RequirementsManager

Enrico Marchionni
enrico.marchionni@studio.unibo.it

August 16, 2024

Abstract

ReM[1] is an administrative software that makes you minimize time and maximize result in the *Software Engineering* process of collecting and creating requirements.

---

[1] *ReM*, Requirements Manager.

Contents

Analysis

It's requested to realize a database that allows a team to optimize Software's Analysis by managing Requirements creation. Each one of them descends from a Request that could be created by a team member or by the project customers.

Preliminary

Interview

The aim is to manage requirements during all phases of software production and revision storing away all final and in production data relative to releases already published or yet to publish. Everything has to start from a *Request*. Once it has been created the relative *Requirement/s* can be developed.

The system admin has to create users and the relative database linked to a software. Requests can be created by a team member or a customer. Requirements can be developed by team members only. Requests and therefore Requirements can obviously be added or even edited in later releases. It's not important to keep track of every update or change, the important is to keep track of final data just before a release. The timeline can be added by a team member only and domain conditions has to be respected.

Requirements have to be structured with a title, a type (functional, non functional), a version, a description, a body, one or more files, progress data, a creation time and a last modified time. Moreover for a requirement it has to be known the user that did the first and last modifies and an history of updates with times and users that did them. Furthermore is important to consider that requirements could be arranged in a tree structure in the most complex cases.

Requests structure consists of a title, a type, a version, a description, a body, one or more files, a creation time and a last modified time. Besides for a request it has to be known the user that did the first and last modifies and an history of updates with times and users that did them.

A requirement has to be associated with one and only one request, a request can be associated with multiple requirements.

A request has to be approved by a team member before developing it into a requirement. After the approval a customer has to accept the request approval, only then the relative requirement can be developed. When the approval is done the request mustn't be able to be modified.

Users has to be saved with personal info such as username, name, surname, email, phone and optionally the company name.

The releases should also save a short description and a name that has to be an identifier between the timelines. It's important to permit the creation of a new timeline only if every requirement was completed, or this mustn't be permitted. In addition every request has to be approved, formalized into requirement and completed before the version creation.

Every data that was affected by the version cannot be overridden so their last modified time should be previous to the version creation time.

Glossary

A glossary of terms with description, synonyms and links used in Interview is here explained in Table 1.

Term	Description	Synonyms	Links
Request	Customer request of what the system should do		Requirement, release
Requirement	Formal description of what the system should do		Request, release
	Requirements + Requests	Data	
Editor	User that can add Requirements	Team member	
Guest	User that can only add Requests	Customer	
User	Editor + Guest	User	Editing, versioning
Release	A timeline that confirms the fact that a software version was released	Version, timeline	Data

Table 1: Glossary of terms

Requirements

This section aims to restructure requirements according to the Preliminary phase.

Functional

- General

  - It is demanded to realize a database that manages requirements;
  - Information that have to be stored are **Requests**, **Requirements**, **Users** and **Releases**;

- Users

  - **Users** are structured with username, name, surname, email, phone and optionally the company name;

- **Users** are divided into two groups:

    * **Editors** - they approve *Requests*, develop relative *Requirements* and decides *Releases*; in some cases they could also create *Requests*;

    * **Guests** - they create *Requests* that once approved cannot be edited but will be developed into *Requirements*;

- Requests

    - **Requests** are structured with a title, a type, a version (positive integer), a description, a body, one or more files, a creation time and a last modified time;

- Requirements

    - **Requirements** are structured with a title, a type (functional, non functional), a version (positive integer), a description, a body, one or more files, progress data (a percentage), a creation time and a last modified time;

    - **Requirements** could be arranged in a tree structure;

- Releases

    - **Releases** are structured with a unique name (software version name), a short description and the creation time;

    - **Releases** are timelines that can be created only if all *Requests* had been approved and converted into already fully developed *Requirements*;

- Conditions

    - all **Requests** and **Requirements** relative to a **Release** have to be historicized;

    - **Requests** and **Requirements** could be modified or even invalidated in later **Releases**;

    - a **Requirement** has to be associated with one and only one **Request**, a **Request** can be associated with multiple **Requirements**.

Non-Functional

- Data access has to be really fast;

- CRUD[2] operations should be non blocking in final GUI[3];

---

[2]*CRUD*, Create, read, update, delete.

[3]*GUI*, Graphical user interface.

Actions

The mainly requested actions are:

1. Subscribe a new user;

2. View a request;

3. Create a request;

4. Update a request;

5. Approve a request;

6. View a requirement;

7. Create a requirement;

8. Update a requirement;

9. Disable a requirement;

10. Create a release;

11. Show requirements and requests relative to a specific release;

12. View single requirement history (every change for each release);

13. Obtain an average of the progress time for completing a branch of the requirements tree;

Conceptual design

In production I chose to use a mixed strategy to build the E/R diagram.

A first version of the E/R diagram explained in 1. This diagram represents the skeleton of the E/R diagram. Here I made the "mistake" of mixing time concepts with static ones. In fact Requests and Requirements are single entity sets but their historic of a Version (Release) is not well managed.

Figure 1: Version 1
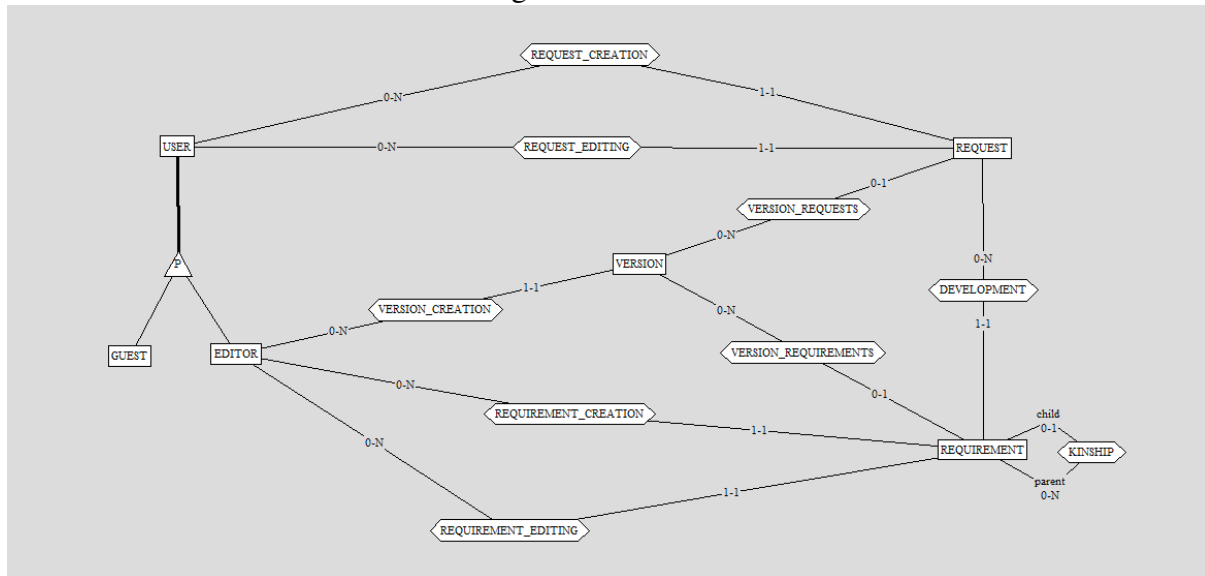
A second version of the E/R diagram explained in 2. Here a clean schema shows how Requests and Requirements should be traced in their historic versions. Furthermore is important to watch that an historic version is always created when a new Release is published, independently from Requests or Requirements changes. This could lead to an inefficient use of storage...
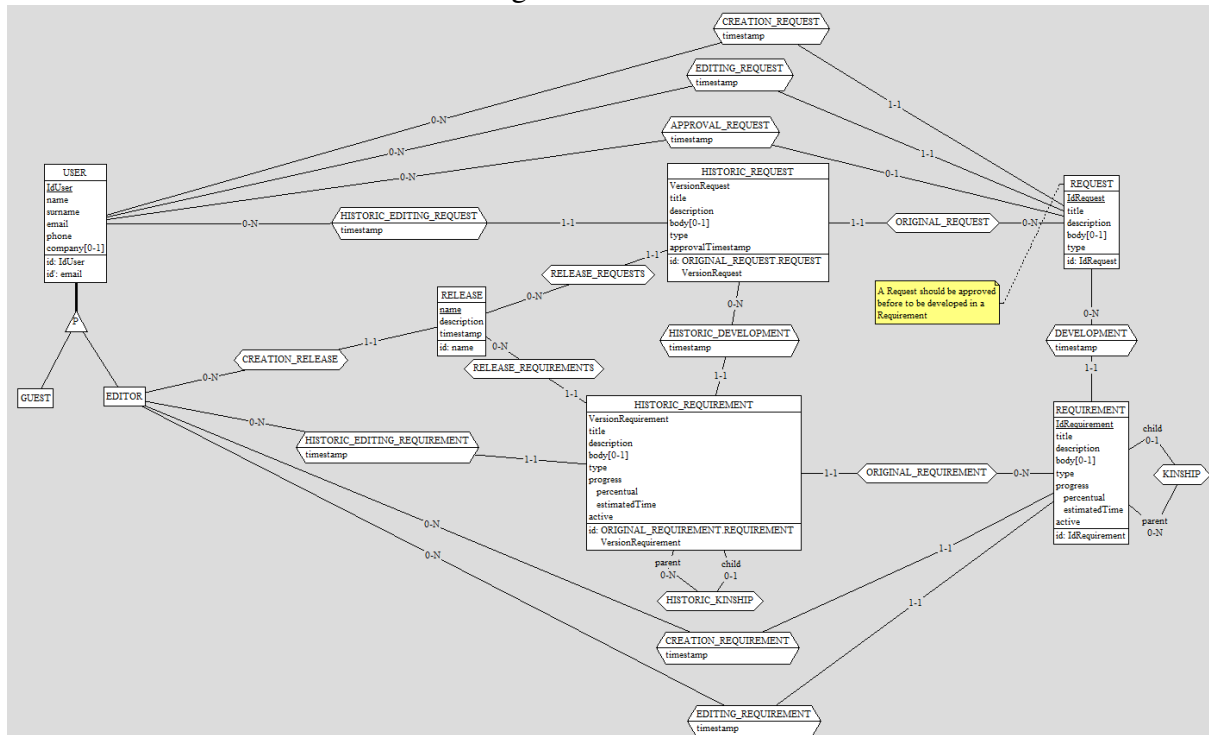

Figure 2: Version 2
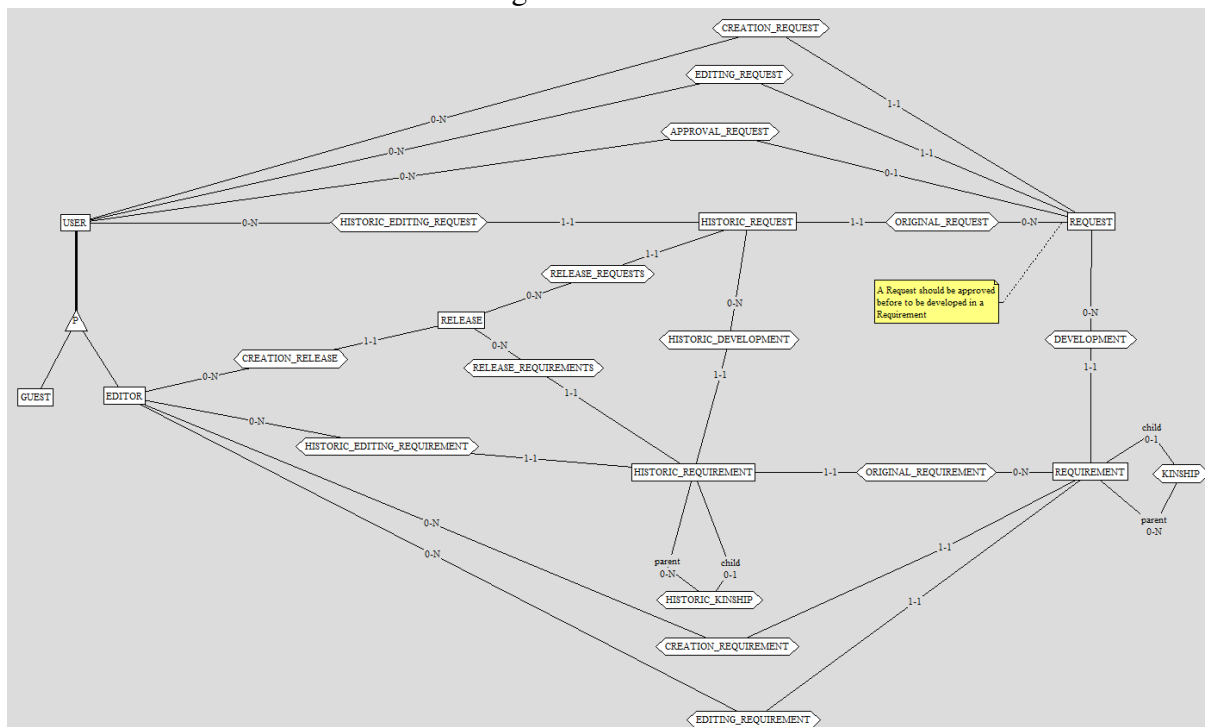
A third version of the E/R diagram explained in 3. Here every attribute is present and shows which data are stored for each entity/relationship.

Figure 3: Version 3

Logical design

Table of volumes

Data and expected volumes are described in Table 2.

Concept	Construct	Volume
USER	E	100
GUEST	E	90
EDITOR	E	10
REQUEST	E	50
HISTORIC_REQUEST	E	40
REQUIREMENT	E	70
HISTORIC_REQUIREMENT	E	50
RELEASE	E	5
DEVELOPMENT	R	70
HISTORIC_DEVELOPMENT	R	50
KINSHIP	R	5
HISTORIC_KINSHIP	R	5
CREATION_RELEASE	R	5
RELEASE_REQUESTS	R	40
RELEASE_REQUIREMENTS	R	50
CREATION_REQUEST	R	50
EDITING_REQUEST	R	50
HISTORIC_EDITING_REQUEST	R	40
APPROVAL_REQUEST	R	45
CREATION_REQUIREMENT	R	70
EDITING_REQUIREMENT	R	70
HISTORIC_EDITING_REQUIREMENT	R	50
ORIGINAL_REQUEST	R	40
ORIGINAL_REQUIREMENT	R	50

Table 2: Volumes of data

Operations and frequency

Operations and frequency are described in Table 3.

Code	Operation	Frequency
1	Subscribe a new user	10 every month
2	View a request	200 every day
3	Create a request	2 every day
4	Update a request	5 every day
5	Approve a request	1 every day
6	View a requirement	200 every day
7	Create a requirement	2 every day
8	Update a requirement	50 every day
9	Disable a requirement	1 every month
10	Create a release	2 every year
11	Show requirements and requests relative to a specific release	50 every day
12	View single requirement history (every change for each release)	10 every day
13	Obtain an average of the progress time for completing a branch of the requirements tree	20 every day

Table 3: Operations and frequency