Technical Report

Information Visualization in Production Systems Engineering

Felix Rinker[1,2]
Laura Waltersdofer[1,2]
Manuel Schüller[2]
Dietmar Winkler[1,2]

[1] Christian Doppler Laboratory for Security and Quality Improvement in the Production System Life Cycle, Institute for Information and Systems Engineering, TU Wien, Vienna, Austria
<firstname>.<lastname>@tuwien.ac.at

[2]TU Wien, Information Systems Engineering, Institute for Information and Software Engineering, Vienna, Austria
<firstname>.<lastname>@tuwien.ac.at

Technical Report No. CDL-SQI 2019-15
Issued: June 2019

Citation: F. Rinker, L. Waltersdorfer, M. Schüller, D. Winkler „Information Visualization in Production Systems Engineering", Technical Report CDL-SQI 2019-15, TU Wien, Vienna, Austria, June 2019.

Information Visualisation in Production Systems Engineering

Felix Rinker[1,2], Laura Waltersdorfer[1,2], Manuel Schüller[1], and Dietmar Winkler[1,2]

[1] Institute of Information Systems Eng., Technische Universität Wien, Austria
`<firstname>.<lastname>@tuwien.ac.at`
[2] Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle (CDL-SQI), Technische Universität Wien, Austria
`<firstname>.<lastname>@tuwien.ac.at`

Abstract. Production Systems Engineering (PSE), the planning of production systems, is a highly complex tasks, involving domain experts from various domains, such as mechanical, electrical and software engineering. Data files, that describe entire plants, or single working cells can reach a large size (around 30 MB to several GBs and containing thousands of elements) with various relationships and dependencies between the different system components. Domain experts work in their discipline-specific tools, which do not take into account dependencies from other disciplines. In order to increase consistency and awareness of the influence on other disciplines, the visualisation of Multi-Disciplinary Engineering (MDE) data is a promising method. Therefore, we identified requirements and use cases for such a visualisation and designed and implemented a software prototype. We evaluate the effectiveness of our Information Visualisation (InfoVis) approach in comparison to a standard tool, the AutomationML Editor and report our results.

Keywords: production systems engineering · data visualisation · engineering data · process improvement · information visualisation · multi-disciplinary engineering.

1 Introduction

Production systems contain various different components, which are again made up of multiple other components, e.g. coil cars with motors and which is connected to a conveyor. In industrial settings the number of such components can rise up to thousands of single components, which are clustered in hierarchical structures depicting important dependencies and relationships in relation to neighbouring elements. Multiple engineers from different disciplines are involved in this highly complex engineering task in a so-called round-trip engineering process, meaning that changes from other disciplines get incorporated in the work of one workgroup. This collaborative engineering takes place because of market pressure and fastened release cycles. The simultaneous contributions and adaptions to the common model result into big data file sizes, up to 30 MB, which are hard to perceive and manage. An advantageous feature for this knowledge-intensive working process would be the visualisation of the engineering information. However, so far, useful approaches for this kind of specialised data are missing: AutomationML (AML) is a common data format, used in the engineering of production systems, based on Extensible Markup Language (XML). Therefore, simple XML editors can show the data in large text list, which is useful for the automated data exchange, but unfortunately very hard to read for domain experts, and almost impossible for non-experts for more than simple use cases.

Lüder *et al.* [6] identified the following major challenges: *Unclear requirements and benefits of data exchange for stakeholders* and *the complexity of integrating heterogeneous engineering tools and data*. Change and consistency management are so far also highly important topics but do not receive attention independently but solely within the scope of single workgroups because improvements are mainly done within the groups themselves, but not across organisational barriers. One possible reason for this behavior could also be the ignorance of effects on other disciplines and systems components outside of the scope of the own discipline.

Making the element and attribute changes and dependencies between elements visible to the stakeholders can increase the efficiency and the quality of the overall engineering data exchange process, which is of major importance for the quality of a production system [1]. Each discipline has an own views and on their data, thus an adaptive software system that fulfills the stakeholders specific requirements is essential to find application in real-world use-cases and to support the work process.

Information Visualisation (InfoVis) is a field of research, which is concerned with how humans perceive data and how to improve this process by creating guidelines and best practices [7]. Raw, unstructured large data sets are challenging to be analysed by humans if they are not presented in a well-designed way, especially graphical means support an efficient and effective data processing [9] by reducing the user's cognitive load, search effort and recognition of patterns and changes of data. Nevertheless, more data is generated and aggregated in almost every domain, the kind of data visualisation that is chosen for the information is essential.

Therefore, we want to apply InfoVis methods to Production Systems Engineering (PSE) to improve the representation of engineering data to increase the efficiency and effectiveness of the data integration process.

The following research questions (RQ) will be answered:

RQ1. What are main requirements and elements in production systems engineering?

InfoVis is a general research area to design data accordingly to the user's specific needs to improve the perception and support an informed decision-making person. However, the methods are not designed to support the production systems domain. In order to adapt these methods to the domain's specific needs, in Section 2 we will report related work information visualisation and the process of production systems engineering to pair up with the most optimal design choices for identified challenges field of PSE. Requirements and use cases for production systems engineering are derived and explained in Section 4.

RQ2. Can information visualisation methods make PSE engineering more efficient?

To address this research question, we derive requirements on a visualisation tool and real-world use cases in Section 3. Section 5 reports on an evaluation of the effectiveness and effort of the proposed information visualisation methods in a comparative case study between the AML Editor and our proposed software prototype based on the Keystroke-Level Method (KLM). Section 6 discusses the findings and limitations. Section 7 concludes and proposes future research work. From the research, we expect the following contributions for the process improvement community. The use cases and Engineering Data Exchange (EDEx) process give Information System Engineering (ISE) researchers insight into the needs for visualising engineering information PSE in order to improve process efficiency.

2 Related work

This section summarizes related work on data visualisation in production systems engineering, and InfoVis.

2.1 Data structures in Production Systems Engineering

Due to the hierarchical structure of plant topologies, a tree-like data structure is a common tool to visualise the key concepts: Each element, except the root has one parent element and may have one or multiple child elements. Computer-Aided Engineering eXchange (CAEX), a XML-based format for modeling plant data is one example of this representation form. There are multiple well-researched approaches to visualise trees: The most relevant ones being *Rooted Trees*, *Tree Maps*, *Radial Trees* and *Force-directed Tree* that will be discussed in this section.

Rooted Trees Hierarchical relations are easily distinguishable in this representation: The *Rooted Tree*, depicted in Fig. 1 (a), has a single root node which is in the top level position, from which the child nodes move downwards. Nodes on the same hierarchical level share the same vertical depth. However, for large data structures this is a sub-optimal representation from, because such a tree would require the user to zoom and search along the vertical space of the tree.

Radial Trees In comparison to the rooted tree, the representation of a *Radial Tree*, shown in Fig. 1 (b), has a central positioned root node form which all other nodes are grouped in circles around, which leads to a more efficient utilization of the available space.

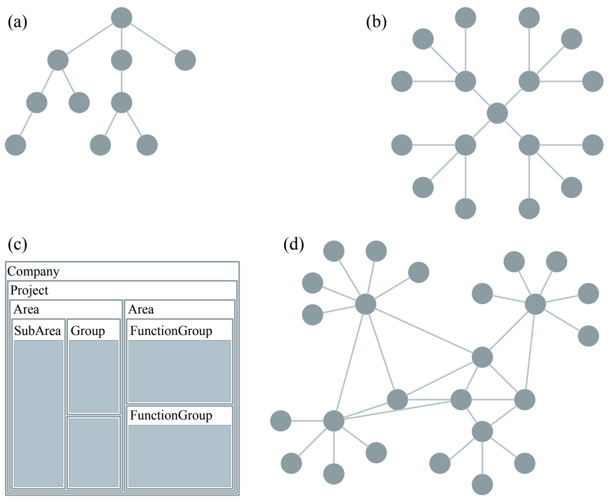


Fig. 1. Tree visualizations: (a) *Rooted Trees* (b) *Radial Trees* (c) *Tree-Maps* (d) *Force-directed Tree*

Tree Maps The *Tree Map*, depicted in in Fig. 1 (c), represents the data in rectangular sections, accordingly to the size of the data it represents. This is an advantage compared to node-link graphs and thus better suited to display larger data structures. However, cross-references cannot be visualized with this approach.

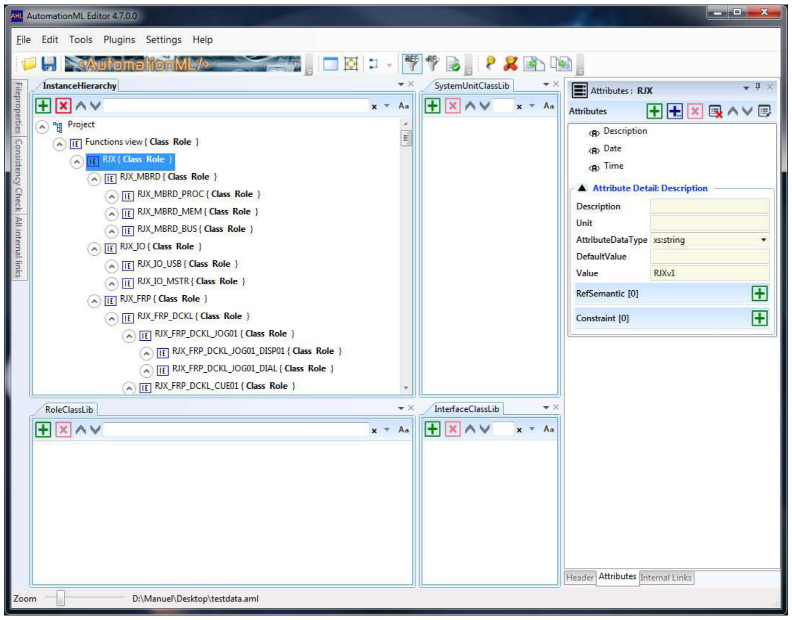
Force-directed Tree Force-directed graphs, shown in Fig. 1 (d), are usually created by algorithms simulating the position of nodes based on force simulation (force between nodes, attracting or repelling each other). This method oftentimes do not create graphs that are to humans aesthetically attractive, however the usage of available space is more ideal in this case.

2.2 AutomationML

AML is a data format, based on XML, especially designed for storing and exchanging plant engineering information and to reduce the complexity of heterogeneous data sources. AML files are described in hierarchical form, also called *Instance Hierarchy*, which is made up of *Internal Elements*. The standard extends other already established standards, such as CAEX for the hierarchy and COLLADA for geometric and kinematic data. Behavior and processes of logical data are modelled in PLCopenXML. Because AML is a XML-based file format, it can be displayed with any text editor or software capable of viewing XML. A more sophisticated form is the AutomationML Editor [3].

[3] https://www.automationml.org/o.red.c/dateien.html?cat=1

AutomationML Editor The AutomationML Editor is the recommended tool to handle AML files by the Automation e.V (association) [4]. Currently there are not many software alternatives available to manage AML files. In Fig. 2 we can see that the editor uses a intended tree structure to visualize the project hierarchy, a well-established method for representing such structures. Elements can be expanded to their sub-nodes, also the side panel helps to understand the project hierarchy very well. However, a major disadvantage of this editor is the poor utilization of the given screen space. The tree representation mostly expands downwards, which means that especially for larger structures the user has to scroll and search extensively to look for components. The search function also quite limited, it is case limited and to find an element the complete name need to be spelled. Searching for an attribute value is not possible at all until now.



Project
Functions view { Class Role }
RJX { Class Role }
RJX_MBRD { Class Role }
RJX_MBRD_PROC { Class Role }
RJX_MBRD_MEM { Class Role }
RJX_MBRD_BUS { Class Role }
RJX_IO { Class Role }
RJX_IO_USB { Class Role }
RJX_IO_MSTR { Class Role }
RJX_FRP { Class Role }
RJX_FRP_DCKL { Class Role }
RJX_FRP_DCKL_JOG01 { Class Role }
RJX_FRP_DCKL_JOG01_DISP01 { Class Role }
RJX_FRP_DCKL_JOG01_DIAL { Class Role }
RJX_FRP_DCKL_CUE01 { Class Role }

Description
Date
Time

Attribute Detail: Description

Description	
Unit	
AttributeDataType	xs:string
DefaultValue	
Value	RJXv1

RefSemantic [0]
Constraint [0]

D:\Manuel\Desktop\testdata.aml

Fig. 2. AutomationML Editor

2.3 Data visualisation in PSE

[3] and [4] analysed differences between XML standards and mining knowledge between plant objects. Vathoopan *et al.* describe how mechatronic models can be visualised to enable model-based automation engineering [10] and report their

[4] https://www.automationml.org

experiences from their prototype. Romero *et al.* report findings of their prototype to visualise plant dependencies [8].

3 Design and Requirements Engineering for the Visualisation of Engineering Data

This section introduces requirements, use cases, and main elements for the visualisation of PSE data.

3.1 Requirements for an Engineering Data Visualisation

Following the *design science cycle* in [11], we set up an initial problem investigation in workshops, outlining the context and problem space of research, and deriving the following requirements addressing the challenges introduced in Section 1.

Requirement 1. Project Hierarchy. This requirement describes the hierarchical topology of engineering data, the different concepts are usually ordered in a hierarchy, the parent node being the plant.

Requirement 2. Cross-References. This requirement describes relationships and dependencies between different concepts and attributes. To prevent consistency errors, the connections between different disciplines should be made visible.

Requirement 3. Data structure size. This requirements describes the fact that engineering data structures are usually quite big, with a multitude of elements. Engineers need to have a concise overview over this structure in an efficient manner.

Requirement 4. Discipline-neutral view. All disciplines should benefit from the visualisation of engineering data. To achieve this requirement the data should be represented in a discipline-neutral way, if needed.

3.2 Use Cases for Evaluation

The use cases were derived from the basic functions that the AutomationML Editor provides (import an AML file, add a component) plus basic functions it does not provide (search for attribute values).

UC-1 Import & Export of an AML file To support realistic round-trip engineering, the import (UC-1.1) and export (UC-1.2) of AML files is essential as the basis for data management. A user's first step is usually the import of a AML file in order to view or edit the imported data. Exporting an AML file is necessary, to share the data with other stakeholders or to import the data into another tool for more advanced data manipulation functions.

UC-2 Navigate in Project Data. Engineers need to navigate through project data in order to efficiently process engineering data: This can mean *To show only the components that are relevant to one specific engineering discipline* (UC-2.1), *To get an overview of the specific discipline* or *To show in which other views the component is related to* (UC-2.2), to get an overview over dependencies to other disciplines.

UC-3 Search in Project Data. This use case concerns the search for specific values in the project data: Engineers need to be able *to search for a name of a component* (UC-3.1) , and *to search for a attribute of a component* (UC-3.2).

UC-4 Modify the Project Data. The user also needs to be able to interact with the data: *To add a component* (UC-4.1), *To edit a component* (UC-4.2), *To move component in hierarchy* (UC-4.3), *To remove a component* (UC-4.4).

4 Solution Approach

In this section we will discuss our solution approach based on the required capabilities and use cases in Section 3.

4.1 Visualisation of Project hierarchy

We decided to use a node-ink graph as main visualisation method for the project structure. Main benefit of this form is that users intuitively are aware of the hierarchical structure of components without being trained for this visualisation technique.

Furthermore, the *Radial Tree* visualisation was chosen since the usage of available space is much more efficient in comparison to other techniques. The project hierarchy is visualised by differently colored tree branches show in Fig. 3. The maximum of simultaneously displayed was chosen to be two levels because of human's working memory limits. By default component names are hidden to reduce unnecessary information, by hovering over the component the name of the component is displayed.

4.2 Visualisation of Cross-references

Tree visualisations are not the best form to visualise non-hierarchical relationships and dependencies, therefore the chosen *Radial Tree* approach needs adaptations to accommodate to the requirements described in section 3. One example for a cross-reference is the component of an electrical motor that is described in multiple disciplines: Electrical engineers specify the required power supply while mechanical engineers specify the dimensions of the motors and other parameters. Therefore, we visualise the Instance Hierarchy by drawing connections between the nodes with links, if selected. The rest of the tree is greyed out during the selection to let the user focus on the relevant connections.

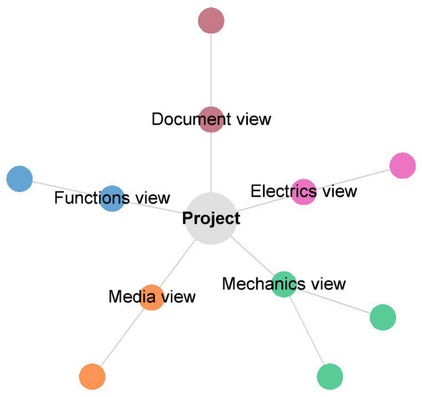


Fig. 3. Disciplines view in radial tree view

4.3 Space-efficient Visualisation

Certain decisions were made to de-clutter the visual space, and to make the most out of the limited available space:

A force-directed graph algorithm was implemented to arrange the nodes in an optimal way.

Labels were only displayed where needed, for example the label of the root node, for nodes representing views *Mechanical view* and currently selected nodes.

Color is used to visualise the view the nodes belongs to, the electrics view is colored in pink, orphan nodes (without a parent), are colored in red to bring attention to them.

Shapes In our case, shapes codify hierarchical information about the node:
Circles denote components that have one or more sub-components.
Squares denote components without any sub-component.
Triangles denote orphan nodes.

Size This feature was only used for the root node, being bigger than the others.

4.4 Software Prototype

A prototype, consisting of a frontend web application and a simple backend service for data model management, was built to show the feasibility of the concepts and as a foundation for further validation. The frontend web application is based on Angular 6 [5]. Further noteworthy libraries are the reactive programming library RXJS [6] and the JavaScript (JS) library Data-Driven Documents (D3)

[5] https://angular.io/
[6] http://reactivex.io/

[7]. The backend service is based on Spring Boot [8]. The User Interface (UI) as shown in Fig. 4 consists of a main view, which displays the hierarchy of the current project. Nodes can be double-clicked to re-arrange the view around them. Another panel is the details panel, to view the details of the currently selected node. Currently the following information are displayed here, *details* (name, ID, parent node), *attributes*, *values*, *views* (e.g. mechanical, electrical) and *actions* (add, remove, edit). The search function is also implemented, so that user can search components by name or attribute.

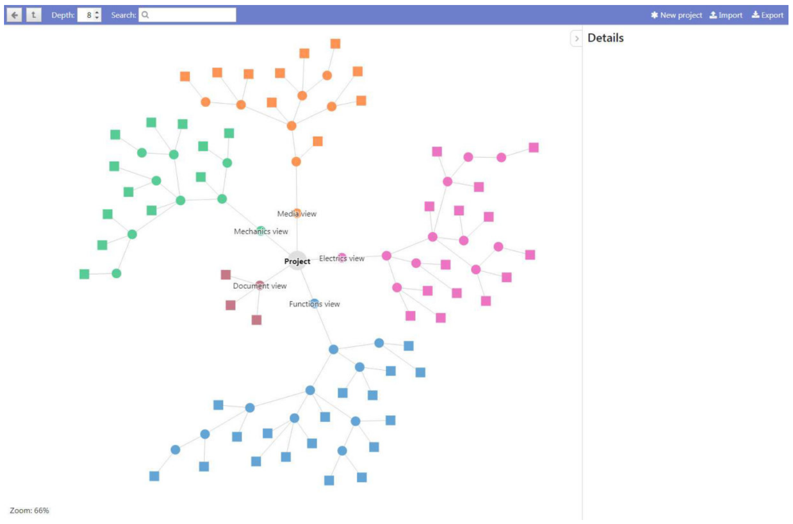


Fig. 4. User Interface of the software prototype

5 Performance Evaluation

This section reports on the evaluation of the applied visualisation techniques in the PSE domain. To evaluate whether the proposed solution approach and the developed prototype supports the process performance for domain experts in real-life use cases we measured via the KLM, first described in [2].
The aim of this method is to measure the needed execution time by expert users. To achieve this, performed tasks are broken into atomic keystroke-level actions. For so-called *Operators*, a standard set of actions and empirical data has been collected that represents the average execution time frames, which can be observed in Table 1.

[7] https://d3js.org/
[8] https://spring.io/projects/spring-boot

Shorthand	**Operator**	**Execution time (seconds)**
K	Keystroke	0.12 1.2 (typically 0.28)
T(n)	Type sequence of n characters	n x K
P	Point with mouse to target on display	1.1
B	Press or release mouse button	0.1
BB	Click mouse button	0.2
H	Move hand to keyboard or mouse	0.4
M	Mental act of routine thinking	0.6 - 1.35 (typically 1.2)
W(t)	Waiting for the system to respond	t

Table 1. List of standard KLM *operators*, including *shorthands* and estimated *execution time* [5].

The method can also take into account *Mental Operators*, breaks where the user stops the action and thinks either to find a function in the UI or recalling an information etc. Finally, all execution times are added up, so the time efficiency between different systems can be measured.
To test an initial evaluation, a relatively small and simple data set was created to measure performance between the AutomationML editor and the software prototype. To imitate the multi-disciplinary engineering environment, that is common in PSE, 98 components were created, which were modelled in different views (disciplines).

The following tasks were measured, according to the derived requirements and grouped by the use cases introduced in Section 3.2:

UC-1 Import & Export of an AML file
- 1.1 Import an AML file
- 1.2 Export an AML file

UC-2 Navigate in Project data
- 2.1 Showing only components relevant to a specific engineering discipline
- 2.2 Showing which other views a component is related to

UC-3 Search in Project data
- 3.1 Searching for a component by name
- 3.2 Searching for a component by attribute value

UC-4 Modify the Project data
- 4.1 Add a new component
- 4.2 Editing the details of a component
- 4.3 Changing the hierarchy of a component
- 4.4 Removing a component

5.1 Results

In this section we show the calculations of the first use case *Import & Export of a AML file* in both AutomationML editor and the prototype. The shorthand of the operators are printed in bold font. The rest of the results will be presented in Table 2, and the more extensive version of the calculations can be found in XX.

Task 1: Importing an AutomationML file AutomationML editor:

```
Initiate the import (decide to carry out the task) M
Find, point to and click the "Open a file" button M, P, BB
Wait for the "Select file" dialog to open W(0.5)
Find the desired file in the shown list M
Point to to the desired file P
Double click the left mouse button 2BB
Check that the root component is visible in the Instance Hierarchy window M
```

Total time = $4\mathbf{M} + 2\mathbf{P} + 3\mathbf{BB} + \mathbf{W}(0.5)$ = 4*1.2 + 2*1.1 + 3*0.2 + 0.5 = 8.1 sec

Proposed solution:

```
Initiate the import (decide to carry out the task) M
Find, point to and click the "Import" button M, P, BB
Wait for the "Select file" dialog to open W(0.5)
Find the desired file in the shown list M
Point to the desired file P
Double click the left mouse button 2BB
Wait for the graph to initialize W(1.5)
Check that the root component is visible in the main view M
```

Total time = $4\mathbf{M} + 2\mathbf{P} + 3\mathbf{BB} + \mathbf{W}(1.5) + \mathbf{W}(0.5)$ = 4*1.2 + 2*1.1 + 3*0.2 + 1.5 + 0.5 = 9.6 sec

Task 2: Exporting an AutomationML file AutomationML editor:

```
Initiate the export (decide to carry out the task) M
Find, point to and click the "File" dropdown menu M, P, BB
Wait for the "File" dropdown menu to open W(0.5)
Find, point to and click the "Save as..." menu item M, P, BB
Wait for the "Save as" dialog to open W(0.5)
Decide what name the new file should be given M
Move hand from mouse to keyboard H
Enter the name of the new file ("export.aml") T(10)
Hit the Enter key K
Wait for the file to be stored W(0.5)
```

Total time = $4\mathbf{M} + 2\mathbf{P} + \mathbf{H} + \mathbf{K} + 2\mathbf{BB} + \mathbf{T}(10) + 3\mathbf{W}(0.5)$ = 4*1.2 + 2*1.1 + 0.4 + 0.28 + 2*0.2 + 10*0.28 + 3*0.5 = 12.38 sec

Proposed solution:

```
Initiate the export (decide to carry out the task) M
Find, point to and click the "Export" button M, P, BB
Wait for the "Save as" dialog to open W(0.5)
Decide what name the new file should be given M
```

```
Move hand from mouse to keyboard H
Enter the name of the new file ("export.aml") T(10)
Hit the Enter key K
Wait for the file to be stored W(0.5)
```

Total time = $3\mathbf{M} + \mathbf{P} + \mathbf{H} + \mathbf{K} + \mathbf{BB} + \mathbf{T}(10) + 2\mathbf{W}(0.5)$ = 3*1.2 + 1.1 + 0.4 + 0.28 + 0.2 + 10*0.28 + 2*0.5 = 9.38 sec

Task 1: Importing an AutomationML file AutomationML editor:

```
Initiate the import (decide to carry out the task) M
Find, point to and click the "Open a file" button M, P, BB
Wait for the "Select file" dialog to open W(0.5)
Find the desired file in the shown list M
Point to to the desired file P
Double click the left mouse button 2BB
Check that the root component is visible in the Instance Hierarchy window M
```

Total time = $4\mathbf{M} + 2\mathbf{P} + 3\mathbf{BB} + \mathbf{W}(0.5)$ = 4*1.2 + 2*1.1 + 3*0.2 + 0.5 = 8.1 sec

Proposed solution:

```
Initiate the import (decide to carry out the task) M
Find, point to and click the "Import" button M, P, BB
Wait for the "Select file" dialog to open W(0.5)
Find the desired file in the shown list M
Point to the desired file P
Double click the left mouse button 2BB
Wait for the graph to initialize W(1.5)
Check that the root component is visible in the main view M
```

Total time = $4\mathbf{M} + 2\mathbf{P} + 3\mathbf{BB} + \mathbf{W}(1.5) + \mathbf{W}(0.5)$ = 4*1.2 + 2*1.1 + 3*0.2 + 1.5 + 0.5 = 9.6 sec

Task 2: Exporting an AutomationML file AutomationML editor:

```
Initiate the export (decide to carry out the task) M
Find, point to and click the "File" dropdown menu M, P, BB
Wait for the "File" dropdown menu to open W(0.5)
Find, point to and click the "Save as..." menu item M, P, BB
Wait for the "Save as" dialog to open W(0.5)
Decide what name the new file should be given M
Move hand from mouse to keyboard H
Enter the name of the new file ("export.aml") T(10)
Hit the Enter key K
Wait for the file to be stored W(0.5)
```

Total time = $4\mathbf{M} + 2\mathbf{P} + \mathbf{H} + \mathbf{K} + 2\mathbf{BB} + \mathbf{T}(10) + 3\mathbf{W}(0.5)$ = 4*1.2 + 2*1.1 + 0.4 + 0.28 + 2*0.2 + 10*0.28 + 3*0.5 = 12.38 sec

Proposed solution:

```
1  Initiate the export (decide to carry out the task) M
2  Find, point to and click the "Export" button M, P, BB
3  Wait for the "Save as" dialog to open W(0.5)
4  Decide what name the new file should be given M
5  Move hand from mouse to keyboard H
6  Enter the name of the new file ("export.aml") T(10)
7  Hit the Enter key K
8  Wait for the file to be stored W(0.5)
```

Total time = $3\mathbf{M} + \mathbf{P} + \mathbf{H} + \mathbf{K} + \mathbf{BB} + \mathbf{T}(10) + 2\mathbf{W}(0.5)$ = 3*1.2 + 1.1 + 0.4 + 0.28 + 0.2 + 10*0.28 + 2*0.5 = 9.38 sec

Task 3: Comparing two versions of an AutomationML file AutomationML editor:

```
1  Initiate the comparison (decide to carry out the task) M
2  Find, point to and click the "Project" component in the project tree M, P, BB
3  Right click on the "Project" component BB
4  Wait for the context menu to open W(0.5)
5  Find, point to and click on the "Expand tree" menu item M, P, BB
6  Repeat steps 2-5 for the second file 2M, 2P, 3BB, W(0.5)
7  Find a component that contains leafs (components without sub-components) M
8  Find the same component in the second file M
9  Compare if both components have the same number of sub-components M
10 Repeat steps 7-9 until the structure visible on the screen is compared 8 x 3M
11 Point to the Instance Hierarchy window P
12 Scroll down the project tree 2 sec
13 Point to the Instance Hierarchy window of the second file P
14 Scroll down the project tree 2 sec
15 Repeat steps 7-9 until the structure visible on the screen is compared 10 x 3M
16 Notice that the "RJX_FRP" component of the second file has less sub-components M
17 Compare the contents of the "RJX_FRP" component in both files in detail M
18 Notice that the "RJX_FRP_DCKR" component is missing in the second file M
```

Total time = $65\mathbf{M} + 6\mathbf{P} + 6\mathbf{BB} + 2\mathbf{W}(0.5)$ + 2*2.0 = 65*1.2 + 6*1.1 + 6*0.2 + 2*0.5 + 2*2.0 = 90.8 sec

Proposed solution:

```
1  Initiate the comparison (decide to carry out the task) M
2  Find the button that increases the depth of the shown project tree M
3  Point to the button P
4  Click the button until the whole project structure is visible 5 x BB
5  Zoom out to fit the whole structure on the screen 2 sec
```

```
6  Repeat steps 2-6 for the second file M, P, 5BB, 2 sec
7  Pick a "view" subtree to compare M
8  Find the same subtree in the second file M
9  Compare the general shape and size of the subtree M
10 Repeat steps 7-9 until all view subtree shapes have been compared 4 x 3M
11 Notice that the green subtree ("Mechanics view") is smaller M
12 Zoom in so the green subtree fills the whole screen 2 sec
13 Find a component that contains leafs (components without sub-components) M
14 Find the same component in the second file M
15 Compare if both components have the same number of sub-components M
16 Repeat steps 13-15 until the subtree is compared 8 x 3M
17 Notice that one of the components of the first file has more sub-components M
18 Point to and double click that component P, 2BB
19 Point to and double click that component in the second file P, 2BB
20 Read the name of one of the sub-components M
21 Check if a sub-component with the same name is present in the second file M
22 Repeat steps 20-21 until all sub-components have been checked 2 x 2M
23 Notice that the "RJX_FRP_DCKR" component is missing in the second file M
```

Total time $= 54\mathbf{M} + 4\mathbf{P} + 14\mathbf{BB} + 3{*}2.0 = 54{*}1.2 + 4{*}1.1 + 14{*}0.2 + 3{*}2.0$ $= 78.0$ sec

Task 4: Showing only the components relevant to a specific engineering discipline AutomationML editor:

As of now, the AutomationML editor does not provide a straight forward way of meeting the postconditions of this task entirely. There is the possibility of expanding only the components of the ”Electrics view” subtree, but this does not hide any other parts of the project hierarchy. Another way is by selecting the ”Electrics view” component, copying it and pasting it into a newly created AutomationML file. As this option is quite cumbersome and could potentially cause consistency errors within the data, it was not considered a valid solution for the task.

Proposed solution:

```
1 Initiate the filtering (decide to carry out the task) M
2 Find the "Electrics view" component M
3 Point to the "Electrics view" component P
4 Double click the "Electrics view" component 2BB
5 Wait for the system to only show sub-components of "Electrics view" anymore W(0.5)
```

Total time $= 2\mathbf{M} + \mathbf{P} + 2\mathbf{BB} + \mathbf{W}(0.5) = 2{*}1.2 + 1.1 + 2{*}0.2 + 0.5 = 4.4$ sec

Task 5: Showing which other views a component is contained in AutomationML editor:

1. Initiate the inspection (decide to carry out the task) **M**
2. Find, point to and double click (expand) the "Project" component **M, P, 2BB**
3. Find, point to and double click (expand) the "Mechanics view" component **M, P, 2BB**
4. Find, point to and double click (expand) the revealed "RJX" component **M, P, 2BB**
5. Check if "RJX" contains a component named "RJX_IO" **M**
6. Notice that "RJX" and thus the "Mechanics view" contains "RJX_IO" **M**
7. Find, point to and double click (expand) the "Electrics view" component **M, P, 2BB**
8. Find, point to and double click (expand) the revealed "RJX" component **M, P, 2BB**
9. Check if "RJX" contains a component named "RJX_IO" **M**
10. Notice that "RJX" and thus the "Electrics view" contains "RJX_IO" **M**
11. Find, point to and double click (expand) the "Media view" component **M, P, 2BB**
12. Find, point to and double click (expand) the revealed "RJX" component **M, P, 2BB**
13. Check if "RJX" contains a component named "RJX_IO" **M**
14. Notice that "RJX" and thus the "Media view" contains "RJX_IO" **M**
15. Find, point to and double click (expand) the "Document view" component **M, P, 2BB**
16. Find, point to and double click (expand) the revealed "RJX" component **M, P, 2BB**
17. Check if "RJX" contains a component named "RJX_IO" **M**
18. Notice that "RJX" and thus the "Document view" does not contain "RJX_IO" **M**
19. Conclude that all views except for the "Document view" contain "RJX_IO" **M**

Total time = 19**M** + 9**P** + 18**BB** = 19*1.2 + 9*1.1 + 18*0.2 = 36.3 sec

Proposed solution:

1. Initiate the inspection (decide to carry out the task) **M**
2. Find, point to and double click (expand) the "Functions view" component **M, P, 2BB**
3. Find, point to and double click (expand) the revealed "RJX" component **M, P, 2BB**
4. Find, point to and click (select) the revealed "RJX_IO" component **M, P, BB**
5. Contemplate the colors of the revealed components of the same name **M**
6. Notice that a brown-colored component is missing **M**
7. Recall that the color brown is assigned to components of the "Document view" **M**
8. Conclude that all views except for the "Document view" contain "RJX_IO" **M**

Total time = 8**M** + 3**P** + 5**BB** = 8*1.2 + 3*1.1 + 5*0.2 = 13.9 sec

Task 6: Searching for a component with a specific name AutomationML editor:

1. Initiate the search (decide to carry out the task) **M**
2. Find, point to and double click (expand) the "Project" component **M, P, 2BB**
3. Find, point to and click (select) the "Mechanics view" component **M, P, BB**
4. Hit the Ctrl + F keys (actives the search function) **T(2)**
5. Find, point to and click the revealed search bar **M, P, BB**
6. Move hand from mouse to keyboard **H**
7. Enter the name of the searched component ("RJX_FRP_DCKL_JOG01_DIAL") **T(23)**
8. Hit the Enter key **K**
9. Find the highlighted component **M**
10. Validate that the component name is the one that was searched for **M**

Total time = 6**M** + 3**P** + **H** + **K** + 4**BB** + **T**(25) = 6*1.2 + 3*1.1 + 0.4 + 0.28 + 4*0.2 + 25*0.28 = 18.98 sec

Proposed solution:

```
1 Initiate the search (decide to carry out the task) M
2 Find, point to and click the search bar M, P, BB
3 Move hand from mouse to keyboard H
4 Enter a part of the searched component's name ("01_dial") T(7)
5 Go through the two results of the revealed search result list M
6 Recall that the color green is assigned to components of the "Mechanics view" M
7 Find the result with the green color coding M
8 Find, point to and click the "Select" button of that search result M, P, BB
```

Total time = 6**M** + 2**P** + **H** + 2**BB** + **T**(7) = 6*1.2 + 2*1.1 + 0.4 + 2*0.2 + 7*0.28 = 12.16 sec

Task 7: Searching for a component with a specific attribute value AutomationML editor:

As of now, the AutomationML editor does not provide a search function that takes attribute values into account. The only way of meeting this task's postconditions would be traversing the project's components one by one. As this process is not feasible for large hierarchies, it was not considered a valid solution for the task.

Proposed solution:

```
1 Initiate the search (decide to carry out the task) M
2 Find, point to and click the search bar M, P, BB
3 Move hand from mouse to keyboard H
4 Enter a part of the searched for attribute value ("Level le") T(8)
5 Go through the two results of the revealed search result list M
6 Recall that the color pink is assigned to components of the "Electrics view" M
7 Find the result with the pink color coding M
8 Find, point to and click the "Select" button of that search result M, P, BB
```

Total time = 6**M** + 2**P** + **H** + 2**BB** + **T**(8) = 6*1.2 + 2*1.1 + 0.4 + 2*0.2 + 8*0.28 = 12.44 sec

Task 8: Showing detailed information of a component on demand AutomationML editor:

```
1 Initiate the inspection (decide to carry out the task) M
2 Find, point to and click (select) the "RJX_MBRD_MEM" component M, P, BB
3 Find the ID value in the details panel on the right of the window M
```

```
4  Read the ID value M
5  Find, point to and click (open) the "Attributes" tab of the details panel M, P, BB
6  Find the "Description" attribute in the details panel M
7  Read the "Description" attribute value M
```

Total time $= 7\mathbf{M} + 2\mathbf{P} + 2\mathbf{BB} = 7{*}1.2 + 2{*}1.1 + 2{*}0.2 = 11.0$ sec

Proposed solution:

```
1  Initiate the inspection (decide to carry out the task) M
2  Find, point to and click (select) the "RJX_MBRD_MEM" component M, P, BB
3  Find the ID value in the details panel on the right of the window M
4  Read the ID value M
5  Find the "Description" attribute in the details panel M
6  Read the "Description" attribute value M
```

Total time $= 6\mathbf{M} + \mathbf{P} + \mathbf{BB} = 6{*}1.2 + 1.1 + 0.2 = 8.5$ sec

Task 9: Adding a new component AutomationML editor:

```
1   Initiate the addition (decide to carry out the task) M
2   Find, point to and click (select) the "RJX_IO" component M, P, BB
3   Right click on the "RJX_IO" component BB
4   Wait for the context menu to open W(0.5)
5   Find the "Add..." menu item M
6   Point to the "Add..." menu item BB
7   Wait for the context menu to expand W(1.0)
8   Find, point to and click the "Internal Element" menu item M, P, BB
9   Find, point to and double click (rename) the new component M, P, 2BB
10  Move hand from mouse to keyboard H
11  Press and hold the Delete key (delete the default name) 1 sec
12  Enter the new component's name ("RJX_IO_LINE") T(11)
13  Hit the Enter key K
14  Move hand from keyboard to mouse H
15  Find, point to and click the "Add attribute" button in the side panel M, P, BB
16  Find, point to and double click (rename) the new attribute's name M, P, 2BB
17  Move hand from mouse to keyboard H
18  Press and hold the Delete key (delete the default name) 1 sec
19  Enter the new attribute name ("Description") T(11)
20  Hit the Enter key K
21  Move hand from keyboard to mouse H
22  Find, point to and click the "Value" input field M, P, BB
23  Move hand from mouse to keyboard H
24  Enter the new attribute value ("Line in") T(7)
25  Move hand from keyboard to mouse H
26  Find, point to and click an empty location on the screen  M, P, BB
```

Total time $= 9\mathbf{M} + 7\mathbf{P} + 6\mathbf{H} + 2\mathbf{K} + 11\mathbf{BB} + \mathbf{T}(29) + \mathbf{W}(1.0) + \mathbf{W}(0.5)$
$+ 2{*}1.0 = 9{*}1.2 + 7{*}1.1 + 6{*}0.4 + 2{*}0.28 + 11{*}0.2 + 29{*}0.28 + 1.0 + 0.5 +$

$2*1.0 = 35.28$ sec

Proposed solution:

```
Initiate the addition (decide to carry out the task) M
Find, point to and click (select) the "RJX_IO" component M, P, BB
Find, point to and click the "Add" button in the side panel M, P, BB
Wait for the "Add new element" dialog to open W(0.5)
Find, point to and click the "Name" input field M, P, BB
Move hand from mouse to keyboard H
Enter the last part of the new component's name ("_LINE") T(5)
Move hand from keyboard to mouse H
Find, point to and click the "Add attribute" button M, P, BB
Find, point to and click the "Name" input field of the new attribute M, P, BB
Move hand from mouse to keyboard H
Enter the new attribute name ("Description") T(11)
Hit the Tab key K
Enter the new attribute value ("Line in") T(7)
Hit the Enter key K
```

Total time $= 6\mathbf{M} + 5\mathbf{P} + 3\mathbf{H} + 2\mathbf{K} + 5\mathbf{BB} + \mathbf{T}(23) + \mathbf{W}(0.5) = 6*1.2 + 5*1.1 + 3*0.4 + 2*0.28 + 5*0.2 + 23*0.28 + 0.5 = 22.4$ sec

Task 10: Editing the details of a component AutomationML editor:

```
Initiate the editing (decide to carry out the task) M
Find, point to and click (select) the "RJX_IO" component M, P, BB
Find, point to and click the "Value" input field in the side panel M, P, BB
Move hand from mouse to keyboard H
Press and hold the Return key (delete the current value) 1 sec
Enter the new attribute value ("IO panel v2") T(11)
Find, point to and click an empty location on the screen  M, P, BB
```

Total time $= 4\mathbf{M} + 3\mathbf{P} + \mathbf{H} + 3\mathbf{BB} + \mathbf{T}(11) + 1.0 = 4*1.2 + 3*1.1 + 0.4 + 3*0.2 + 11*0.28 + 1.0 = 13.18$ sec

Proposed solution:

```
Initiate the editing (decide to carry out the task) M
Find, point to and click (select) the "RJX_IO" component M, P, BB
Find, point to and click the "Edit" button in the side panel M, P, BB
Wait for the "Edit element" dialog to open W(0.5)
Find, point to and click the "Value" input field of the attribute M, P, BB
Move hand from mouse to keyboard H
Press and hold the Return key (delete the current value) 1 sec
Enter the new attribute value ("IO panel v2") T(11)
Hit the Enter key K
```

Total time $= 4\mathbf{M} + 3\mathbf{P} + \mathbf{H} + 3\mathbf{BB} + \mathbf{T}(11) + \mathbf{W}(0.5) + 1.0 = 4*1.2 + 3*1.1 + 0.4 + 3*0.2 + 11*0.28 + 0.5 + 1.0 = 13.68$ sec

Task 11: Moving a component to a new position in the hierarchy tree
AutomationML editor:

```
1  Initiate the editing (decide to carry out the task) M
2  Find the "RJX_IO" component M
3  Point to to the "RJX_IO" component P
4  Press the left mouse button B
5  Find the "RJX_MBRD" component M
6  Point to to the "RJX_MBRD" component P
7  Release the left mouse button B
8  Wait for the "Drag & Drop" dialog to open W(0.5)
9  Find, point to and click the "Move" button in the dialog M, P, BB
```

Total time = $4\mathbf{M} + 3\mathbf{P} + \mathbf{BB} + 2\mathbf{B} + \mathbf{W}(0.5)$ = 4*1.2 + 3*1.1 + 0.2 + 2*0.1 + 0.5 = 9.0 sec

Proposed solution:

```
1  Initiate the editing (decide to carry out the task) M
2  Find, point to and click (select) the "RJX_IO" component M, P, BB
3  Find, point to and click the "Change parent" button in the side panel M, P, BB
4  Find, point to and click (select) the "RJX_MBRD" component M, P, BB
```

Total time = $4\mathbf{M} + 3\mathbf{P} + 3\mathbf{BB}$ = 4*1.2 + 3*1.1 + 3*0.2 = 8.7 sec

Task 12: Removing a component AutomationML editor:

```
1  Initiate the removal (decide to carry out the task) M
2  Find, point to and click (select) the "RJX_FRP_MXR_EQL" component M, P, BB
3  Move hand from mouse to keyboard H
4  Hit the Delete key K
5  Wait for the "Confirm deletion" dialog to open W(0.5)
6  Hit the Enter key K
7  Move hand from keyboard to mouse H
8  Find, point to and double click (expand) the "Mechanics view" component M, P, 2BB
9  Find, point to and double click (expand) the "RJX" component M, P, 2BB
10 Find, point to and double click (expand) the "RJX_FRP" component M, P, 2BB
11 Find, point to and double click (expand) the "RJX_FRP_MXR" component M, P, 2BB
12 Find, point to and click (select) the "RJX_FRP_MXR_EQL" component M, P, BB
13 Move hand from mouse to keyboard H
14 Hit the Delete key K
15 Wait for the "Confirm deletion" dialog to open W(0.5)
16 Hit the Enter key K
17 Move hand from keyboard to mouse H
18 Repeat steps 8-16 with the "Media view" 5M, 5P, 9BB, 2H, K, W(0.5)
```

Total time = $12\mathbf{M} + 11\mathbf{P} + 6\mathbf{H} + 5\mathbf{K} + 19\mathbf{BB} + 3\mathbf{W}(0.5)$ = 12*1.2 + 11*1.1 + 6*0.4 + 5*0.28 + 19*0.2 + 3*0.5 = 35.6 sec

Proposed solution:

```
Initiate the removal (decide to carry out the task) M
Find, point to and click (select) the "RJX_FRP_MXR_EQL" component M, P, BB
Find, point to and click the "Remove" button in the side panel M, P, BB
Wait for the "Remove element" dialog to open W(0.5)
Make sure all views are selected M
Find, point to and click the "Delete" button in the dialog (confirm) M, P, BB
```

Total time = $5\mathbf{M} + 3\mathbf{P} + 3\mathbf{BB} + \mathbf{W}(0.5) = 5*1.2 + 3*1.1 + 3*0.2 + 0.5 =$ 10.4 sec

	AutomationML editor	Proposed solution
Task 1 (Import)	**8.1**	9.6
Task 2 (Export)	12.38	**9.38**
Task 3 (Comparison)	90.8	**78.0**
Task 4 (Specific discipline)	-	**4.4**
Task 5 (Other views)	36.3	**13.9**
Task 6 (Search by name)	18.98	**12.16**
Task 7 (Search by attribute)	-	**12.44**
Task 8 (Detailed information)	11.0	**8.5**
Task 9 (Add)	35.28	**22.4**
Task 10 (Edit)	**13.18**	13.68
Task 11 (Move)	9.0	**8.7**
Task 12 (Remove)	35.6	**10.4**

Table 2. The estimated execution times for all tasks (in seconds), as performed with the AutomationML editor and the proposed solution. The lower time of each task is marked bold.

Summary

6 Discussion

This section discusses results of the research questions introduced in Section 1.

RQ1. What are main requirements and elements in production systems engineering? We derived requirements and actions that are usually performed by engineers in multi-disciplinary environments in section 3. The following requirements were identified: *The ability to represent project hierarchy*, *The capability to represent cross-references between components*, *An efficient way to represent large data structures* and *A discipline-neutral view.* Furthermore, we derived the following four basic use cases described in Section 3.2, which are representative for the daily work of an engineer in PSE: *UC-1 Import & Export of an AML file*, *UC-2 Navigate in Project Data*, *UC-3 Search in Project Data*, *UC-4 Modify the Project data.*

RQ2. Can Information Visualisation methods make PSE engineering more efficient? To answer this research question, we designed and implemented a web-based prototype to test whether our design decisions could have a positive impact on the effectiveness of handling engineering data. Moreover, we tested the prototype compared to the AutomationML editor with the KLM.

Limitations. There are certain limitations to this work: The prototype in its current form, only covers the discussed use cases, however there are more use cases that are necessary for the daily work in PSE. Furthermore, the generated test data was relatively simple and small in comparison to real-world production system data sets. Furthermore, it does not cover the whole functionality of the AML standard, such as SystemUnitClasses, RoleClasses, InterfaceClasses, InternalLinks and references to external resources. Moreover, the names of components, represented in different views were named the same over all views, which is not always the case. The evaluation of KLM is a static calculation method, and does not test the data with real participants, who generate more realistic data. These limitations should be covered in future work to ensure increased validity for the results.

7 Conclusion and Future Work

Visualisation in PSE has not gained much attention so far, although the benefits of implementing a sensible solution are manifold: A better understanding of project relationships and dependencies between disciplines has the potential to decrease defects and effort for data integration and to increase data quality and chances for successful completion of the project, on the other hand.

We introduced and investigated PSE use cases and information visualisation methods to improve the data perception in PSE. Our proposed solution has the mentioned drawbacks and shortcoming that need to be overcome in future research. However, the identified gaps in research, the requirements for visualising engineering data and proof of concept in form of a developed software prototype contribute to the field of process improvement. Our results are an initial step to gather knowledge in the area of applying information visualisation methods in the PSE domain to improve the process quality in the engineering data exchange process.

Future Work. To overcome the limitations of the basic test data and to evaluate the findings of this paper additional research is needed. Suggestions for future work is to configure the prototype with special test data-sets for each discussed use case and to collect empirical data instead of relying on the KLM. Usability tests or field studies would be viable approaches. In this work we focused on visualising the project hierarchy, to make it more useful for real-world applications, more functions of AML should be added.

Acknowledgment. The financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital & Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References

1. Stefan Biffl, Arndt Lüder, Felix Rinker, and Laura Waltersdorfer. Efficient engineering data exchange in multi-disciplinary systems engineering. In *International Conference on Advanced Information Systems Engineering*, pages 17–31. Springer, 2019.
2. Stuart K. Card, Thomas P. Moran, and Allen Newell. *The psychology of human-computer interaction*, volume 15. CRC Press, 1983.
3. Thomas Holm, Lars Christiansen, Markus Göring, Tobias Jäger, and Alexander Fay. Iso 15926 vs. iec 62424comparison of plant structure modeling concepts. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pages 1–8. IEEE, 2012.
4. Tobias Jäger, Alexander Fay, Thomas Wagner, and Ulrich Löwen. Mining technical dependencies throughout engineering process knowledge. In *ETFA2011*, pages 1–7. IEEE, 2011.
5. David Kieras. Using the keystroke-level model to estimate execution times. Technical report, University of Michigan, 2001.
6. Arndt Lüder, Johanna-Lisa Pauly, Konstantin Kirchheim, Felix Rinker, and Stefan Biffl. Migration to AutomationML based Tool Chains - incrementally overcoming Engineering Network Challenges. In *5th AutomationML User Conference*, 2018.
7. Riccardo Mazza. *Introduction to information visualization*. Springer Science & Business Media, 2009.
8. David D Romero, Tone-Grete Graven, and Nina F Thornhill. Towards the development of a tool for visualising plantwide dependencies. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–4. IEEE, 2014.
9. K Stuart. Card, jock d. mackinlay, and ben shneiderman. *Readings in Information Visualization: Using Vision to Think*, 1999.
10. Milan Vathoopan, Hendrik Walzel, Waldemar Eisenmenger, Alois Zoitl, and Benjamin Brandenbourger. Automationml mechatronic models as enabler of automation systems engineering: Use-case and evaluation. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 51–58. IEEE, 2018.
11. Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.