3-5-2015

Understanding the Elusive Black Box of Artifact Mutability

Jens Pöppelbuß

Matthias Goeken

Understanding the Elusive Black Box of Artifact Mutability

Jens Pöppelbuß[1] and Matthias Goeken[2]

[1] University of Bremen, Bremen, Germany
jens.poeppelbuss@uni-bremen.de
[2] University of Applied Sciences of Deutsche Bundesbank, Hachenburg, Germany
matthias.goeken@bundesbank.de

Abstract. Statements on artifact mutability are considered a core component of design theories, but the understanding of this phenomenon is fragmented and limited. To mitigate this issue, we build a framework of artifact mutability that is structured into six generic dimensions: paradigmatic perspective, intentionality, drivers, scope, temporality, and artifact layers. We review existing design theories in the light of these dimensions. With this paper, we show that our current knowledge about artifact mutability is still disconnected and mostly linked to specific artifact types. We are also able to characterize artifact mutability as a multi-facetted topic that has found little attention in existing articles that propose design theories. From a theoretical perspective, we advance the understanding of this phenomenon. From a practical perspective, the framework is expected to help tackling mutability in the design and adaptation of IT artifacts.

Keywords: Mutability, Artifact, Framework, Design Theory, Design Science.

1 Introduction

IT artifacts exhibit mutability [1] as they usually cannot be used "out of the box" ([2], p. 48) in specific organizational contexts and evolve over time while being used. In Information Systems (IS) research, the phenomenon of artifact mutability has received increasing attention through the proposition of the anatomy of design theory by Gregor and Jones in 2007 [3]. This anatomy defines artifact mutability as one of its core components and thus explicitly urges IS researchers to make statements about artifact mutability when developing new design theories.

However, looking at design theories published in recent research articles suggests that the understanding of artifact mutability is still fragmented and limited. On the one hand, design-oriented researchers have developed approaches to deal with artifact mutability in various contexts already for a long time, especially in the fields of conceptual modeling [4], method engineering [5], and software engineering [6]. These different existing research streams have progressed independently from each other and have rarely been connected to design theorizing yet. On the other hand, research from a more behavioral perspective has just recently begun to investigate the role of

12th International Conference on Wirtschaftsinformatik,
March 4-6 2015, Osnabrück, Germany

Pöppelbuß, J.; Goeken, M. (2015): Understanding the Elusive Black Box of Artifact Mutability, in: Thomas. O.; Teuteberg, F. (Hrsg.): Proceedings der 12. Internationalen Tagung Wirtschaftsinformatik (WI 2015), Osnabrück, S. 1557-1571

users in tinkering and tailoring IT artifacts in use [7–9]. Correspondingly, Gregor and Jones emphasize that "the ways in which IT artifacts emerge and evolve over time […] are key unresolved issues for our field and ones that will become even more problematic in these dynamic and innovative times." ([3], p. 326).

As to now, artifact mutability remains a vague and elusive concept that is not well understood [10] and needs to be defined more clearly. It currently serves as a black box that encompasses a broad range of phenomena related to the design and use of various artifacts. Even the term artifact alone can refer to very different things [11], rendering the term artifact mutability even more unclear. Furthermore, it is criticized that existing guidelines for design science research are not sufficient for devising robust design theories that account for artifact mutability [7]. Although artifact mutability is considered a core component of a design theory, there are no established guidelines how to flesh this out. In this regard, Sjöström et al. [10] recommend that IS researchers should combine design theorizing with existing knowledge from fields like software engineering where many works already promote artifact mutability.

Against this backdrop, we inductively develop a framework of artifact mutability that is grounded in existing knowledge on the mutability of four specific artifact types including constructs, models, methods, and instantiations. We further apply this framework in a review of articles that propose design theories. This paper contributes to a better understanding of artifact mutability that will hopefully help researchers in preparing the statements about this phenomenon in their design theories.

In the remainder, we discuss the background of design theory and artifact mutability (2). After presenting our research approach (3), we gather artifact type-specific dimensions of mutability and consolidate them into a generic framework (4). We provide a first applicability check of this framework by using it as the analytical lens in our literature review (5). The paper closes with a discussion and conclusions (6).

2 Theoretical Background

Since its publication in 2007, "The Anatomy of a Design Theory" by Gregor and Jones [3] has gained much attention and can be considered as the de-facto blueprint for design theories. A design theory "gives explicit prescriptions (e.g., methods, techniques, principles of form and function) for constructing an artifact" ([12], p. 620), based on knowledge of both IT and human behavior [3]. The anatomy defines eight components of a design theory, including six core components (purpose and scope, constructs, principles of form and function, artifact mutability, testable propositions, justificatory knowledge) and two additional ones (principles of implementation and expository instantiation). *Artifact mutability* is one of the core components and is defined as the "changes in state of the artifact anticipated in the theory, that is, what degree of artifact change is encompassed by the [design] theory." ([3], p. 322) Including artifact mutability into design theorizing was a novel aspect of their anatomy [3].

To better reflect the dynamic nature of artifacts, Gregor and Iivari [1] introduce the new term *semizoa*. They conceptualize semizoa as "mutable systems that exhibit some of the characteristics of living creatures and that are only in part designable."

([1], p. 3) They further distinguish between eight different degrees of mutability that semizoa as a design product can exhibit, from rather stable nilpotent systems to highly mutable systems that can be easily redesigned (see p. 17 in [1] for an overview).

In existing literature, we further find different perspectives on artifact mutability. A key distinction is that between artifact mutability as the purposeful design of adaptable artifacts (in-design) and the evolution of artifacts (in-use) over time [10]. As for the former, mutability-in-design refers to the design ideal of mutable and flexible artifacts that are adaptable to various organizational contexts. As for the latter, mutability-in-use reflects that IT artifacts are typically not immutable end results of design processes, but inherently dynamic [1]. Here, the term *secondary design* can be used to describe changes "where functions and content emerge during interaction, modification, and embodiment of the system in use" ([8], p. 662). As such, artifact mutability refers to the behavior of the artefact when implemented in a specific context [13].

According to Sjöström et al. [10], both mutability-in-design and mutability-in-use can be subject to either a process or a product view. The process view focuses on how designers design and users appropriate the artifact. The product view focuses on the artifact as the result of these processes which exhibits, e.g., a flexible software architecture developed by the designers or features that can be configured by the users.

3 Research Approach

In this study, we follow an inductive research approach that comprises the following phases (Fig. 1). First, we search for existing works from various fields for potential dimensions of artifact mutability. We structure our search according to the four specific types of IT artifacts that are commonly distinguished as outputs of design science research in the IS discipline (constructs, models, methods, and instantiations [14, 15]). By focusing on the specific artifact types we avoid the vagueness of IT artifacts in general and bring together so far disconnected research streams. The first phase results in a set of 19 IT artifact type-specific dimensions that provide the grounding for the following phase. Second, we consolidate this set into an artifact type-generic framework. We do this by collating the specific dimensions in search for similarities, resulting in six generic dimensions. Third, we apply the consolidated framework in a review of existing design theories. This review serves as an applicability check for the proposed framework and also points to the merits and shortcomings of existing design theories with regard to the core component of artifact mutability.

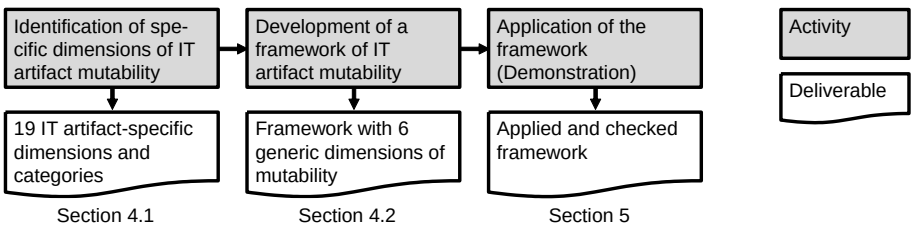

Fig. 1. Research Approach

4 Developing a Framework for IT Artifact Mutability

4.1 Identifying specific dimensions of IT artifact mutability from related work

In the following, the four artifact types of constructs, models, methods, and instantiations serve as a structure for our search for possible artifact type-specific dimensions of mutability. The resulting set of dimensions is summarized in Table 1.

Construct Mutability. *Constructs* form the vocabulary or the set of symbols of a domain. They build the basis for defining problems and specifying solutions [15]. In the IS discipline, constructs are mainly designed in the form of modeling notations, but also glossaries, taxonomies, and ontologies. The *mutability of constructs* can be observed based on modeling notations evolving over time. For example, the Business Process Model and Notation (BPMN) has gone through some evolutionary steps since its initial publication, the latest version being 2.0 published in 2011. At the same time, many of the constructs offered by such a semantically rich notation like BPMN are hardly used in practice [16]. Organizations and individual users obviously adapt and trim available modeling notations by only using a subset of constructs that appears useful. Hence, we identify the drivers of mutability as one dimension of construct mutability. Precisely, we distinguish between developer-driven, user-driven, and intermediator-driven mutability (D01 in Table 1). In the first case, a developer (this can be individuals or institutions) of a notation effectuates a change due to deficiencies or enhancement needs. In the second case, users modify (sets of) constructs according to their needs or preferences. In the third case, a tool developer that implements a notation that deviates from the original specification or an instructor that teaches a modeling notation differently from the original specification can be intermediators that also lead to a mutation of constructs. Furthermore, so called meta-models can be used to give a formalization of the constructs of a modeling notation, e.g., an Entity-Relationship-Model (ERM) that describes the elements of the Event-Driven Process Chain (EPC) used for process modeling [4]. Accordingly, modeling notations can take two different roles in artifact mutability, being a subject of mutability or a supporting tool for documenting and /or achieving mutability (D02 in Table 1).

Model Mutability. *Models* are sets of statements expressing relationships between constructs [15]. They are abstractions and representations that help describe as-is and to-be states. Business process models, for instance, can be used for analyzing pitfalls in current activities and for devising improved ways of operations. *Model mutability* has been a key interest to researchers in the field of reference models. Reference models are conceptual models that are developed with the intention of being reused for different, but similar purposes. They provide best or common practices and thus frequently serve as a starting point and blueprint for the creation of organization- or project-specific models. The adaptation of reference models to specific application contexts, e.g., in terms of business characteristics and user groups, can be supported by configuration and adaptation mechanisms [4] (D03). Configuration mechanisms rely on predefined configuration points that are already included in the reference model. They reduce a total model containing information for all application contexts by selecting only the information that is actually needed for the specific context [4].

In contrast to configuration mechanisms, adaption mechanisms do not tailor a total model but allow for creative freedom (D04 in Table 1) in the adaptation process [4, 17]. Such mechanisms can be applied to different modeling layers [4], i.e., the model layer, the meta model layer and the meta-meta model layer (D05). Apart from reference models, we also find discussions on model mutability in the field of model-driven software engineering. Wenzel and Kelter, for instance, see a key challenge of *model evolution* in tracing the changes that happen to model elements during the software development process [18]. In database management, *schema evolution* has also been subject of a long discussion [19]. Schema evolution refers to the problem of evolving a database schema in response to changes in the modeled reality. Approaches to schema evolution have frequently been transferred to *ontology evolution*, which is defined as the "timely adaptation of an ontology and a consistent propagation of changes to the dependent artifacts." ([20], p. 12). From the existing works we can also derive different directions of action (D06): In a proactive manner, the model is the blueprint for change (model change is supposed to have an impact on reality, e.g., people work according to a revised process model), whereas a reactive change is given if the model change follows a change in the modeled reality. Noy and Klein [19] further distinguish between traced (where the series of changes is known) and untraced ontology evolution (where two versions of an ontology exist without any knowledge about intermediate steps) (D07 in Table 1).

Method Mutability. *Methods* are sequences of steps used to perform a task [15]. Typical examples are algorithms, procedures, or guidelines. The field of method engineering (ME) has emerged since the early 1990s with the intention to design and adapt methods for systems development. As it is unlikely that a method designed at a particular time will fit all future circumstances, this stream of research has been expanded by ideas of situational method engineering (SME) in the last decade [5]. As for SME, Bucher et al. [21] distinguish between *situational method configuration* and *situational method composition* (D08). In the former case, situation-specific changes have to be foreseen and planned when a situational method is developed. In the latter case, spontaneous and more liberal combination of method parts or fragments (orchestration) happen that are not foreseen at design-time [21]. Henderson-Sellers et al. [5] further distinguish between two ways of engineering a method: *creating a method ab initio* (starting with a set of method parts), and *method tailoring* (modifying an existing method). The former contains the mechanisms configuration and composition as discussed above [5]. Furthermore, mutability can either happen on the level of method design (i.e., the artifact described in a design theory) or on the instance of a method (i.e., the "method in action"), used by a particular group of people [22]. Summarizing these aspects, we identify three levels of method mutability: mutability defined at the meta-level of a method (through configurations or compositions), mutation which takes place at the level of the method (especially through tailoring), and mutation at the level of the method instance ("method in action"; D09). A further related aspect is raised by Börner et al. [23] who describe the concept of an "emergent method". They consider a method as emergent if a pattern develops in the absence of intentions or even despite them. This is contrasted with a predetermined method, i.e., a "settled/statutory method" (D10)

Instantiation Mutability. *Instantiations* are realizations of constructs, models, or methods [15]. They are valuable for demonstrating the utility of artifacts of other types (e.g., methods) and the general feasibility of their implementation in software. The mutability of instantiations has been discussed in Computer Science under the terms software product lines and software evolution. The basic idea of *software product lines* (SPL) is that software can be constructed from reusable parts that allow for variability [24]. A software product line contains a certain amount of variation points and preplanned mechanisms allowing for (intended) mutability. The following variability dimensions are observable from the literature on SPL (D11-D15):

- *Locus of variability:* Bühne et al. [25] distinguish different loci, e.g., variability in *functionality* (basic vs. extended functions) or variability in *processes* (variations in processes due to external factors leading to the same result).
- *Direction of accomplishment: Negative* variability means that features are masked or removed. *Positive* variability describes the selection of features [26].
- *Focus of variability:* While *essential* variability is related to the requirement of the client, *technical* variability is what occurs in the process of realizing it [27].
- *Phases of the SPL development process:* Variability can be found in different phases, including *domain analysis*, *domain design*, and *domain implementation* [28].
- *Binding time:* This dimension describes the point in time when the variability is bound by selecting an appropriate variant [29].

In addition, research on *software evolution* investigates the long-term changes and transformations of a software product throughout its life cycle. The basic assumption is that applications cannot be implemented once and for all. Instead, new releases are deployed resulting from maintenance activities. The following dimensions (D16-D19) are observable from the literature on software evolution:

- *Viewpoints:* The descriptive viewpoint represents a "nounal view" focusing on understanding the evolution of software. The design-oriented viewpoint takes a "verbal view" and deals with the means to direct and control software evolution [6, 30, 31].
- *Categories of maintenance:* Mens [31] distinguishes between perfection maintenance (modification to improve performance or maintainability), corrective maintenance (reactive modification to correct discovered faults), adaptive maintenance (modification to keep software usable in changing environments), and preventive maintenance (modification for preventing problems before they occur).
- *Software type:* Lehman and Ramil [6] distinguish between different software types, including socially embedded systems (E-type programs, "E" stands for "evolving") and unambiguously specified systems (S-type programs, "S" stands for "specified").
- *Areas of evolution:* Lehman and Ramil [6] furthermore distinguish between *software ab initio implementation* (e.g., incorporation of user feedback during software development), s*oftware system evolution* (new versions, releases, and upgrades during run-time), *evolution of the application in its domain* (co-evolution with the contextual processes and domains), *software process evolution* (related to devel-

opment processes, methods, software paradigms, and technologies), and *process model evolution* (models describing development processes are also subject of evolution).

Table 1. IT Artifact-Specific Mutability Dimensions

	ID	*Dimension*	*Categories*						
constructs	**D01**	**Drivers of mutability**	Developer-driven	Intermediator-driven mutability	User-driven				
	D02	**Role of notation**	Mutability of the notation itself	Mutability utilizing a notation					
model	**D03**	**Mechanism type**	Configuration	Adaptation					
	D04	**Degrees of freedom**	Low: Selecting components from a predefined total model	High: Creative modifications, extensions, revisions, and tailoring					
	D05	**Model layer**	Model layer	Meta model layer	Meta-meta model layer				
	D06	**Direction of action**	Proactive and normative	Reactive and descriptive					
	D07	**Mutability trace**	Traced	Untraced					
method	**D08**	**Ways to engineer a method**	Creating a method ab initio (configuration / composition)	Method tailoring					
	D09	**Level of method mutability**	Meta level of method design	Level of method design	Instance level of a method				
	D10	**Origin of method**	Settled/statutory method	Emergent method					
instantiation	**D11**	**Locus of variability**	Functionality	Processes	Quality	Data formats	User interfaces	Information	…
	D12	**Direction of accomplishment**	Negative variability	Positive variability					
	D13	**Focus of variability**	Essential variability	Technical variability					
	D14	**Phases of SPL development**	Variability in domain analysis	Variability in domain design	Variability in domain implementation				
	D15	**Binding time of variability**	Build time	Compile time	Startup time	Run time			
	D16	**Viewpoints on software evolution**	Nounal view	Verbal view					
	D17	**Categories of maintenance**	Perfection	Corrective	Adaptive	Preventive			
	D18	**Software type**	S-type programs (non-mutable)	E-type programs (mutable)					
	D19	**Scope/areas of evolution**	Software ab initio implementation	Software system evolution	Evolution of the application in its domain	Software process evolution	Process model evolution		

4.2 Consolidation of Dimensions into a Generic Framework

In what follows, we develop a generic framework of artifact mutability that is grounded in the previously gathered artifact type-specific dimensions. We generalize them into a set of generic dimensions for artifact mutability. We do this by referring to the specific dimensions (D01 to D19) and further related literature. Table 2 gives an overview of the resulting six generic dimensions of our framework.

Table 2. Generic Mutability Dimensions

Mutability Dimensions	*Reference to Artifact Type-specific Dimensions*	*Reference to Further Literature*
Paradigmatic perspective	D06, D07, D16, D03, D04, D08. D12, D17	[13]
Intentionality	D01, D07, D10, D17	[1, 7, 9]
Scope	D11, D18, D19	
Drivers	D01	[32]
Temporality	D15	[10]
Artifact layers	D02, D05, D09, D14	[1]

First, we identify two *paradigmatic perspectives* to deal with artifact mutability (D16): One perspective is describing and explaining (and possibly predicting) artifact mutability as a phenomenon in the sense of behavioral research. This perspective also covers the identification of successful strategies to deal with mutability, e.g., whether to tackle it proactively or reactively (D06) and whether to trace it systematically or not (D07). The other perspective is to focus on mutability as a design objective, e.g., by developing adaptation and configuration mechanisms to better cope with the phenomenon and, in doing so, to manage mutability (D03, D04, D08. D12, and D17).

Second, we identify the dimension of *intentionality*, which is grounded in D01, D07, D10, D17, and [9]. On the one hand, scholars argue that there are *unintended* mutations of an artifact caused by "tinkering and secondary design" and "bricolage" [7]. D01 and D10 point out that users in particular can be drivers of unintended (or at least unplanned) artifact mutability. On the other hand, mutability can be *intended* and be designed into an artifact to some degree [9]. Correspondingly, IS and Computer Science researchers have been continuously developing mechanisms for artifact maintenance, configuration, composition, adaption, and tailoring in order to better manage and control mutability (see above, D03, D04, D08. D12, and D17). The intentionality dimension is related to traceability (D07), supposing that unintended mutations of artifacts are often also untraced. In addition, there is a relation between intentionality and the origin of method (D10) or the origin of artifacts in general.

Third, we identify the dimension of *scope*. D11 exhibits that mutability can refer to various objects and aspects of an artifact (functions, quality, processes). The broader conceptualization of Lehman and Ramil [6] (see D18 and D19) also illustrates the various areas of mutability, incorporating entities ranging from users, technologies, task, domain, and context to development processes and process models.

Fourth, we identify the dimension of *drivers*. Based on D01 and [32], the previously mentioned entities (e.g., users and technology) not only define the scope, but can also be seen as relevant drivers of mutability.

Fifth, we identify the dimension of *temporality*. D15 distinguishes different points in time where evolution can take place and where variability is bound to a specific variant. Different points in time with respect to mutability can also be found in [10].

Finally, we identify the dimension of *artifact layers***.** Generally, the notion of layers seem applicable to different types of artifacts (e.g., constructs (D02), models (D05), methods (D09), or instantiations (D14)). Similarly, Gregor and Iivari [1] discuss artifact mutability in the light of two layers: changes to the IS structure/schema and changes to the IS state.

5 Review of Design Theories

In this section, we review design theories that make statements concerning artifact mutability. To identify relevant academic articles, we conducted a structured literature search with the help of Google Scholar, ISI Web of Knowledge, and EBSCOhost (using the Business Source Premier database). First, we selected the paper by Gregor and Jones [3] as the starting point for our search in all three databases as it represents the de-facto blueprint for design theories and we assume that papers proposing a design theory within IS are very likely to cite it. The paper has been cited 702/161/3 times according to Google Scholar/ISI Web of Knowledge/EBSCOhost (as of 2014-11-09). As we were particularly interested in papers that not only propose design theories but also make statements regarding artifact mutability, we searched for the terms "artifact mutability" and "artefact mutability" (to cover both British and American English) within the set of 702/161/3 manuscripts. Unfortunately, EBSCOhost does not offer a dedicated feature for this and analyzing the titles and abstracts of the three citing papers yielded no article that proposes a design theory at all. In ISI Web of Knowledge, the "Refine Results" feature using the two search terms did not return any results as it does not support a fulltext search of the citing articles. For our further search for relevant articles, we therefore relied on Google Scholar only as we assume that it is very likely that their 702 hits also cover the 161/3 hits of the other databases. Searching the 702 citing articles for "artifact mutability" and "artefact mutability" led us to 109 and 37 hits. We scanned the total of 146 hits for articles that present design theories (or at least novel artifacts) and discuss artifact mutability. We excluded theses (Master/PhD) and book chapters to guarantee that the papers have undergone a review process. As a result, we got a short list of 8 journal and 12 conference papers. We categorized them according to the different IT artifact types. One paper each proposes a set of constructs and a model. With ten manuscripts, design theories for methods are presented most often. We found eight papers on instantiations (Table 3).

All of the 20 manuscripts address artifact mutability explicitly. Many only devote one sentence or a bullet point (e.g., [33, 34]), while a few others give a complete paragraph on this component (e.g., [32, 35–37]). The one paper that presents *constructs* (a technique for modelling intents for strategic alignment) does not discuss mutability

in detail. The authors refer to it as the evolution and maturation of models that can be created with the proposed technique over time [38]. The one paper that presents a *model* (a template for collecting teaching cases) points to specific sections of the template accounting for mutability. Unfortunately, this is not discussed any further [39].

Table 3. Categorization of Artifacts

Artifact Type	#	*Artifact Descriptions*
Constructs	1	• Reference architecture of enterprise intent (vocabulary, rules, and structure) [38]
Model	1	• Teaching case collection template [39]
Method	10	• Method framework and models for context-aware process design [33] • Guidebook on managing IS integration [40] • End-to-end demand management process [32] • Prescriptive knowledge for guiding organizations to implement Enterprise Architecture Management [35] • Evaluation guidelines for futures research [41] • Framework for performing an IS analysis [42] • Approach to analyzing and designing business processes [43] • Customer satisfaction-oriented IT vendor management [44] • Multi-ontology topology of the strategic landscape [36] • Technological process of creating business software based on core functionality known from web 2.0 [45]
Instantiation	8	• Educational on-line information security laboratories [46] • Health care quality registers [47] • application for mobile devices [48] • IT systems that support convergent and divergent thinking [37] • Collaborative systems [34] • Two-factor authentication system [49] • System that monitor and manages vehicles [50] • Sales configurator [51]

As for *methods*, we find many manuscripts that point to a fit between the method and its application context. One paper presents three variants of a process that may be applicable in varying contexts [32]. Another paper identifies four factors that can lead to an adaptation of the proposed method, including the organizational setting, the type of process under consideration, the degree of formalization of the respective process description, and organizational change [35]. [44] mention that varying goals, organizational routines, and governance structures may have an impact on the implementation of their design theory. [42] also mention that their framework must be adapted to fit the environment. [43] state that there are both generic guidelines relevant for several kinds of processes and domain-specific ones with a limited applicability. Some further authors point to the generic capability of their methods: [36] write that the method that they initially designed and tested for businesses is equally applicable in public organizations, while [45] write that paying respect to special needs is inherent to the method itself. Moreover, we find one reference to suggestions for improvement

from users as a source for method mutability [40]. Finally, [41] refer to artifact mutability as a matter of future research as their framework has not been applied yet.

As for *instantiations*, suggestions for improvements from (prospective) users during build-time and run-time are frequently mentioned as a source of artifact mutability [46–48]. [49] write that trial runs of their system may lead to refinements. Some manuscripts also emphasize a context-dependence when implementing an instantiation that follows the proposed design theory (e.g., [37]). [51] describe the "built-in flexibility" (p. 72) of their sales configurator that is achieved through internal interface that allow for the replacement of major software components. Dealing with collaborative technologies, [34] write that system features will change as teams evolve.

Analyzing the mutability of different artifact types based on the six generic dimensions, we can ascertain the following (as information given by the papers on constructs and models is limited, we focus on mutability of instantiations and methods):

Paradigmatic perspective: We see that mutability of an *instantiation* is mainly described as something that happens during build-time (e.g., due to user feedback [40, 46–48, 50]) or will happen during run-time (e.g., future trial runs [49]). However, it seems not to be proactively managed, rather, it is observed, documented, and reacted upon (except for [51] who refer to system flexibility). In contrast, mutability of *methods* for varying contexts is a feature that is explicitly addressed in the design process.

Intentionality: We cannot identify any reference to unintended changes in our sample. The mutability of *instantiations* in response to user feedback is generally presented as a positive and accepted phenomenon in our set of papers. We also see that mutability is purposefully designed into artifacts, especially in *methods* to make them applicable in different contexts (e.g., [32]).

Scope: The scope of mutability is frequently not defined clearly. Authors mostly point to suggestions for improvements (esp. for *instantiations*), but do not limit these to specific aspects or components of the artifact. An exception is the design theory for systems that support convergent and divergent thinking [37], where the authors clearly point to some core constructs (i.e., different data sources, tags, and tag trees for retrieving data) that are expected to vary depending on different contexts. Two papers mention specific mutability points for *methods*, i.e., selection of different random distributions for calculations [42] or the definition of specific guidelines [43].

Drivers: Many papers mention possible drivers of mutability. In our sample we see that multiple factors of the organizational context can trigger mutability. Possible factors comprise industry (e.g., business vs. public administration, [36]), organizational setting and complexity [32], goals, organizational routines, and governance structures [32, 44]). Context is also considered as something dynamic, i.e., it also evolves over time [34, 35] leading to the mutation of artifacts as a result.

Temporality: We see that *instantiations* are mutable in build-time (e.g., during an iterative development process) and run-time (e.g., through modifications based on user feedback). Mutability of *methods* seems to be rather a matter of startup-time, if we transfer this term from SPL (see also D15). Methods are typically selected and tailored at the beginning of a project in response to the project-specific context.

Artifact layers: We do not find explicit discussions in our sample that mutability can happen on different layers. However, we see that complex artifacts comprising

both constructs and models [38] naturally address multiple layers, i.e., a model and a meta-model layer. The corresponding paper, however, points to the mutability of models only, as these mature in a specific organization over time [38]. As for the template proposed by [39], this is not expected to change. However, it encompasses fields that can be used to document changes in the way teaching cases have been performed, i.e., mutability only happens on the instance layer and not on the type layer.

6 Discussion and Conclusions

From this study, we identify the following themes that also stimulate future research. First, artifact mutability is a multi-facetted topic that has been subject to research already for some time. We were able to identify different research streams that have dealt with the variability, configuration, adaptation, and evolution of artifacts and that provided the grounding for our framework. Hence, artifact mutability can be considered an umbrella term that has the potential to connect different fields like reference modelling, situational method engineering, and software evolution. To bring this idea forward, we plan to extend and revise the presented framework in future work.

Second, our current knowledge about mutability is still closely linked to the artifact type and this also shapes the varying understandings of this term. To give an instance, the mutability of methods is a common objective of (primary) design, whereas the mutability of instantiations is mostly considered as something that happens after the implementation in a specific context, e.g., through secondary design processes.

Third, IS researchers have devoted little attention to artifact mutability so far when developing design theories. Although the anatomy of a design theory [3] has been cited hundreds of times, we only found 20 design theory articles that address mutability. There is also an unbalanced distribution with a majority of articles dedicated to methods and instantiations, but rarely constructs and models. Future research should investigate the underlying reasons for both the low adoption level and the imbalance.

Fourth, the core component of artifact mutability is only a side issue in design theory articles. Those researchers who address it do this only briefly. It seems that they feel obligated to comment on mutability without making it truly a subject matter of their research. In this regard, our framework can help IS researchers in formulating more substantive statements on artifact mutability as part of design theories. It also provides a reference for review and publication processes of design theory articles that discuss artifact mutability. However, future research is also needed that reflects on the general relevance of the artifact mutability component for design theories.

The presented work is beset with some limitations. First, the framework's grounding is probably not exhaustive. We presented existing approaches prominent to IS research and related disciplines, in particular Computer Science. Our intention was to lay a foundation and show that there are already works on artifact mutability although they do not use this exact term. Second, the identification of dimensions and our suggestions for a framework are subjective in nature. Other researchers might have come to a different set of dimensions with different categories. As for now, we also do not present categories for the generic dimensions. Third, with our search strategy, we

excluded design theory articles published prior to Gregor and Jones [3] that possibly also address artifact mutability. Forth, we relied on the popular categorization of IT artifact types by March and Smith [15] although there are alternative ones [11, 52].

Concluding, this study contributes a framework that comprises six generic dimensions of artifact mutability, grounded in a larger set of dimensions from existing work. Both might be used as a reference and guidance for researchers when addressing mutability in their work. Furthermore, we demonstrated the usefulness of the framework as an analytical lens for reviewing existing design theories. These contributions advance the understanding of mutability in the IS discipline and will inform researchers and reviewers when developing, presenting, or reviewing design theories.

References

1. Gregor, S., Iivari, J.: Designing for Mutability in Information Systems Artifacts. Information Systems Foundations: Theory, Representation and Reality. pp. 3–24. ANU E Press, Canberra, Australia (2007).
2. Ahlemann, F., El Arbi, F., Kaiser, M.G., Heck, A.: A process framework for theoretically grounded prescriptive research in the project management field. International Journal of Project Management. 31, 43–56 (2013).
3. Gregor, S., Jones, D.: The anatomy of a design theory. Journal of the Association for Information Systems. 8, 312–335 (2007).
4. Becker, J., Delfmann, P., Knackstedt, R.: Adaptive reference modeling: integrating configurative and generic adaptation techniques for information models. Reference Modeling. pp. 27–58. Springer (2007).
5. Henderson-Sellers, B., Ralyté, J., Agerfalk, P., Rossi, M.: Situational Method Engineering. Springer (2013).
6. Lehman, M.M., Ramil, J.F.: Software evolution and software evolution processes. Annals of Software Engineering. 14, 275–309 (2002).
7. Hovorka, D.S., Germonprez, M.: Tinkering, tailoring and bricolage: Implications for theories of design. AMCIS 2009 Proceedings. (2009).
8. Germonprez, M., Hovorka, D., Gal, U.: Secondary design: A case of behavioral design science research. Journal of the Association for Information Systems. 12, 662–683 (2011).
9. Germonprez, M., Hovorka, D., Collopy, F.: A theory of tailorable technology design. Journal of the Association for Information Systems. 8, 351–367 (2007).
10. Sjöström, J., Agerfalk, P.J., Lochan, R.A.: Mutability Matters: Baselining the Consequences of Design. MCIS 2011 Proceedings (2011).
11. Alter, S.: The concept of "IT artifact" has outlived its usefulness and should be retired now. Information Systems Journal. (2014).
12. Gregor, S.: The nature of theory in information systems. MIS Quarterly. 611–642 (2006).
13. Piirainen, K.A., Briggs, R.O.: Design theory in practice – making design science research more transparent. Service-Oriented Perspectives in Design Science Research. pp. 47–61. Springer (2011).
14. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Quarterly. 28, 75–105 (2004).

15. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decision Support Systems. 15, 251–266 (1995).
16. Zur Muehlen, M., Recker, J.: How much language is enough? Theoretical and practical use of the business process modeling notation. Advanced information systems engineering. pp. 465–479. Springer (2008).
17. Vom Brocke, J.: Design principles for reference modeling: Reusing information models by means of aggregation, specialisation, instantiation, and analogy. Reference Modeling for Business Systems Analysis. 47–75 (2007).
18. Wenzel, S., Kelter, U.: Analyzing model evolution. Proceedings of the 30th International Conference on Software Engineering. pp. 831–834. ACM (2008).
19. Noy, N.F., Klein, M.: Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems. 6, 428–440 (2004).
20. Maedche, A., Motik, B., Stojanovic, L.: Managing multiple and distributed ontologies on the Semantic Web. The VLDB Journal. 12, 286–302 (2003).
21. Bucher, T., Klesse, M., Kurpjuweit, S., Winter, R.: Situational Method Engineering. Situational Method Engineering: Fundamentals and Experiences. pp. 33–48. Springer (2007).
22. Offermann, P., Blom, S., Levina, O., Bub, U.: Proposal for Components of Method Design Theories. Business & Information Systems Engineering. 2, 295–304 (2010).
23. Börner, R., Goeken, M., Rabhi, F.: SOA Development and Service Identification. Situationsspezifische Methoden zur Serviceidentifikation in serviceorientierten Architekturen. pp. 115–171 (2012).
24. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines. Springer, Heidelberg (2013).
25. Bühne, S., Halmans, G., Lauenroth, K., Pohl, K.: Variabilität in Software-Produktlinien. Software-Produktlinien – Methoden, Einführung und Praxis. pp. 13–24. dpunkt Verlag, Heidelberg (2004).
26. Schaefer, I.: Variability Modelling for Model-Driven Development of Software Product Lines. VaMoS 2010 Proceedings. pp. 85–92. , Linz, Austria (2010).
27. Halmans, G., Pohl, K.: Communicating the variability of a software-product family to customers. Software and Systems Modeling. 2, 15–36 (2003).
28. Thiel, S., Hein, A.: Architekturentwicklung. Software-Produktlinien – Methoden, Einführung und Praxis. pp. 81–92. , Heidelberg (2004).
29. Svahnberg, M., Van Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Software: Practice and Experience. 35, 705–754 (2005).
30. Lehman, M.M., Ramil, J.F.: Software evolution—Background, theory, practice. Information Processing Letters. 88, 33–44 (2003).
31. Mens, T.: Introduction and Roadmap: History and Challenges of Software Evolution. Software Evolution. Springer (2008).
32. Legner, C., Löhe, J.: Improving the Realization of IT Demands: A Design Theory for End-to-End Demand Management. ICIS 2012 Proceedings. , Orlando, Florida (2012).
33. Ploesser, K., Recker, J., Rosemann, M.: Supporting Context-Aware Process Design: Learnings from a Design Science Study. Business Process Management Workshops. pp. 97–104. Springer (2011).
34. Zhang, X., Venkatesh, V., Brown, S.A.: Designing collaborative systems to enhance team performance. Journal of the Association for Information Systems. 12, 556–585 (2011).

35. Löhe, J., Legner, C.: Overcoming implementation challenges in enterprise architecture management: A design theory for architecture-driven IT Management (ADRIMA). Information Systems and e-Business Management. 12, 101–137 (2014).
36. Aaltonen, M., Holmström, J.: Multi-ontology topology of the strategic landscape in three practical cases. Technological Forecasting and Social Change. 77, 1519–1526 (2010).
37. Müller-Wienbergen, F., Müller, O., Seidel, S., Becker, J.: Leaving the beaten tracks in creative work–A design theory for systems that support convergent and divergent thinking. Journal of the Association for Information Systems. 12, 714–740 (2011).
38. Woolridge, R.W., Hale, J.E., Hale, D.P.: Towards a reference architecture of intent for information systems strategic alignment. AMCIS Proceedings. Toronto, Canada (2008).
39. Molka-Danielsen, J., Mundy, D., Hadjistassou, S., Stefannelli, C.: Good Practice in Virtual Worlds Teaching: Designing a Framework through the Euroversity Project. IRIS Selected Papers of the Information Systems Research Seminar in Scandinavia. pp. 41–52 (2012).
40. Carlsson, S.A., Henningsson, S., Hrastinski, S., Keller, C.: Socio-technical IS design science research: Developing design theory for IS integration management. Information Systems and e-Business Management. 9, 109–131 (2011).
41. Piirainen, K.A., Gonzalez, R.A., Bragge, J.: A systemic evaluation framework for futures research. Futures. 44, 464–474 (2012).
42. Närman, P., Buschle, M., Ekstedt, M.: An enterprise architecture framework for multi-attribute information systems analysis. Software & Systems Modeling. 1–32 (2013).
43. De Bruyn, P., Huysmans, P., Oorts, G., Mannaert, H., Verelst, J.: Using Entropy's Justificatory Knowledge for a Business Process Design Theory. HICSS 2014 Proceedings. pp. 3717–3726 (2014).
44. Urbach, N., Ahlemann, F., El Arbi, F.: Towards a Design Theory for Customer Satisfaction-Oriented IT Vendor Management. AMCIS 2014 Proceedings (2014).
45. Böhringer, M.: Towards a design theory for applying web 2.0 patterns to organisations. ECIS 2011 Proceedings. , Helsinki, Finland (2011).
46. Iqbal, S., Päivärinta, T.: Towards a design theory for educational on-line information security laboratories. Advances in Web-Based Learning (ICWL 2012). pp. 295–306. Springer (2012).
47. Keller, C., Gäre, K., Edenius, M., Lindblad, S.: Designing for complex innovations in health care: Design theory and realist evaluation combined. DESRIST Proceedings (2009).
48. Andersson, B., Keller, C.: Harness mobility: managing the off-task property. Global Perspectives on Design Science Research. pp. 258–269. Springer (2010).
49. Ting, S.L., Tsang, A.H.: A two-factor authentication system using Radio Frequency Identification and watermarking technology. Computers in Industry. 64, 268–279 (2013).
50. Andersson, B.: Harnessing handheld computing–managing IS support to the digital ranger with defensive design. Service-Oriented Perspectives in Design Science Research. pp. 62–76. Springer (2011).
51. Tiihonen, J., Männistö, T., Felfernig, A.: Sales Configurator Information Systems Design Theory. 16th International Configuration Workshop. pp. 67–74. , Novi Sad, Serbia (2014).
52. Offermann, P., Blom, S., Schönherr, M., Bub, U.: Artifact types in information systems design science–a literature review. Global Perspectives on Design Science Research. pp. 77–92. Springer (2010).