Confirming Design Guidelines for Evolvable Business Processes Based on the Concept of Entropy

Peter De Bruyn, Dieter Van Nuffel, Philip Huysmans and Herwig Mannaert
Normalized Systems Institute (NSI)
Department of Management Information Systems
University of Antwerp
Antwerp, Belgium
{peter.debruyn, dieter.vannuffel, philip.huysmans, herwig.mannaert}@uantwerpen.be

***Abstract*—Contemporary organizations need to be agile at both their IT systems and organizational structures (such as business processes). Normalized Systems theory has recently proposed an approach to build evolvable IT systems, based on the systems theoretic concept of stability. However, its applicability to the organizational level, including business processes, has proven to be relevant in the past and resulted a.o. in a set of 25 guidelines for designing business processes. In subsequent work, the Normalized Systems theory was confirmed and extended based on the concept of entropy from thermodynamics. Therefore, this paper explores whether the guidelines which have been proposed for business processes from an evolvability point of view can be confirmed or extended from the entropy reasoning as well. More specifically, the validity of 9 business process design guidelines is investigated for this purpose. Our results indicate that the investigated guidelines are rather consistent among both approaches: guidelines required to attain evolvability seem to enable low entropy (i.e., complexity) and vice versa.**

***Keywords*—Business Processes; Complexity; Entropy; Normalized Systems.**

I. INTRODUCTION

Lack of organizational agility is often attributed to a lack of IT agility [1] as IT systems ensure the support or even automation of business processes. Consequently, organizational changes need to be reflected in both the business processes and their supporting information systems. This means that, instead of focusing solely on IT systems, attention for the design and agility of the business processes is needed as well. The explicit attention for the design of business processes emerged when the implicit work practices were automated using ERP systems [2]. It was recognized that the hard coding of the business processes in software packages resulted in a lack of adaptability of the processes [3]. As a result, the design of business processes gained a central role in organizations, separated from the design of information systems [2]. However, integration of business processes and information systems still needs to be achieved, and agility (or "evolvability") needs to be ensured on both levels.

Normalized Systems (NS) theory offers a theoretically founded way to design software systems which exhibit evolvability based on the systems theory's concept of stability, by proposing a limited set of design theorems [4], [5]. Applying the theory's rationale to the business process level has been shown feasible and resulted a.o. in a set of 25 guidelines for designing evolvable business processes [6]. In subsequent work, NS theory was confirmed and extended based on the concept of entropy from thermodynamics [7]. This extension resulted in additional theorems, while confirming the existing theorems. Therefore, it might be interesting to verify whether the guidelines which have been proposed for business processes can be confirmed or extended from the entropy reasoning as well. This paper explores this research area by applying the entropy reasoning to a set of business process guidelines (which were originally proposed to design evolvable business processes). First, we provide some theoretical background (Section II). Afterwards, the guidelines (Section III) and discussion (Section IV) are presented. Finally, our conclusions are offered in Section V).

II. THEORETICAL BACKGROUND

NS was introduced as a theoretically founded way for deterministically designing software architectures exhibiting a proven amount of evolvability. For this end, the systems theoretic concept of stability is applied [4], [5]. This implies that a bounded input function (e.g., "add data attribute") should result in bounded output values, even as time $T \rightarrow \infty$. It has been proven that at least four theorems should be consistently applied in order to obtain such evolvable software architecture [4], [5]. Violations against these theorems can be observed at compile-time [5].

Later on, the theory has been proven to be applicable to the design of evolvable business processes [6]. Here, business processes are considered at their most elementary level (i.e., the "elementary tasks and elementary sequencing and design of these tasks"). To obtain stability, it is required that changes to individual processes or tasks do not impact other processes or tasks [6]. In order to achieve such Normalized Business Processes (NSBPs), a set of 25 guidelines was developed, based on the four NS theorems [6].

In order to position this research, a clear distinction between the concepts evolvability and flexibility is necessary. Although flexibility also denotes a desired characteristic of business processes, as defined by e.g., [8]: '*the capability to implement changes in the business process type and instances by changing only those parts that need to be changed and keeping other parts stable*"; it differs from evolvability defined as the capability of a modular business process design to adapt to identified change drivers [6]. It also differs from the change patterns research, as that research focuses on how (operationally) processes should be changed to be flexible, whereas

 ISBN: 978-1-61208-304-9

this research focuses on why and how processes should be (structurally) designed in order to support change. Matching the flexibility types of Schonenberg [9], evolvability can be situated within the *Flexibility by Design* type. Nevertheless, designing evolvable business processes actually precedes flexibility as run-time (flexible) design decisions should comply with the requirements of evolvable business processes at design time.

In subsequent research, NS was extended based on the thermodynamic concept of entropy, initially focusing on software architectures again [7]. As entropy is generally associated with concepts as complexity, amount of disorder or available information, it enables the study of the diagnostability of a (software) system. In statistical thermodynamics, entropy is considered proportional to the number of microstates consistent with one macrostate (i.e., its multiplicity) [10]. The macrostate refers to the whole of externally observable and measurable (macroscopic) properties of a system, corresponding to visible output of a software system (e.g., loggings). The microstate depicts the whole of microscopic properties of the constituent parts of the system, such as binary values representing the correct of erroneous outcome of a task (i.e., a unit of processing of which we are interested in independent information about whether it has been executed properly). The higher the multiplicity, the more difficult it becomes to identify the precise origin of an observed error. This approach requires a run-time view of the system [7]. To design information systems exhibiting low entropy, two NS theorems have been confirmed, while two additional theorems were proposed as well [7].

This entropy viewpoint can be applied to business processes as well [11], [12]. Again, a business process is considered to be a flow (i.e., including sequences, selections and iterations) of tasks which perform actions on one or more information objects. Considering their execution allows us to define macrostates and microstates on this level as well. The individual values of, for example, the throughput times of all task instantiations correspond to a microstate. The macrostate of a business process is the (aggregated) information available for an observer (e.g., the total troughput time). Multiple microstate configurations consistent with one macrostate (i.e., multiplicity > 1), makes entropy (and the experienced complexity during diagnostics) increase, and typical management questions more difficult to answer. For instance, it becomes unclear which task or tasks in the business process was (were) responsible for the extremely slow (fast) completion (of this particular instance) ofthe business process

No specific guidelines on how to reduce entropy on this level have been formulated yet. Similar to the software level, it is hypothesized that guidelines to achieve stable business processes might reduce entropy as well. As a first step, we assess in this paper the entropy-reducing capability of the first nine available guidelines of Van Nuffel [6]. More specifically, we investigate whether a violation of each guideline increases the multiplicity (and hence, entropy) of business processes.

III. Comparison of guidelines rationales

In this section, we will systematically investigate the first 9 guidelines as proposed by the work of Van Nuffel [6]. For each guideline, we will first provide a brief description. Next, we explore whether not adhering to this guideline would imply an increase in entropy as we defined it earlier. Guidelines of which violations result in additional entropy are then considered to be suitable for entropy control as well.

The first guideline, "**Elementary Business Process**", requires that a business process should be operating on *one and only one* Information Life Cycle Object (ILCO) [6, p. 107]. Not adhering to this guideline would imply a design in which a business process could be operating on multiple ILCOs. For instance, consider both invoicing and manufacturing steps which are mixed up and interacting in one process, and a problem with the total throughput time of finishing invoices is present. At least two situations in which multiplicity > 1 (and entropy arises), can now occur. First, as the business process is concerned with operations on multiple ILCOs, the problematic throughput time of the invoicing steps can be "compensated" by "normal" throughput times of the manufacturing steps. Consequently, the problematic total throughput time of the invoicing activities would not necessarily raise an "alert", even after for instance hypothesis testing on the overall observed mean versus expected mean. Therefore, multiplicity > 1 (and entropy increases): the status reflected by the macrostate (e.g., no problems are reported ("OK")), is conform to multiple microstates (e.g., both "OK" or "Not OK" for the throughput time of the invoicing steps). Further, not demanding that business processes operate on a single information object, also implies that multiple business processes can be operating (unconsciously) on identical information objects (i.e., duplication and copy/paste might occur). Therefore, chances that the problematic total throughput time of the invoicing activities would raise an "alert" become even smaller, as the information on this concern is not properly separated. This situation correlates with our (reduced) observability interpretation of entropy as pointed out in Section II. Second, in case a problem is observed (i.e., the macrostate signals "Not OK"), multiplicity > 1 as well. Indeed, the macrostate conforms to multiple microstates: the "Not OK" result of the total throughput time might be related to the manufacturing steps, the invoicing steps or both. In order to diagnose the problem unambiguously, the process owner should disentangle all steps in the business process, determine the ILCO they belong to, and analyze to which ILCO the overall problem is actually related. Further, we already noted that not demanding a business process to operate on a single information object might result in multiple business processes operating (unconsciously) on identical information objects (i.e., duplication and copy/paste might occur). If the macrostate of multiple business processes (each implementing (duplicate) invoicing steps) goes to "Not OK", chances of identifying "the invoice" as the problematic concern become even smaller, as the information on this issue is not properly separated. This situation correlates with our (reduced) diagnostability interpretation of entropy as pointed out in Section II. Based on these two situations, we can conclude that not adhering to this guideline implies an increased amount of entropy in the business process instantiation space. Therefore, we state that the guideline is suitable for entropy control as well.

The second guideline, "**Elementary Life Cycle Information Object**", defines a *LCIO as an information object not exhibiting state transparency* [6, p. 114]. Combined with guideline 1 this implies that a business process is related to

one information object not exhibiting state transparency. In this context, an information object is considered state transparent if it adheres to the NS Separation of States principle and the object has no proper state transitions which should be made explicit [6, p. 118]. Not adhering to this guideline would imply two possible situations: (1) the identification of an information object as a LCIO when it already exhibits state transparency, or (2) not recognizing a non-state transparent information object as a LCIO. Regarding the first situation, the creation of an additional LCIO (and a corresponding business process) for an information object of which the states are already fully reflected by another LCIO, does neither increase of decrease entropy. Indeed, no additional information regarding the microstate configuration is retained or lost (the information regarding the states of one particular LCIO instance is simply duplicated) by identifying this additional LCIO. Regarding the second situation however, an information object not exhibiting state transparency which does not get recognized as a LCIO, will generate an increase in the degree of entropy (i.e., multiplicity > 1). Indeed, as in such case no state transparency regarding the concerning information object is attained, information about its state transitions (and hence, the microstate configuration) is lost. Expressed differently, a multiplicity > 1 will arise during and after execution-time as the macroscopic observations regarding this information object cannot be traced to individual tasks represented by states (i.e., a myriad of microstates are possible). This situation correlates with both our (reduced) observability and diagnostability interpretations of entropy as pointed out in Section II. Consequently, this guideline is not strictly necessary to control entropy in the context of the first situation: theoretically speaking, a state transparent information object can be identified as a LCIO without increasing entropy (albeit without any thinkable benefit). However, the second situation shows that not adhering to this guideline can imply an increased amount of entropy in the business process instantiation space when a non-transparent information object is not recognized as a LCIO. Therefore, we state that the guideline is largely suitable for entropy control and advice its application for this purpose as well. We would further like to add that this guideline actually quite nicely illustrates the core reasoning of designing business processes based on the entropy rationale: for every task of which separate information might be valuable (constituting a so-called "information unit"), a separate state should be defined and related to the information object it is operating on. Therefore, each information object not exhibiting state transparency should be considered as a LCIO, thereby storing information of each individual task performed on it, at its most fine-grained level.

The third guideline, "**Aggregated Business Process**", states that in order to represent an aggregated business process, an aggregated LCIO has to be introduced (p. 121). This guideline relates to the fact that certain aggregated business processes might be necessary to several reasons. First, the orchestration of different business processes (each operating on a single LCIO) by a distinct business process might be necessary. For instance, consider an Order-to-Cash process in which several sub-processes —such as "order entry process", "procurement process", "production process", etcetera— are each individually and successively called, waiting for completion, upon which the next (set of) sub-process(es) is called, completed, etcetera. Second, different (both internal or external) stakeholders might require different perspectives (such as aggregations) due to, for instance, their own functional domain. For instance, in case of very complex business processes, one can imagine that clients or certain actors at a higher management level might be primarily interested in the mere "milestones" (e.g., "order received", "order produced", "order shipped") of a business process, instead of the possible hundreds of more fine-grained states the product might be in during its lifecycle. The guideline under consideration prescribes that such *aggregated processes may only be introduced for orchestrating purposes and in case the business processes under consideration are not able to be designed solely based on guidelines 1 and 2*. Once more, not adhering to this guideline would imply two possible situations: (1) designing an aggregated business process while a redesign based on guidelines 1 and 2 would be possible, or (2) not recognizing a business process for orchestrating purposes while a redesign based on guidelines 1 and 2 is not possible. The first situation would clearly imply an unnecessary combination of two concerns and therefore a violation of guidelines 1 and 2 (as a redesign based on them is still possible). Given the fact that both guidelines were proven to mostly result in an increase of entropy when not adhered to, this situation would equally result in an increase of entropy. The second situation would lead to not recognizing a "combined concern": while each of the underlying concerns have their own LCIO and corresponding business process, the orchestration or "interfacing" between them might constitute a genuine concern as well. This orchestration might entail a relevant information unit and therefore necessary to keep track of when one's aim is to minimize entropy. Imagine an Order-to-Cash process tracking the Order Entry Process, (possibly multiple) Procurement Processes, Production Processes, Delivery Processes, etcetera. While each of these processes clearly designate their own LCIO and therefore, business process, the orchestration between them is crucial to be monitored as well. Indeed, tracking interfacing issues in this Order-to-Cash Process constitutes relevant information (macroscopically) and in case a customer complains about a lately delivered order (i.e., the macrostate), the specific business process (instance) which is causing this delay (Order Entry, Procurement, etcetera) should be identifiable (i.e., the specific microstate) Not identifying the necessary aggregated process would therefore lead to multiple microstates consistent with one macrostate. This situation correlates with both our (reduced) observability and diagnostability interpretations of entropy as pointed out in Section II. We can therefore conclude that not adhering to this guideline implies an increased amount of entropy in the business process instantiation space and state that the guideline is suitable for entropy control as well.

Guideline 4, "**Aggregation Level**", requires that *tasks performed on a different aggregation level should denote a separate business process* (p. 124). An "aggregation level" in this particular guideline is mainly to be understood as focusing on the multiplicities of different information objects (i.e., the different perceived aggregations). For instance, a typical Order within a company might be conceived as being associated with several Product processes, where this Product process at its turn might then again be associated with multiple Part processes. Not adhering to this guideline would imply that it is possible for a business process to execute sequences of

 ISBN: 978-1-61208-304-9

tasks situated at different "aggregation levels". Suppose one business process performing a sequence of tasks on a "parent" information object (e.g., "Product") and sequences of tasks on its "child" information objects (e.g., different "Part" instances). As one could argue that such business process is operating on multiple LCIOs, our first two arguments are highly parallel to those of guideline 1. First, such business process design would not guarantee that systematic problems regarding, for instance, the overall throughput time of the sequence of tasks performed on the "child" information object are observed. Indeed, they might become "compensated" by "normal" throughput times of the other tasks, therefore not necessarily raising an "alert" to the observer. Hence, multiplicity > 1 (and entropy increases): multiple microstates ("throughput times OK" and "throughput times Not OK") are consistent with one macrostate ("no problems are reported"). This situation correlates with our (reduced) observability interpretation of entropy as pointed out in Section II. Second, in case a problem is observed (i.e., the macrostate signals "Not OK"), multiplicity > 1 as well. Indeed, the macrostate conforms to multiple microstates: the "Not OK" result of the overall process might be related to the sequence of tasks performed on the "parent" information object, the "child" information object or both. This situation correlates with our (reduced) diagnostability interpretation of entropy as pointed out in Section II. Third, no instance traceability regarding the multiple processed Parts within the single business process seems feasible in such design. Therefore, the same states regarding the "child" information object sequence are activated several times during the execution of the business process. This makes adequate state-tracking (cf. guideline 2) impossible. As a result, the business process owner cannot make the distinction between situations in which the problematic throughput time might be associated with all Part instances in general (i.e., a "systematic" recurring problem) or with one Part instance in particular (and in such case, which specific Product instance). Also in this third situation, this implies multiplicity > 1: one macrostate (i.e., a problem is observed) is consistent with multiple microstate (i.e., the problem is due to Part instance 1, or 2, ..., or all Part instances): certain parts of the microstate configuration are simply not captured during process execution. Based on these two situations, we can conclude that not adhering to this guideline implies an increased amount of entropy in the business process instantiation space. Therefore, we state that the guideline is suitable for entropy control as well.

Guideline 5, "**Value Chain Phase**", states that the *follow-up of an organizational artifact resulting from a value chain phase should denote a different business process* (p. 132). A value chain phase refers to the rather generic, often recurring structure and parts within aggregated business processes in manufacturing organizations (e.g., Order Entry, Procurement, Production, etcetera), such as for instance described by the SCOR reference model. Not adhering to the above described guideline could lead to the following two situations: (1) the steps related to these value chains are incorporated into the aggregated (i.e., orchestrating) business process, or (2) no more grained steps related to each of these value chain phases are discerned and no states regarding them is kept. In the first situation, this would imply a violation of guidelines 1 as multiple LCIOs (e.g., Order Entry, Procurement, Procurement) are combined into one business process. Further, guideline 4 would be violated as well because most often, these value chain phases have one-to-many or many-to-many relations. Indeed, a Customer Order can typically be related to multiple Purchase Orders and/or Production Orders. The second situation would imply violations regarding guidelines 2 (i.e., no LCIO is identified for several non-state transparent information objects) and 3 (i.e., an aggregated business process is designed when there are still some opportunities for redesign based on guidelines 1 and 2). A situation in which no relevant states regarding the tasks constituting a value chain phase should be identified, seems rather unlikely as this would allow to model almost all necessary activities of a typical manufacturing company within one business process having 5 to 8 tasks. Consequently, as we should earlier how violations regarding guidelines 1 to 4 result in multiple microstates consistent with one macrostate, we can conclude that violating this guideline would generate a multiplicity > 1 as well. Therefore, we state that the guideline is suitable for entropy control as well.

Guideline 6, "**Attribute Update Request**", states that *a task sequence to update an attribute of a particular LCIO that is not part of its business process scenarios, is represented by an Attribute Update Request business process* (p. 135). This guideline is subject to two specific conditions. First, it has to concern an update operation for which one single functional task is not sufficient to complete the update request, but rather a sequence (i.e., "process") of activities is required. Second, it concerns update requests which are not part of a branch within the regular business process scenarios. Consequently such procedures can be instantiated several times and during several different "states" of the lifecycle of the information object regarding which the update request is actually aimed at. Additionally, such process (verifying for instance the validity of updating a certain information object attribute with a certain new value) will typically differ for each individual attribute. Not adhering to this guideline would imply that tasks for handling an attribute update request, not part of the regular business process scenario, becomes incorporated into the flow of the LCIO of which the attribute is requested to be update. Again, such situation can be seen as a violation regarding several of the above mentioned guidelines. Indeed, not separating such task sequences would lead to a business process operating on multiple ILCOs and —at the same time— one concern being dispersed over several places within one business process (i.e., all the life cycle states in which the update request is allowed), thereby violating guideline 1. Second, the design would make the proper tracking of states impossible as at any point of the business process execution (thereby indirectly violating guideline 2) as each time an update request is initiated, the state of the regular business process is suddenly (possibly repeatedly) changed to states regarding this update request. Third, as attribute update requests can be performed several times during one instance of the "parent" business process, both concerns relate in a one-to-many multiplicity, thereby violating guideline 4. Consequently, as we showed earlier how violations regarding guidelines 1, 2 and 4 result in multiple microstates consistent with one macrostate, we can conclude that violating this guideline would generate a multiplicity $>$ 1 as well. Therefore, we state that the guideline is suitable for entropy control as well. Indeed, from an organization diagnostics (i.e., entropy) viewpoint, it clearly makes sense to separate such sequence of tasks for future reference. For

 ISBN: 978-1-61208-304-9

instance, the calculation of certain measures and the solution for certain managerial questions such as: "how often are such requests accepted/denied and for which reason" or "can we see any relation between the outcome of the update requests and its input values" are only able to be solved in an efficient way when this task sequence is properly separated in its own business process module and not unconsciously repeated in other places throughout the business process repository.

Guideline 7, **Actor Business Process Responsibility**, states that *tasks, of which the task allocation genuinely belongs to a different business process owner, should be designed into a separate business process* (p. 139). This guideline only applies in very stringent cases. For example, in case legislation or internal audit rules prescribe that different owners should be responsible for other (parts of) task sequences, this guideline applies. Mostly, the guideline is applicable when different parts of a task sequence are performed by different organizations. In such cases, the respective task allocations are logically situated at one of these different organizations as well. From an entropy viewpoint, let us consider the case in which the mentioned guideline is not adhered to. In such case, a business process could consist of a combination tasks which belong to genuinely different business process owners. Each task still has an attribute regarding which actor is allowed or required to perform the task. However, no information is available regarding who is doing the task allocation (e.g., the manager of organization who determines who is doing what). If such information should be retained, the appropriate level seems to be the business process level, as it concerns a sequence of multiple tasks. In case this information is relevant but however no distinct business process would be designed, a multiplicity > 1 (and hence, entropy) arises as one macrostate (e.g., a problem regarding the overall process) complies with multiple microstates (was the task allocation responsibility situated at person A, B, or C?). This situation correlates with our (reduced) diagnostability interpretation of entropy as pointed out in Section II. Therefore, in case the information regarding task allocation responsibility is relevant, a different business process should be identified from an entropy viewpoint to allow for this task allocation responsibility to be traceable. Indeed, this guideline calls to create an additional level of "process responsibility" (i.e., who allocates tasks among different actors and takes responsibility that they are carried out adequately), in addition to the responsibility for one or multiple tasks. Therefore, we state that the guideline might be suitable for entropy control as well. However, in line with the work of Van Nuffel [6] we stress that identifying additional business processes based on this guideline should be done with extreme precaution to avoid unnecessary additional business processes and, hence, only in cases where a different task allocation responsibility is relevant for diagnostability purposes.

Guidelines 8 and 9 as proposed by Van Nuffel [6], propose two specific business process types to be identified. Guideline 8, "**Notifying Stakeholders**" states that the *communication of a message to stakeholders (in the correct format, incorporating fault handling, etcetera) constitutes a distinct business process* (p. 143). Guideline 9, "**Payment**" states that the *payment of a particular amount of money to a particular beneficiary should equally constitute a distinct business process* (p. 146). Not recognizing these two concerns as distinct business processes could again create two possible situations: (1) integrating the tasks for the notification and payment in other business processes or (2) not specifying their constituting tasks at all. It is clear that the first situation would violate guideline 1 (multiple ILCOs operating within one business process) and 4 (for example, multiple notifications can be sent within the scope of one "parent" business process instantiation). The second situation would violate guideline 2 as a non-state transparent information object is not identified as a separate LCIO. Consequently, as we showed earlier how violations regarding guidelines 1, 2 and 4 result in multiple microstates consistent with one macrostate, we can conclude that violating this guideline would generate a multiplicity > 1 as well. Therefore, we state that guideline 8 and 9 are suitable for entropy control as well. Obviously, designing these task sequences as separate business processes is useful from an organizational diagnostics (i.e., entropy) viewpoint as. Indeed, both the payment of a particular amount in a particular format to a particular beneficiary at the right time, as well as communicating a certain message in a particular format at the right time while maintaining integrity, are often recurring functionalities within typical business processes. As a consequence, due to their frequently occurring nature, a business process owner would typically be interested in certain characteristics of each of these separately recurring tasks sequences: how long do they take to execute, how many times do they result in an error, etcetera. Focusing on these aspects might generate considerable efficiency gains as, for instance, improving the quality metrics or throughput time of the payment process with 5% might entail huge organizational effects as the changes are "expanded" throughout the whole organization. However, these analyses and improvements can only be performed when "payments" and "notifications" are designed into separate business processes. Otherwise, systematic problems regarding one of the concerns might not be noticed (cf. the observability issue of Section II) or might not be unambiguously traced to the right concern (cf. the diagnostability issue of Section II)

IV. Discussion, limitations and future research

This paper aims to contribute to our research line on how to prescriptively design business processes regarding certain criteria (such as low complexity and high evolvability). In earlier work, a set of prescriptive guidelines has been proposed from the stability perspective, and the applicability of entropy to study process complexity has been reported. The main focus in this paper was to verify whether the already existing guidelines from the stability viewpoint align with this entropy reasoning [12], [11]. Due to page limitations, we were only able to investigate a small subset of the guidelines of Van Nuffel for this purpose [6]. We found that most of the investigated guidelines are rather consistent among both approaches: guidelines required to attain evolvability seem to enable low complexity and vice versa. A small exception was noticed for guidelines 2 and 7. Regarding the former, it was observed that —theoretically— entropy does not increase when a state transparent information object is identified as a LCIO. Regarding the latter, it was argued that the application of the specific guideline should be performed even more thoughtfully and exceptionally from an entropy viewpoint as its necessity in many situation seems not really compelling.

This consistency might seem surprising, since the evolvability analysis focuses on the mere *design-time* of business

 ISBN: 978-1-61208-304-9

processes, which means that the harmful effects its aims to resolve (the so-called "combinatorial effects") are situated on this perspective: a functional change which causes N changes in the business process design. In contrast, the complexity analysis focuses on avoiding harmful effects during *execution-time*: a multiplicity > 1 (which we could coin as an "uncertainty effect") only manifests itself when the business processes are executed. While these effects are caused by choices made at design-time, this distinction illustrates the need for more insight at the execution-time of business processes. Current business process modeling notations (e.g., BPMN) focus primarily on design-time models. Moreover, the criteria both approaches use to delineate and identify the different business processes and their constituting tasks, differ. The evolvability approach employs the concept of "*change drivers*" (i.e., parts within the business process design which are assumed to change independently) to identify and isolate concerns, whereas the complexity approach employs the concept of "*information units*" (i.e., these parts within the business process design of which independently traceable information is assumed to be needed later on). Since most of the stability-related guidelines largely align with our entropy reasoning, we might conclude that the concerns which should be used to delineate and identify business processes or tasks are determined by the union of "change drivers" and "information units". Given the additional, more in-depth analysis of the entropy approach by incorporating the execution-time perspective (e.g., the importance of traceability), additional concerns which do not seem to be necessary from the evolvability perspective, might indeed be potentially identified in future research. Moreover, this preliminary analysis is limited to the first nine guidelines of Van Nuffel [6], and future research should elaborate on the consistency of other guidelines.

Notwithstanding the limitations and need for future research, this paper can claim a number of contributions. First, we further contributed to the enterprise and business process engineering field by elaborating on the usefulness to take an entropy perspective for studying the complexity of business processes. Second, we validated the suitability of a set of (already existing) business process design guidelines in this context as a first step towards a Design Theory [13]. In literature, it is generally acknowledged and even encouraged that such design efforts are guided by principles from related scientific fields (i.e., "kernel theories") [14], such as the concept of entropy from thermodynamics. Third, Design Science research acknowledges logical reasoning as one the possible evaluation methods in design science [15]. Therefore, next to our efforts performed in earlier work, this paper constitutes an additional validation base for the applicability of (a part of) the guidelines of Van Nuffel [6].

V. Conclusion

Contemporary organizations need to be agile regarding both their IT systems and organizational structures (such as business processes). Normalized Systems theory has recently proposed an approach to build evolvable IT systems, based on the systems theoretic concept of stability. However, its applicability to the organizational level, including business processes, has proven to be relevant in the past and resulted a.o. in a set of 25 guidelines for designing business processes. This paper investigated the validity of 9 of these guidelines from another theoretical perspective, more specifically, entropy from thermodynamics. We concluded that the investigated guidelines are rather consistent among both approaches: guidelines required to attain evolvability seem to enable low complexity (i.e., entropy) and vice versa. However, future research is definitely needed in this domain: for instance, 14 guidelines are still to be investigated and additional guidelines might potentially be investigated from the entropy perspective as well.

Acknowledgment

P.D.B. is supported by a Research Grant of the Agency for Innovation by Science and Technology in Flanders (IWT).

References

[1] E. Overby, A. Bharadwaj, and V. Sambamurthy, "Enterprise agility and the enabling role of information technology," *European Journal of Information Systems*, vol. 15, no. 2, pp. 120–131, 2006.

[2] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7pmg)," *Inf. Softw. Technol.*, vol. 52, no. 2, pp. 127–136, Feb. 2010.

[3] L. Brehm, A. Heinzl, and M. Markus, "Tailoring erp systems: a spectrum of choices and their implications," in *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 2001.

[4] H. Mannaert, J. Verelst, and K. Ven, "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability," *Science of Computer Programming*, vol. 76, no. 12, pp. 1210–1222, 2011.

[5] ——, "Towards evolvable software architectures based on systems theoretic stability," *Software: Practice and Experience*, vol. 42, no. 1, pp. 89–116, January 2012.

[6] D. Van Nuffel, "Towards designing modular and evolvable business processes," Ph.D. dissertation, University of Antwerp, 2011.

[7] H. Mannaert, P. De Bruyn, and J. Verelst, "Exploring entropy in software systems : towards a precise definition and design rules," in *The Seventh International Conference of Software Engineering Advances (ICSEA)*, 2012, pp. 84–89.

[8] S. P. Regev, G. and R. Schmidt, "Taxonomy of flexibility in business processes," in *Proceedings of the 7th Workshop on Business Process Modelling, Development and Support (BPMDS'06)*, 2006.

[9] M. R. R. N. M. N. Schonenberg, H. and W. van der Aalst, "Process flexibility: A survey of contemporary approaches," in *Advances in Enterprise Engineering I*, 2008, pp. 16–30.

[10] L. Boltzmann, *Lectures on gas theory*. Dover Publications, 1995.

[11] P. De Bruyn, P. Huysmans, H. Mannaert, and J. Verelst, "Understanding entropy generation during the execution of business process instantiations: An illustration from cost accounting," in *Advances in Enterprise Engineering VII*, ser. Lecture Notes in Business Information Processing, H. Proper, D. Aveiro, and K. Gaaloul, Eds. Springer Berlin Heidelberg, 2013, vol. 146, pp. 103–117.

[12] P. De Bruyn, P. Huysmans, G. Oorts, and H. Mannaert, "On the applicability of the notion of entropy for business process analysis," in *Proceedings of the second international symposium on Business Modeling and Software Desgin (BMSD2012)*, B. Shishkov, Ed., 2012, pp. 128–137.

[13] S. Gregor and D. Jones, "The anatomy of a design theory," *Journal of the Association for Information Systems*, vol. 8, no. 5, pp. 312–335, 2007.

[14] J. Walls, G. Widmeyer, and O. El Saway, "Building an information system design theory for vigilant eis," *Information Systems Research*, vol. 3, no. 1, pp. 36–59, 1992.

[15] F. Müller-Wienbergen, O. Müller, S. Seidel, and J. Becker, "Leaving the beaten tracks in creative work - a design theory for systems that support convergent and divergent thinking," *Journal of the Association for Information Systems*, vol. 12, no. 11, pp. 714–740, 2011.

 ISBN: 978-1-61208-304-9