Titel

Autor

***Abstract*—hallo [1] [2] [3]**

I. Introduction

Context

Over time, customers demand more and more specific products at small batch sizes. Engineer to order is a part of the production chain in which a product is produced and customized according to exact specifications. Such customization is becoming more and more important to increase customer satisfaction and thus also to increase the value of the company.

An important part of this individual production consists of regular meetings. Through constant exchange, customer requirements can be discussed and specified. Through agile product management is it possible to adjust the product in the developing process if necessary.

Digitization enables such agile product management, as this would be laborious and sometimes impossible without databases and digital spreadsheets.

Problem

Requirements management is complicated by the fact that CAD data and documentation are kept separate. By keeping the documentation in Excel lists or documents, some engineering artifacts are lost.

Since the customer does not have access to the requirements or the Excel lists, agreements, task planning and scheduling are difficult.

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.

Solution

To address these issues, we developed a product development platform modeled after GitHub. CAD models can be uploaded to our platform. The platform offers a user management and the possibility to add members with different rights to a CAD model. Authorized members can upload different versions of the CAD model, just like on GitHub. Milestones can be created for each CAD model. Issues can be added for each milestone, on the basis of which changes can be made to the model. For each issue there is a discussion thread that can be opened and closed. Once all issues have been resolved, the respective milestone is considered completed. 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

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.

Requirement specifications

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.

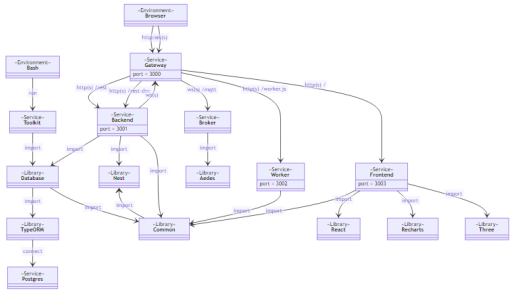

Fig. 1. Packages

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. 8 on page 4]. 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. 15 on page 5].

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.

Worker: This package is not used yet. It is used to parallelize processes for later extensions of the software.

Broker: This package is not used yet. It is used to send routing messages between different services for later extensions of the software.

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.

Classes: The following figure [see Fig. 2 on page 3] shows the data model of the developed API in the backend. In order to clearly define which data of a class is changed, added and constant, each class is divided into three subclasses that inherit from each other. Thus, the classes are divided into UpdateData, AddData and the entity itself. AddData inherits from UpdateData and the Entity inherits again from AddData. The respective Entity subclasses serve as passing parameters for the API methods. Each entity has an ID as primary key and a deleted attribute. The find methods of the API are defined

in such a way that they return only objects, where the deleted attribute is false. So if data is deleted, it is only marked with a deleted flag and will not be found again. If data is created, then the necessary references are generated, such as the associated product or the selected user. These attributes are comparable with foreign keys and are located in the AddData classes. The UpdateData classes contain attributes of the respective entity, which can be changed by the API in the later course.

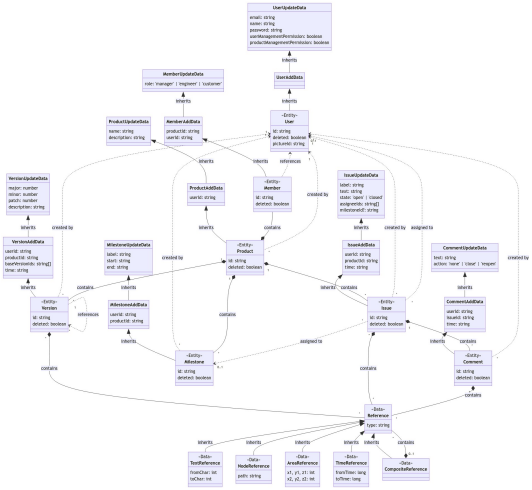

Fig. 2. Class diagram

Entities: The entity-relationship model [see Fig. 3 on page 3] has a simpler structure like the class diagram. It consists of the exact entities that are used in the interface and API. Since this is only the data store, the classes are not structured in three subclasses. One table is reserved in the database for the entire object. Since the objects of the API and the database are not identical, these must be converted with each write and read process. The database entities are mapped to tables with TypeORM.

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 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. 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

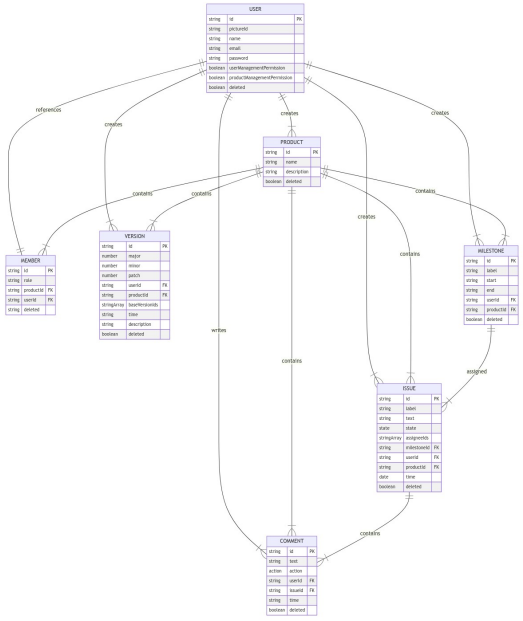

Fig. 3. Entity relationship diagram

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 returned from the backend to the frontend.

User management:
Product management:
Version management:
Issue management:
Milestone management:
Member management:
Product settings:

Permission model

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 for the user management 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. 4 on page 4]. 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. 19 on page 6]. 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. 5 on page 4]. 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.

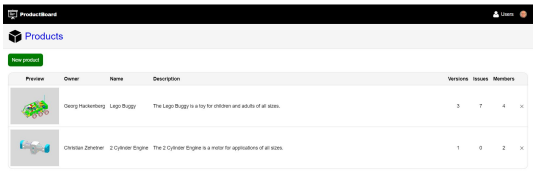

Fig. 4. Startpage

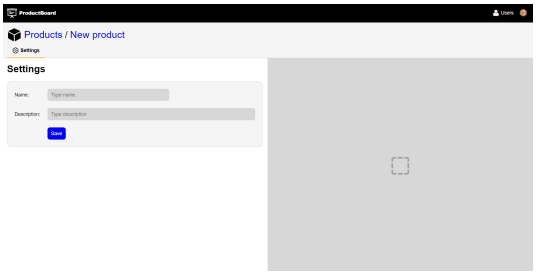

Fig. 5. Add new product

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. 6 on page 4]. 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. 7 on page 4]. Save leads back to the user overview and shows the new or changed user in the table.

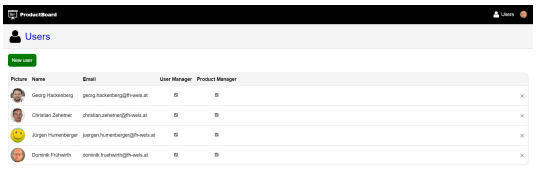

Fig. 6. User management

Version view: Clicking on a product takes you to the version view [see Fig. 8 on page 4]. You can also use the toolbar to jump to other pages such as Issues, Milestones Members or

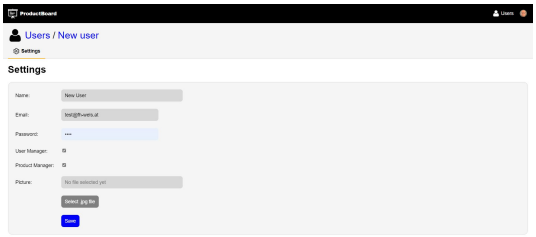

Fig. 7. User settings

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. 9 on page 4]. 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.

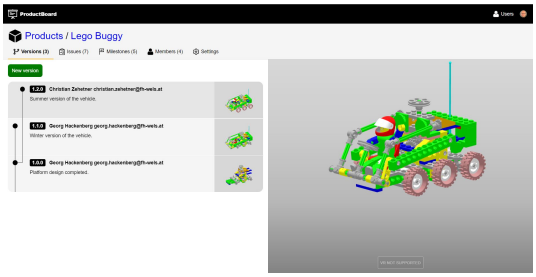

Fig. 8. Version view

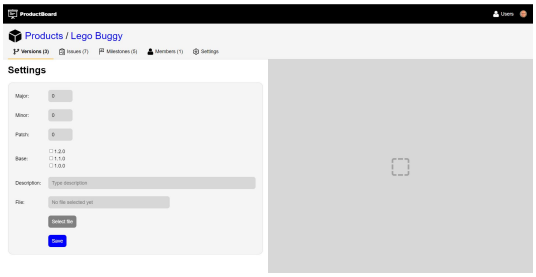

Fig. 9. Versionsettings view

Issue view: By clicking on the Issues link, you access the Issue view [see Fig. 10 on page 5]. 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. 11 on page 5]. 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.

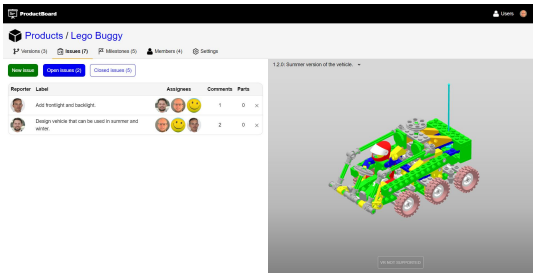

Fig. 10. Issue view

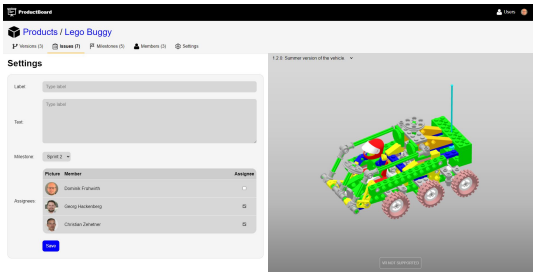

Fig. 11. Issuesettings view

Comment view: Clicking on an issue in the Issue view opens the corresponding Comment view [see Fig. 12 on page 5]. 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. 13 on page 5]. 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. 11 on page 5]. 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. 14 on page 5]. 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 is selected, a table with the attached issues is displayed [see Fig. 15 on page 5]. This table is identical to the one in the

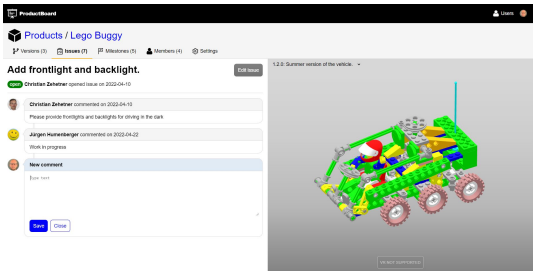

Fig. 12. Comment view

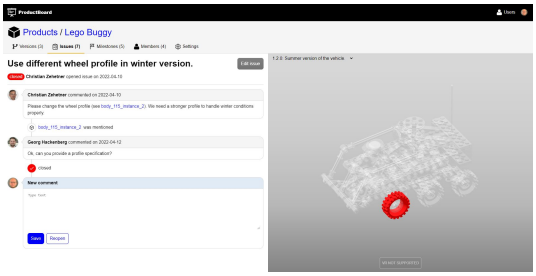

Fig. 13. Selected part

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. 16 on page 6]. Here the attributes of a milestone can be adjusted and saved. The new or edited Milestone than show up in the Milestone view.

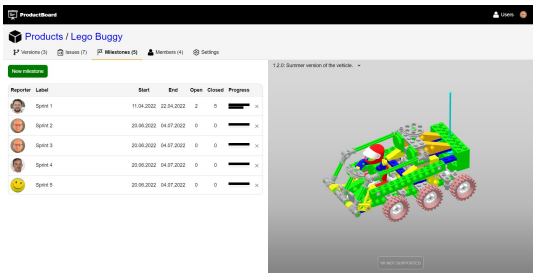

Fig. 14. Milestone view

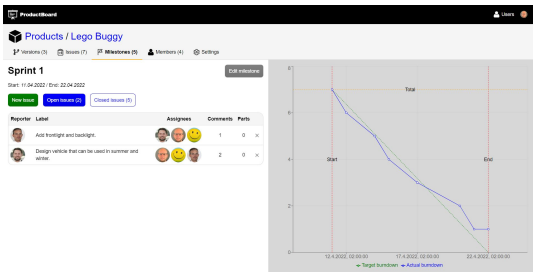

Fig. 15. Sprint view

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

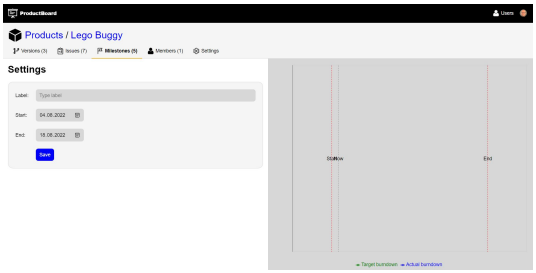

Fig. 16. Milestone Settings view

to the selected product [see Fig. 17 on page 6]. The table shows the user picture and the name of the user. The column Role defines which rights the respective member has. At the 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. 18 on page 6]. If you click on X, the user disapears and the text field appears again. However, if Save is clicked, the current member is saved as specified and displayed in the member view.

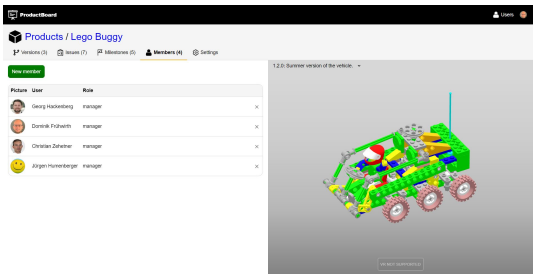

Fig. 17. Member view

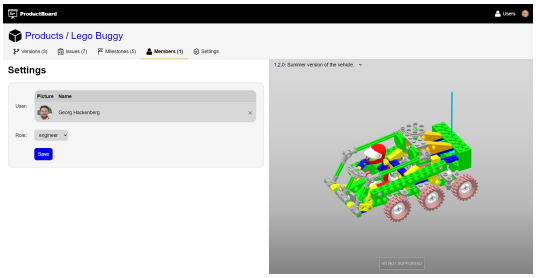

Fig. 18. Membersettings view

Productsettings view: In this view the attributes Name and Description of the selected product can be changed [see Fig. 19 on page 6]. After clicking the Save button the changes are visible in the product overview

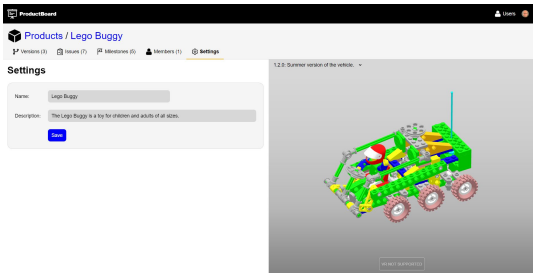

Fig. 19. Productsettings view

IV. Critical Evaluation

V. Conclusion

Summary

Outlook

References

[1] Georg Hackenberg, Christoph Richter, and Michael F. Zäh. A multi-disciplinary 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.