bound to the Title property of the view model one would use the following markup:

```
<TextBox Text="{Binding Title}" Height="25"></TextBox>
```

The binding for text fields works both ways by default, the text fields value will initially be populated from the view models Title property and if the user edits the value through the text field the value will be updated back to the view model automatically once the field loses focus.

*The main window* was implemented with a Microsoft Ribbon for WPF (Microsoft 2013a) component on the top of the window. The bottom area of the window consists of an AvalonDock docking windows control where various child controls can be docked depending on the context (AvalonDock 2013). A screenshot of the main window showing the ribbon, product tree, task list and requirement tree view controls can be seen below in Figure 24.

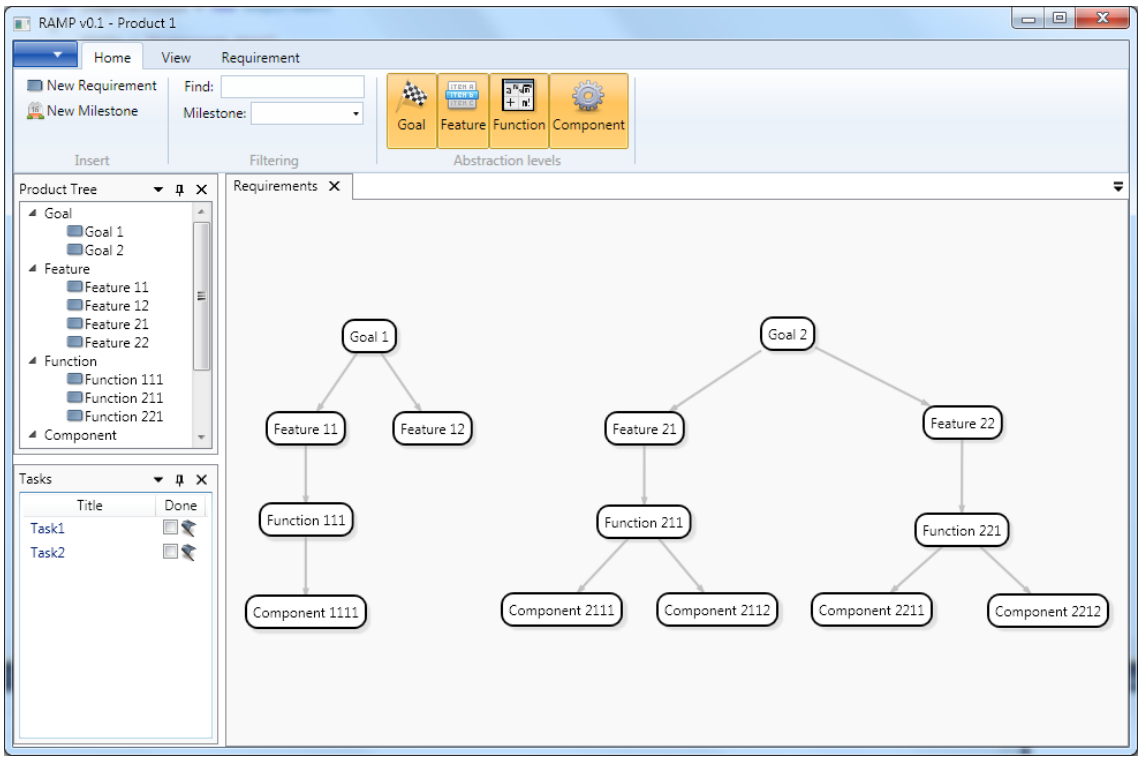


**Figure 24** Screenshot of RAMP desktop client prototype main window.

*The requirement tree view* was implemented using the Graph# graph layout framework (Graph# 2013). Wrapper classes were implemented to represent abstraction levels and requirements as a directed graph so that they could be rendered as a tree.

The tree view seems quite effective in visualizing relations between requirements, however it also seems that the tree structures take a large amount of screen real estate compared to for example a table presentation. This might become an issue in products with a very large number of requirements, though the issue can probably be mitigated with the filtering options available in the home ribbon tab, e.g. free text search and toggling the visibility of different abstraction levels as needed. The Graph# framework also supports zooming and panning of the graphs which could be utilized to navigate the requirements. The requirement tree view could also be linked with the product tree on

the left so that if the user selects a requirement in the product tree it is automatically centered in the requirement tree view also.

*The requirement grid view* was implemented using the DataGrid user control that is included in the WPF framework. Each abstraction level is shown as a groping row with all requirements for that level shown under it as child rows. A screenshot of the grid view can be seen in Figure 25.

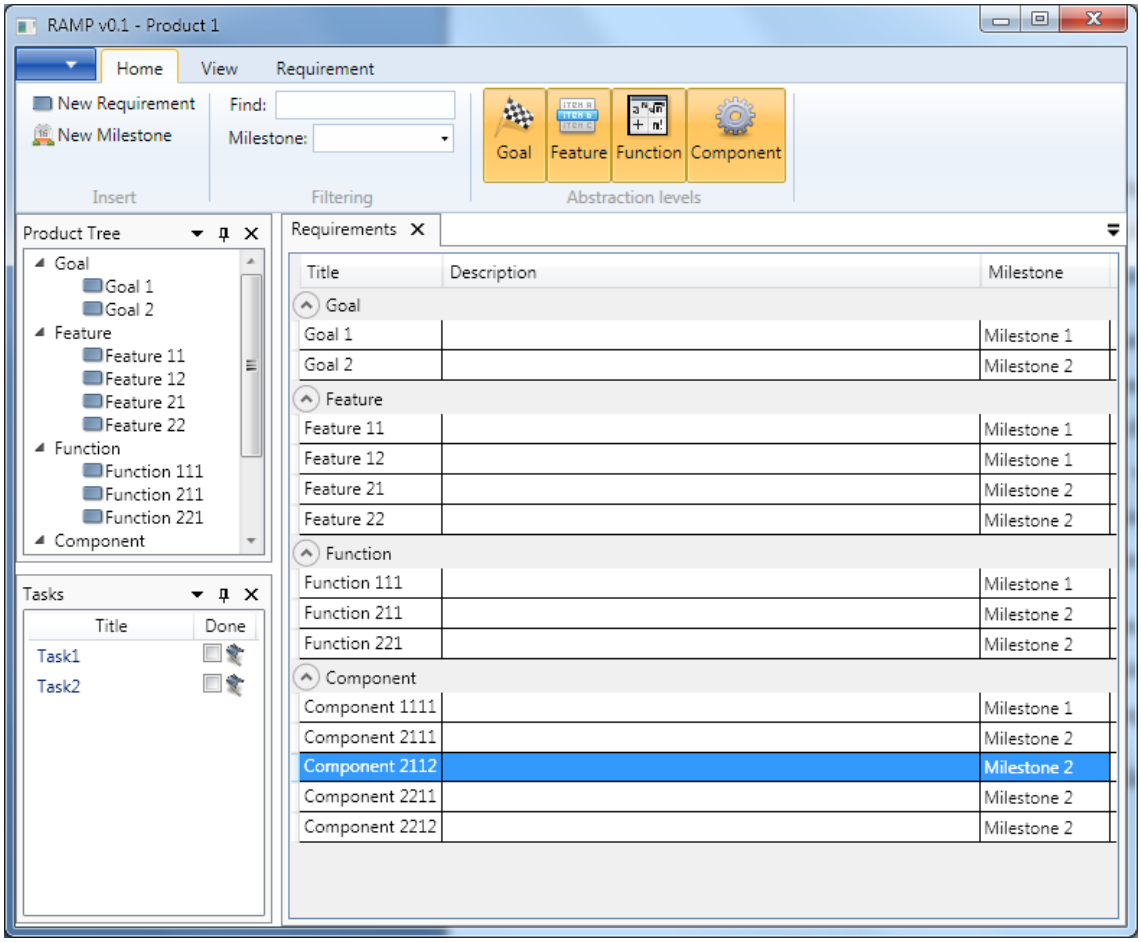


**Figure 25** Screenshot of the requirement grid view.

It was noted that while the grid presentation is compact and seems to work well for viewing a large amount of requirements at once, it is hard to describe relations between a high level requirement and various related break-down requirements in this grid format.

One possibility to visualize relationships between requirements in the grid view could be to hide all unrelated requirements once a requirement row is selected and only show requirements that are directly related to the selected requirement.

Basic versions of the requirement and milestone editing windows were also implemented.

# 6. CONCLUSIONS

The Requirements Abstraction Model was reviewed and evaluated as one solution to market driven requirements engineering. Related literature such as the agile requirements refinery and quality function deployment was also reviewed.

RAM seems like a reasonably lightweight and easy to adopt requirement engineering process. There is also some empirical evidence to suggest that it does improve requirement quality in exchange for some extra effort. The clearly defined constructs defined by RAM (abstraction levels, attributes, roles, states) seem like they can be quite easily translated into the design of a new requirements engineering tool. Hence RAM was chosen to be the basis for the design of RAMP.

The key concepts of RAM were applied into the architecture, domain model, use cases and the user interface design of a requirement engineering tool called RAMP. The primary output of this thesis is the architecture and design of RAMP. A proof-of-concept prototype of the RAMP desktop client was also implemented and presented as part of this thesis.

A minimum viable product could be developed based on the work in this thesis. A case study with some early adopters could be constructed to look for an improvement in requirements quality and to evaluate whether end users find the tool more valuable over traditional Microsoft Excel spreadsheets or other requirement engineering tools. It would also be interesting to see whether a dedicated tool such as RAMP makes it easier for organizations to adopt the Requirements Abstraction Model and how RAM performs compared to other models.

# REFERENCES

Akao, Yoji (1997). *QFD: Past, Present, and Future*. International Symposium on QFD 1997, Linköping, Sweden [online] [cited 19.1.2014]. Available on the World Wide Web: <URL: http://www.qfdi.org/QFD_History.pdf>.

AvalonDock (2013). AvalonDock [online] [cited 23.6.2013]. Available on the World Wide Web: <URL:http://avalondock.codeplex.com/>.

Brooks, Fred P. (1987). No Silver Bullet — Essence and Accidents of Software Engineering. *IEEE Computer* [online] 20:4 [cited 20.1.2013], 10-19. Available on the World Wide Web: <URL: http://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet/>. Freed P. Brooks also authored the well known "The Mythical Man-Month: Essays on Software Engineering" book about software engineering and project management.

Bugnion, Laurent (2013). *MVVM Light Toolkit* [online] [cited 23.6.2013]. Available on the World Wide Web: <URL:http://www.galasoft.ch/mvvm/>.

Carlshamre, Pär & Björn Regnell (2000). *Requirements Lifecycle Management and Release Planning in Market-Driven Requirements Engineering Processes*. International Workshop on the Requirements Engineering Process: Innovative Techniques, Models, and Tools to support the RE Process, 6th-8th of September 2000, Greenwich UK, preceding the DEXA Conference. Available on the World Wide Web: <URL:http://dl.acm.org/citation.cfm?id=790511>.

Christel, Michael G. & Kyo C. Kang (1992). Issues in Requirements Elicitation. *Technical Report CMU/SEI-92-TR-012* [online] [cited 20.1.2013]. Available on the World Wide Web: <URL:http://www.sei.cmu.edu/reports/92tr012.pdf/>.

Dinu, Valentin & Prakash Nadkarni (2007). Guidelines for the effective use of entity–attribute–value modeling for biomedical databases. *International Journal of Medical Informatics* [online] 76 [cited 12.5.2013], 769-779. Available on the World Wide Web: <URL:http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2110957/>.

Ferraiolo, David F. & D. Rishard Kuhn (1992). Role-Based Access Controls. 15th National Computer Security Conference, Baltimore MD [online] [cited 12.5.2013], 554-563. Available on the World Wide Web: <URL: http://csrc.nist.gov/groups/SNS/rbac/documents/ferraiolo-kuhn-92.pdf>.

Gorschek, Tony & Claes Wohlin (2006). Requirements Abstraction Model. *Requirements Engineering Journal* [online] 11:1 [cited 20.1.2013], 79-101. Available on the World Wide Web: <URL: http://www.wohlin.eu/rej06.pdf>.

Gorschek, Tony, Per Garre, Stig BM Larsson & Claes Wohlin (2007). Industry Evaluation of the Requirements Abstraction Model. *Requirements Engineering Journal* [online] 12:3 [cited 20.1.2013], 163-190. Available on the World Wide Web: <URL: http://www.wohlin.eu/rej07.pdf>.

Gossman, John (2005). *Introduction to Model/View/ViewModel pattern for building WPF apps* [online] [cited 28.4.2013]. Available on the World Wide Web: <URL: http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx>.

Graph# (2013). Graph# [online] [cited 23.6.2013]. Available on the World Wide Web: <URL:http://graphsharp.codeplex.com/>.

Haag, Stephen, M. K. Raja & L. L. Schkade (1996). Quality function deployment usage in software development. *Communications of the ACM* [online] 39:1 [cited 10.5.2013], 41-49.

Karlsson, Joachim (1997). Managing software requirements using quality function deployment. *Software Quality Journal* [online] 6:4 [cited 10.5.2013], 311-326.

Koski, Jouko (2003). *Quality Function Deployment in Requirements Engineering: A Review and Case Studies* [online] [cited 7.4.2013]. MBA Thesis. Helsinki University of Technology, Executive School of Business. Available on the World Wide Web: <URL: http://www.soberit.hut.fi/core/reports/mba-jouko-koski.pdf>.

Microsoft (2013a). *Microsoft Ribbon for WPF* [online] [cited 23.6.2013]. Available on the World Wide Web: <URL: http://msdn.microsoft.com/en-us/library/ff799534.aspx>.

Microsoft (2013b). *Use the Ribbon instead of toolbars and menus* [online] [cited 23.6.2013]. Available on the World Wide Web: <URL: http://office.microsoft.com/en-us/help/ use-the-ribbon-instead-of-toolbars-and-menus-HA010089895.aspx>.

Microsoft (2013c). *XAML in WPF* [online] [cited 23.6.2013]. Available on the World Wide Web: <URL: http://msdn.microsoft.com/en-us/library/ ms747122(v=vs.100).aspx>.

Microsoft Patterns & Practices Team (2009). *Microsoft Application Architecture Guide, 2nd Edition* [online] [cited 28.4.2013]. Available on the World Wide Web: <URL: http://msdn.microsoft.com/en-us/library/ee658109.aspx>.

Poel, Ibo Van De (2007). Methodological problems in QFD and directions for future development. *Research in Engineering Design* [online] 18 [cited 7.4.2013], 21-36. Available on the World Wide Web: <URL: http://www.clemson.edu/ces/cedar/images/f/f7/ME870-QFD-Poel.pdf>.

The Standish Group International, Inc. (1995). *The Chaos Report*. Technical report [online] [cited 28.4.2013]. Available on the World Wide Web: <URL: http://www.csus.edu/indiv/v/velianitis/161/ChaosReport.pdf>.

Vlaanderen, Kevin, Slinger Jansen, Sjaak Brinkkemper & Erik Jaspers (2011). The agile requirements refinery: Applying SCRUM principles to software product management. *Information and Software Technology* [online] 53 [cited 28.4.2013], 58-70. Available on the World Wide Web: <URL: http://www.sciencedirect.com/science/article/pii/S0950584910001539>

Vähäniitty, Jarno & Kristian Rautiainen (2008). Towards a conceptual framework and tool support for linking long-term product and business planning with agile software development. *International Conference on Software Engineering* [online] 25–28 [cited 10.5.2013]. Available on the World Wide Web: <URL: http://www.soberit.hut.fi/sprg/projects/atman/publications/international%20research%20publications/200805_Towards%20a%20Conceptual%20Framework_SDG2008.pdf>

Wang, Shaohua Alex, Yang Fann, Huey Cheung, Frank Pecjak, Barg Upender, Adam Frazin, Raj Lingam, Sarada Chintala, Gladys Wang, Marc Kellogg, Robert L. Martino & Calvin A. Johnson (2004). Performance of Using Oracle XMLDB in the Evaluation of CDISC ODM for a Clinical Study Informatics System. *Proceedings of the 17th IEEE Symposium on Computer-Based Medical Systems* [online] [cited 12.5.2013] 594. Available on the World Wide Web: <URL: http://dl.acm.org/citation.cfm?id=1009377.1009532>