INTERNATIONAL DESIGN CONFERENCE – DESIGN 2020
https://doi.org/10.1017/dsd.2020.37

AN MBSE-BASED APPROACH FOR THE DEFINITION AND EVALUATION OF ENGINEERING IT ARCHITECTURES

J. Heihoff-Schwede ✉, L. Kaiser and R. Dumitrescu

Fraunhofer IEM, Germany

✉ joerg.heihoff-schwede@iem.fraunhofer.de

Abstract

As the complexity of the systems increases, so does the complexity of designing a suitable engineering IT architecture. Challenges reach from the definition of required and consistent functionalities and interfaces to the evaluation, which combination of IT tools fulfils the required functionality, usability and interoperability in the best way. Thus, we provide a procedure, methodology and modelling support for the definition of functional and logical engineering IT architectures and their holistic evaluation. The approach is part of an overall procedure and demonstrated in an example.

Keywords: engineering IT, architectural design, design evaluation, modelling, systems engineering (SE)

1. Introduction

Technical systems and their environment are changing significantly. For example, Mechatronic Systems emerge to complex intelligent technical systems (ITS) (Gausemeier et al., 2013). While the system's complexity increases, the engineering must adapt (Törngren and Grogan, 2018). Engineering comprises, besides organizational aspects, processes, methods, information models and information technology (IT) tools (Abramovici, 2018). Engineering IT describes the required IT infrastructure to support the aforementioned aspects of engineering and consists of IT tools and their interfaces. As a consequence of evolution, suitable engineering approaches, such as Systems Engineering (SE), are currently expanded in research and implemented in enterprises. SE is an interdisciplinary approach to successfully design complex systems (Gausemeier et al., 2013), generating - besides other engineering approaches - a rising number of heterogeneous artefacts (information carrier) (Alvarez-Rodríguez et al., 2019). The engineering approaches comprise new engineering processes and methods, which pose new challenges to the engineering IT such as traceability or end-to-end configuration management (Fisher et al., 2014). In addition, digitalization has a major impact on engineering. One example being Model-based systems engineering (MBSE), facilitating the use of granular, computer-based models instead of document-based information (Friedenthal et al., 2012). MBSE demonstrates that more and more artefacts become digital, requiring a defined structure and inducing a need for end-to-end continuity.

This necessitates a broad set of functionality regarding the engineering IT, often leading to a large number of combined tools (Alvarez-Rodríguez et al., 2019). On the one hand, the tools need to and can be combined in various, synergetic ways, especially since they have to communicate via interfaces, which may be incompatible or offer limited capabilities. On the other hand, the resulting architectures meet the requirements in terms of functionality and quality (e.g. usability) very differently.

In this paper, we introduce a systematic approach to support the definition of an engineering IT architecture and to evaluate which set of deployable software components should be implemented

regarding the holistic fulfilment of requirements. Therefore, we discuss the existing challenges in the definition and evaluation of engineering IT architectures and present existing contributing approaches (Section 2). In Section 3, we introduce a systematic procedure supported by a guiding methodology and a modelling support for engineering IT architecture design and evaluation. We apply the approach with help of a consistent example, acting as a first indicator for validation. We summarize the results and give an outlook in Section 4.

2. Challenges and current approaches in engineering IT architecture design and evaluation

2.1. Recent challenges

There are two **major drivers for the complexity of the engineering IT**. First, the number and diversity of engineering approaches and resulting artefacts is rising (Alvarez-Rodríguez et al., 2019). This means new engineering processes, defining what to do (ISO, 2015) and methods defining how to do (Graner, 2013), need to be implemented. SE represents one example comprising a broad set of processes and methods which need to be supported by the engineering IT. Since SE is an interdisciplinary approach, it inherently leads to cross-domain cutting, heterogeneous artefacts. Moreover, these artefacts need to be controlled by overarching, consistent lifecycle management processes such as configuration management (ISO, 2015; Fisher et al., 2014). In addition, traceability is required by normative aspects (ISO, 2010) or engineering methods (Friedenthal et al., 2012) implying the linkage of different artefacts. Second, digitalization has impact on engineering. Artefacts are digitized. They need to be represented by a defined structure of data, e.g. demonstrated by MBSE. The digital transformation goes beyond and implies continuity of data along IT-systems (Bauer et al., 2015) to support end-to-end processes and automation, e.g. required for the concept of digital twin.

These facts lead to **challenges in designing the engineering IT** (C) since the complexity drivers raise new requirements to be fulfilled by the engineering IT. While there are new and highly interrelated processes and methods producing numerous aspects, it is challenging to define the *required functionality and requirements* towards an engineering IT (C1)**.** While tools only have a limited set of functionality, the broad set of processes and methods requires also a *broad set of tools* (C2). Since processes and methods require linked data, the tools need to be interconnected. However, the *interfaces are often incompatible* or lack the required functionality, making the design of continuous engineering IT architectures challenging (C3) (Barth et al., 2012). Besides interconnectivity, each tool has *different characteristics regarding functionality and quality* (e.g. usability) (ISO, 2011). This raises the question which combination of tools fits the requirements of processes, methods and stakeholders best. In practice this often leads to a dilemma since the tools with best coverage in functionality and quality do often lead to federated approaches with complex interfaces, impeding continuity and causing high implementation and maintenance effort (Lu, 2019) (C4). Thus, IT tools are often implemented ad-hoc and lack a holistic view on the software capabilities and strategy (Holligan et al., 2017). A systematic and guided approach is required.

A further problem area is in the communication and documentation of the results from analysis and design activities of the engineering IT architecture. Models seem to be an appropriate way to support in this field (The Open Group, 2018). The process of engineering IT design involves heterogeneous groups of stakeholders with different backgrounds and concerns, e.g. process/method designers, applying engineers, IT architects and developers or tool vendors (Lu, 2019). The information gathered in the analysis and design process need to be captured and represented in a way being *understandable for the different stakeholders* (C5). Since the intention of the documentation is communication and specification, it needs to be clear how to document which aspects of the analysis and design process to *ensure unambiguity and standardization* (C6) (Becker et al., 1995). Moreover, the documentation needs to *assure traceability*, since change impact analyses and reasoning need to be assured along the various artefacts created in the analysis and design phases (C7).

The challenges induce the following **needs** (N) for this paper, summarized in Figure 1. First, these problems require an approach to *define engineering IT architectures*, supporting the analysis of

https://doi.org/10.1017/dsd.2020.37 Published online by Cambridge University Press

requirements and required functions. The approach has to contribute to the definition of software interfaces and consideration of their compatibility within the architecture definition **(N1)**. It has to support the *evaluation of IT architectures* regarding functional and quality requirements, considering a holistic engineering IT architecture. In case of trade-offs, the approach will give extensive decision support **(N2)**. Second, the approach needs to ensure an unambiguous and defined documentation in form of a *modelling framework*, easy to understand for a broad set of stakeholders, covering all relevant aspects of analysis and design definition in a traceable way **(N3)**. Third, the approach should give a *systematic and guided procedure* without extensive tailoring, to allow complete and precise specifications **(N4)**.

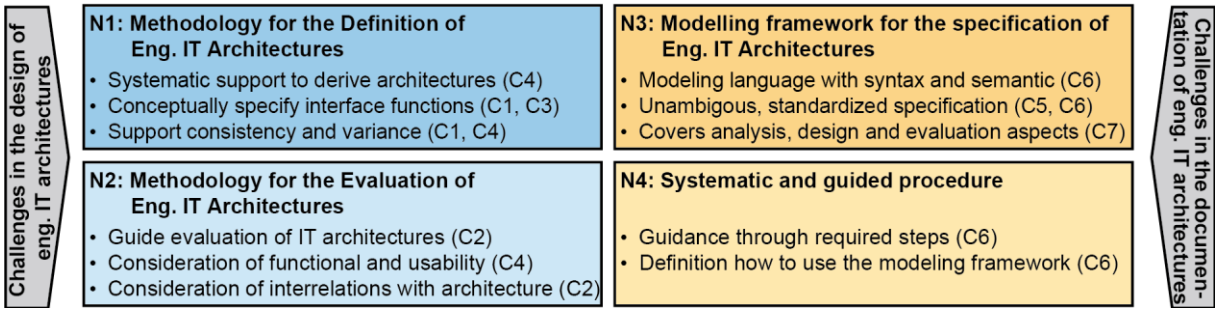

Figure 1. Summary of challenges and needs arising from the problem analysis

2.2. Available contributions for engineering IT architecture definition and evaluation

There are various approaches from different backgrounds which can support to analyse requirements, to define an IT architecture and/or to evaluate software tools. Thus, we briefly present and discuss some available contributions below. Table 1 shows a brief summary.

Table 1. Overview of the presented approaches and their satisfaction of needs

Legend ○ Fully satisfied ◐ Partially satisfied ○ Not satisfied	Satisfaction of needs defined in section 2.1			
Approach	**N1**	**N2**	**N3**	**N4**
TOGAF	●	○	◐	◐
PDM Selection by Stark	○	●	○	◐
Morphology by Hoffman & Heimes	◐	○	○	○
MBPE by Huth el al.	◐	○	●	◐
MBSE suite design by Friedland et al.	●	○	●	●
MBSE suite tool study by Friedland et al.	○	●	◐	●
Tool-chain development framework by Lu	◐	◐	◐	◐

The field of Enterprise Architecture Management (EAM) approaches generally addresses the goal to analyse enterprises and to develop a suitable IT solution. They cover a broad field from business analysis, the design of IT architectures to planning and implementation. While different approaches exist, they have similarities. A well-known example is **The Open Group Architecture Framework (TOGAF)** (The Open Group, 2018). Its Architecture Development Method (ADM) offers an 8-phase methodological approach to develop a vision, architectures for business, information systems and technology. Additional phases support the implementation and architecture management. TOGAF includes a meta-model core-package with several extensions. TOGAF proposes possible diagrams and matrices for documentation and is neutral regarding a modelling language. Concerning engineering IT architecture, particularly the development of the information systems architecture is relevant, consisting of the parts of data architecture and application architecture. The application architecture describes the information system services allocated to logical or physical application components. Analogously, the data architecture maps required data entities to logical or physical data components. While TOGAF describes procedural steps which include architecture

https://doi.org/10.1017/dsd.2020.37 Published online by Cambridge University Press

definition, it lacks tangible aids to systematically derive an engineering architecture based on identified requirements. The evaluation of software tools or architectures is not considered. Since TOGAF is neutral of language, it defines modelling artefacts, but does not give guidelines how to realize them in a model in an unambiguous way. The approach is very generic, thus specific characteristics of engineering are not considered and adaption to the specific context is complex.

Beside generic EAM approaches, contributions for the application in engineering exist. Stark (2015) describes a best practice for the **selection of Product Data Management (PDM) systems**. The procedure covers phases for preparation, the analysis of requirements, the evaluation including identification and benchmarking of potential PDM systems based on scenarios and deployment. The analysis comprises the documentation of the business objectives and current situation, including product data, users of product data, existing applications and PDM systems. The evaluation of a suitable PDM system is based on the scenarios and defined criteria. Other approaches are similar since they describe processes to evaluate software tools based on requirements with help of a cost-utility analysis (Schütte and Vering, 2011). While these approaches typically include a procedure and methods for evaluation and trade-offs, they often do not include methodologies for the systematic identification and analysis of requirements as evaluation criteria. Moreover, the approaches focus on the selection of a single software tool, e.g. a PDM system, and not a landscape of interconnected tools.

Huth et al. (2018) present the **model-based process engineering** approach which represents and analyses process, product system (the technical system under development) and the IT system based on UML, SysML and BPMN. Guidance is provided by a four-step modelling procedure. The authors focus on the application in engineering and thus contribute an approach for integrated analysis and representation. While the approach offers a procedure, it lacks a methodology and modelling guideline describing how to perform the procedural steps. The analysis is limited since engineering methods and requirements are not explicitly covered. Furthermore, the approach seems to define only high-level architectures and does not consider the evaluation and selection of particular IT systems.

A **morphology for the design of engineering IT systems** is presented by Hoffmann and Heimes (2018). They consider four aspects: technology, organization, process and data relating categories. A multi-dimensional architectural space is created by the breakdown of each aspect. For example, a technical characteristic might be “interfaces”, which can be characterized as “generic” or “specific”. It provides recommendations based on goals for an idealized engineering IT architecture. The approach supports to identify basic potentials and variables for the design of engineering IT infrastructures. However, it is not capable to analyse the current situation of an organization in detail to gather specific requirements. The definition of a particular IT architecture and selection of tools is not considered.

Friedland et al. developed two different approaches addressing the problem area. The first approach by Friedland et al. (2016) describes an **SE approach for MBSE tool suite design** which specifies work processes to capture the needs of the tool-users by use cases, processes steps and unit tasks. The unit tasks are specified by requirements. In a further step, an information model is created to specify the data and their relationships, moreover logical IT systems and their interfaces are allocated. The work results are documented within a model according to a meta-model. While the approach covers requirements analysis based on use cases and processes, it does not explicitly consider engineering methods. The evaluation and selection of a logical architecture of tools is not covered and only considers MBSE tools.

In a second approach, Friedland et al. (2017) describe the **SE-based trade study of an MBSE tool**. There is no explicit relation to the first approach. It covers a pre-phase to collect different levels of use cases and requirements. Phase derives a set of measurable evaluation criteria which are evaluated by tool vendors within a request for information and then evaluated by the requester. In phase 2, test scenarios are defined, presented within the solutions of selected tool vendors and finally evaluated to select the best tool. While a systematic procedure and methods for the evaluation are given, the approach does not consider how to evaluate holistic tool-combinations or architectures but single or simple combinations of tools. There is no information how to document or model the work results.

Lu (2019) presents an **approach to develop tool chains for the engineering of cyber-physical systems**. The procedure covers the definition of tool-chain stakeholders which deliver views, being classified in four categories. The views are captured as requirements and constraints are defined. Based on techniques, which aim to formalize the requirements and their context, tools and their interactions are

https://doi.org/10.1017/dsd.2020.37 Published online by Cambridge University Press

defined to derive a tool-chain concept. The approach is highly formal using domain specific models within a process management system to generate OSLC interfaces. While the approach allows the creation of integrated tools-chains by automated generated OSLC interfaces within a structured procedure, it does insufficiently consider other forms of interfaces such as proprietary interfaces. The required formal nature and models are hard to understand by a broad set of stakeholders. The approach does not support a semi-formal definition of tool-architectures. The individual identification and elicitation of requirements as evaluation criteria by analysing engineering processes and methods within a structured approach do not seem to be covered. Essential parts of the approach are not published yet.
None of the approaches allow the integrated architecture definition (N1) and tool evaluation (N2) based on a systematic procedure (N3) within a fully traceable, tool-supported model. The presented approaches either lack the full coverage from architecture to an evaluated set of tools or don't provide a modelling-framework which is suitable to be understood and applied by a broad set of stakeholders.

3. Approach for the definition and evaluation of engineering IT architectures

SE is an approach to successfully realize complex systems. In this work we define the Engineering IT as the system-of-interest (SoI) and use techniques of (MB-)SE for its design. The fundamental goal is the definition of an engineering IT architecture comprising of deployable software tools fulfilling a set of Evaluation Criteria in the best way. A systematic analysis of the processes and methods and resulting artefacts, their relations and use cases as described in Heihoff-Schwede et al. (2019) needs to be performed as pre-requisite to deliver input for the following procedure. Our approach consists of two phases and four tasks, which may be divided into steps, presented in Figure 2. They can be proceeded iteratively, e.g. to discover necessary support-functions. Phase 1 considers the derivation of functions and corresponding evaluation criteria which are resulting in a functional engineering IT architecture, acting as solution-free description **(Section 3.1)**. In the next task, suitable software components are identified and consistent combinations of allocations to functions are identified, resulting in consistent sets of software components **(Section 3.2)**. While we present the definition of the logical IT architecture as a separate step **(Section 3.3),** it should be proceeded parallel to the variant identification. The resulting component architecture describes the IT components and characterizes the resulting interfaces. In phase 2 the component architecture variants are evaluated with respect to their criteria coverage **(Section 3.4)**. All work results are captured in an integrated model based on OMG SysML and modelling guidelines, allowing end-to-end traceability from process/method analysis to the evaluated IT architecture.
The application is demonstrated and validated on the example of a toolset which should be able to handle textual requirements in combination with architecture artefacts (e.g. in a visual notation such as SysML) for the development of ITS (generically described in ISO, 2015). The analysis of activities, artefacts and use cases resulted in required tool functionality to import customer requirements, to edit, store and manage the lifecylce of requirements and to administrate the tool and to generate automated reports. Since the textual requirements are linked to architecture elements (context & functional elements), interfaces to the architecture modelling and its sub-functions need to be considered. While we use techniques of (MB-)SE for IT design, we analyse requirements and MBSE as SoI. The example is consistent with and based on the approach presented in Heihoff-Schwede (2019, 2019a).

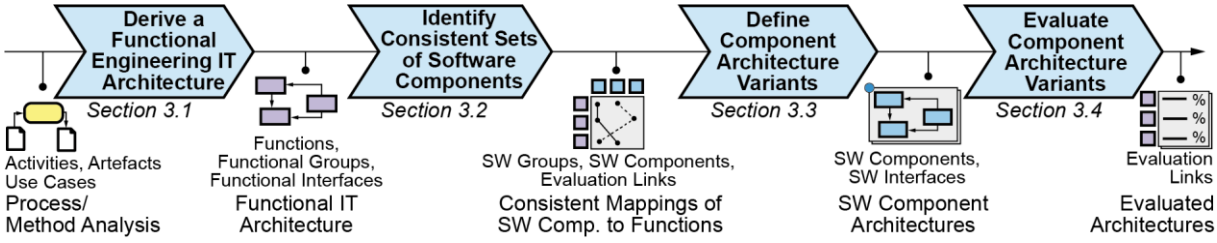

Figure 2. Tasks (middle), major artefacts and model elements (bottom) of the approach

https://doi.org/10.1017/dsd.2020.37 Published online by Cambridge University Press

3.1. Derive a functional engineering IT architecture

To identify potential interfaces and their required functionality, a functional engineering IT architecture is developed. It has to reflect the functionality and quality requirements determined by engineering processes, methods, information models (Gausemeier and Plass, 2014). The first step is to **deduce a functional structure**. The input is based on the approach described in Heihoff-Schwede et al. (2019). It systematically analyses process and method activities to identify engineering artifacts, including their structure and relations as well as Use Cases. The functional structure is deduced based on heuristics as described in Heihoff-Schwede et al. (2019a). The result is a hierarchical structure consisting of Functional Groups describing clusters of Functions with a high internal complexity. On a second level, the Functions define functionality which shall be realized by a tool. The Function must be as granular that it can be assigned to a single tool. They typically act as K.O.-criteria in case they cannot be fulfilled. On a third level, Evaluation Criteria can be subordinated to Functions, describing a desired functionality or quality to be delivered by a tool. This structure allows a systematic step-by-step breakdown based on the analysis models. As the Functions will be part of the functional architecture, the complexity of the architecture can be reduced while having a detailed and structured set of Evaluation Criteria. Figure 3 shows an example of a functional breakdown. This is the input for the following approach of this paper.

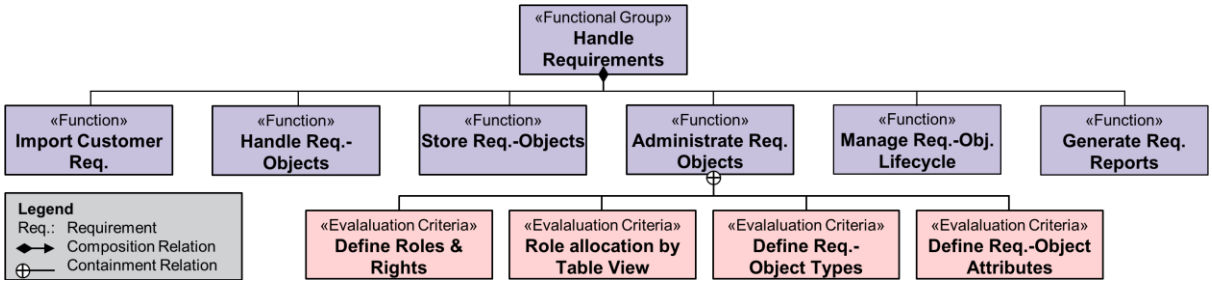

Figure 3. Example of a functional breakdown including Evaluation Criteria

The second step is about to **create a functional engineering IT architecture**. The purpose of the solution-free architecture is to describe required functionalities and their relations. The functional architecture can be used to analyse and specify the interfaces, especially regarding data provided by them. It is built by connecting the Functions within Functional Groups (identified in the first step) by flows within a SysML IBD. Flows exchanging engineering artefacts described in a process or method model can be linked to the flow by using item flows. The artefacts identified in the analysis are systematically analysed for the need of flows. E.g. a non-editable customer spec. needs to be imported and passed to the function to handle native textual requirements. Other data such as control information can be specified textually. The result is a functional architecture as presented in Figure 4.

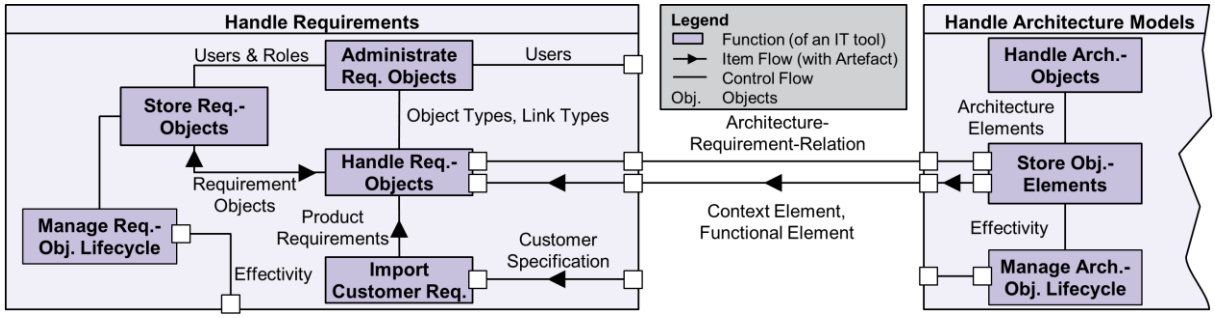

Figure 4. Exemplary functional architecture of the required IT tools

3.2. Identify consistent sets of software components

While the first task described in section 3.1 produced a functional, solution-free architecture, the tasks described in 3.2 and 3.3 aim to identify a logical engineering IT architecture, consisting of deployable software components, which satisfies the required functionality and is consistent in its interfaces.

https://doi.org/10.1017/dsd.2020.37 Published online by Cambridge University Press

The first step targets to **define eventual constraints**, e.g. architecture strategies (federative vs. integrative), legacy systems which need to be preserved or are already set for the architecture.
The second step aims to **identify suitable solutions** which can be allocated to the Functions described in section 3.1. Initially, Software Groups (SG) - clusters of potential software tools (e.g. Requirements Management (RM) tools or PLM solutions) - are identified to define the search space. Software Components which are likely to contribute to the Functions are identified for each Software Group. Software Components are deployable, often commercially available IT applications which can be stand-alone tools (e.g. a SysML Modeler) or parts of a set of applications, e.g. of a PLM solution.
Goal of the third step is to **allocate Software Components to Functions** of the functional architecture by help of a 2-dimensional morphologic box (Zwicky, 1967). Within the model, the morphologic box can be represented by a matrix (e.g. Functions in columns, Software Components in rows). Consistent combinations of IT tools within one variant are expressed by Evaluation Links between Functions and Software Components. The step is repeated to identify different alternative sets of consistent Software Components, resulting in multiple architecture variants. E.g. the function "Import Customer Req." may be allocated to a distinct requirements parsing tool or an RM tool including the function. The Evaluation Links which are not consistent in all considered variations are handled as variation points to allow managing all variants in one model but creating variant specific diagrams based on variant management.

3.3. Define component architecture variants

The component architecture is the logical architecture representing a specific consistent variant. This means the Software Components of a consistent variant are set in relation to each other. The focus is on the interfaces resulting between the various Software Components. The interfaces are defined according to the functional flows which need to be analysed in detail. Since in reality often technical or functional restrictions apply, the interfaces are additionally categorized by attributes which define the availability (e.g. out of the box or customized), the implementation risk and the required initial and maintenance effort. Each interface can be optionally specified by an interface block to define characteristics such as exchanged information or services. This is typically done by IT specialists with in-depth knowledge about the concept of the interface. An example is given in Figure 5, where e.g. the interface between the requirements parser (SW-C3) an RM tool for requirement handling (SW-C4) and a PDM systems backbone for requirements storage (SW-C1) is specified as a Requirements Interchange Format (ReqIF) interface, resulting in Variant 1. Variant 2 only uses components of the PDM system to cover all functions. The component architecture supports the step by step consistency analysis described in section 3.2 and gives important information about the required effort for the realization of the software interfaces. Different baselines of a component variant can be defined to support the migration or road mapping of later implementation.

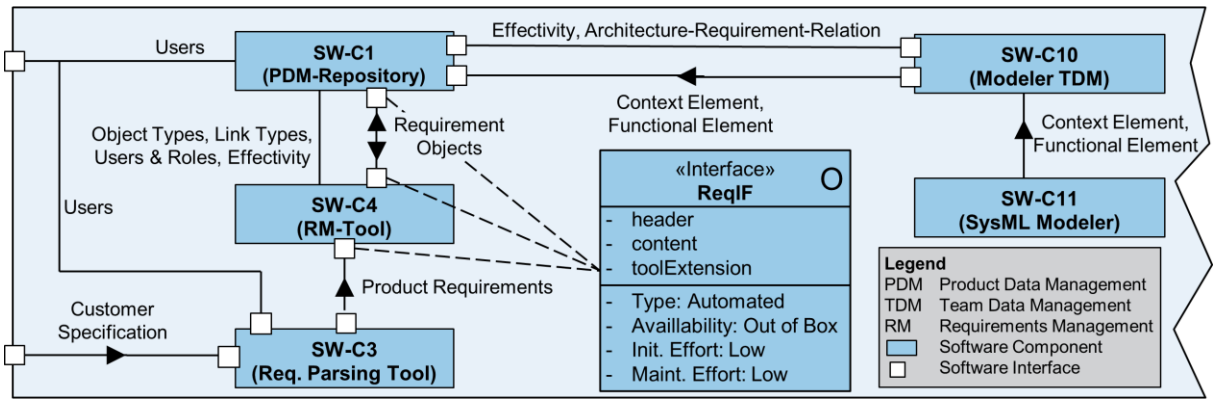

Figure 5. Component architecture Variant 1 with Software Components and Interfaces

3.4. Evaluate IT architecture variants

In a final task the component architecture variants shall be evaluated in detail. Due to the evaluation-effort, only a limited amount of variants can be evaluated in depth. In the previous tasks, incomplete or

https://doi.org/10.1017/dsd.2020.37 Published online by Cambridge University Press

inconsistent component architecture variants should be already excluded or pre-evaluated iteratively to reduce the final set. ISO (2011) defines eight dimensions of software quality: Functional suitability, usability and compatibility which have direct impact on the primary users and performance efficiency, reliability, security, maintainability and portability which can be best assessed by IT specialists. Thus, this paper mainly focusses on the first group. The following steps are performed for each component architecture variant, all with the same set of Functions, Evaluation Criteria and Use Cases.

For the evaluation of **functional suitability** all Evaluation Criteria, Functions and Function Groups are weighted and evaluated with a utility analysis according to Zangemeister (1976). The value is stored as an attribute of the respective elements. Next, the evaluation is performed for each Function. All Evaluation Criteria of a Function are rated on an ordinal scale regarding their fulfilment and corresponding implementation effort. Scale items are: Fulfilled; Parametrization; Extension (migration-capable coding) or Customization (not migration-capable coding). The ordinal values are transformed by a value table to cardinal utility values (e.g. Fulfilled ≙ 100%; Parametrization ≙ 90%). The percentage value can be adjusted according to the implementation capabilities to perform extensions or customizations, e.g. a lower percentage for Customization in case of a small company, where a customization would be disproportionate. The **usability** is evaluated for each Use Case created in the analysis phase. They describe typical and repetitive tool interactions between users and tools to have comparable and defined situations to be evaluated. They comprise a textual description, pre- and postconditions and a trigger. The Use Cases are linked to functions, allowing usability evaluation on a Function level. The evaluation is guided by usability-evaluation criteria (see e.g. ISO, 2006) within the application of established evaluation approaches. A pragmatic approach is e.g. the usage of the User Experience Questionay (UEQ), which uses a 7-point ordinal Likert scale, transformed to a cardinal scale by predefined values (ranging from -3 to +3). The approach aggregates six dimensions (Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation, Novelty), which can be stored as separate values in the model. Finally, each rating for Evaluation Criteria and Use Cases represents a value of a utility value analysis and is stored as a value of the Evaluation Link which is typically a variation point. An example of one component architecture variant is presented in Table 2. The **compatibility** inherently describes a quality between two or more software components (ISO, 2011). Thus, it is evaluated for the overall solution instead of on a Function level. Each connector in the architecture is weighted and rated regarding the expected functionality described in the specification of the connector (Fulfilled; Partially fulfilled; Insufficiently fulfilled - each with respective percentage value to be transformed to a cardinal scale) to extend the connector attributes specified in section 3.3. Each connector can be highlighted by a traffic light colour-scheme to support identification of weak spots in compatibility. A further aspect is financial **cost**, which should include initial cost and on ongoing cost. They can be variant-specific defined for each function and defined by static or dynamic capital budgeting methods. For the overall evaluation, all four categories are weighted and considered, allowing the holistic comparison of the architecture variants. As a result, Variant 2 may have no effort regarding interfaces due to the PDM integration, but lacks functionality and usability while Variant 1 offers better performance in functionality and usability and only little implementation effort to extend the existing ReqIF interface of the Software Components.

Table 2. Exemplary results of functional evaluation based on evaluation criteria (left) and usability evaluation based on use cases (right) for one architecture variant

V1 Functional Evaluation for Architecture Variant 1	Relev. [%]	Tool	Rating [%]
Handle Requirements	10		87,5
Administrate Req.-Objects	30		75
Define Roles & Rights	25	SW-C1	100
Role allocation by Table View	25	SW-C1	100
Define Req.-Object Types	25	SW-C1	50
Define Req.-Object Attributes	25	SW-C1	50
…	…		…

V1 Usability Evaluation for Architecture Variant 1	Relev. [%]	Rating [-3 to 3]
Handle Requirements	10	2,1
Administrate Req.-Objects	30	1,35
Create new User	25	2
Create new Role	20	2,2
Assign role to User	25	1,2
Search User	15	-0,3
…	…	…

https://doi.org/10.1017/dsd.2020.37 Published online by Cambridge University Press

4. Conclusion & outlook

The approach delivers a procedure, methodology and modelling support for the systematic definition of consistent engineering IT architecture variants as well as their evaluation. This comprises a functional architecture which describes required functionality and prospective tool interfaces. It acts as the foundation to define logical architecture variants which are evaluated regarding functionality, usability, and holistic compatibility within an integrated model.

The Validation was performed according to Pedersen et al. (2000). *Structural Validation* is mainly given by using well-known constructs such as OMG SysML, morphologic box and utility analysis. Method consistency is demonstrated by in- and outputs for each task (see Figure 2) which are likely suitable to generated the expected results. The approach has been validated in industrial application from which the example problem was derived. The example problem is based on a subset of processes of an SE standard (ISO, 2011). The goal was to develop a suitable Engineering IT Architecture to support the example processes and methods. This is typical in practice if method/process are modified or tools need to be changed or introduced which are major cases the presented approach is designed for. Regarding *performance validation*, the approached proved to be useful to deliver more detailed and higher completeness regarding analysis results (activities and artefacts). TOGAF in contrast only identified major artefacts, which were not systematically broken down based on engineering methodologies. Furthermore, the presented approach delivers a wider range of suitable alternatives of IT architectures by a systematic discovery of the solution space and consistency assurance by interface descriptions and analysis compared to approaches without dedicated methodology to derive architectures such as Friedland et al. (2016). A functional analysis in combination with a morphology and conceptual interface descriptions fulfils need N1. Atomic Evaluation Criteria with Functions as a stable and unified structure, even for different architectures, allow comparability and satisfy N2 in combination with the UEQ-based evaluation of usability. This allows more transparency in contrast to aggregated rating items such as used in the evaluation approach according to Stark (2015) or Friedland et al. (2017). A modelling language provides syntax and semantics to allow an integrated model covering all aspects from analysis to evaluation (fulfilling N3). None of the examined approaches fully supported the required model elements and concepts, e.g. for rating evaluations. N4 is fulfilled by a procedure and modelling guidance. All stated needs are fulfilled by the presented approach. We expect usefulness of the approach beyond the example problem, which will be validated more detailed in further publications.

Acknowledgements

This research was funded by the German Federal Ministry of Education and Research (BMBF) and European Social Fund (ESF) in the project IviPep, grant number 02L15A120. The contents of this publication are the sole responsibility of the authors.

References

Abramovici, M. (2018), Engineering smarter Produkte und Services, Plattform Industrie 4.0 STUDIE, acatech - Deutsche Akademie der Technikwissenschaften, Munich, Germany.

Alvarez-Rodríguez, J.M., Zuñiga, R.M. and Llorens, J. (2019), "Elevating the meaning of data and operations within the development lifecycle through an interoperable toolchain", *INCOSE International Symposium*, Vol. 29 No. 1, pp. 1053-1071. https://doi.org/10.1002/j.2334-5837.2019.00652.x

Barth, M. et al. (2012), Evaluation of the openness of automation tools for interoperability in engineering tool chains, *IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA)*, IEEE, Piscataway, NJ, USA. https://doi.org/10.1109/ETFA.2012.6489542

Bauer, W., Herkommer, O. and Schlund, S. (2015), "Die Digitalisierung der Wertschöpfung kommt in deutschen Unternehmen an: Industrie 4.0 wird unsere Arbeit verändern", *Zeitschrift für wirtschaftlichen Fabrikbetrieb*, Vol. 110 No. 1-2, pp. 68-73. https://doi.org/10.3139/104.111273

Becker, J., Rosemann, M. and Schütte, R. (1995), "Grundsätze ordnungsmäßiger Modellierung", *Wirtschafts-informatik*, Vol. 37 No. 5, pp. 435-445.

Fisher, A. et al. (2014), "Model Lifecycle Management for MBSE", *INCOSE International Symposium, Las Vegas, NV, USA, June 30-July 3, 2014*, INCOSE, San Diego, Vol. 24, pp. 207-229. https://doi.org/10.1002/j.2334-5837.2014.tb03145.x

https://doi.org/10.1017/dsd.2020.37 Published online by Cambridge University Press

Friedenthal, S., Moore, A. and Steiner, R. (2012), *A Practical Guide to SysML - The Systems Modeling Language*, Morgan Kaufmann, Boston. https://doi.org/10.1016/C2010-0-66331-0
Friedland, B. et al. (2017), "Conducting a Model Based Systems Engineering Tool Trade Study Using a Systems Engineering Approach", *INCOSE International Symposium*, Vol. 27 No. 1, pp. 1087-1099. https://doi.org/10.1002/j.2334-5837.2017.00414.x
Friedland, B., Malone, R. and Herrold, J. (2016), "Systems Engineering a Model Based Systems Engineering Tool Suite: The Boeing Approach", *INCOSE International Symposium*, Vol. 26 No. 1, pp. 386-398. https://doi.org/10.1002/j.2334-5837.2016.00167.x
Gausemeier, J. et al. (2013), *Systems Engineering in industrial practice*, Heinz Nixdorf Institut, University of Paderborn, Paderborn, Germany.
Gausemeier, J. and Plass, C. (2014), *Zukunftsorientierte Unternehmensgestaltung*, Carl Hanser, München. https://doi.org/10.3139/9783446438422
Graner, M. (2013), *Der Einsatz von Methoden in Produktentwicklungsprojekten: Eine empirische Untersuchung der Rahmenbedingungen und Auswirkungen*, Springer Gabler, Wiesbaden. https://doi.org/10.1007/978-3-658-01278-6
Heihoff-Schwede, J., Kaiser, L. and Dumitrescu, R. (2019a), "An MBSE-based approach for the functional analysis of engineering IT architectures", *Tag des Systems Engineering, München, 6.-8. November, 2019*, Gesellschaft für Systems Engineering, Bremen
Heihoff-Schwede, J., Kaiser, L. and Dumitrescu, R. (2019), "An MBSE-based approach for the analysis of requirements towards engineering IT architectures", *Proceedings of the 5th Annual IEEE International Symposium on Systems Engineering (ISSE), Edinburgh, Scottland, UK, October 1-3, 2019*. https://doi.org/10.1109/ISSE46696.2019.8984568
Hoffmann, J. and Heimes, P. (2018), "Informationssystem-Architekturen produzierender Unternehmen für die Digitalisierung gestalten", *HMD Praxis der Wirtschaftsinformatik*, Vol. 55 No. 5, pp. 984-1005. https://doi.org/10.1365/s40702-018-0440-8
Holligan, C., Hargaden, V. and Papakostas, N. (2017), "Product lifecycle management and digital manufacturing technologies in the era of cloud computing", *2017 International Conference on Engineering, Technology and Innovation, Madeira, Portugal, June 27-29, 2017*, IEEE. https://doi.org/10.1109/ICE.2017.8279980
Huth, T. et al. (2018), "Model-based Process Engineering - An approach to integrated product system and process modelling", *INCOSE EMEASEC/GfSE TdSE 2018, Berlin, Germany, November 5-7, 2018*.
IEC (2010), IEC61508: Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements, International Electrotechnical Commission, Geneva.
ISO (2006), ISO 9241-110:2006: Ergonomics of human-system interaction - Part 110: Dialogue principles, International Organization for Standardization, Geneva.
ISO (2011), ISO/IEC 25010:2011: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, International Organization for Standardization, Geneva.
ISO (2015), ISO/IEC/IEEE 15288:2015 International Standard - Systems and software engineering - System life cycle processes, International Organization for Standardization, Geneva.
Lu, J. (2019), *A Framework for Cyber-physical System Tool-chain Development: A Service-oriented and Model-based Systems Engineering Approach, [PhD Thesis]*, KTH Royal Institute of Technology.
Pedersen, K. et al. (2000), "Validating Design Methods & Research: The Validation Square", *Proceedings of DETC '00, 2000 ASME Design Engineering Technical Conferences, September 10-14, 2000, Baltimore, MD*, American Society of Mechanical Engineers, New York.
Schütte, R. and Vering, O. (2011), *Erfolgreiche Geschäftsprozesse durch moderne Warenwirtschaftssysteme - Produktübersicht marktführender Systeme und Auswahlprozess*, Springer Verlag, Berlin. https://doi.org/10.1007/978-3-642-20523-1
Stark, J. (2015), *Product Lifecycle Management (Volume 1) - 21st Century Paradigm for Product Realisation*, Springer, Cham. https://doi.org/10.1007/978-3-319-17440-2
The Open Group (2018), The Open Group Architecture Framework - Version 9.2.
Törngren, M. and Grogan, P.T. (2018), "How to Deal with the Complexity of Future Cyber-Physical Systems?", *Designs*, Vol. 2 No. 4. https://doi.org/10.3390/designs2040040
Zangemeister, C. (1976), *Nutzwertanalyse in der Systemtechnik*, Wittemann, Munich.
Zwicky, F. (1969), *Discovery, invention, research through the morphological approach*, Macmillan, New York. https://doi.org/10.1126/science.163.3873.1317