Reducing ambiguity during enterprise design

Marné de Vries

Room 3-22, Engineering Building II, University of Pretoria

marne.devries@up.ac.za

+27 12 420 2038

ORGID: 0000-0002-1715-0430

Keywords

Enterprise Engineering; Enterprise Design; Enterprise Intentions; Enterprise Requirements; Design Domains.

Acknowledgements

We would like to thank all participants of this study for their active participation and willingness to contribute towards the development of a *method for enterprise intentions concept clarification*.

Abstract

Requirements elicitation is one of the most important phases in the design process and applied by many engineering disciplines. A more recent application of the design process is to design the enterprise as an artefact, also called enterprise engineering (EE). Even though there are limits to formal enterprise design due to enterprise complexity, strategic intentions are not realised spontaneously or accidently. Intentional enterprise design is required, starting with the strategic context, eliciting enterprise intentions. Similar to the ad hoc evolution of enterprises, EE as a discipline also developed in a fragmented way, with enterprise design knowledge mostly encapsulated in several enterprise design *approaches*. A previous study analysed eight different enterprise design/alignment *approaches*, inductively developing a common framework to represent and compare these *approaches* in terms of four main components. One of the components represent the *scope of enterprise design/alignment* in terms of three dimensions: *design domains, intentions and constraints*, as well as *enterprise scope*. Since existing *approaches* use inconsistent means of defining the first dimension, namely the *design domains*, previous work already provide some guidance on demarcating *design domains* in a more consistent way. This article focuses on the second dimension, i.e. *intentions and constraints*, and the need to distinguish between different *intention-related* concepts to reduce possible ambiguity. The study applies design science research to develop a *method for enterprise intentions concept clarification* (MEICC) as a theoretical contribution. The study also offers a practical contribution, demonstrating how the MEICC was used to clarify *intention-related* concepts that feature within a specific approach, namely Hoogervorst's approach. A *coding strategy* (including *coding conditions*, a *refined codebook* and a *coding method*), developed for Hoogervorst's approach via MEICC, is presented as a secondary contribution, since the *coding strategy* will also be useful to practitioners that use Hoogervorst's approach.

1. Introduction

Requirements elicitation, also called requirements gathering, is the process of discovering system problems and solution requirements for a system by gathering knowledge from stakeholders who have a direct or indirect influence on the requirements [1,2]. Establishing design requirements, sometimes termed *problem definition*, is one of the most important elements in the design process and applied by many engineering disciplines [3,4]. A more recent application of the design process, is to *design the enterprise as an artefact*, also called enterprise engineering (EE) [5]. Yet, enterprise design is by no means simple, since enterprises rank amongst the highest in complexity, i.e. level eight on Boulding's [6] nine-level complexity scale. Even though there may be limits to formal enterprise design due to enterprise complexity, the realisation of strategic intentions and successfully addressing *areas of concern* do not occur spontaneously/incidentally [7]. Enterprises are intentionally created entities and are created via *design* when *design* is defined as: "courses of action aimed at changing existing situations into preferred ones" [8].

Multiple theorists and practitioners address enterprise complexity by developing structured *approaches* for enterprise design [9]. Similar to the ad hoc evolution of enterprises, EE as a discipline also developed in a fragmented way, with enterprise design knowledge mostly encapsulated in several enterprise design *approaches* [10]. De Vries et al. [9] studied eight different enterprise design/alignment *approaches*, inductively developing a common framework to represent and compare these *approaches*. The enterprise evolution contextualisation framework (EECM) indicates that the analysed approaches can be represented by *four main components*:

(1) A *belief/paradigm of value-creation* that the approach offers.

(2) Three *dimensions* that represent the extent of enterprise design/alignment scope.

(3) *Mechanisms and practices* that are used to ensure consistent design/alignment for the intended design/alignment scope.

(4) *Approach classifiers* that represent common *patterns of design* detected in the existing design approaches.

Based on the *belief/paradigm of value-creation*, a particular approach usually offers various *mechanisms and practices* to align a certain part of the enterprise as defined by the *dimensions* [9]. Even though EECM provides a common reference framework to compare existing approaches [11], EECM is neither prescriptive in evaluating the components of an existing approach, nor prescriptive in guiding a theorist/practitioner when s/he develops a new approach. EECM represents the enterprise design/alignment scope, namely the *dimensions*. The three *dimensions* include:

(1) *Design domains* that represent abstract sub-systems within the enterprise that should be designed/aligned.

(2) *Concerns and constraints* as emerging concerns/expectations/intentions of different stakeholders within the enterprise design process.

(3) *Enterprise scope* that reflects formal implementation structures that are often used to organise employees within an enterprise-as-a-legal-entity, e.g. departments, programmes and projects.

De Vries [12] already critiqued existing approaches for their inconsistent means of defining the first dimension, namely the *design domains* [12]. Contributing towards the prescriptive knowledge within the EE discipline, this article focuses on the second dimension, i.e. *concerns and constraints*. As will become evident in this article, the research process itself highlighted the problematic nature of the word *concerns* and its negative connotation; hence, we have replaced the word *concerns* with *intensions* in this article. The study addresses the following questions:

- What methodologies and techniques are currently available to guide elicitation of *intentions and constraints*?
- What techniques/methods are available to clarify concepts associated with *intentions and constraints* for an existing or new enterprise design approach?
- What *method* should be developed to clarify concepts associated with *intentions and constraints* for an existing enterprise design approach?
- How useful is the newly developed *method*?

Using *design science research*, we develop a *method for enterprise intention concepts clarification* (MEICC) as the main contribution of this article. The MEICC is primarily a practical contribution, since it is useful to clarify concepts related to *intentions and constraints* when a new enterprise design approach is designed *or* when an existing enterprise design approach provides limited guidance on distinguishing between concepts related to *intentions and constraints*. As an example, MEICC should be useful in the following scenario: An existing approach identifies two different concepts that embody enterprise design *intentions*, namely *areas of concern* and *requirements*. However, the approach provides limited or ambiguous descriptions and examples of *areas of concern* and *requirements*. MEICC should be used to iteratively refine, clarify and possibly re-phrase the concepts.

The rest of the paper is structured as follows. Section 2 provides additional background on one of the EECM dimensions, namely *intentions and constraints*, elaborating on how existing *approaches* within the EE discipline and associated techniques to represent this dimension, the deficiencies of existing techniques and the need to provide more guidance by developing the MEICC. The section concludes with some background on an existing EE approach, namely Hoogervorst's approach, since we use his approach for demonstrating the MEICC. Section 3 presents the research methodology, i.e. design research that was used for developing the MEICC, whereas section 4 presents the constructional parts of the MEICC. Section 5 provides a demonstration of the MEICC when applied to Hoogervorst's approach, as well as a *coding strategy* when practitioners use Hoogervorst's *intention-related* concepts. We discuss our findings in section 6, concluding on limitations and recommendations for future work.

2. Background

Although many disciplines contribute new knowledge about the nature, behaviour and design of the enterprise [13], EE is an emerging discipline that focuses on the holistic governance and design of the enterprise, aligning the intended functions/objectives of the enterprise with its construction [14,5]. In general the term *enterprise* refers to a business, company, firm, corporation, organisation or institution, i.e. a purposeful social entity of human endeavour [15].

Numerous theoretical *approaches* exist to guide enterprise design, each based on a particular conceptualisation of the enterprise and its parts [9]. As discussed before, existing enterprise design *approaches* may be represented by four main components of which *three* are usually evident [9]:

(1) The author(s) of a theoretical approach usually focus on a particular enterprise design-related phenomenon, formulating a *belief/paradigm of value-creation* to address the phenomenon.

(2) Approach author(s) may also be explicit in stating their intended design scope via three explicit *dimensions*, i.e. design domains, concerns & constraints, and enterprise scope.

(3) They usually present a number of *mechanisms and practices* to ensure consistent enterprise design across the intended *dimensions*.

In terms of the third component, *mechanisms and practices*, De Vries et al. [9] identified a not-exhaustive list of nine categories of *mechanisms and practices*. Yet, a theoretical *approach* seldom encapsulates all nine categories. One of the categories, i.e. *methodology*, often features within an enterprise design *approach*. A *methodology* stipulates a systematic design process for designing/aligning the *enterprise as an artefact* across the enterprise *dimensions*. The methodology should ensure that relevant enterprise *intentions and constraints* are addressed within the *design domains* of the enterprise.

2.1 Existing approaches within related disciplines

Since EE is still an emerging discipline [5], focusing on designing the enterprise as an *artefact*, it draws on other existing mature disciplines that also involve the design of *artefacts*, such as systems engineering [16,17] and software engineering [13]. Within their design processes, mature disciplines have already developed *methodologies* to ensure that the initial design *intentions* are systematically translated into structural components of the artefact.

Within the *systems engineering* discipline three main methodologies exist, namely the V-model, incremental and hybrid methodologies [17]. Within *software engineering*, multiple *methodologies* exist to reduce the semantic gap between software systems as *artefacts* and their operational environment, such as ASPIRE, SIKOSA, TROPOS and i* [18,19].

2.2 Existing methodologies and techniques within related disciplines

Existing *methodologies* provide guidance on the process for eliciting design *intentions*, i.e. facilitating various activities to elicit, model and manage *intentions*. In terms of *modelling*, the i* framework, [20] provides a means to trace design *intentions* as *goals* throughout the artefact design process and up to the phase where constructional alternatives are defined and compared [21,19]. Yet, Horkoff and Yu [19] believe that the reliability of *goal-oriented* requirements engineering depend on a number of factors such as various interpretations of goal model syntax, measurement choices for goal satisfaction, and the level of participation of the user. Since the initial elicitation of enterprise *intentions/concerns* is still exploratory in nature, high-level, abstract and hard-to-measure [22] automated procedures associated with goal modelling analysis is often difficult. Horkoff and Yu [22,23] suggest an alternative procedure for these initial phases, encouraging stakeholder involvement and model improvement *via iteration*. Horkoff and Yu developed a systematic and interactive analysis procedure for eliciting and evaluating goals during enterprise modelling using i* [23]. Their procedure involved multiple stakeholders to express their viewpoints, allowing multiple iterations and discussions to refine the goal models, alternative constructional solutions, also evaluating the alternative constructs. Although the models intend to represent the *intentions/goals* for constructing a part of the enterprise-as-an-artefact, Horkoff and Yu do not demarcate the enterprise into *design domains* and their methodology does not focus on the holistic design of the *entire enterprise*.

Existing methodologies differ in how they classify concepts related to design *intentions*. The *agile systems engineering* methodology classify *intentions* as functional/stakeholder requirements to specify system *functions* and *other requirements* (e.g. operational, logistics, usability, user interface, maintainability, certification, project and constraints) [17], whereas CORE (capability oriented requirements engineering), refers to system *capabilities*, rather than system *functions* [24]. Horkoff and Yu [22] acknowledge two types of design *intentions*, namely soft goals and goals. Although we should expect that existing methodologies use similar concepts, but with different interpretations, the authors of such methodologies do not always provide sufficient descriptions and examples to clarify and distinguish between the concepts.

2.3 Existing approaches, methodologies and techniques within enterprise engineering

Within the EE discipline, design team members also clarify their understanding via conceptual models and design specifications to represent design *intentions* within *design domains* [9]. De Vries argued that *design domains* should be defined in a more consistent way to reduce ambiguity during design [12]. Similar to other design-related disciplines, EE researchers also acknowledge the need to address various stakeholder *intentions* [25] (see Chapter 21 of TOGAF 9.1) that need be explicit [26,7] and constantly examined and monitored [27]. The Open Group indicates that multiple documents may be useful for extracting stakeholder *intentions*. Zachman [28] indicates that multiple *intentions* for different stakeholders need to be addressed during enterprise design. Although the Zachman framework provides some concepts related to *intentions*, associated with different stakeholders, such as motivation types, business end, system end, technology-and-tool end [28], it offers little methodological guidance to translate the concepts into enterprise constructions [29]. Smith [30] offers a methodology that includes a strategizing phase to state concepts associated with design *intentions*, including mission, vision, strategies tactics, goals and objectives, with some guidance on principles (see p 65 of [31]), but with little guidance on clarifying other *intention*-related concepts (see p 47 of [32] and p 17 of [31]).

Giachetti [13] provides a generic approach for enterprise systems design. Although he provides some methodological guidance on eliciting design *intentions*, advocating traceability of identified design *intentions*, there is no indication of how the design *intentions* should be addressed within certain enterprise *design domains* or constructs. He identifies a number of *intention*-related concepts, indicating that “a distinction is made between goals, objectives, requirements, and constraints”, also differentiating between functional and non-functional requirements. Furthermore, he suggests that a number of techniques are used to elicit design *intentions*, including data-gathering techniques (documents, interviews, observations, questionnaires and workshops), problem analyses (using cause and effect analyses and causal loop analyses), stakeholder analyses and requirements analysis. In clarifying *intention*-related concepts, Giachetti [13] provides some guidance on how the concepts *vision*, *goal* and *objective* should form a hierarchy and providing some guidance on how to state an *objective*. In terms of the concept *functional requirement*, Giachetti [13] suggests that the SMART (Specific, Measurable, Attainable, Realistic, and Time-bound) criteria are used. For the concept *non-functional requirement*, he provides a number of examples, such as reliability, performance and sustainability. In terms of *constraints*, Giachetti [13] indicates that constraints are “restrictions on other requirements” originating from reasons such as physical limitations of resources and environmental regulatory rules.

2.4 Need to reduce ambiguity when eliciting enterprise intentions

A key criterion for defining design *intentions* is that they need to be unambiguous, understandable and verifiable [17]. Emphasising the need to reduce ambiguity during the *process* of requirements elicitation, ambiguity in communication is regarded as a major obstacle [33,1] . Since human beings participate as design team members, they need to formulate a common understanding of the *design domains* that need to be constructed, as well as the design *intentions* that need to be addressed. Within the EE discipline, we believe that a consistent representation of design *intentions* should reduce ambiguity. In section 3 we suggest that a new method (i.e. the MEICC) is required to distinguish between different kinds of design *intentions* prior to extracting and modelling design *intentions* for an enterprise. Our suggested method incorporates the use of a codebook and applies natural language to clarify *intention-related* concepts. We reduce ambiguity by avoiding plurality (see [34]), also preventing lexical, syntactic, semantic, pragmatic ambiguity, and vagueness (see [35]).

In section 3 we suggest that the newly-developed MEICC should be useful to clarify intention-related concepts for an existing or new enterprise design approach and we motivate our choice to demonstrate its use, using Hoogervorst's [7] approach. The next section provides some background on Hoogervorst's approach.

2.5 Hoogervorst's iterative approach

Hoogervorst [15,14,7] acknowledges that an enterprise is a morphogenic, dynamic, socio-cultural system that will only exist/survive if it is capable of reacting to contingencies concerning, for example, consumers, competitors, weather (especially agricultural enterprises) and economics. The enterprise as a system should have sufficient variety or regulating ability to address the variety it faces [15,36]. Traditional approaches reduce or attenuate enterprise variety through rules, regulations and management directives, which create rigidity and the inability to properly respond to the variety they face [15]. In contrast, Hoogervorst [15] suggests a different approach to deal with variety, namely to increase the enterprise *regulating variety*. He presents an *iterative approach* that guides and converts design *intentions* into enterprise constructions within different *design domains*.

Hoogervorst's approach provides a multi-disciplinary inquisitive approach with relevant stakeholders, starting with the *strategic context*. As indicated in Fig. 1, *strategic contexts* have to *indicate* certain *areas of concern* (alternatively defined as *performance areas*), whereas detail *requirements* are defined to *deal with* the *areas of concern*. Since *requirements* and *areas of concern* need to be *addressed* in *design domains*, it is possible to formulate *design principles* that will *address* the *areas of concern* if enterprise designers *apply* the *design principles* to one or more *design domains*. Although enterprise design encapsulates elicitation of detailed *requirements*, Hoogervorst [7] focuses more on formulating *design principles* that are primarily used to ensure unity and integration across the implemented *design domains* of the enterprise.

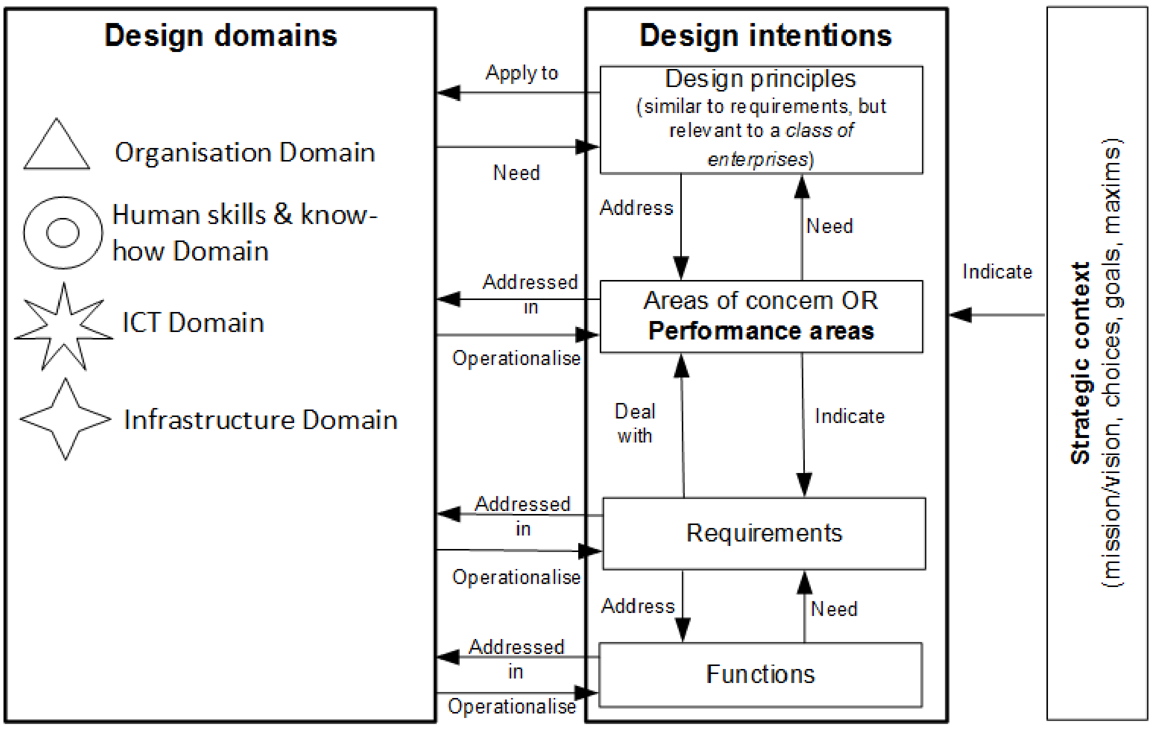

Fig. 1 Iterative enterprise design approach, based on Hoogervorst [7]

Following Hoogervorst's guidance to involve multiple team members, Van der Meulen [37] experimented with the *iterative approach*, but indicated that additional guidance was required when translating *design intentions* into enterprise constructions within *design domains*. Following Gauss & Weinberg's advice [38] to clarify words and concepts, reducing interpretation ambiguity, Van der Meulen [37] argued that a codebook was required to define and distinguish between concepts associated with Hoogervorst's *iterative approach*. As indicated in Fig. 1, two categories of concepts exist, namely *design domains* and *design intentions*. *Design domains* (e.g. *organisation*, *ICT*, *infrastructure* and *human skills & know-how*) are abstract or real areas for which design activities are required, whereas *design intentions* (i.e. *design principles*, *areas of concern*, *requirements* and *functions*) provide functional or constructional guidance for creating/changing the construction of *design domains*.

Van der Meulen [37] experimented with Hoogervorst's [14] demarcated *design domains*, but experienced difficulties during the process of sense-making when his design team applied emerging *design principles* to the main *design domains.* Based on his suggestion to use the *basic system design process* to re-define *design domains* in a more consistent way [37], a subsequent study suggested a new set of *design domains* [12]. The newly demarcated *design domains* are already reflected in Fig. 1. Hoogervorst's [7] initial framework included an additional design domain, called the *business domain*. Yet, since the business domain represent functions of the enterprise, rather than its construction [12], Fig. 1 presents *functions* as *design intentions*.

The study in [12] highlights multiple concurrent design cycles that exist between using systems and object systems in an enterprise [12]. We provide a *short definition* for each *design domain* and *typical models or modelling languages* that are used to represent the particular design domain. The interested reader is referred to [12] for additional elaboration on typical design cycles that are associated with different design domains.

Organisation domain: The organisation domain encapsulates *actor roles*, implemented by human beings, that form relationships due to their interactions and communications when they perform *production acts* [39]. Dietz [39] suggests *aspect models* that represent the *essence of enterprise operation* in a coherent, comprehensive, consistent and concise way.

ICT domain: The ICT domain incorporates software applications, databases and ICT hardware [39]. Different representations are used to communicate ICT designs, such as *unified modelling language* (UML) models [40] and *wire-framing* models [41].

Infrastructure domain: Infrastructure entails facilities and other non-ICT technologies that support *actor roles* and their *production acts*. Enterprises within different industries may require different *representations* of infrastructure, based on the type of *production acts* that should be supported. The educational industry, for instance, may apply web-based *3D interactive campus models* to visualize learning facilities.

Human skills & know-how: Human skills & know-how constitutes human *abilities and skills* required when executing *production acts*, as well as *coordination acts*. Skills and know-how are often represented in *curricula vitae*.

Although it was necessary to clarify the definition for the main enterprise *design domains* as distinguished from design *intentions,* this article primarily focuses on a method (i.e. MEICC) to clarify concepts that are related to design *intentions*.

3. Research methodology

The study forms part of an existing project that applies *design science research (DSR)*, where existing concepts within the EE discipline exist, but require further clarification. DSR is a research methodology that developed from the *design science* knowledge area. Whereas *design science* reflects on *design* as a *topic of investigation* to explore almost any design related subject, DSR uses *design* as a *method for investigation* [42], aiming to create "solutions to specific classes of relevant problems by using a rigorous construction and evaluation process" [43]. Simon [8] differentiates *design science* from other paradigms as follows: "Whereas natural sciences and social sciences try to understand reality, design science attempts to create things that serve human purposes". Hevner et al. [44] state a key differentiator between *routine design* and DSR, namely that DSR contributes to the "archival knowledge base of foundations and methodologies." The fundamental principle of DSR is that "knowledge and understanding of a design problem and its solution are acquired in the building and application of an artefact" [44]. Knowledge and action form a cycle, in which knowledge is used to *create works*, and works are evaluated to build knowledge [45]. March & Smith [46] identified four design artefacts that are produced by information system related DSR, i.e. *constructs*, *models*, *methods*, and *instantiations*.

When discussing epistemological perspectives of *design science research* (DSR), i.e. perspectives on producing "true" knowledge via DSR, Niehaves [47] argues in favour of *epistemological diversity* and to establish *constructive pluralism* within DSR. Although the *seven guidelines* provided by Hevner et al. [44] on conducting research and evaluating knowledge within DSR, follow a rather implicit *positivist epistemology*, Niehaves [47] mapped the *seven principles of Klein & Myers* [48] for interpretive field studies to *five* of the *seven guidelines* of Hevner et al. [44] to demonstrate the possibility of an *interpretivist* stance when conducting DSR. Even thought the *seven guidelines* offered some guidance on DSR, Peffers et al. [49] believe that different genres have developed within the DSR community, characterising genres in terms of their focus, research process, role of theory, and evaluation. As an example, the *IS design theory* genre of Gregor and Jones [50] *focuses* on the presentation of design theories and their conceptual-oriented *evaluation*. In contrast, the *DSR methodology* genre of Peffers et al. [51] *focuses* on applicable artefact-development and its *evaluation* that should be outcome-oriented and practical.

Following the *DSR methodology* genre of Peffers et al. [51], this study focuses on the development of an artefact, i.e. we suggest that a new *method* (i.e. MEICC) is developed as an artefact. *Methods* define processes or guidance on how to solve problems, ranging from mathematical algorithms to informal, textual descriptions of "best practice" [46]. In accordance with Gregor & Hevner's [52] knowledge contribution framework, our newly-developed MEICC can be considered as an *improvement* study, since a known problem is addressed.

3.1 The design science research design cycle

Referring to the DSR phases of Peffers et al. [51], this article addresses the *first four phases* of the DSR cycle as follows:

Identify a problem: EE researchers already acknowledge the need to address various stakeholder *intentions* [25] that need be explicit [26,7] and need to be constantly examined and monitored [27]. The Open Group [25] indicates that multiple documents may be useful for extracting stakeholder *intentions* and Giachetti [13] proposes various means of data-gathering to elicit design *intentions*. We believe that existing EE approaches need additional rigour in clarifying and classifying the emerging design *intentions*, prior to the iterative extraction and translation of *intentions* into enterprise design specifications.

Define objectives of solution: We suggest that a *method for enterprise intention concepts clarification* (MEICC) is developed to reduce ambiguity, distinguishing between design *intention-related* concepts for an existing or new enterprise design approach.

Design and develop: Prior to designing the MEICC, we considered different techniques that would address MEICC's *solution objectives*. In section 3.2 we elaborate on the different techniques that could be considered in addressing the *solution objectives* for MEICC, motivating the use of a codebook. The study applied principles of reducing ambiguity [34,35] and principles of codebook design [53] to construct the MEICC, a method that uses a codebook to refine concepts related to design *intentions* in terms of a *code label, condensed definition, full definition, example, what it is not, how to state* and *cues* for coding.

Demonstrate: Hoogervorst [7] presents an approach that is focused on design, i.e. eliciting *intentions*, translating *intentions* into *design principles* that would guide future designs within *design domains.* Similar to other existing EE approaches, Hoogervorst [7] provides some guidance in further classifying design *intentions*. Yet, Van der Meulen [37] experimented with Hoogervorst's *iterative approach*, but indicated that additional guidance was required to clarify the concepts. Although Van der Meulen [37] already used a codebook to distinguish between *intention-related* concepts associated with Hoogervorst's approach, he only involved a single participant to validate his codebook. We also use Hoogervorst's approach to demonstrate the newly-developed MEICC, but involve multiple participants with several iterations of code-refinement.

Although we only discussed the *first four phases* of the DSR cycle, we elaborate on future work in section 6 of this article.

3.2 Different techniques for addressing the solution objectives

Different techniques can be considered to address the main *solution objectives* for the MEICC. This section elaborates on three possible techniques and the rationale for using two of the techniques during the development of the MEICC.

3.2.1 Conceptual models and ontologies

Conceptual models were initially used to provide a pictorial representation of several ideas as to increase understanding and communication of a system or domain among different stakeholders within the information system discipline [54]. An example of a conceptual model is Fig. 1, used to represent concepts associated with the iterative enterprise design approach of Hoogervorst. Although conceptual modelling was initially focused on defining user requirements for information systems on different levels [55,56], conceptual models also provide a common understanding amongst stakeholders within many different areas, ranging from enterprise strategic management, service sciences and technology-enhanced learning [57]. Yet, conceptual modelling techniques are also criticized, since many of the techniques lack adequate specification semantics for the terminology of the underlying models, leading to inconsistent interpretations and uses of knowledge [58]. Ontologies are useful to overcome these issues, since ontologies describe a set of things associated with a particular theory or system of thought [59]. Ontologies can be applied to articulate and reason about the contents of a conceptual model [60], describing the structure and behaviour of the modelled domain [61]. Even though Verdonck et al.'s [62] empirical research indicated that ontology-based conceptual modelling (OBCM) increases the quality of the conceptual models when modelling advanced concepts of a domain, research subjects had to receive training in OBCM modelling "over a period of several months". Within the context of this study, we believe that a conceptual model, such as Fig. 1 provides some understanding of the concepts associated with an existing enterprise design approach. Thus, similar to what Guizzardi et al. [63] did, it is possible remove possible ambiguity and conceptual overlap of concepts by starting with the existing concepts promulgated in an enterprise design approach and develop an ontology from the existing concepts that is grounded in the unified foundational ontology (UFO). Since participants of the ontology development team need to be well-trained in OBCM, we did not incorporate OBCM when developing the MEICC. In section 6.2 we still acknowledge the value of using OBCM and suggest future research to experiment with OBCM as a means to clarify design *intention-related* concepts associated with an enterprise design approach.

3.2.2 Focus group discussions

Giachetti [13] suggests that *requirement workshops* are used for requirements elicitation, since "they can be designed to encourage consensus concerning the requirements for a particular system capability or feature". He lists several benefits, including its: (1) Involvement of stakeholders across organizational boundaries; (2) Interactive, dynamic and cooperative nature; and (3) Ability to provide structure to the capturing and analysis of requirements. Yet, Giachetti [13] provides limited guidance on structuring a *requirements workshop*, indicating that a joint requirements planning (JRP) workshop is often used to elicit requirements for *information system development*. Krueger & Casey [64] refer to a more generic form of workshop, namely *focus group discussions* that are used to perform data-gathering on a *particular topic* in a structured way. The format of a focus group discussion is similar to that of a *requirement workshop* and we believe the guidelines presented by Krueger & Casey's [64] would also be applicable when it is used to distinguish between different design *intention-related* concepts in a participative way. In section 4.2, we indicate how we incorporated Krueger & Casey's [64] guidelines into the MEICC, partially addressing the main *solution objectives* for the MEICC, i.e. reducing ambiguity when *systematic* and *participatory* focus group discussions are used to iteratively distinguish and refine design *intention-related* concepts of an enterprise design approach.

3.2.3 Using a codebook

Guest et al. [53] believe that the codebook is one of the most critical components of *thematic analysis*. Similar to grounded theory, *thematic analysis* is used to interpret free-flowing text, moving beyond counting explicit words or phrases and "focus on identifying and describing both implicit and explicit ideas within data". Since multiple analysts may be involved when analysing multiple documents, Guest et al. [53] use a codebook to ensure consistent interpretation of text, monitoring and improving interpretation consistency by assessing the level of intercoder-agreement. Codebook development is a discrete analysis step within thematic analysis, where the "observed meaning in the text is systematically sorted into categories, types and relationships of meaning" [53]. The codebook thus assists in clarifying the meaning of an identified theme, defining the identified theme as a code within a codebook. Based on various different studies in different settings over many years, MacQueen et al. [65] indicated that a code definition needs to be clarified using: (1) code label; (2) short definition; (3) full definition; (4) when to use; (5) when not to use; and (6) an example. The rigour of the codebook increases if the code definitions are refined in an iterative way, resolving interpretation discrepancies, until and acceptable level of intercoder-agreement is obtained [53]. We believe that a codebook and its iterative refinement could be useful to incorporate within the MEIC. Partially addressing the main *solution objectives* for the MEICC, i.e. reducing ambiguity when distinguishing between different design *intention-related* concepts, we believe that a codebook can be used to *define*, *refine* and *document* the design *intention-related* concepts, removing possible interpretation discrepancies that may exist amongst different design team participants until intercoder-agreement is obtained.

The next section elaborates on the newly-developed MEICC and how we incorporated *focus group discussions* and a *codebook* within the structural components of the MEICC.

4. Method for enterprise intension clarification

The design of MEICC is based on the principle that ambiguity should be reduced by following a *systematic*, *iterative* and *participatory* process on refining design *intention-related* concepts that are associated with an existing or new enterprise design approach. Using natural language as a means of documenting and communicating the design *intention-related* concepts via a codebook, we suggest that the codebook designer should follow two main iterations of codebook refinement.

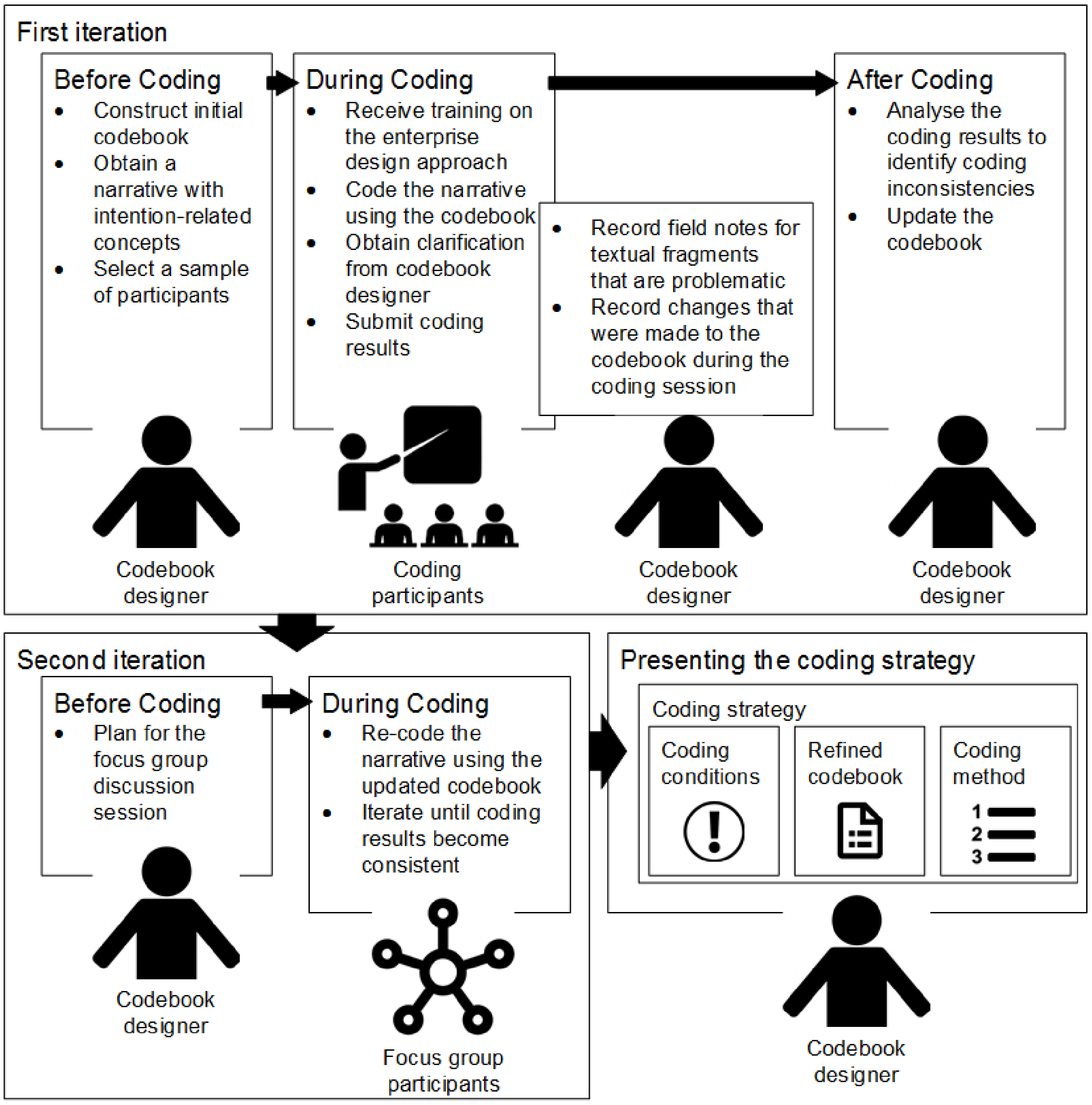
First iteration
Before Coding
Construct initial codebook
Obtain a narrative with intention-related concepts
Select a sample of participants
Codebook designer
During Coding
Receive training on the enterprise design approach
Code the narrative using the codebook
Obtain clarification from codebook designer
Submit coding results
Coding participants
Record field notes for textual fragments that are problematic
Record changes that were made to the codebook during the coding session
Codebook designer
After Coding
Analyse the coding results to identify coding inconsistencies
Update the codebook
Codebook designer
Second iteration
Before Coding
Plan for the focus group discussion session
Codebook designer
During Coding
Re-code the narrative using the updated codebook
Iterate until coding results become consistent
Focus group participants
Presenting the coding strategy
Coding strategy
Coding conditions
Refined codebook
Coding method
Codebook designer

Fig. 2 The method of enterprise intention concepts clarification (MEICC)

4.1 First iteration

The purpose of the first iteration is to assess whether the existing *intention-related* concepts, associated with a selected enterprise design approach, provide sufficient clarity and distinction when concepts are applied during documental coding.

4.1.1 Before coding

Prior to coding, the codebook designer needs to construct or obtain the following inputs:

- An *initial codebook* should be constructed by the codebook designer, referring back to the initial definitions about *intention-related* concepts that are provided by the author(s) of a chosen enterprise design approach. The codebook designer should refine *intention-related* concepts as to avoid plurality [34], preventing lexical, syntactic, semantic, pragmatic ambiguity, and vagueness [35]. Based on Guest et al.'s [53] guidance on codebook design, every *intention-related* concept should be defined in terms of a *code label*, *condensed definition*, *full definition*, *example*, *what it is not*, *how to state* and *cues* that may be useful during coding.
- A *narrative* that incorporates *intention-related* concepts should be obtained, such as minutes of a management meeting for *a selected enterprise*.
- A sample of *participants* should be selected for the coding exercise. The participants should have some knowledge about the industry of the *selected enterprise*.

4.1.2 During coding

The codebook designer has to start the coding session by introducing a selected enterprise design approach to the participants. Then, *participants* need to use the *codebook* and *narrative* to perform content coding. The codebook designer should not over-explain the *intention-related* concepts. Rather, the participants should perform coding; using the codebook on face value. Since the codebook designer should gain an understanding about the reasons for coding deviations, s/he should ensure that:

- Participants are encouraged to request clarification from the codebook designer for fragments in the narrative that are difficult/problematic to code.
- A participative approach is followed to allow for real-time identification of problems regarding the codebook. Any adaptations to the codebook should be communicated to all participants while coding proceeds.
- Field notes are used to track feedback from participants on problematic textual fragments in the *narrative*, as well as the changes that were communicated to participants regarding codebook changes.
- Participants submit their *coding results* to the codebook designer once they have completed the coding exercise.

4.1.3 After coding

After the participative session, the codebook designer has to analyse the *coding results* to identify coding inconsistencies and reflect on possible reasons for inconsistencies. The main deliverable of the first iteration is an *updated codebook*, based on the participant feedback and the *coding results*.

4.2 Second iteration

For the second iteration, a focus group should be used to obtain feedback on the *updated codebook*. As reported by Krueger & Casey [64], focus groups provide an opportunity to obtain rich data from a small and diverse group of people familiar with the study. A focus group would be a suitable instrument to perform formative evaluation of the *updated codebook*, since the number of participants are limited and differences in coding results are discussed in more depth.

4.2.1 Before coding

The codebook designer uses the following guidelines of Krueger & Casey [64] to plan for a focus group discussion:

1. *Group size.* The group should be conducted with 4 to 12 people, led by a skilled moderator [64]. The group has to be small enough for everyone to have an opportunity to share insights, but also large enough to provide diversity of perceptions. If a topic is complicated, 10 to 12 participants are risky and may produce trivial results. Also, when time is limited (90 minutes), the size of the group should be restricted [64].
2. *Time.* Focus group sessions are typically two hours long [64].
3. *Questioning route*. Krueger & Casey [64] present practical guidance regarding the sequence of questions (e.g. general to specific), phrasing of questions (using open-ended questions, keeping questions simple), and revising questions if clarification is required.
4. *Sampling*. Random sampling is of limited value in focus group research, since the intent of a focus group is to understand and provide insights - not to generalise [64].
5. *Analysis*. Data can be captured in different forms as the basis for analysis, e.g. transcripts, abbreviated transcripts, notes, and memory [64]. Note-based analysis relies on field notes and is audio recorded as a backup to clarify confusing aspects contained in the notes [64].

4.2.2 During Coding

Similar to the first iteration, *focus group participants* need to use the *updated codebook* to re-code the same *narrative* that was used during the first iteration. The main intent of the second iteration is to obtain feedback from the participants, adapting the codebook until participants' *coding results* become consistent. Thus, multiple sessions of coding-and-feedback may be required.

4.3 Presenting the coding strategy

The main output of the MEICC is a *coding strategy* that may be used in combination with the selected EE approach. Whereas the first two iterations of MEICC experimented with a preliminary codebook, with the intent to refine the codebook based on the inconsistencies of the *coding results*, the last part of MEICC presents a *coding strategy* that should be useful to an *enterprise designer* when s/he extract and classify enterprise *intentions* for a particular enterprise design scope. We suggest that the *coding strategy* consists of *coding conditions*, a *refined codebook* and a *coding method*.

4.3.1 Coding conditions

One of the main *conditions* for coding is that coders receive training on good coding practices, such as guarding against over-interpretation of documental text [66]. The *enterprise designer* should provide guidance in defining the scope and purpose of coding, since the coding results may differ drastically based on its pre-defined scope and purpose. As an example, one *enterprise designer* may only be interested in identifying design *intentions* related to an *enterprise-wide* scope, whereas another *enterprise designer* may be interested in identifying design *intentions* that are applicable to a specific *department* within the enterprise.

4.3.2 Refined codebook

The *refined codebook* provides formal definitions for concepts of a selected EE *design approach*. The codebook needs to provide additional guidance to reduce inter-coder inconsistencies when the codebook is used to code *documental text*. The multiple coding iterations (e.g. first iteration discussed in section 4.1 and second iteration discussed in section 4.2) will indicate that some coding limitations exist, since it may be difficult to identify certain design *intention-related* concepts via documental coding alone. The *refined codebook* has to emphasize these limitations, highlighting concepts that cannot be identified via documental coding alone, e.g. using the phrase ╟Not to be used for documental text coding╢.

4.3.3 Coding method

The coding method should provide a method to coders to ensure that the coding process produces consistent *coding results*. The multiple coding iterations of MEICC (e.g. first iteration discussed in section 4.1 and second iteration discussed in section 4.2) may indicate a *practical sequence* of coding, e.g. some codes may belong to an aggregate coding family. Thus, it may be easier to identify the code families first and then the sub-codes. The coding method should include *standards for tagging* text e.g. using dotted-underline for fragments that comply with the definition of a particular concept, and using super-script notation to associate a fragment with one instance of a concept, e.g. quality^{I1} and quantity ^{I2}.

5. Demonstration

Applying the MEICC to Hoogervorst's approach, this section presents the coding feedback for the *first iteration* in section 5.1, followed by the coding feedback for the *second iteration* in section 5.2. Section 5.3 concludes with a *coding strategy* that may be used by *enterprise designers* that adopt Hoogervorst's approach at their enterprise.

5.1 First iteration and results

5.1.1 Before coding

- For the *initial codebook*, the codebook designer extracted *intention-related* concepts presented by Hoogervorst [7,14], as well as the codebook that was developed by Van Der Meulen [37], clarifying some of the codes to reduce ambiguity. For each code, the codebook designer also referred to examples from a fictitious tertiary education institution, rather than re-using the agricultural industry examples provided by Van Der Meulen [37]. The selected coding participants all graduated from a tertiary education institution; hence, they would have some knowledge about its primary activities.
- An existing *narrative* was adapted to represent the minutes of a departmental meeting at a fictitious tertiary education institution.
- A sample of 36 *participants* were involved in the coding exercise.

5.1.2 During coding

The codebook designer first introduced Hoogervorst's iterative approach to participants, as presented in Fig. 1, broadly distinguishing between *design domains* and *design intentions*. The codebook designer then applied all the guidelines presented in section 4.1.2 to facilitate the coding process. Table 1 summarises the clarification that was required and the subsequent adaptations that were made to the codebook. The adaptations were communicated immediately to all 36 participants while they were busy with coding.

Table 1 Feedback during participation to adapt the codebook

It 1	**Clarification required and adaptations made to the codebook**
1	A participant inquired if fragments that indicated "suggested" *key actions* should be coded, since it is not mandated. Another participant raised a similar question on whether the fragment "if possible" should be coded. **Adaptations made to the codebook:** *Codebook change to guide further coding:* A codebook change was made to guide all analysts as participants, stating that if it is not clear from the text that the committee has made a firm decision about an intention, then analysts should not code the fragment as an intention. *Changes to the generic codebook:* Since Hoogervorst's *iterative approach* highlights the holistic view of the enterprise; the codebook should state that **all** principles and requirements are identified during coding. If the status of a principle or requirement is clear from the documental text, analysts should also use 3 themes to classify the status as *suggested*, *mandated* or *implemented*.
2	A participant inquired if the word *concern* should be an indication of an *area of concern*. **Adaptations made to the codebook:** *Codebook change to guide further coding:* The codebook was changed to indicate that analysts should not interpret the word *concern* as an indication of an *area of concern*, but rather interpret the intention of the entire fragment. *Changes to the generic codebook:* A similar change was made for the generic codebook.
3	A participant indicated that the documental text showed that a particular *area of concern* (i.e. continuity of power supply) has already been addressed. The participant inquired if the fragment should still be coded as an *area of concern*. **Adaptations made to the codebook:** *Codebook change to guide further coding:* The codebook was changed to indicate that analysts should not code *areas of concern* that have been resolved. *Changes to the generic codebook:* Since a previous/resolved *area of concern* may still be of concern for other management units within the enterprise, the generic codebook should indicate that *areas of concern* may be further qualified according to their status. Coders should use 3 themes to classify the status as *suggested*, *mandated* or *implemented*.
4	A participant indicated that a requirement may deal with multiple *areas of concern*. Furthermore, cause-and-effect relationships may exist between *areas of concern*. The coding however does not allow

It 1	Clarification required and adaptations made to the codebook
	for cause-and-effect analyses. **Adaptations made to the codebook:** *Codebook change to guide further coding:* The codebook was changed to indicate that analysts should only identify *areas of concern* that are directly related to requirements, not identifying indirect cause-and-effect relationships that may exist between *areas of concern*. *Changes to the generic codebook:* The participant's inquiry refers to the complexity that exists within enterprises and their behaviour. Coding merely provides an interpretation of existing documental text. Coding of documents that encapsulate strategic intent, provides a starting point for identifying *design intentions*. Yet, iterative enterprise design requires multiple iterations, inquiry and debate about *design intentions* and their cause-and-effect relationships. Since coding should not over-interpret text, the codebook has to emphasise that analysist should only associate a requirement with specific *area(s) of concern* if the associations are evident in the text.
5	A participant inquired about the scope of identifying intentions, i.e. should the coder only highlight those *areas of concern* that are relevant for the department or for other parts of the enterprise. **Adaptations made to the codebook:** *Codebook change to guide further coding:* The codebook was changed to indicate that coders should only identify *areas of concern* that are relevant for the fictitious engineering department. *Changes to the generic codebook:* The approach suggested by Hoogervorst takes a holistic view on enterprise design, whereas the case presented to the participants reduced the scope to a particular engineering department within an educational institution. The participant however raised a valid question about *scope*. The approach needs to facilitate identification of *design intentions* for different *design domains* and its embedded constructs. The codebook was updated to emphasise inclusivity, accommodating different levels of scope, when identifying *design intentions*.

5.1.3 After coding

Participants had to submit their *coding results* to the codebook designer for further analysis. The codebook designer analysed the coding results to determine if participants coded the text in accordance with the intended code meanings, tabling possible reasons for inappropriate coding in Table 2.

Table 2 Incorrect way-of-coding and codebook adaptations

It 1	Inappropriate coding and codebook adaptations
6	The word *need* in a sentence is interpreted as a requirement, whereas the phrase itself does not indicate a future requirement that should be incorporated in future designs. **Codebook adaptation:** Codebook should warn that the word *need* does not necessarily indicate a requirement.
7	The word *requirements* is used as a cue for a requirement, even if the requirement is not applicable for designing the future version of the enterprise.

It 1	Inappropriate coding and codebook adaptations
	Codebook adaptation: Codebook should warn that the word *requirement* does not necessarily indicate a requirement.
8	Coders had to do additional interpretation to assess whether some prescriptive phrases would require multiple *key actions* and be classified as principles rather than requirements. **Codebook adaptation:** Although the exercise allowed for clarification, coding was primarily based on documental text. Practical use Hoogervorst's *iterative approach* would require additional discussion, involving domain experts to classify a prescriptive phrase as a principle or requirement.
9	One phrase about transportation problems and the effect on timeous submission of deliverables have been phrased as two different *areas of concern*: (1) accessibility to campus; and (2) timeous submission of deliverables. **Codebook adaptation:** The identification of cause-and-effect relationships between *areas of concern* should not be part of coding. Iterative enterprise design requires multiple iterations, inquiry and debate about design intentions and their cause-and-effect relationships. Since coding should not over-interpret text, the codebook has to emphasise that coders should only associate a *requirement* with a specific *area of concern* if the association is evident in the text.
10	Coders vary on classifying a prescriptive statement as a *principle* or *requirement*. **Codebook adaptation:** According to the *iterative approach*, *design principles* address *areas of concern* whereas *requirements* deal with *areas of concern*. *Key actions* should be specified for every *design principle* and are qualified as "formal programs or projects, investigations, pilots and projects" [14] Hoogervorst [14] states that requirements may also be published using a similar four-tier structure than for design principles (i.e. statement, rationale, implications and key actions). Since *requirements* and *design principles* share common characteristics and qualifiers, their distinction is problematic. One way to distinguish between the two concepts is to use *design scope*, since a *design principle* is applicable to a broader scope of enterprise design. The concepts *design principle* and *requirement* should be consolidated in the codebook and renamed as *prescriptor*. The codebook should also be updated to clarify concepts used to qualify a *prescriptor*, i.e. the statement, rationale, implications and key actions. Sub-themes should be defined for these four concepts.
11	Some phrases state *action rules* that guide actor roles to perform coordination acts. Yet, the concept of *action rules*, as defined by Dietz [39] has not been introduced as part of the codebook. Hence, coders classified *action rule* phrases as *principles* or *requirements*. **Codebook adaptation:** Knowledge about the construction models of the *organisation design domain* is necessary to distinguish between action rules and requirements. The codebook should be adapted to add an *action*

It 1	Inappropriate coding and codebook adaptations
	rule example to section *What it is NOT* of a *prescriptor's* definition.
12	Some of the analysts coded fragments that should be interpreted as *requirements* incorrectly as *key actions* associated with the principle *department must align with the main objective of EAA*. **Codebook adaptation:** The codebook should indicate that the *key actions* must transform as-is designs into to-be designs. The *key actions* should be qualified as a sub-theme in the codebook.
13	Some of the coders coded suggested implementations as requirements. **Codebook adaptation:** *Design intentions* should guide the future construction or re-construction of *design domains*. The codebook is not clear on whether suggested implementation actions may be considered as constructional requirements. Since requirements could only be confirmed during an iterative and participative process, the codebook should be changed to indicate the implementation status of the intention. Analysts should also use 3 themes to classify the status as *suggested*, *mandated* or *implemented*.
14	Although the codebook stated that all *areas of concern* had to be stated as a variable that could either increase or decrease, coders did not always apply the codebook instruction. Some analysts indicated the desired direction (e.g. increase) of the variable. Example: A coder would incorrectly phrase an *area of concern* as *Increasing the number of admissions*, which already indicates the desired direction (i.e. increasing). The correct way of stating the *area of concern* would be *number of admissions*. **Codebook adaptation:** The codebook needs to be adapted to incorporate the concept of *desired direction*, by adding a sub-code.

The suggested improvements were incorporated in an *updated codebook* in preparation for a second iteration of experimentation and feedback.

5.2 Second iteration and results

For the second iteration, each focus group participant received a handout of the *updated codebook* (detailed code definitions for different design *intention-related* concepts) as well as a *narrative* that encapsulates the fictitious minutes of a departmental meeting used for the first iteration.

5.2.1 Before coding

We applied the guidelines stipulated in section 4.2 as follows:

1. *Group size.* Since multiple focus group discussions were not possible, we invited 11 participants to participate in the focus group discussion, of which 7 invitees attended the discussion. The diversity of the group was reflected in gender (male:female, 3:4) race (white:coloured:black:indian, 3:2:1:1) and industry

background (public, private, manufacturing and services sectors). The second iteration was divided into two sessions. At the end of the *first session*, 3 participants left due to other commitments. The number of participants for the *second session* was thus reduced to 4 participants, still representing different genders (male:female, 1:3) race (white:coloured:indian, 2:1:1) and industry background (public, private, manufacturing and services sectors).

2. *Time.* We conducted a two-hour session, divided into two coding-and-feedback sessions with a short break between sessions.
3. *Questioning route*. The codebook designer applied the guidelines of Krueger & Casey [64] regarding the sequence of questions, phrasing of questions, and revising questions if clarification was required. The intent was to understand the source of inconsistent coding amongst the participants.
4. *Sampling*. From the initial set of 36 participants, we invited 11 participants that requested clarification on the codebook during the first iteration of coding. All participants were busy with post-graduate studies.
5. *Analysis*. Since note-based analysis is sufficient when the purpose of the study is narrowly defined [64], we applied note-based analysis for the study.

5.2.2 During Coding

For the *first session*, focus group participants were requested to read the *updated codebook* and provide feedback on the *clarity* of the codebook and the code qualifiers (i.e. *condensed definition*, *full definition*, *example*, *what it is not*, *how to state*, and *cues* for each code). The feedback from the first session is presented in Table 3, as well as the adaptations that were made to the codebook prior to the second session.

At the end of the *first session*, 3 participants left due to other commitments. The number of participants for the second session was thus reduced to 4 participants and the codebook designer (moderator). The reduction of participants facilitated coding comparisons and discussions on coding differences.

Results for first coding-and-feedback session

Next, we present the results of the two coding-and-feedback sessions.

Table 3 Feedback during first participation session to adapt the codebook

It 2	Feedback and adaptations made to the codebook
1	One participant indicated that the word *concern* has a negative association. The effect is that coders will only identify fragments that are associated with *problematic* areas. The participant suggested that *area of concern* should be replaced by KPI (key performance indicator). Participants also suggested other alternatives, such as *areas to address* and *areas of interest*. KPI's are used for control, rather than for design. The phrase *areas to address* does not integrate well with other concepts in Fig. 1, especially when relationships between concepts are discussed. The word *interest* may also induce ambiguity. **Adaptations to the codebook:** The codebook was adapted by replacing *area of concern* with *performance area*.
2	Existing coding definitions should be supplemented with the graphical representation of concepts (such as depicted in Fig. 1) to visually depict the relationships between concepts. One participant also

It 2	Feedback and adaptations made to the codebook
	suggested that an entity relationship diagram (ERD) may be useful to indicate cardinalities between concepts. Yet, the coder should be skilled in using ERD's. **Adaptations to the codebook:** The graphical representation should be used in combination with the codebook. An ERD will not be added to the codebook, since it may be intimidating for coders that are not skilled in using ERD's.
3	For the *how to state* qualifier of *performance area* both quantitatively-measured and qualitatively-measured examples should be provided. **Adaptations to the codebook:** Two examples need to be included: 1. An example for a *performance area* measured qualitatively: the performance area called *quality of assessment* can either *increase* (improve) or *decrease* (deteriorate) due to enterprise (re-)design initiatives/projects. 2. An example for a *performance area* measured quantitatively: the performance area called *cost*, measured in Rand, can either increase (improve) or decrease (deteriorate) due to enterprise (re-)design initiatives/projects.
4	Coding skills is a prerequisite for using the codebook to code text. **Adaptations to the codebook:** The codebook should indicate that training is a prerequisite for coding.
5	*Rationale*, stipulated per prescriptor, may be misinterpreted. Additional clarification is needed. **Adaptations to the codebook:** The codebook should provide additional clarification by adding a question: *Why should the prescriptor be added to guide design?*
6	*Implications*, stipulated per *prescriptor*, may be incorrectly interpreted as *consequences*. Additional clarification is needed. **Adaptations to the codebook:** The codebook should provide additional clarification by adding a question: *What new constructions or constructional changes should be made to new/existing design domains or other constructs?*

Results for second coding-and-feedback session

For the *second session*, an incremental coding strategy was applied. The intent was to incrementally refine the codebook until participants' coding results were consistent. Thus, the *narrative* (departmental minutes) was divided into sections and all participants had to individually code the same section within the document. Once a section has been coded, each participant had to report on their coding results, i.e. association of textual fragments with specified codes in the codebook. Coding differences were discussed and the codebook was adapted accordingly. Four subsequent sections were coded. For the first section that had to be coded, participants followed a *concurrent coding strategy*, i.e. identifying both *areas of concern*, *prescriptors* and *prescriptor* sub-codes. For the second coding section, participants followed an *iterative coding strategy*, i.e. first identifying the *areas of concern* in the section and then identified *prescriptors* and associated sub-codes. Table 4 presents the results of the different coding strategies, additional feedback and codebook adaptations.

Table 4 Feedback during second participation session to adapt the codebook

It 2	Coding differences and adaptations to the codebook
7	Coding results differed. Participants indicated that more guidance was required regarding the *coding strategy*. Since they used a concurrent coding strategy when coding the *first section* of the documental text, they experimented with an iterative coding strategy for the *second section*. Participants agreed that it is easier to identify the grand *performance areas* during a first iteration of coding and *prescriptors* (and sub-codes, i.e. *statement*, *rationale*, *implication(s)*, *key action(s)*) during a second iteration of coding. **Adaptations to the codebook:** As suggested by the participants, the *coding strategy* has to indicate sequence, especially for novice coders: 1. For the first coding iteration: Identify *performance areas*. 2. For the second iteration: Identify *prescriptors*, i.e. fragments that are prescriptive in nature. Also identify fragments associated with the identified *prescriptor* that provide a *rationale* for the *prescriptor*.
8	For the *first section* some fragments were coded as sub-codes (*Prescriptor–Rationale* and *Prescriptor–Implication*) without identifying the *Prescriptor–Statement*. **Adaptations to the codebook:** The *coding strategy* should add a condition for identifying sub-codes, i.e. the coder should first identify a fragment that indicates the existence of a *Prescriptor–Statement* before identifying fragments for its rationale, implication(s) or key action(s).
9	For the *second section*, coding of the fragment "the country requires more engineers" differed, i.e. not coded at all, coded as a *performance area* and coded as a *Prescriptor-Statement*. The word *requires* may be misleading. Participants also reasoned that a pre-defined scope (relevance to the country versus relevance to the enterprise) will affect coding and may be considered to be *out of scope*. **Adaptations to the codebook:** The *coding strategy* should specify the scope and purpose of coding.
10	For the *third section*, coding was consistent, except for sub-codes associated with the identified *prescriptor*. Participants indicated difficulty in coding fragments that are related to historic implementations, since historic implementations do not necessarily direct future designs. Some suggested that sub-coding (i.e. coding fragments as a *Prescriptor-Rationale*, *Prescriptor-Implication* and *Precriptor-KeyAction*) should not be performed if the documental text refers to historic implementation. Others reasoned that it may be useful to keep record of historic design guidance.

It 2	Coding differences and adaptations to the codebook
	Also, differences in coding indicated that the distinction between *Prescriptor-Statement, Prescriptor-Implication* and *Precriptor-KeyAction* is problematic. The definition of a *Prescriptor-Statement* and *Prescripor-Implication* does not facilitate distinction, since both refer to constructional consequences and are stated in a prescriptive format. **Adaptations to the codebook:** The concepts defined in the codebook are still useful to domain specialists, especially if an iterative process is used to identify, classify and debate on emerging design intentions. Yet, if institutional documents are used for identifying design intentions, the *coding strategy* should only include concepts that are easily distinguishable. The *Prescriptor-Statement* should be further refined by adding a sub-code *Prescriptor-Construct* to ensure that coders would easily identify a prescriptor. Since the *Prescriptor-Implication* is similar to the definition of *Prescriptor-Statement*, the *Prescriptor-Implication* should refer back to the *Prescriptor-Statement* for a definition. Furthermore, the *Prescriptor-Statement* should be updated to indicate that a *prescriptor* may have several *implications* that are also stated as *prescriptors*.
11	For the *fourth section*, all participants identified a single fragment as a *performance area*, i.e. “students are not reading articles, case studies and content posted via EMS”. Yet, their way of stating the *performance area* differed: (1) Learnability of students; (2) Amount of reading; (3) Student’s engagement with academic content; and (4) Competence in reading via EMS. All four statements comply with the codebook instruction for stating a performance area. Yet, some of the statements provide a different interpretation to the text. **Adaptations to the codebook:** As indicated in Table 3 (It-2, Item 4), coders should receive sufficient training on coding to ensure that they do not add their own interpretations to the text.

At the end of the coding session, participants had an opportunity to reflect on the coding exercise in general. Participants agreed that domain specialists had to be involved to improve the quality of coding. One participant suggested that design *intention* identification and validation should be done during a collaborative session. The facilitator of the session should use the distinctions made in the codebook to guide elicitation and refinement of *performance areas* and *prescriptors*.

5.3 Presenting the coding strategy

The results of the two iterations were beneficial in assessing the practical use of existing concepts related to design *intentions.* The results indicated that the codebook required additional clarity and refinement to distinguish between design *intentions*. In this section, we present the suggested updates in terms of a *coding strategy* that consists of three parts: (1) *coding conditions,* (2) the *updated codebook* and (3) a *coding method*.

5.3.1 Coding conditions

The *enterprise designer* should provide additional background and training on Hoogervorst’s [7] *iterative approach*, referring to Fig. 1 to explain the relationships that tacitly exist between concepts.

5.3.2 Refined codebook

The *refined codebook* (see Table 5) provides formal definitions for concepts that are useful when applying the design approach of Hoogervorst [7]. The *refined codebook* also provides additional guidance to reduce inter-coder inconsistencies when the codebook is used to code documental text.

Table 5 The refined codebook

Code label: **Intention (I)**

Condensed Definition: A *design intention* is used as an aggregating concept for possible enterprise *performance areas, functions* and *prescriptors* that have not yet been classified as *performance areas* or *prescriptors*.

Examples: Refer to the codes for *performance areas, functions* and *prescriptors*.

Code label: **Construct (C)**

Condensed Definition: A *construct* is an enterprise artefact that needs to be designed/re-designed.

Full Definition: The *construct* answers the question: **What should be designed/re-designed at the enterprise?** *Constructs* may refer to certain classes or categories of enterprise artefacts, such as management tools, departments, information systems, curricula or facilities. *Constructs* may also be more specific, referring to an implementation or instance of a class, e.g. an instance of a facility is the Mandela training facility. The concept *construct* enables the speed of intention-generation when a new solution *construct* has to be designed, without constraining the initial conversation with abstract *design domains*. Refer to De Vries [12] for a classification of main *design domains*.

Examples: The Mandela training centre on the north campus (C1). The Engineering department (C2). Product item undergraduate curriculum (C3).

What it is NOT: Constructs of which the enterprise does not have design authority (e.g. a new governmental bill or act), should not be coded as a design *construct*, since they do not form part of the *enterprise design scope*, given the current definition of *enterprise design scope* [67].

Cues: Constructs are nouns or proper nouns. Yet, not every noun or proper noun is an enterprise *construct*.

Code label: **Performance Area (A)**

Condensed Definition: Generic characteristic of an enterprise that must be addressed via enterprise design.

Full Definition: A *performance area* answers the question: **What generic characteristic must be *addressed* via enterprise design in one or more *design domains*?** Conversely, a *design domain* must operationalise one or more *performance areas*. A *performance area* also needs one or more *design principles* to guide enterprise design.

- A fragment of text represents a *performance area* if the fragment represents a *generic characteristic* that should be embedded/addressed during enterprise (re-)design.
- Since *design domains* are abstract concepts, used by enterprise designers to demarcate the enterprise design space, documental text will seldom relate a *performance area* to a *design domain* and it may *not be possible to code* such a relationship. Domain specialists need to discuss possible relationships between an identified *performance area* with abstract *design domains*.
- Documental text may represent strategic intentions for a specific enterprise *construct*. The coder should code all *performance areas* that are relevant for an enterprise or its embedded *constructs*.
- Performance measures are often used as *performance areas*, e.g. if *number of publications* exist as a performance measure at an enterprise, it is also a *performance area*.
- One way of validating the *performance area* is to state it in the form of a *variable* that can either *increase* or *decrease*. As an example, the performance area *cost* can either increase/improve or decrease/deteriorate. Also, *number of publications* can either increase or decrease.
- If the *desired direction* is evident from the documental text, the coder should indicate the direction using a sub-code, i.e. *desire increase* (A-DesireIncr) or *desire decrease* (A-DesireDecr).
- If the *status* of the intention is evident from the documental text, the coder should also indicate the status of the

performance area using sub-codes, i.e. *suggested* (A-Sug), *mandated* (A-Man), *implemented* (A-Done).

Example: Within the context of tertiary education and a department of engineering, some *performance areas* may be *quality of assessment* (A1), *research innovation* (A2) and *cost* (A3).

What it is NOT: A *performance area* merely defines the desired outcome for an operating enterprise. It is *not prescriptive* and does not provide design guidance, whereas a *prescriptor* (principle or requirement) provides prescriptive design guidance that guides the construction of an enterprise *design domain* or its embedded *constructs*.

How to state a new performance area*:* State the *performance area* in terms of a *variable* that can *increase* (improve) or *decrease* (deteriorate). State the *construct* for operationalising the *performance area*, if evident in the documental text. An example for a *performance area* measured qualitatively: *quality of assessment increase* (A1- DesireIncr) An example for a *performance area* measured quantitatively: *cost*, measured in Rand, should *decrease* (A3-DesireDecr).

Code label: **Function (F) ╟Not to be used for documental text coding╢**

Condensed Definition: A utility or capability that must be addressed via enterprise design.

Full Definition: A *function* answers the question: **What generic utility or capability should be *addressed* by an enterprise, a particular *design domain* or *construct*?** Conversely, the enterprise, its *design domains* or *constructs* must operationalise one or more *functions*. A *function* need multiple *requirements* whereas functional *requirements* may address a particular *function*.

- A *function* can only be specified within the context of a user or using system [7]. If the using context is the *environment* of an enterprise-as-legal-entity, *functions* of the *enterprise* should be specified for supporting certain customers within the environment by delivering certain products and services to them. If the using context is the *organisation* design domain, *functions* should be specified for the *ICT* design domain in support of the organisation domain.
- Every *function* should be defined as a black box [7], i.e. transforming an *input* to an *output*.
- The *function* may be related to the enterprise, a particular *design domain* (i.e. F-D) or *construct* (F-C).
- The *function* may be related to one or more *prescriptors* (i.e. F-P).

Example: Within the context of tertiary education institution some *functions* may include *under-graduate education*, transforming *admitted undergraduate students* into *under-graduated students* (F1), *post-graduate education*, transforming *admitted postgraduate students* into *post-graduated students* (F2) and *research,* transforming *undiscovered knowledge* into *new knowledge contributions* (F3).

What it is NOT: A *function* defines the desired utilities for a particular enterprise, its *design domains* or *constructs*. It is *not prescriptive* and does not provide design guidance.

How to state a new performance area*:* State the *function* using adjective(s) + noun, also associating the function with the entire enterprise or a particular *design domain* or *construct*, indicating how an *input* should be transformed into an *output*, e.g. *under-graduate education*, transforming *admitted undergraduate students* into *under-graduated students* (F1) or *under-graduate education* of the *engineering department*, transforming *admitted undergraduate students* into *under-graduated students* (F1-C2).

Code label: **Prescriptor-Construct (P-C)**

Condensed Definition: A prescriptor provides guidance on *how* design of *design domains* **or** their embedded *constructs* must proceed. Since it may be difficult to identify how the *prescriptor* should guide an abstract *design domain*, the coder may initially have to specify how the prescriptor should guide design of a *construct*.

Full Definition: A *prescriptor* answers the question: **How should enterprise design freedom be restricted for design domains or their embedded constructs to guide realisation?** A *prescriptor* provides normative restriction of design freedom and could either be classified as a *design principle* (Pp) or a *requirement* (Pr). A coherent and consistent set of *design principles* guide system design. The reason/rationale for defining a *design principle* is that it will address one or more *performance areas* and *applies to* one or more *design domains*, whereas a *performance area* needs one or more *design principles*. Hoogervorst [7] mentions that *requirements* are sometimes adopted as *design principles* if *design principles* are inadequate.

- Since it may be difficult to distinguish between a *design principle* and *requirement*, the coder should rather focus on identifying a *prescriptor*, i.e. coding fragments that *prescribe* design/re-design of one or more *constructs*. Domain experts should use the coding results to further classify *prescriptors* as *principles* or *requirements*.
- Documental text is not "aware" of the abstract demarcation/definition of *design domains*. Rather, prescriptive phrases often prescribe the design/re-design of one or more tangible *constructs*. By definition, a *prescriptor* should be associated with one or more *construct(s)*. Thus, coders should already identify the *construct(s)* in the documental text, using a sub-code P-C to associate a *prescriptor* with one or more *constructs*. Appropriate coding for associating one *prescriptor* with two separate *constructs* will be P1-C1 and P1-C2.
- If a prescriptive fragment is associated with a *performance area*, indicating that the presciptor will have an effect on the *performance area*, the *rationale* sub-code and *performance area* sub-code should be used (P-R-A). See code labels for *Prescriptor-Rationale* (P-R) and Performance Area (A). Since documental text may not be explicit on the relevant *performance area*, additional participation with domain experts are needed to ensure that every prescriptor is defined in such a way that it addresses one-and-only-one *performance area*.
- Some fragments in the text may be misleading, such as "need" and "requirement". The coder should first consider whether the fragment represents *design guidance* for future (re-)design of the enterprise. If not, the fragment should *not be coded* as a *prescriptor*.
- If the *status* of the intention is clear from the documental text, the coder should also indicate the status of the principle/requirement using sub-codes, i.e. *suggested* (P-C-Sug), *mandated* (P-C-Man), *implemented* (P-C-Done).
- Although the definition states that reason/rationale for defining a *prescriptor* (more specifically a *design principle*) is that it addresses one or more *performance areas*, the coder should not attempt to create an association between a *prescriptor* and *performance area(s)* if the association is not evident in the text. Once coding has completed, domain experts should use the coding results to classify prescriptors as *principles* or *requirements*, associating *principles* with appropriate *performance areas,* and classify *requirements* as functional or non-functional. See code label *Principle-Rationale* for associating a prescriptor with a *rationale* in terms of *performance area(s)*.

Prescriptors (principles or requirements) are usually published according to four descriptors: (1) *Statement*; (2) *Rationale*; (3) *Implication(s)* stated as *Principles*; and (4) *Key action(s)*. Since Hoogervorst [14] focuses on the publication of *principles*, rather than *requirements*, we provide sub-codes for the principle statement (Pp), namely *rationale* (Pp-R); implied *principle* (Pp-Pp) and *key action* (Pp-K).

Example: For a *prescriptor* that addresses the *research innovation* performance area, a *prescriptor* statement would be: *Research-collaboration with international researchers must be encouraged as part of the research process*. Thus, the *construct* that needs to change is the *research process*. The skilled enterprise engineer would reason that *research collaboration* is a performance area that requires two design principles for the *engineering department* as the design domain: (1) *Research output must be measured according to research impact* and (2) *The number of publications may not be used as a research output performance index*.

What it is NOT: A *prescriptor* is not a description of an existing problem, concern or performance area, but provides *prescriptive guidance* on transforming an enterprise *construct* into a future state. Some prescriptive statements (e.g., *late submissions should not be assessed*) guide actors on performing coordination acts [68]. These prescriptive statements should not be coded as *prescriptors*, since they do not provide design guidance for a future state design. Rather, they are *action rules* that apply during enterprise operation.

How to state: Documental evidence of a *prescriptor* should be converted into prescriptive form, using the words *should*, *must* or *may not*. An example of a correct statement: *Research output* ***must be*** *measured according to research impact.* Note that the phrase "must be" is useful to indicate that the prescriptor needs to be verifiable, according to [17].

Cues: Documental evidence may exist that elaborates on possible *changes* to *existing constructs*. Clauses that may provide cues for implications include "design of", "new design", and "need to change".

Code label: **Principle–Rationale (Pp-R) ╟Not to be used for documental text coding╢**

Condensed Definition: Providing justification(s) or motivation(s) for using the *principle* for enterprise design guidance.

Full Definition: Specified per *principle*. The *rationale* answers the question: **Why should the *principle* be used to guide design**? The rationale refers back to the *performance areas* that need to be addressed [14], indicating how a *principle* could contribute towards achievement of identified *performance area(s)*. A sub-code (Pp-R-A) should be used to indicate that the rationale refers to a specific *performance area*, where the "A" in"Pp-R-A" refers to the code label *Performance Area*.

Example: For a principle that addresses the *research collaboration* performance area with the principle statement *research output must be measured according to research impact* the *rationale* could be: *When researchers are measured on research impact, they will engage in collaborative research that is time-consuming, rather than doing individual research.* The stated principle will thus contribute towards the *research collaboration* performance area.

What it is NOT: The *rationale* does not provide additional guidance, but simply justifies the usefulness of the *principle* in guiding enterprise design in addressing *performance area(s)*.

Cues: Documental evidence may exist that elaborates or justifies the identified *prescriptor* in the form of argumentative clauses, such as "the reason for", "since", and "as a result".

Code label: **Principle–Implied Principle (Pp-Pp) ╟Not to be used for documental text coding╢**

Condensed Definition: A *principle* may imply one or more other *principles*.

Full Definition: Specified per main *principle*. The *implied principle(s)* are additional *principles* that can be associated with the main *principle*. The *implied principle(s)* are defined according to the code label *Prescriptor-Statement* - refer to the code label *Prescriptor-Statement* for a definition of a *prescriptor*.

Example: For a *principle* that addresses the *research collaboration* performance area with the principle statement *research output must be measured according to the research impact*, an *implied principle* is: *The number of publications may not be used as a research output performance index*.

What it is NOT: Implied *prescriptors* do not incorporate design-related actions, such as investigate, study, develop etc. Last-mentioned actions should be classified as *key actions* (P-K).

Code label: **Prinicple–Key Action(s) (Pp-K) ╟Not to be used for documental text coding╢**

Condensed Definition: Envisioned design actions to realise a *principle* during enterprise design.

Full Definition: Specified per *principle*. The key action(s) answer the question: **What design actions should be used to change existing constructions to new/re-designed constructions?** Key actions should be specified for every principle and are often encapsulated in "formal programs or projects, investigations, pilots and projects" [14]. They are *design - related actions, mechanisms and practices* used to transform *current (as-is)* enterprise constructions into *future (to-be)* constructions, without specifying the to-be constructions. Key actions precede constructional design and are formulated without yet knowing the chosen constsruct.

Example: For a *principle* that addresses the *research collaboration* performance area with the principle statement *research output must be measured according to research impact,* key actions could be:

(a) *Investigate different ways of calculating research impact*;

(b) *Consult with stakeholders for selecting an appropriate calculation formula for research impact*.

What it is NOT: Key actions are not activities associated with building/implementation. An inappropriate key action would be "Update the performance templates on the performance management software", since such an update already refers to a constructional solution.

Cues: Documental evidence may exist that elaborates on possible design activities. Clauses that may provide cues for actions include action verbs, such as "investigate", "study", "elicit", "define", "evaluate", "develop", and design".

5.3.3 Coding method

The following systematic and iterative process should be followed during coding:

1. Since not all *intention-related* concepts can be clearly distinguished from documental text alone, coders should use all codes except those ones highlighted for exclusion via the phrase: ╟Not to be used for documental text coding╢.
2. Read a paragraph of the document and then identify one or more *performance area(s)*. Wave-underline fragments that comply with the definition of *performance area*. Tag quotes with the same super-script identifier if the text refers to the same *performance area*, e.g. quality of assessment$^{\text{A1}}$ and the evaluation quality$^{\text{A1}}$ Follow additional guidance provided in the codebook on sub-codes for *performance area(s)*.
3. Read the paragraph for a second time and identify *prescriptors*. Dash-underline fragments that comply with the definition for *prescriptor*. Tag quotes with the same super-script identifier if the text refers to the same *prescriptor*, e.g. The module *engineering design*$^{\text{C1}}$ should incorporate more formative assessment$^{\text{P1-C1}}$ and Modules$^{\text{C2}}$ should include more formative assessment$^{\text{P1-C2}}$ Follow additional guidance provided in the codebook on sub-codes for *prescriptor(s)*.

6. Discussion and future work

Requirements elicitation is one of the most important phases in the design process and applied by many engineering disciplines. Enterprise Engineering (EE) is an emerging discipline that applies the design process to design the enterprise as an artefact [5]. Since multiple theorists and practitioners have developed structured *approaches* to address enterprise complexity [9], the EE discipline developed in a fragmented way, with enterprise design knowledge mostly encapsulated in several enterprise design *approaches* [10]. De Vries et al. [9] studied eight different enterprise design/alignment *approaches*, inductively developing a common framework to represent and compare these *approaches* in terms of four main components. One of the components represent the *scope of enterprise design/alignment* in terms of three dimensions: (1) *design domains*, (2) *intentions and constraints*, and (3) *enterprise scope*. De Vries [12] criticized existing approaches for their inconsistent means of defining the first dimension, namely the *design domains* and already provided guidance on demarcating *design domains* in a more consistent way.

Focusing on the second dimension i.e. *intentions and constraints*, a key criterion for defining design *intentions and constraints* is that *intentions* need to be defined in an unambiguous, understandable and verifiable way [17]. Within the EE discipline, we believe that a consistent representation of design *intentions* should reduce ambiguity. The study subsequently suggested a new method (i.e. the MEICC) with the main *solution objective* of reducing ambiguity, distinguishing between design *intention-related* concepts for an existing or new enterprise design approach.

We evaluated three different techniques for addressing the main *solution objective* of MEICC: (1) ontology-based conceptual modelling (OBCM), (2) focus group discussions and (3) using a codebook. Since *OBCM* requires extensive training, it was not incorporated during the development of MEICC. Regarding *focus group discussions*, we incorporated Krueger & Casey's [64] guidelines to reduce ambiguity when *systematic* and *participatory* focus group discussions are used to iteratively distinguish and refine design *intention-related* concepts of an enterprise design approach. A *codebook* was incorporated to *define*, *refine* and *document* the design *intention-related* concepts, removing possible interpretation discrepancies that may exist amongst different design team participants until intercoder-agreement is obtained.

Since Van der Meulen [37] already used a codebook to distinguish between *intention-related* concepts associated with Hoogervorst's [7] approach, we also demonstrated the MEICC using Hoogervorst's approach. Whereas Van der Meulen [37] involved a single participant to validate his codebook, we improved the rigour of the codebook, following an iterative approach for codebook refinement, whilst involving multiple participants. A *coding strategy* (including *coding conditions*, a *refined codebook* and a *coding method*), developed for Hoogervorst's approach via MEICC, was delivered as a secondary contribution.

6.1 Limitations of the study

We acknowledge possible *epistemological diversity* of DSR presented by Niehaves [47]. Following the *DSR methodology* genre in this study, as identified by Peffers et al. [49], we argue that this study demonstrates both a *positivist* and *interpretive* stance towards knowledge *development* and *evaluation*. This study illustrates our *positivist* stance, since the main objective is to produce objective and practical knowledge in the form of a method artefact, the MEICC, as the main knowledge contribution. Yet, our *interpretive* stance is reflected in *demonstrating* the MEICC. Within our *interpretive* epistemological position, we believe that the subjects that participated in the demonstration of the MEICC had an influence on the *coding strategy* that was developed for Hoogervorst's approach. We concur that the participants' understanding and interpretation of Hoogervorst's *intention-related* concepts also had an influence on the clarification of concepts and the construction of a *coding strategy* for Hoogervorst's approach. If a different group of participants were involved during the demonstration of the MEICC, a different *coding strategy* may have resulted.

Peffers et al.'s [49] *DSR methodology* genre focuses on the development of an applicable artefact, indicating that its *evaluation* should be outcome-oriented and practical. Based on inductive analysis of existing design science process elements from various disciplines, Peffers et al. [51] synthesized six possible phases for DSR: (1) Problem identification and motivation; (2) Objectives of a solution; (3) Design and development; (4) Demonstration; (5) Evaluation; and (6) Communication. Although Peffers et al. [51] indicate that the *demonstration* phase may be sufficient to evaluate an artefact, a separate *evaluation* phase may add additional rigor depending on the nature of the problem. Section 3.1 indicates that this study applied the *first four phases* defined by Peffers et al. [51]. In the subsequent paragraphs, we present possible limitations of the study, especially in terms of *phase five*, i.e. the *evaluation* phase.

The *positivist* may reason that the main limitation of the study is that the artefact's utility, quality and efficacy should be *evaluated* more rigorously (see Hevner et al.'s [44] third guideline on design evaluation). Yet, Hevner et al. [44] primarily provide guidance on evaluating an IT artefact, indicating the the IT artefact can be evaluated in terms of "functionality, completeness, consistency, accuracy, performance, reliability, usability, fit within the organization and other relevant quality attributes". Some of the stated measures may also be applicable to a non-IT artefact, such as the MEICC. Evaluating the MEICC in terms of *completeness* and *usability* will provide additional credibility to the practitioner.

The main limitation of the study may also be argued from an *interpretive* stance, indicating that *evaluation* of the artefact need to acknowledge the social setting of the evaluation environment and incorporate more iteration. Niehaves [47] suggests that two of the seven principles for interpretive field studies (see Klein & Myers [48]), are useful to guide the *demonstration and evaluation* phases of DSR. The *principle of hermeneutic cycle* indicates that human understanding depends on continuous iteration. Niehaves [47] suggests that additional completeness criteria need to be specified and used to guide the number of evaluation iterations. The *principle of contextualisation* indicates that the evaluation findings need to stipulate the social setting of the research and evaluation environment [47]. In addition, the study needs to indicate whether the findings are in some way applicable to other situations, also specifying additional criteria for these situations.

6.2 Future work

The limitations of the study indicate that future work is required to further *evaluate* the MEICC. Applying the *principle of hermeneutic cycle*, we suggest at least two additional evaluation iterations.

The first iteration should focus on *completeness* and *usability* measures, when the resulting *coding strategy* of Hoogervorst’s approach, developed via the MEICC, is evaluated within a real-world context. A study is underway to apply the *coding strategy* partially at a real-world enterprise where Hoogervorst’s approach has been adopted in practice. During enterprise design workshops, the *enterprise designer* will be applying the clarified concepts, stipulated in the *refined codebook* that forms part of the *coding strategy*. The *enterprise designer* will be reflecting on the *completeness* and *usability* of the *refined codebook*. Although we excluded ontology-based conceptual modelling (OBCM) as a means to clarify design *intention-related* concepts within MEICC, the evaluation feedback may indicate that we also experiment with OBCM.

For the second evaluation iteration, we suggest that the MEICC is evaluated for a different enterprise design approach, such as TOGAF, developed by the Open Group (see [25]). According to the *principle of contextualisation,* this iteration will be useful to discover additional criteria for using the MEICC when it is demonstrated within the TOGAF context.

7. References

1. Distanont A, Haapasalo H, Vaananen M, Letho J (2012) The engagement between knowledge transfer and requirements engineering. International Journal of Knowledge and Learning 1 (2):131-156

2. Bentley LD, Whitten JL (2007) Systems analysis and design for the global enterprise. 7th edn. McGraw-Hill/Irwin, New York

3. Dym CL, Little P (2009) Engineering design. 3rd edn. John Wiley & Sons, New York

4. Eggert RJ (2010) Engineering design. 2nd edn. High Peak Press, Idaho

5. Dietz JLG, Hoogervorst JAP, Albani A, Aveiro D et al (2013) The discipline of enterprise engineering. International Journal of Organisation Design and Engineering 3 (1):86-114

6. Boulding KE (1956) General systems theory: the skeleton of science. Management Science 2:197-207

7. Hoogervorst JAP (2018) Practicing enterprise governance and enterprise engineering - applying the employee-centric theory of organization. Springer, Berlin Heidelberg

8. Simon HA (1969) The sciences of the artificial. 3rd edn. MIT Press, Cambridge

9. De Vries M, Van der Merwe A, Gerber A (2017) Extending the enterprise evolution contextualisation model. Enterprise Information Systems 11 (6):787-827

10. Lapalme J (2012) Three schools of thought on enterprise architecture. IT Professional 14 (6):37-43. doi:10.1109/MITP.2011.109

11. De Vries M (2013) A classification schema for comparing business-IT alignment approaches. International Journal of Industrial Engineering Theory, Applications and Practice 20 (3-4)

12. De Vries M (2017) Towards consistent demarcation of enterprise design domains. In: De Cesare S, Frank U (eds) Advances in Conceptual Modeling. Springer, Switzerland, pp 91-100

13. Giachetti RE (2010) Design of enterprise systems. CRC Press, Boca Raton

14. Hoogervorst JAP (2009) Enterprise governance and enterprise engineering. Springer, Diemen

15. Hoogervorst JAP (2017) The imperative of employee-centric organizing and the significance for enterprise engineering. Journal of Organizational Design and Engineering 1 (1):43-58. doi:10.1007/s41251-016-0003-y

16. Kossiakoff A, Weet WN, Seymour S, Biemer SM (2011) Systems engineering principles and practice. 2nd edn. John Wiley & Sons, New Jersey

17. Douglas BP (2016) Agile Systems Engineering. Elsevier, https://app.knovel.com/hotlink/toc/id:kpASE00001/agile-systems-engineering/agile-systems-engineering

18. Djouab R, Abran A, Seffah A (2016) An ASPIRE-based method for quality requirements identification from business goals. Requirements Engineering 21:87-106

19. Horkoff J, Yu E (2013) Comparison and evaluation of goal-oreinted satisfaction analysis techniques. Requirements Engineering 18:199-222

20. Guizzardi RSS, Franch X, Guizzardi G, Wieringa RJ (2013) Ontologial distinctions between means-end and contribution links in the i* framework. In: Ng W, Storey VC, Trujillo J (eds) ER 2013, LNCS 8217. Springer-Verlag, Berlin Heidelberg, pp 463-470

21. Chung L, Nixon B, Yu E Using non-functional requirements to systematically select among laternatives in architectural design. In: Proceedings of 1st International Workshop on architectures for Software Systems, 1994. pp 31-43

22. Horkoff J, Yu E (2016) Interactive goal mode analysis for ealry requirements engineering. Requirements Engineering 21:29-61

23. Horkoff J, Yu E (2009) Evaluating goal achievement in enterprise modeling - an interactive procedure and experiences. In: Proceedings of 2nd IFIP WG 8.1 working conference on the practice of enterprise modeling (PoEm'09), LNBIB, vol 39. pp 145-171

24. Dimitrakopoulos G, Kavakali E, Loucopoulos P, Anagnostopoulos D et al (2019) The capability-oreinted modelling and simulation approach for autonomous vehicle management. Simulation Modelling Practice and Theory 91:28-47

25. The Open Group (2011) TOGAF 9.1. http://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html. Accessed 15 January 2019

26. Zachman JA (2009) The Zachman Framework for Enterprise Architecture^{TM}: A Primer for Enterprise Engineering and Manufacturing. http://zachmaninternational.com/index.php/home-article/15#maincol. Accessed 19 November 2009

27. Gharajedaghi J (2011) Systems thinking: managing chaos and complexity. 3rd edn. Elsevier, Burlington, USA

28. Zachman JA (2008) John Zachman's concise defintion of the Zachman Framework. https://zachman.com/about-the-zachman-framework. Accessed 3 April 2019

29. O'Rourke C, Fishman N, Selkow W (2003) Enterprise architecture using the Zachman Framework. Thomson Course Technology, Boston

30. Smith KL (2019) The complete pragmatic family of frameworks. http://www.pragmaticea.com/display-show.asp?ShowName=PragmaticFamily&ModelName=POET.Methods.Overview.Phases.Strategising#entry. Accessed 3 April 2019

31. Smith KL (2017) Enterprise DEBT: A pragmatic approach to enterprise transformation governance, V1.4. Pragmatic EA Ltd, Essex, England

32. Smith KL (2019) Conntecting the DOTS: The Death of "The Business & IT", V1.0. Pragmatic EA Lmt, Essex, England

33. Ferrari A, Spoletini P, Gnesi S (2016) Ambiguity and tacit knowledge in requirements elicitation interviews. Requirements Engineering 21:333-335

34. Berry DM, Kamsties E (2005) The syntactically dangerous all and plural in specifications. IEEE Software 22 (1):55-57

35. Gleich B, Creighton O, Kof L (2010) Ambiguity detection: towards a tool explaining ambiguity sources. Requirements engineering: foundation for software quality Lecture notes in computer science 6182:218-232. doi:http://dx.doi.org/10.1007/978-3-642-14192-8_20

36. Ashby WR (1958) Requisite variety and its implications for the control of complex systems. Cybernetica 1 (2):83-99

37. Van der Meulen T (2017) Towards a useful DEMO-based enterprise engineering methodology, demonstrated at an agricultural enterprise. Dissertation, University of Pretoria

38. Gause DC, Weinberg GM (1989) Exploring requirements: quality before design. Dorset House Publishing, New York

39. Dietz JLG (2006) Enterprise ontology. Springer, Berlin

40. Theuerkorn F (2005) Lightweight enterprise architectures. Auerbach Publications, New York

41. Garrett JJ (2011) The elements of user experience: user-centered design for the web and beyond. 2nd edn. New Riders Press, Berkeley

42. Kuechler W, Vaishnavi V (2008) The emergence of design research in information systems in North America. Journal of Design Research 7 (1):1-16

43. Winter R (2008) Design science research in Europe. European Journal of Information Systems 17:470–475

44. Hevner AR, March ST, Park J, Ram S (2004) Design science in information systems research. MIS Quarterly 28 (1):75-105

45. Owen C (1997) Design research: building the knowledge base. Journal of the Japanese Society for the Science of Design 5 (2):36-45

46. March ST, Smith G (1995) Design and natural science research on Information Technology. Decision Support Systems 15 (4):251-266

47. Niehaves B (2007) On Epistemological Diversity in Design Science - New Vistas for Design-Oriented IS Research? In: 28th International Conference on Infomration Systems. Montreal,

48. Klein HK, Myers MD (1999) A set of principles for conducting and evaluating interpretive field studies in information systems. MIS Quarterly 23 (1):67-94

49. Peffers K, Tuunanen T, Niehaves B (2018) Design science research genres: introduction to the special issue on exemplars and criteria for applicable design science research. European Journal of Information Systems 27 (2):129-139. doi:10.1080/0960085X.2018.1458066

50. Gregor S, Jones D (2007) The anatomy of a design theory. J Assoc Inf Syst 8 (5):312-335

51. Peffers K, Tuunanen T, Rothenberger M, Chatterjee S (2008) A design science research methodology for information systems research. Journal of MIS 24 (3):45-77

52. Gregor S, Hevner A (2013) Positioning and presenting design science research for maximum impact. MIS Quarterly 37 (2):337-355

53. Guest G, MacQueen KM, Namey EE (2012) Applied thematic analysis. Sage, Thousand Oaks, California

54. Siau K (2004) Informational and computational equivalence in comparing information modelling methods, 15 (1) (2004) 73–86. Journal of Database Management 15 (1):73-86

55. Sheer A-W, Hars A (1992) Extending data modelling to cover the whole enterprise. Communications of the ACM 35 (9):166-172

56. Wand Y, Weber RA (2002) Research commentary: information systems and conceptual modelling—a research agenda. Information Systems Research 13 (4):363-376

57. Karagiannis D, Mayer HC, Mylopoulos J (2016) Domain-specific Conceptual Modleing: Concepts, Methods and Tools. Springer, Switzerland

58. Grüninger M, Atefi K, Fox MMS Ontologies to support process integration in enterprise engineering. Computational & Mathematical Organization Theory 6 (4):381-394

59. Honderich T (2006) The Oxford companion to philosophy. Oxford University Press,

60. Corea C, Delfmann P (2017) Detecting compliance with business rules in ontology-based process modeling. In: Leimeister JM, Brenner W (eds) Proceedings der 13. Internationalen Tagung

Wirtschaftsinformatik (WI 2017). St. Gallen, pp 226-240

61. Wand Y, Weber RA (1993) On the ontological expressiveness of information systems analysis and design grammars. Information Systems Journal 3 (4):217–237

62. Verdonck M, Gailly F, Pergl R, Guizzardi G et al (2019) Comparing traditional conceptual modeling with ontology-driven conceptual modeling: An empirical study. Information Systems 81:92-103

63. Guizzardi G, Falbo RA, Guizzardi RSS (2008) Grounding software domain ontologies in the unified foundational ontology: The case of the ODE software process ontology. In: Proceedings XI Iberoamerican Workshop on Requirements Engineering and Software Environments, Recife, Brazil. pp 244-251

64. Krueger RA, Casey MA (2015) Focus groups: a practical guide for applied research. 5th edn. SAGE, Thousand Oaks

65. MacQueen KM, McLellan-Lemal E, Bartholow K, Milstein B (2008) Team-based codebook development: Structure, process, and agreement. In: Guest G, MacQueen KM (eds) Handbook for team-based qualitative research. AltaMira, MD: Lanham, pp 119-135

66. Saldana J (2009) The coding manual for qualitative researchers. Sage Publications, London

67. De Vries M, Gerber A, Van der Merwe A (2015) The enterprise engineering domain. In: Aveiro D, Pergl R, Valenta M (eds) Advances in Enterprise Engineering IX. Springer, Switzerland, pp 47-63

68. Perinforma APC (2015) The essence of organisation. Sapio, www.sapio.nl