Titel

Autor

***Abstract*—Product development is facing new challenges as a result of the demand for increasingly individualized products in small batch sizes and short time to markets. By providing software tools, digitalization is an enabler for agile product development. Stakeholders such as the customer, the project manager, the requirement manager and the solution manager can develop the product together and react quickly to each other's demands. The individual aspects of product development are often considered separately. Each stakeholder uses its own platform to generate outputs. As a result, the connection between the components of product development is missing. Without a common context between the different departments, valuable information and work results are lost. This complicates aspects such as scheduling, priority planning or the definition of new requirements as there is no common basis of information. To solve these issues, we have developed an open source platform on which all relevant results can be managed. Our platform combines project management, requirements engineering and solution engineering to offer a transparent insight into the progress of product development. For the development of the software, tools from web development were used to create a cross-platform user experience built on a client server architecture. The focus was on making the software lightweight and intuitive to use. The software was continuously tested during the development process and showed potential to add value to the project management of products. Nevertheless, important adjustments still need to be made to create a solid user experience. This article describes the development of such a platform and evaluates its benefits, drawbacks and what lessons we have learned. In the Future the software will be further developed in follow-up projects to make it ready for practical use and to extend its functions. There are already plans to extend the software in the direction of a simulation tool and virtual reality.**

[1] [2] [3]

I. Introduction

Context and motivation

Every engineering design process starts with defining the needs to be covered in the resulting product. In order to successfully develop the product, it is important to capture the customer's needs precisely and effectively.

The trend is for products to become increasingly complex and shorter time to markets. To meet today's challenges, companies must shorten the cycle time of new products, reduce the resources used, increase the quality of the product. To meet this business goals, companies need to collect, analyze and apply requirements extensively and effectively.

Requirement engineering plays an important role as a discipline and holds many challenges. Clear requirements and a focus on the end user with management support represent half of the success of future product development.

Through proper requirement management, a product with higher quality can be developed in less time by meeting the right requirements because fewer iterations are required. The customer satisfaction is increased and through a clean documentation it is possible to reuse parts of the process for other products.

By collecting the information and by possible changes of the customer requirements, a lot of data comes up, which makes it difficult to keep an overview of the product development process. It is difficult to manage the large amount of data because of the cognitive capacity of humans. Project management tools help to get an overview of the project progress with visualizations in the form of dashboards.

Such a tool must take into account the dynamics of the requirements in the development process and provide support for collaborative and interdisciplinary cooperation. It should be possible to divide the engineering activities into sprints.

Since such tools do not cover all essential aspects such as project management, requirement engineering and solution engineering, we want to describe the development of such software and evaluate it in the course of this article. The software considers product development as an overall concept. This gives the stakeholders a transparent and interdisciplinary overview of the project progress and enables agile product development.

Problem

Product development is complicated by the fact that CAD data and documentation are kept separate. Because the customer does not have all of these necessary documents available in a clear manner, it is hard to follow the progress of the project. Requirement engineering task planning and scheduling are therefore difficult. By keeping the documentation in Excel lists or other documents, some engineering artifacts are lost. This lack of transparency in product development makes it difficult to implement agile product management. If the customer has no access to the development progress, there is no possibility of prioritizing times and requirements. He also cannot participate in design decisions and shape the product according to his wishes.

Solution

To address these issues, we developed a product development platform. The platform offers user management, member management project management and provides a 3D view for the uploaded CAD models. On our platform, users can create different products. For each product, members, versions, issues, comments and milestones can then be added and edited. As in GitHub, different versions can be created. Each version of the product contain a 3D CAD model that can be viewed like in a 3D CAD software. Members can be registered for a product which possess different permissions. With the help of the customer member role, the customer can participate in agile project management and track the current status of

product development. Issues, Comments and Milestones serve as project management tools. Milestones are filled with issues that can be open or closed. A milestone is considered complete when all associated issues are closed. Each issue contains a communication channel where the respective task can be evaluated. On the platform, the CAD model can be viewed in a 3D view and is connected to the integrated product management tools.

Outline

This article is structured as follows: In Section II we discuss related work on the problem defined previously. In Section III we present our original solution to the problem at hand. Then, in Section IV we evaluate our solution with respect to different criteria. Finally, in Section V we summarize our learnings and provide an outlook on future work.

II. Related work

Scientific approach

Commercial approach

III. Our solution

For the implementation of the project, the requirements for the software are defined first. The requirements are still general in this, first section and become more concrete during the development of the design. Following the requirement definitions a detailed solution is compiled step by step. The most important concepts of this software are thereby the software architecture, the data model, the function model, the permission model and the user interface. Thus, the software can be developed according to the defined concepts and evaluated afterwards.

Conception

The goal to be achieved is to combine project management with CAD data to create a compact and lightweight solution. Customers and companies can evaluate and control the development status on a common basis. For this purpose, the platform GIT was considered as a reference, which already offers a similar solution in the area of software development. Following this example, a platform is to be developed that combines the design process, project management and communication with the customer. For this purpose, clear user roles with corresponding permissions must be defined. For example, the app needs a user administration with an associated login function. At the top level, it must be possible to define which users are allowed to create products and edit the list of users. The person who creates a product is both owner and manager and can add product members who have different permissions. Depending on the distributed rights, members can add versions to the respective product. Each added version contains a new CAD model with version number, previous version and description. For each product it should be possible to create issues, which are filtered by open and closed issues. In each issue there is a communication channel in which the issues can be discussed. Furthermore, specific components of the model can be selected and referenced, and an issue can be closed or reopened. Another level of product management is built by the milestones. Milestones can be created, and different issues can be added to them. A list of open and closed issues as well as a chart show the progress of each milestone. Finally, the platform must provide settings for each product to define the member list and the product properties. All data generated on the platform should be stored in a database. The software should be as lightweight as possible and intuitive to use. The customer should be in the foreground and the user interface should be built from the user's point of view. In this way, a platform can be created that clearly combines product development, product management and communication with the customer.

Architecture

In the first step of the conception the architecture of the software is planned. The following picture [see Fig. 1 on page 2] shows the general structure of the software. It is a full stack app written in Typescript and consists of gateway, frontend, backend, common and the database at its core. Further, packages such as toolkit, worker and broker extend the scope of functions.

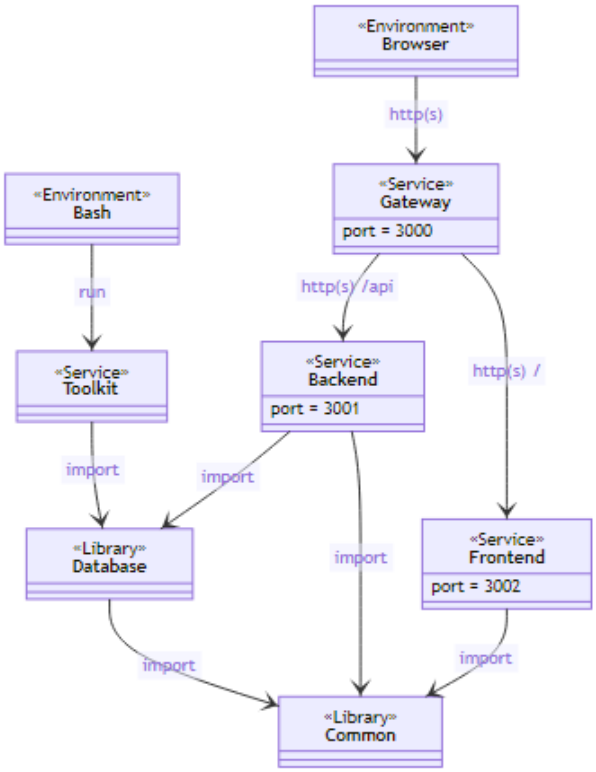

Fig. 1. Packages

Gateway: The gateway runs at port 3000. It establishes a connection between the services backend, broker, worker and frontend. When the app is started the gateway serves as the entry address. It displays the frontend, which itself runs on port 3003, while the other services are running in parallel. To

provide this functionality the gateway uses the express and the proxy library.

Frontend: The frontend provides an interface to make changes to the data via the offered functions of the developed API. To create the user interface React version 17 was used. It uses React states that can be altered with React hooks. The frontend communicates via HTTP requests with the backend to create, read, update and delete data with help of the API located in the backend. This communication runs with the library Axios. To display the uploaded cad models three.js was used. It offers a 3d view of the selected cad file. The view of the model can be adjusted by zooming and rotating, as well as positioning [see Fig. 9 on page 7]. The library Recharts provides a variety of different charts and graphs. Recharts is used to display a burn down chart in a specific user interface component [see Fig. 16 on page 9].

Backend: The backend of the app is built with the framework nest.js. This area of the software is the core component and provides those functions to meet the previously defined requirement specifications. Using nest.js, an API was created that interacts with the data using CRUD commands and sends it to the frontend. For that, it receives commands from the frontend and then executes the corresponding function of the API. The backend is controlled via HTTP requests. Therefore, the commands can be executed without the frontend. Good possibilities are offered by the program Postman or with Swagger. A Swagger integration was done in the backend during the development process. Swagger provides good documentation about the API and allows its functions to be executed without the frontend.

Common: The package common provides the necessary interfaces for the communication of the frontend and backend. In this section the data classes like product, version, issue, etc. for the different functions are defined. The rest file contains the interfaces for the HTTP requests. They define exactly which data may be sent from the frontend and which may be received again. For this purpose, the CRUD methods with transfer parameter and return value are defined in each interface of a data class.

Database: To store the data permanently a PostgreSQL database is used. It runs in a Docker container on port 5432. The database uses the data classes defined in the interface as entities. For the object relational mapping TypeORM is used.

Toolkit: The toolkit package provides functionality to fill the database with test data. Based on the classes defined in the interface, the respective objects are instantiated here. These objects are filled with fictitious data to test the functionality of the app. After the objects are instantiated, they are loaded into the database. For this purpose asynchronous function calls are executed.

Data model

The core of the software is in the backend. Via the API, the data can be manipulated using CRUD methods. This section describes the data model of the app. The necessary classes and entities are based on the interfaces in the common package. The difference between the data in the backend and in the database is that each entity class in the backend is divided into update, add and the base class. In the database, the classes are not divided. For example, the class product is mapped to a table in the database.

Entities: The data model contains following entities [see Fig. 2 on page 4]: User, Product, Version, Issue, Comment, Milestone, Member. At the top level of the data model is the user, who can create multiple products if he has the permission to do so. For each product versions, issues, comments, milestones and members can be created. The entity User has personal attributes like the profile picture name, email, password and the two permissions to create and change users or to create products. Finally, like any other entity, the user has a deleted flag. If a delete method is executed, this is set to true. The find and get methods of the API are implemented to search only for objects that are not deleted. When deleting, care must be taken which dependencies must also be set to deleted. If a user is deleted, no further data is removed from him. The user is only marked as deleted on the user interface. All data generated by the user will remain. All other entities are linked to the product entity. The product itself does not provide much information, because it will be filled with information later by other references. If a product is deleted, all versions, issues, comments, milestones, and members must also be set to deleted. The versions contain important information and the 3D CAD model. To create a versioning, the attributes major, minor and patch are specified for a version. the baseVersionIds are used to identify the previous version. By knowing the current version and previous version, a graphical overview can be built on the UI. Multiple issues can be added for each project. Label and text contain information about the issue. The state distinguishes between open and closed issues. So it is possible to filter for completed and not completed issues. The attribute assigneeId is a string array. Here the userIds of the users are specified who have the task to solve the issue. The milestoneId can be used to add the issue to a milestone. For each issue one or more comments can be created. These comments have a text, an author and a corresponding issue as well as the time of the post. With the attribute action an issue can be closed or reopened by the comment. For each product one or more milestones can be defined. These have a label and clear start and end dates. UserId and productId serve as foreign keys. Later it can be counted how many issues per milestone are still open or already closed. This can be used to display a graphical representation of the project progress in the frontend. For each product one or more members can be added. These members have the foreign keys productId and userId. The attribute role can have three states: manager, engineer, customer. Depending on the role, the corresponding product member has different permissions to interact with the respective product.

Function model

The functions implemented in the API allows to create, read, update and delete data. This would not even require a graphical user interface. All functions offered by the API are called via HTTP requests. These requests can also be made via the

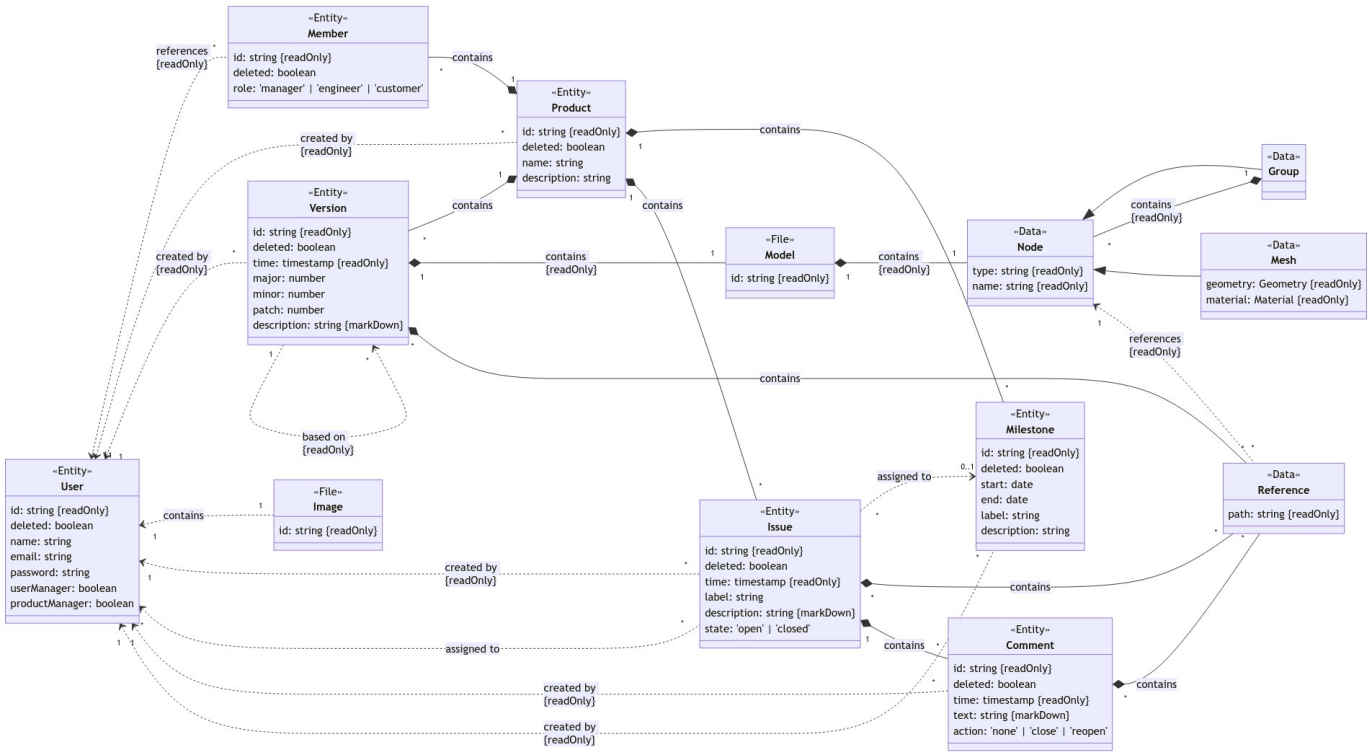

Fig. 2. Data model

console or other software such as Postman or Swagger. Thus, the functional model in combination with the data model is the foundation of this application. The API is built with nest.js. One rest entity contains of controller, service and module. In operation, an HTTP request is performed in the frontend. The data flows from the respective view via a request manager to a rest method, where the respective Axios request is triggered. In the backend, the nest controller receives the request and sends it on to the nest service. The nest service implements the necessary API methods to make changes to the data. These functions are all asynchronous and have a Promise as return value. After processing, the result is returned. This tunnel is provided with various permission checks to verify that the user or product member has the permission to perform this action. The interface in the common package controls the communication between frontend and backend. In the nest service the methods find, add, get, update, delete and convert are available. The convert method parses the received database object back into an object that can be further processed by the software.

User management: Users can be created, modified and deleted here. The condition for this is that the respective user has the User Manager Permission. When a user is created, the associated data object is filled with the respective attributes. Permissions can be assigned to allow the new user to create and edit other users and to create products. For this the two attributes user management permission and product management permission have been added. The API offers the following methods for interaction with user objects:

- findUsers(query?: string, productId?: string)
- addUser(data: UserAddData, file?: File)
- getUser(id: string)
- updateUser(id: string, data: UserUpdateData, file?: File)
- deleteUser(id: string)

The `findUsers` method searches for all users of the system from the database. It can be searched either via a query by user name or via the product id. The parameters query and productId are optional. Depending on which parameter is specified, the search process runs differently. The users found in the database are converted into user objects with the convert method, pushed into an array and returned. For the addUser method the transfer parameter is needed, which contains the information about the new user. The second parameter holds the profile picture and is optional. The UserAddData object is generated in normal operation via the User settings view and sent to the backend. GetUser searches for a specific user based on the userId and returns the object after conversion. UpdateUser needs the userId, the UserUpdateData object and optionally a file containing a profile picture. DeleteUser sets the deleted flag of the particular user to true. The get and find methods are implemented that way, to search for objects that are not deleted. The convert method is in each class and parses the database entities back into objects for further processing.

Product management: This part of the API provides methods to make modifications to product objects. Here products can be created, modified and deleted. The following bullet points show the methods that are available for interacting with products:

- findProducts()
- addProduct(data: ProductAddData)
- getProduct(id: string)
- updateProduct(id: string, data: ProductUpdateData)
- deleteProduct(id: string)
- convert(user: ProductEntity)

The findProducts method searches in the database for all products that are not deleted. For this no passing parameter is necessary. The other methods work according to the same principle as in user management. It is searched with the corresponding productId for the object and displayed or its properties get changed. When deleting a product, it must be ensured that all objects that are attached to a product are also deleted. So if a product is deleted, all versions, issues, milestones and members must also be set to deleted.

Version management: Only by referencing a version to a product it gets further descriptive properties and the 3D CAD model. Again, the API provides CRUD methods to interact with versions:

- findVersions(productId: string)
- addVersion(data: VersionAddData, file: File)
- getVersion(id: string)
- updateVersion(id: string, data: VersionUpdateData, file?: File)
- deleteVersion(id: string)
- convert(user: VersionEntity)

Unlike user management, a file must be specified when a version is created. This is an STL file, which is then stored locally. This CAD model can be exchanged when updating the version.

Issue management: These methods are available to create, read, update and delete issues:

- findIssues(productId: string, milestoneId?: string, state?: 'open' or 'closed')
- addIssue(data: IssueAddData)
- getIssue(id: string)
- updateIssue(id: string, data: IssueUpdateData)
- deleteIssue(id: string)
- convert(user: IssueEntity)

The findIssues method can search for issues in the database in several ways. An issue is always connected to a product. Thus, the productId must always be passed. Aditionally, the milestoneId parameters can be given to the method to search for issues that belongs to a specific milestone. The state parameter allows to filter the issues by closed and open. This is especially useful to be able to filter for completed or uncompleted tasks for the purpose of product management. When deleting issues, all comments belonging to the respective issue must also be deleted.

Comment management: The following list shows the methods that can be performed to interact with the comment objects:

- findComments(issueId: string)
- addComment(data: CommentAddData)
- getComment(id: string)
- updateComment(id: string, data: CommentUpdateData)
- deleteComment(id: string)
- convert(user: CommentEntity)

An issue can have one or more comments. These comments are bound to an issue and the issue is bound to a product. The findComments method stores all comments for a given issueId in an array and returns it.

Milestone management: Milestones are a container that can be filled with issues by reference. The CRUD methods are listed below:

- findMilestones(productId: string)
- addMilestone(data: MilestoneAddData)
- getMilestone(id: string)
- updateMilestone(id: string, data: MilestoneUpdateData)
- deleteMilestone(id: string)
- convert(user: MilestoneEntity)

The methods have analogous purposes as already with the other data objects. When deleting a milestone, it must be ensured that the references to the linked issues are removed. For this purpose, the milestoneId attribute of each associated issue is set to null again.

Member management: The API provides the following methods for member management:

- findMembers(productId: string, userId?: string)
- addMember(data: MemberAddData)
- getMember(id: string)
- updateMember(id: string, data: MemberUpdateData)
- deleteMember(id: string)
- convert(user: MemberEntity)

Members are always linked to the respective product. Additionally, the userId can be used to search for one specific member. The findMembers method returns an array of members that can then be displayed on the user interface. The other methods are similar to those of the other entities

Permission model

The permission model offers a number of possible restrictions on the platform so that not every user has all freedoms. This is especially important when cooperating with customers. The customer should only have the possibility to evaluate existing products. For this he can see the products he is registered for and use the given product management functions. Creating new users and products should only be possible by the developers of the respective product. They also organize the rights of each user. The permission model is divided into two levels. The first level is the user level and the second level is created by the product members. This system is deeply integrated in the backend. Each method of the API checks with each call whether the respective user or the member role has the necessary permissions for the execution. All permission of the respective user and member roles are stored in a separate permissions.ts file.

User level: The User entity has the two attributes user management permission and product management permission. Only a user who has the user management permission can see the Users button in the upper right corner of the frontend and create or edit users there. With the product management permission it is possible to create new products. If a new product is created, the respective user is automatically also

a product member and receives the member role manager. The permissions of each user can be adjusted afterwards in the user settings [see Fig. 8 on page 7]. So it is possible to give multiple users the permissions for user administration and product administration. For example, a user and product manager can exist for each department The following table [see Fig. 3 on page 6] shows the division of permissions into user management and product management on user level.

	user manager	product manager
find user	y	n
get user	y	n
read user	y	n
update user	y	n
delete user	y	n
create product	n	y

Fig. 3. Permissions on user level

Member level: For permission management at member level, three roles are provided. These roles are: manager, engineer and customer. The manager is the one who created the product and has the permission to add more members. So he can add more managers, who in turn have all the rights over the management of the product. This is useful when the product development covers several departments. The second role is the engineer. He is involved in the product development process. He has no rights to change the product or its members. However, he can freely create and edit versions, issues, comments and milestones. The last role is the customer who has the possibility to observe the product development process. He can follow the progress of the project, but has no permission to change anything on the platform. In the current version of the software the rights of the customer are still very strict. The permission system is implemented in such a way that it can be changed with few adjustments. If the customer needs writing permissions, this can be easily changed in the code. For this purpose, there is a separate file in the backend, which implements exactly these two tables for the permission management. The corresponding table [see Fig. 4 on page 6] shows an overview of the respective permissions on member level.

	manager	engineer	customer
read product	y	y	y
update product	y	n	n
delete product	y	n	n
find version	y	y	y
create version	y	y	n
read version	y	y	y
update version	y	y	n
delete version	y	y	n
find issue	y	y	y
create issue	y	y	y
update issue	y	y	n
delete issue	y	y	n
find comment	y	y	y
create comment	y	y	n
read comment	y	y	y
find milestone	y	y	y
create milestone	y	y	n
read milestone	y	y	y
update milestone	y	y	n
delete milestone	y	y	n
find member	y	y	y
create member	y	n	n
read member	y	y	y
update member	y	n	n
delete member	y	n	n

Fig. 4. Permissions on member level

Interface model

The user interface provides a convenient way to interact with the functional model to modify and display data. For the frontend the library React.js version 17 was used. The data is managed in react states and manipulated with react hooks. Typescript was used for programming. The pages were designed with simple CSS without additional tools. The user interface offers a consistent design, which runs through the entire system. The header of the user interface offers three buttons. A click on ProductBoard leads back to the start page. On the right side there is the user administration and the currently logged-in user. The button to view the users is only visible if the current user has the user management permission. The rest of the page below the header adapts to the corresponding content.

Start page: After a successful login with username and password you will see the start page. This page lists all available products in a table [see Fig. 5 on page 7]. For each product in the table a preview is shown. The other columns show the attributes Owner, Name, Description, Versions, Issues and Members. The X on the right provides the possibility to delete the corresponding product. The owner is the person who created the product. Name and Description are defined when the product is created and can be changed later in the Product Settings [see Fig. 20 on page 9]. The columns on the right show how many versions exist for this product, how many issues have been created and how many members have access to the product. By clicking on New product you get to a separate page where you can add a new product [see Fig. 6 on page 7]. This button is only visible when the corresponding user has product management permission. After entering name and description the new product appears in the product list on the start page. Only by creating a new version for a product a CAD model with further information is added.

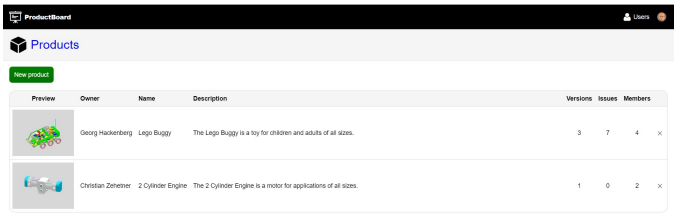

Fig. 5. Startpage

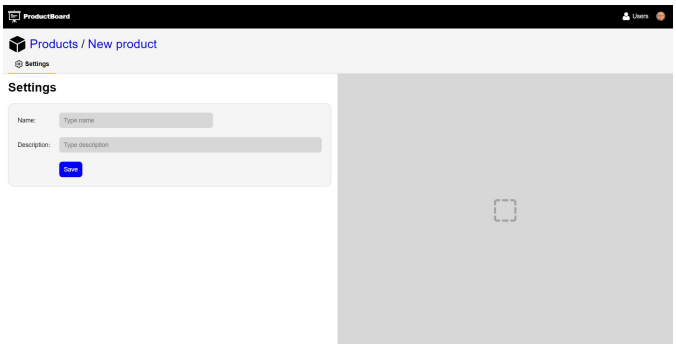

Fig. 6. Add new product

User management: This view is only accessible if the active user also has the authorization. By clicking on Users this page is called. If the user does not have user management permission, the button is not visible. On this page all users with profile picture, name, email and permissions are displayed [see Fig. 7 on page 7]. You can delete a user by clicking on the X on the right side. If a user is clicked on, he can be edited via the User settings. The button New user also leads to the user settings where you can provide information and create a new user by clicking on the save button [see Fig. 8 on page 7]. Save leads back to the user overview and shows the new or changed user in the table.

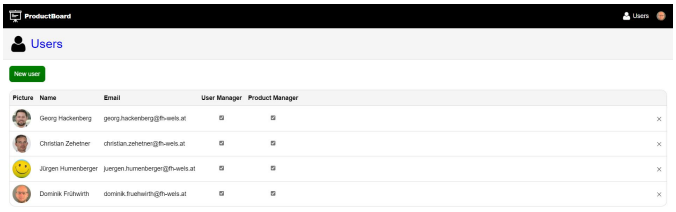

Fig. 7. User management

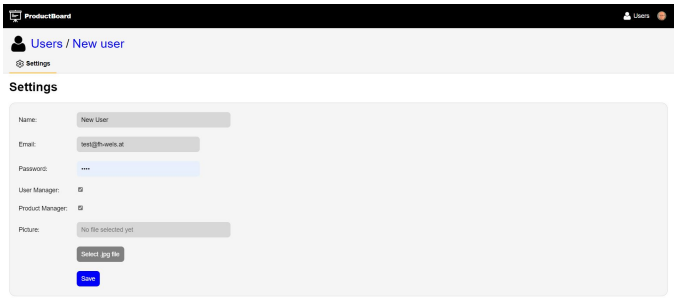

Fig. 8. User settings

Version view: Clicking on a product takes you to the version view [see Fig. 9 on page 7]. You can also use the toolbar to jump to other pages such as Issues, Milestones Members or Settings. Next to the links a number in brackets shows how many objects per category have already been created. The left side of the Version view shows the created versions. On the left side there is a tree structure similar to Git. In the middle is the corresponding version number with the owner of the version inclusive email and a short description. Each version offers a preview. By clicking on the respective version, the 3D view on the right side also changes and shows the selected model. The 3D view was built with three.js. It allows to rotate, move and zoom the model. With a click on New version you get to the version settings [see Fig. 10 on page 7]. There you can enter information for a new version and select an STL file. Depending on the selected base version, the version view shows the new version with the corresponding new tree structure after pressing the Save button.

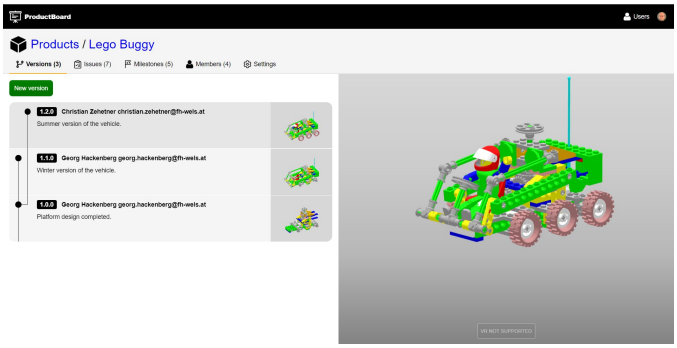

Fig. 9. Version view

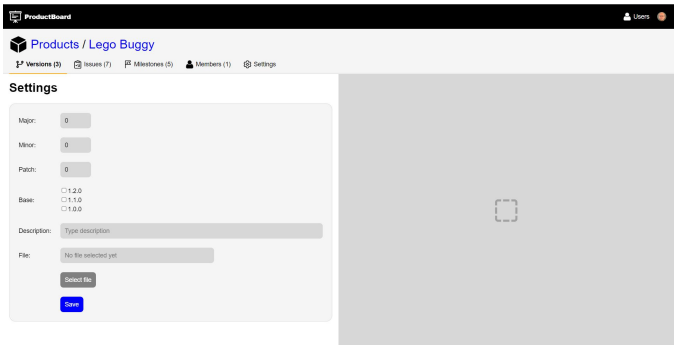

Fig. 10. Versionsettings view

Issue view: By clicking on the Issues link, you access the Issue view [see Fig. 11 on page 8]. Here the created issues are displayed in a table. The two buttons Open Issues and Closed Issues can be used to filter the list accordingly. The table shows the reporter who created the issue, the associated label, the assignees and how many comments and marked parts are in the conversation channel. The Issue settings view allows to create new issues for the product [see Fig. 12 on page 8]. The label, the text, the milestone and the assignees can be defined. An existing milestone can be selected with the dropdown menu. An issue must not be assigned to a milestone. This choice

lies by the user. The Save button closes the settings, and you return to the Issues view where the new issue is visible. All views with 3D View offer the possibility to select a desired version for viewing. In the version view the version can be clicked directly. In the other views the version can be selected via a dropdown menu. This menu is located in the upper left corner of the 3D View.

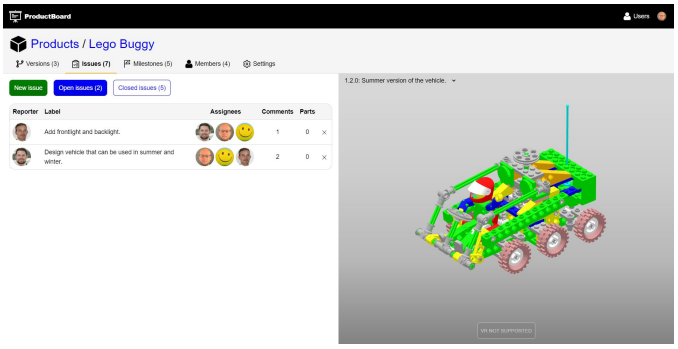

Fig. 11. Issue view

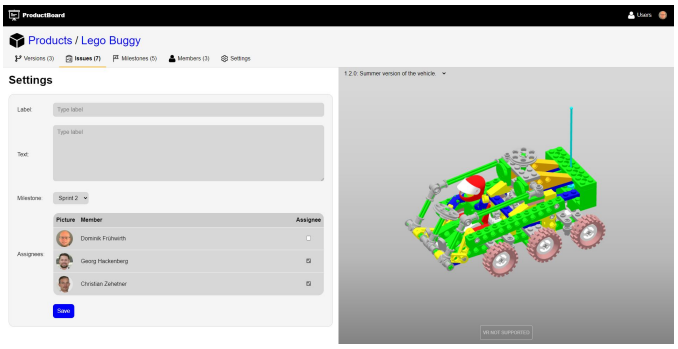

Fig. 12. Issuesettings view

Comment view: Clicking on an issue in the Issue view opens the corresponding Comment view [see Fig. 13 on page 8]. Here you have the possibility to discuss the issue. You can click on a component of the 3D model in the Comment view to reference a component [see Fig. 14 on page 8]. If a post is created or a part is selected in the comment view, the corresponding counter in the issue view table is increased. A comment can be used to close an issue by clicking Close. This issue will then be found in Closed Issues in the Issue view. With the comment function it is also possible to reopen the issue in the same way. The Close button displays the text Reopen when an issue is closed. In the upper right corner of the Comment view there is a button to edit the selected issue. For example, the issue can be assigned to another Milestone or other attributes can be changed [see Fig. 12 on page 8]. A click on Save makes the changes visible in the Issue view.

Milestone view: The Milestone view can be accessed via the Milestones link [see Fig. 15 on page 8]. A table shows who created the milestone, its name, start date, end date and the progress like open and closed issues. For each milestone two progress bars are displayed. The first one shows the date progress and the second one the issue progress. If a milestone

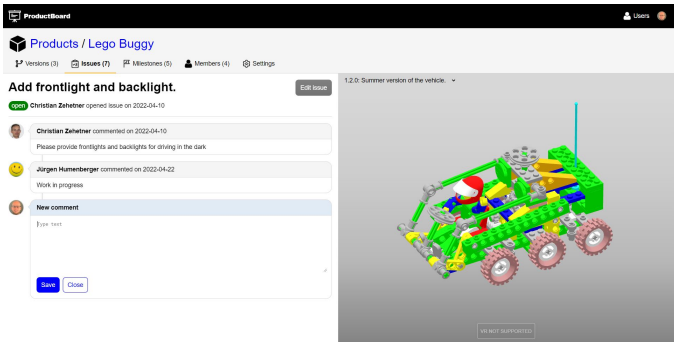

Fig. 13. Comment view

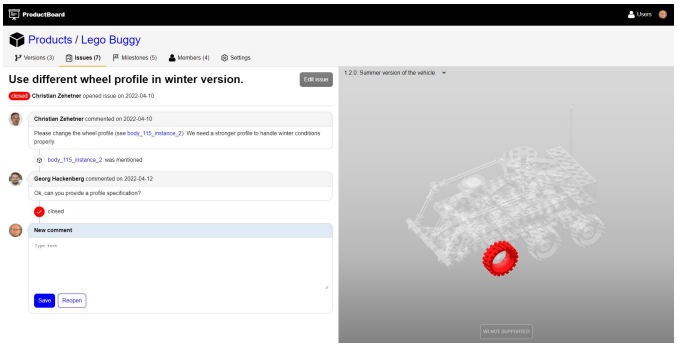

Fig. 14. Selected part

is selected, a table with the attached issues is displayed [see Fig. 16 on page 9]. This table is identical to the one in the Issue view. Here you can also filter by open and closed issues. On the right side a burn down chart is displayed which shows the current progress of the milestone. The chart shows the start date, the end date, the number of issues and the progress until the current day. A click on New Milestone or Edit Milestone leads to the Milestone Settings [see Fig. 17 on page 9]. Here the attributes of a milestone can be adjusted and saved. The new or edited Milestone than show up in the Milestone view.

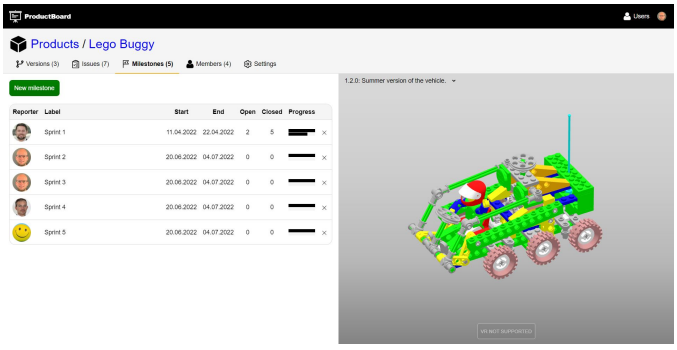

Fig. 15. Milestone view

Member view: To distribute the rights for a product, members are added to an existing product via the user interface. In the member view, a table shows all members who have access to the selected product [see Fig. 18 on page 9]. The table shows the user picture and the name of the user. The column Role defines which rights the respective member has. At the

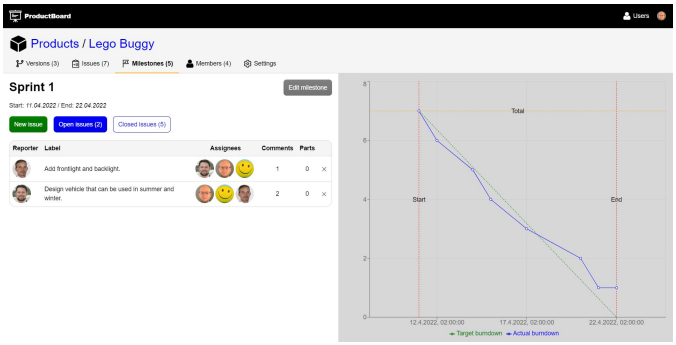

Fig. 16. Sprint view

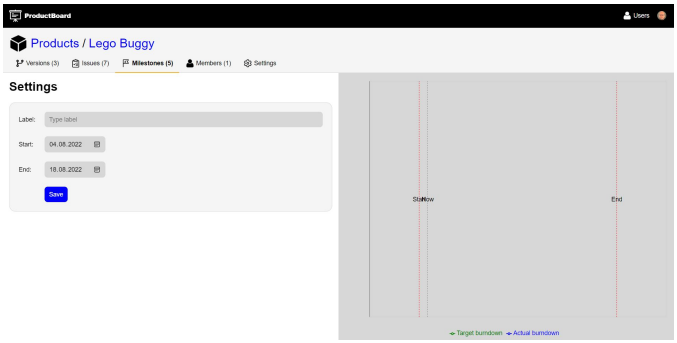

Fig. 17. Milestone Settings view

moment there are three roles: manager, engineer, customer. As with every overview table, objects can be deleted from the list by clicking on the X button. The button New Member leads to the Member settings. Here you can enter the name of a potential member into a text field. When typing, a list appears in the lower area that filters for each new letter. If you click on a user from the list, this user is selected and a member role can be assigned to him [see Fig. 19 on page 9]. If you click on X, the user disappears and the text field appears again. However, if Save is clicked, the current member is saved as specified and displayed in the member view.

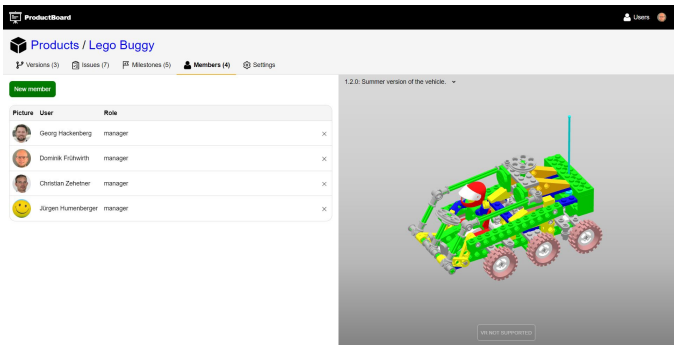

Fig. 18. Member view

Productsettings view: In this view the attributes Name and Description of the selected product can be changed [see Fig. 20 on page 9]. After clicking the Save button the changes are visible in the product overview

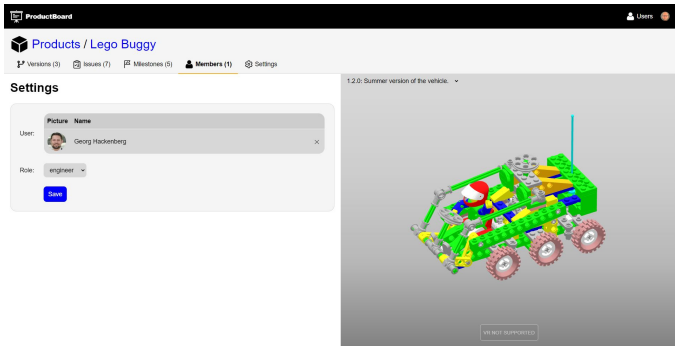

Fig. 19. Membersettings view

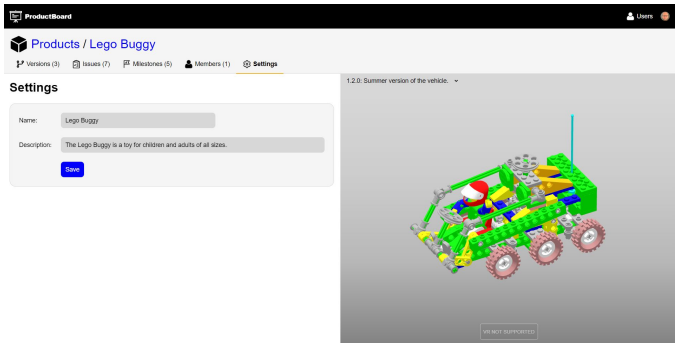

Fig. 20. Productsettings view

IV. Critical Evaluation

To solve the issues defined at the beginning of the article, we have developed the software ProductBoard. This software combines agile project management with product development. It was developed as a team of two people and the development time is currently one year. Due to the short development time, the software has of course not yet reached its final state. Nevertheless, it already covers a wide range of required issues. The development was done with state-of-the-art technology. The clean software architecture makes it easy to extend and maintain the software. As a single user, the software could already be tested at this stage. The user interface is responsive, and the software is intuitive to use. At this point, the app is still designed for a single user. There are still no functionalities built in that several PCs can access the data system simultaneously. This requires a revision in the backend to be able to react live to changes and display them again in the frontend. Another issue is that the data is consistent across all clients. One solution is a server client architecture with a web socket. For an adaptation to the customer, the permission system must also be adapted individually. Furthermore, the software has to be tested for stability. The current state of the project shows that it is possible and realistic to generate a benefit for agile product management and product development. In the next steps, the app can be made ready to enable communication between multiple clients. For this purpose, the current state can be continued and further developed without a complete revision.

V. Conclusion

Summary

To create a common basis for product development and product management, the software ProductBoard was developed. The advantage of this combination is that no engineering artifacts are lost. By bundling on a single app, a better overview of the process and more transparency is created. The customer can participate in agile product management and at the same time has the current status of all versions of the products. For this purpose, users can create and manage products on the platform. These products have versions, issues, comments and milestones. Members can also be added to the products, which have different permissions. There are three roles: manger, engineer and customer. The software offers versioning similar to GitHub. Based on the previous version and the current version, the history can be displayed graphically. Via issues, it is possible to assign tasks to one or more assignees. These tasks can be discussed in the associated conversation channel. After completion of a task, it can be marked as closed. Milestones are used to create an overview of the issues. Through the milestones the progress of the project can be tracked well. For the development of the software we decided to use tools from web development. This enables a cross-operating system user experience and can be well-built on a client server architecture. The three main components are the backend, the frontend and the database. In the center is the data model and the API for read and write operations to the data model. The frontend serves as a generic layer for triggering operations and as a graphical representation of data. To enable operation on company and customer level a permission model was implemented. This can be customized according to the customer's needs by adjusting the customer's permissions individually. The functionality from the user's point of view was tested each time a new functionality was added. The tests have shown us that it is already possible to manage products on a PC using the software. In order to be ready for a wider range of users, the app still needs to be adapted. Due to the clear separation of frontend and backend, it is possible to extend the app in this direction without major reconstruction measures.

Outlook

According to our tests, the app has the potential to generate real added value for customers and companies. In order for the app to run on multiple devices at the same time and access a central backend, adjustments still need to be made. Furthermore, it has to be ensured that the software runs stable independent of all user inputs. This will be checked in a test phase. The architecture of the software was designed in such a way that other files can be uploaded in the future instead of CAD data. We plan to expand the scope of the app in the future. One idea is for example the management of electrical schematics. Like 3D CAD files, these can be designed and evaluated according to customer requirements. The software can also evolve in the direction of a simulation platform. It would be possible to upload entire simulated processes and support them with project management tools. For example, mechanical or electrical systems could be simulated directly in the View. For this, only the 3D view would have to be replaced by the corresponding functionality. Furthermore, we see potential in VR applications. With the help of VR, communication between companies and customers could be greatly improved during the project phase. The customer would have the possibility to view the 3D CAD model with all its versions in virtual space. This would allow the customer to move the model freely in space and thus capture even more engineering artifacts. These functions are already partially implemented in the software and will be extended in the future. A follow-up project is already in the planning phase and will be launched in the next few months. The follow-up project will complement this software with VR functions to enable an evaluation of the product development process in virtual space.

References

[1] Georg Hackenberg, Christoph Richter, and Michael F. Zäh. A multidisciplinary modeling technique for requirements management in mechatronic systems engineering. *Procedia Technology*, 15:5–16, 2014. 2nd International Conference on System-Integrated Intelligence: Challenges for Product and Production Engineering.

[2] Christoph Legat, Jakob Mund, Alarico Campetelli, Georg Hackenberg, Jens Folmer, Daniel Schütz, Manfred Broy, and Birgit Vogel-Heuser. Interface behavior modeling for automatic verification of industrial automation systems' functional conformance. *at - Automatisierungstechnik*, 62(11):815–825, 2014.

[3] Sabine Teufl and Georg Hackenberg. Efficient impact analysis of changes in the requirements of manufacturing automation systems. *IFAC-PapersOnLine*, 48(3):1482–1489, 2015. 15th IFAC Symposium onInformation Control Problems inManufacturing.