On the Variability Dimensions of Normalized Systems Applications: Experiences from Four Case Studies

Peter De Bruyn, Herwig Mannaert and Philip Huysmans

Department of Management Information Systems
Faculty of Applied Economics
University of Antwerp, Belgium
Email: {peter.debruyn,herwig.mannaert,philip.huysmans}@uantwerp.be

Abstract—**Normalized Systems Theory aims to create software systems exhibiting a proven degree of evolvability. While its theorems have been formally proven and several applications have been used in practice, no real overview of the typical types or dimensions along which such Normalized Systems software applications can evolve is present. Therefore, this paper presents several cases in which its different variability dimensions are illustrated. Based on these cases, a more general overview of four variability dimensions for Normalized Systems software applications is proposed: changes regarding the application model, expanders, craftings and technological options.**

Keywords*–*Evolvability; Normalized Systems; Variability dimensions; Case Study

I. INTRODUCTION

This paper extends a previous paper which was originally presented at the EMPAT track on evolvable modularity patterns at the PATTERNS conference 2018 [1].

The evolvability of information systems (IS) is considered as an important attribute determining the survival chances of organizations, although it has not yet received much attention within the IS research area [2]. Normalized Systems Theory (NST) was proposed as one theory to provide an ex-ante proven approach to build evolvable software by leveraging concepts from systems theory and statistical thermodynamics [3]–[5]. The theory prescribes a set of theorems which are necessary conditions to obtain evolvable software and proposes a set of patterns to generate significant parts of software systems which can obey to these theorems. While it has been suggested that software created in this way exhibits evolvability, the main dimensions of evolvability or variability facilitated by the theory have nevertheless not yet been thoroughly discussed. Additionally, while some NST cases have been documented in extant literature [6]–[10], the overall number of cases is still fairly limited and their analysis has never been focused on the different dimensions of evolvability which were possibly present. This paper attempts to tackle both mentioned gaps by first discussing the case of a new (i.e., not previously documented) NST software application, which was built and used for the management of process evaluations of master dissertations at the faculty of the authors. Based on our experiences with this case, we will document variations that occurred along several dimensions: the model (business entities) of the application, the craftings (customizations on top of the generated code), the technology used and the version of the code generators themselves. Next to this new case, we will reinterpret several previous cases which were documented earlier in other work [7]–[10]. So while the cases themselves are not new, our perspective and way of analyzing the cases is. It is also the discussion of these additional (previously reported) cases in this evolvability dimensions context which is the main addition of this paper when compared to our initial contribution at PATTERNS 2018 [1]. In this way, we are able to identify and discuss the different dimensions along which variations in an NST application can arise and provide illustrations for each of them from different cases and examples. Consequently, these dimensions are also important indications with respect to the main areas in which an NST application can evolve throughout time.

The remainder of this paper is structured as follows. In Section II, we briefly present NST as the theoretical basis on which the considered software applications were built. Section III provides some general context regarding the newly reported educational case as well as its analysis in terms of evolvability dimensions. Section IV focuses on some of the earlier presented cases but analyzes them from a different angle than before, i.e., also in terms of evolvability dimensions. We offer a discussion in Section V and our conclusion in Section VI.

II. NORMALIZED SYSTEMS THEORY

The case applications we will present and analyze in the following sections, are based on NST. This theory has been previously formulated with the aim of creating software applications exhibiting a proven amount of evolvability [3]–[5]. More specifically, the goal is to eliminate the generally experienced phenomenon in which software systems become more difficult to maintain and adapt as they become bigger and evolve throughout time [11].

NST is theoretically founded on the concept of *stability* from systems theory. Here, stability is considered as an essential property of systems. Stability means that a bounded input should result in a bounded output, even if an unlimited time period is considered. In the context of information systems, this implies that a bounded set of changes should only result in a bounded impact to the information system, even in cases where an unlimited time period and growth of the system is taken into account (i.e., considering an unlimited systems evolution). Put differently, it is demanded that the impact of changes to an information system should not be dependent on the size of the system to which they are applied, but only on the size and property of the changes to be performed. Changes dependent on the size of the system are called *combinatorial*

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

effects. It has been formally proven that any violation of any of the following *theorems* will result in combinatorial effects (thereby hampering evolvability) [3]–[5]:

- *Separation of Concerns*, stating that each concern (i.e., each change driver) needs to be separated from other concerns in its own construct;
- *Action Version Transparency*, stating that an action entity should be able to be updated without impacting the action entities it is called by;
- *Data Version Transparency*, stating that a data entity should be updateable without impacting the action entities it is called by;
- *Separation of States*, stating that all actions in a workflow should be separated by state (i.e., being called in a stateful way).

The application of the theorems in practice has shown to result in very fine-grained modular structures within a software application. Such structure are, in general, difficult to achieve by manual programming. Therefore, NST proposes five *elements* (action, data, workflow, connector and trigger) that serve as design patterns [4], [5]:

- *data element*: a set of software constructs encapsulating a data construct (including a set of convenience methods, such as get- and set-methods, and providing remote access and persistence), allowing data storage and usage within an NST application;
- *action element*: a set of software constructs encapsulating an action construct (providing remote access, logging and access control), allowing the execution of (units of) processing functionality within an NST application;
- *workflow element*: a set of software constructs allowing the execution of a sequence of action elements (on a specific data element) within an NST application;
- *connector element*: a set of software constructs enabling the interaction of an NST application with external systems and users in a stateful way;
- *trigger element*: a set of software constructs enabling the triggering of action elements within an NST application, based on error and non-error states.

Based on these elements, NST software is generated in a relatively straightforward way through the use of the *NST expansion mechanism*. First, a model of the considered universe of discussion is defined in terms of a set of data, action and workflow elements. Next, NST expanders generate parameterized copies of the general element design patterns into boiler plate source code. Several layers can be discerned in this code: a shared layer (not containing any reference to external technologies), data layer (taking care of data services), logic layer (taking care of business logic and transactions), remote or proxy layer (taking care of remote access), control layer (taking care of the routing of incoming requests to the appropriate method in the appropriate class in the proxy layer) and view layer (taking care of presenting the view to be rendered by the user interface, such as a web browser). This generated code can, if preferred, be complemented with *craftings* (custom code) to add non-standard functionality that is not provided by the expanders themselves at well specified places (anchors) within the boiler plate code. The boiler plate code together with the optional craftings are then compiled (built) so that the application can be deployed.

III. Educational case

In this section, we will first introduce the educational case in-depth. Next, we analyze the case both in general and along several potential evolvability dimensions.

A. Case introduction

The new case we present in this paper is situated within an educational context and concerned with the master thesis evaluations at the Faculty of Applied Economics of the University of Antwerp. At the university, master students writing their dissertation are not only evaluated with regard to the end result (i.e., the thesis itself) but also (for a minor part) with regard to the process they go through in order to arrive at that end result (e.g., their communication and reporting skills, problem-solving attitude, etcetera during the project). This "*process evaluation*" is built around a set of specific evaluation criteria for students of this faculty, based upon the pedagogic vision of the faculty. More specifically, depending on the trajectory a student is following, the thesis advisor(s) need(s) to assess a student two or three times on four skill dimensions (each comprising a set of specific skills to be rated from insufficient up to very good) during the completion of his or her master thesis.

In this context, the *procesEval* application, based on NST, was created around 2013. Up to that moment, the process evaluation was either performed on paper or had to be registered via a customized part of the university's online learning and course management system. While the paper based evaluation was considered as generating administrative overhead (the results had to be manually copied into the university's database systems by the administration) and providing little overview for the thesis advisors (e.g., when performing the second process evaluation they could not easily consult the first process evaluation in order to make a more objective comparison), the electronic variant in the online learning and course management system was considered cumbersome from a usability perspective (e.g., users complaining about the amount of clicks required to perform "simple" actions or experiencing difficulties in order to find the information they are looking for).

The faculty management decided to develop an NST application to manage the process evaluations. This choice was made for several reasons. First, the expertise on how to build NST applications was present within the faculty itself as the theory (and the adjoining code expanders) was the output of research projects of faculty members. Second, as the software system would be developed by members of the faculty itself as well, the developers were highly knowledgeable about the inner working of the faculty (administration) and the associated (functional) requirements. And third, evolvability and maintainability were considered to be import quality aspects of the software system to be developed as the process evaluation was anticipated to remain an important part of the student evaluations for several years to come (but could be subject to some further fine-tuning or redirection in the future). Given the situation of the project as sketched above, it was

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

Figure 1. A general screenshot of the procesEval application.

expected that the application could be developed in a rather short development trajectory without too many hurdles (i.e., no significant risk related to the technology was present and the application domain was well known and understood).

The application was developed in the beginning of 2013. In the academic years 2013–2014 and 2014–2015, a first pilot test with a set of key users (technological savvy and proactive faculty members) was conducted. In the academic years 2015–2016 and 2016–2017, the set of test users was gradually enlarged up to the level at which all thesis supervisors could use the procesEval application if they wanted, but could still use the paper version if preferred. As of the academic year 2017–2018, all faculty members were expected to use the NST procesEval application for the administration of the master thesis process evaluations. Apart from minor (usability) adjustments, the project has been completed without major problems. Currently, on a yearly basis, about 45 faculty members manage the process evaluation of roughly 500 students via the procesEval application.

Figure 2. A screenshot of a specific process evaluation within the procesEval application.

In Figure 1, a screenshot of the procesEval application is shown (the names of the students are blurred out to assure anonymity, the names of the labels are Dutch as this is the administrative language of the organization). Here, one can notice that a supervisor can get an overview of all the students he or she is supervising in the current academic year. By selecting a particular student, a set of tabs appears below the first table providing further details regarding his/her (earlier) evaluations or working sessions (e.g., meetings) and documents (e.g., preliminary thesis version). Figure 2 shows a screenshot of one particular process evaluation. The procesEval application therefore manages all process evaluations (typically 2-3) of all master dissertations (as of 2017–2018) of multiple academic years. Based on the provided information, the application automatically generates overview reports of the evaluations

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

and sends emails to students and supervisors with information regarding their evaluations, as well as reminders (e.g., when a particular process evaluation is due).

B. Case analysis

We will first provide a general overview of our case analysis, and then zoom into a set of relevant variability dimensions that could be discerned at the level of the case.

1) General overview: An NST application typically consists of a set of base components (which are reused in several or even most applications), as well as one or multiple non-base components (typically specific for the considered application). The base components used within the procesEval application consisted of 29 data elements, 7 task elements and 1 flow element. The non-base component used within the procesEval application consisted of 14 data elements, 8 task elements and 4 flow elements. As a consequence, relatively speaking, the NST application was still rather small: it comprised about 63 NST elements.

2) Model variations: By using the NST approach, the procesEval application could be extended and adapted at the level of the model (i.e., the definition of the different element instances for the considered application domain). For instance, additional elements could be added: next to the registration of three possible process evaluations for each student, some working documents and information regarding working sessions (e.g., what was agreed upon by the student and his supervisor during a meeting) could be added to the model. After re-generating the application based on this updated model, this functionality becomes available in the new version of the application. Similarly, existing (i.e., earlier created) components could be added to the model. For example, a notification component was added to the procesEval application as that component contained the functionality to automatically trigger emails and could be leveraged to enable the automatic report delivery (of the process evaluations to the students, supervisors and administration). While the model could be changed in terms of data elements and components, this also holds for all types of other possible changes within the model. More specifically, the following types of adaptations can be performed to create different variations of the application:

- the addition, update or deletion of a component (i.e., a set of data, task and flow elements);
- the addition, update or deletion of a data element definition (its fields with its types and field options, finders, data element options, child elements);
- the addition or deletion of a task element definition (the specific implementation of a task is a crafting, see below);
- the addition, update or deletion of a flow element definition and its accompanying default state transitions.

It should be remarked that the determination and evolutions of such model is completely technology-agnostic (i.e., it does not require any specification in programming language specific terminology). For instance, the specification of the model (in terms of elements and their properties) is currently stored in an XML file, not containing any references to the (background) technology of the current reference implementation (i.e., Java). Based on this model, boiler plate source code for each of the layers can be created.

3) Crafting variations: Once the model is converted (expanded) into boiler plate source code, additional code (so-called “craftings”, which are custom made for an application) could be added between predefined anchors (insertions) or in additional classes (extensions). This way, non-standard functionality can be incorporated within the application as well. In total, the procesEval application contained 22 classes with insertions and 29 additional classes (extension). For instance, specific coding had to be added to make sure that a supervisor logged into the application can only view those master dissertations which he/she is supervising in the concerning year (i.e., dissertations supported by other supervisors or those of the previous year should not be visible). For this purpose, a few lines of code were added in the MasterThesisFinderBean class determining the fetching of the results viewable for a particular user. These FinderBean classes are expanded as part of the data layer: enforcing the filter of master dissertations at the level of the data layer ensures that no data from other users can be retrieved by the currently logged in user. Consequently, this crafting only impacts the data layer, while the remaining layers have no impact resulting from this change: they perform their functionality handling the (filtered) data offered by the MasterThesisFinderBean.

Additionally, a set of screentips was added to assist the user when filling-in the process evaluation (e.g., summarizing the meaning of each of the evaluation criteria in case of a mouse-over). The expanded NST code base supports this functionality by providing a helpInfo Knockout binding. Specific screentips can be added by including a crafting using this Knockout binding, and referring to a certain key. At run-time, the specific values for the required keys can be added in instances of HelpInfo data elements. This enables the configuration of the screentips even when the application has already been deployed. Note that only the view layer is customized for this functionality. This makes sense, since it is purely a useability concern, not impacting actual business logic. However, it is dependent on the specific technology used in the view layer (i.e., Knockout), and should be reprogrammed when a different technology is used.

Next, as mentioned before, the procesEval application also needed to create and send reports summarizing the content of the process evaluations. The definition of these reports (i.e., the items to be included and the corresponding layout) is considered to be a separate functionality, and should therefore be contained in a task element. The expanders provide all boilerplate code needed to execute this task in the NST application, and only the specific report generating functionality needs to be added as a crafting. The actual implementation of the execution of a task element is clearly separated, allowing versions and variations of the task implementation to co-exist. Currently, reports are generated using Jasper Reports. This requires the addition of a Jasper template file to the code base, and some code to fill the parameters to be inserted into this template. The additional processing logic is completely contained in the logic layer.

These craftings were added in a gradual and iterative way to the application: each time a particular additional functionality was added or improved, a new version of the overall application could be built and deployed. Furthermore, it can be remarked that each of these craftings were situated at another layer (i.e., data, view and logic).

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

4) Infrastructural technology variations: The procesEval application could be generated by using various different underlying infrastructural technologies. For instance, whereas a prototype of the application is typically demonstrated by using an HSQL database, most production systems are deployed while using a PostgreSQL database. Nevertheless, one can choose for SQLServer and MySQL databases as well. Further, the procesEval can be built by using different build automation frameworks (i.e., Ant and Maven). And finally, the procesEval could also be generated by using different controlling (Cocoon, Struts2, or combination Struts2-Knockout) and styling frameworks (plain style or using Bootstrap). In practice, the Struts2-Knockout and Bootstrap were used in the production environment. Changing the choice of a particular infrastructural technology in the procesEval only impacts those layers depending on the purpose of the technology (e.g., the database selection impacts the data layer, whereas the GUI framework selection impacts the view layer).

5) Expander version variations: The expanders (i.e., the programming logic used to convert the model into boiler plate source code according to the infrastructural technologies chosen) evolves throughout time as well. This way, when considering the current procesEval project duration (2013–present), 8 different production versions were deployed while using the same model and craftings (as the expanders provide backwards version compatibility). In each of these production versions, the new or improved possibilities of the expanders could be used. For instance, in one particular version of the expanders, information regarding a Date field did no longer have to be entered manually but could be selected by using a more advanced date picker. And, more relevant in the context of the procesEval, another particular version of the expanders allowed the automatic creation of summarizing graphs on certain fields. For example, it would now be possible to inspect the number of master dissertations who did not yet receive a first process evaluation versus those who did in a visual way. In order to use the date picker, no changes in the model or the craftings are required. In order to use the status graphs, only one additional specification in the model (i.e., an option indicating that a graph for a particular field should be created) needs to be added. Clearly, the precise set of layers that is impacted due to an expander update depends on the type of modifications performed in that particular version update (logic related, view related, etcetera).

IV. Reinterpretation of Existing Cases

In this section, we will look at some previously documented and analyzed cases of NST applications. While the initial publications were not specifically directed towards the illustration of the variability dimensions present within these systems, we will now aim to see to which extent we can find indications of such variability dimensions in them. This should allow us to verify whether the variability dimensions identified in the educational case study above also appear in other cases, thereby increasing the validity of our study.

A. Budget management application

A first case which we published in earlier work concerns a budgeting application for a local Belgian government [7]. The administration of the concerned local government organization was required to track its allocated budgets meticulously and in a very fine-grained way (including the division of budgets in subbudgets, their reservation, changes to the budgets, etcetera). The goal of the application was to provide the functionality for users to have a clear overview and tracking of budgets and subbudgets, budget assignments, changes on them, and so on. While the organization originally had the possibility to perform these activities via Microsoft Excel by using pivot tables, the long term goal was to integrate the application performing these analyses with other functionalities such as project management, budget reporting and simulations. During the analysis of this case, we noticed that the development of this replacing application was not trivial [7]. Indeed, many people are used of working with the popular spreadsheet program Excel, which offers many flexible and versatile analysis options in a user friendly way. In order to be able to meet the high standards of the end users, it was therefore decided to approach the application development in a very iterative and gradual way. In a first stage, attention was almost exclusively devoted to the development of the functionalities related to budget management and the usability of that part for end users. It was only later on that the project started to focus on the realization of a larger application which also incorporated some of the additional functionalities as mentioned above. Therefore, both the optimization of the universe of discussion within the budgeting functionality, as well as the initial (exclusively budgeting oriented) and later phases (focusing on the other functionalities as well) can be seen as different versions of the model throughout time (each time be expanded into working prototypes or working applications). Therefore, this case clearly illustrated the relevance of *model variations*.

During the case, craftings had to be added at various places as well. First, some code was required to provide additional graphical features. That is, a more advanced user interface with more sophisticated screens was needed (compared to those that were by default provided by the code expanders at that point in time). Such more advanced (composed) screens would allow users to inspect budget specifications over various levels concurrently (year, department, article) or from different angles/perspectives (departments, types of activities), thereby replicating behavior somewhat similar as the previously used pivot tables. Next to that, several customizations were present for specific calculations (logical operations). These calculations were very context specific for the organization and domain at hand and were directed towards issues such as the on-the-fly calculation of the currently available budget based on all previous budgets, the verification that budget calls were not exceeding the available budget, etcetera. As these craftings were refined over time, they illustrate the relevance of the *crafting variations* within this case. However, probably even more interesting, while the logic related craftings were very specific for this application that needed to be developed, the graphical extensions (i.e., the composite screens displaying multiple data elements having a one-to-many relationship on one screen) were considered to be useful for other (current and future) NST applications. Stated otherwise, these craftings were regarded as being somewhat generic. As a consequence, over time, some of these graphical extensions have been included in the code expanders. As soon as this happened, the more advanced screens became available for other (already existing or newly developed) NST applications. Therefore, this case clearly illustrated the relevance of *expander variations*.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

B. Infrastructure monitoring application

A second case which has been published earlier involved an application for an organization providing hardware and software for the monitoring of infrastructure (e.g., checking the correct functioning) such as power supplies, airconditioning and so on [8]. Whereas the report of the previous case (discussed in Section IV-A) provided a general overview of the application as a whole, the current case was reported in a temporal way, i.e., four phases were discussed in which the application evolved from its original status to its current status (at the time of publication). Additionally, the case was somewhat atypical for an NST application as it involved one of the first applications (re)developed according to this approach. In particular, the following four phases were distinguished in the concerning case:

- *phase 1*: Initially, the application was designed in a rather monolithic way without explicit attention to modularization while using a Microsoft Access database and a Visual Basic application. This was a version of the application without any use of the NST approach;
- *phase 2*: The application was redeveloped and designed in another technology stack using Java 2 Enterprise Edition (J2EE) with Enterprise JavaBeans 2.1 and the Cocoon framework. With NST not yet formulated and the element expansion mechanism not yet developed, the software system was mainly developed manually but taking into account industry best practices and the (implicit) heuristic knowledge which would later on result into the NST theorems. This resulted in a recurrent structure throughout the application similar to the later on developed NST elements;
- *phase 3*: In the following version, a significant part of the application was defined by using descriptor files describing certain recurring constructs (such as the need to persist a certain type of data) for which the code was then generated by one of the first versions of the pattern expanders. Next to updating the application to a new version of its code base, some additional functionalities (e.g., regarding FAQs and asset management) were added by generating additional data elements for them. Some custom code (e.g., for authorization requirements and user interfaces) was added in separate files;
- *phase 4*: In a final documented phase of the application, a switch to other controls and protocols for the infrastructure was made. At the same time, a newer version of the NST expanders was used in which custom code could be added between specifically located anchors within the code that could be harvested and injected during regeneration later on.

Based on the description of the phases as provided above, we can not only distinguish the relevance of the evolvability dimensions described earlier in the context of the instructure monitoring application, but also gather some information on their historical occurrence throughout time. As in phase 1, a non-NST approach was adopted, no explicit evolvability dimensions were present. The case report mentioned difficulties in order to adapt the application, including duplications and lack of flexibility. In phases 2 and 3, the patterns or elements were introduced (first in a somewhat implicit way, later on in a more explicit way allowing for automatic code generation) and improved for (largely) the same functional requirements. Therefore, the *expander* variability dimension was introduced at this point. In phase 3, due to the addition of some functionality based on the same patterns, the *model* variability dimension was illustrated as well. Finally, in phase 4, the harvesting mechanism allowed the easier migration of custom code from one application version to another. Therefore, this last phase also illustrates the occurrence of the *crafting* variability dimension in this context. One might remark that the last variability dimension, regarding the technological or *infrastructural options* does not seem to covered by the case at hand. While it is true that during the actual use of the NST expanders no significant technological or infrastructural changes have been performed, the transition of phase 1 to phase 2 was partly motivated by the fact that the technologies used for the creation of the initial software application were not adequate to work with in a distributed and multi-user environment. Therefore, at least the relevance of this variability dimension could certainly be argued for in the context of this case as well.

C. Integration applications

Finally, a set of four enterprise application integration cases was presented in [8]. For the purpose of this paper, with our focus on variability dimensions, one case is particularly interesting as it illustrates the infrastructural technology variations possible within this context and this variability dimension was somewhat less prominently present in the cases discussed in Sections IV-A and IV-B. The case was conducted within a multinational human resources consulting firm for which web-based access to 180 data entities needed to be provided from a legacy application (which was using a PL/SQL Oracle database). At the time the case was carried out, the NST expanders did not provide support for the PL/SQL Oracle database as required by the case organization. Therefore, the NST expanders had to be adapted for this possibility. Once this operation was performed, the generated NST application was able to connect with the database of the case organization. Next to that, all previously existing default functionality typically present in an NST application (and not impacted by the required changes in the logic and data layers) were available as well (e.g., the out-of-the-box CRUD screens for all the data elements within the application). And, as this project required an adaptation of the expanders, the possibility to link to PL/SQL databases became as of then available for all other current or future NST applications. Therefore, this case clearly illustrated the relevance of *infrastructural technology variations*.

V. DISCUSSION

Based on the above discussion of NST and its cases, we will discuss two broad areas in this section. First, in Section V-A, we will analyze the offered variability dimensions in the cases in a somewhat more general way. While we illustrated the possible variability dimensions using only one in-depth and three smaller cases, we anticipate that the proposed categorization can be generalized to a large extent as it also aligns with the general "degrees of freedom" available during

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

the development and maintenance of an NST application. Next, in Section V-B, we discuss some general implications that the existence of these kind of evolvability dimensions has for the management of NST projects and the role of the analyst in particular.

A. Application level evolvability dimensions

Based on our analysis as presented above, we identify four variability dimensions, as visualized in Figure 3.

First, as represented at the top of the figure, the modeler should select the *model* he or she wants to expand. Such a model is technology agnostic (i.e., defined without any reference to a particular technology that should be used) and represented by a blue puzzle (i.e., each puzzle piece represents a defined element, with the columns corresponding to data, task, flow, trigger and connector elements). Such a model can have multiple versions throughout time (e.g., being updated or complemented) or concurrently (e.g., choosing between a more extensive or summarized version). As a consequence, the figure contains multiple blue puzzles that are put behind each other and the chosen model represents a variability dimension (represented by the green bidirectional arrow).

Second, the *expanders* (represented by the trapezoid in the figure) generate (boiler plate) source code by taking the specifications in the chosen model as its *arguments*. For instance, for a data element Person, a set of java classes PersonBean, PersonLocal, PersonRemote, PersonDetails, etcetera will be generated. This code can be called boiler plate code as it provides a set of standard functionalities for each of the elements within the model. Nevertheless, one could argue that this set of standard functionalities is already quite decent as it contains the possibilities to provide standard finders, master-detail (waterfall) screens, certain display options, document upload/download functionality, child relations, etcetera. The expanders themselves evolve throughout time. Typically, in each new version, a set of bugs of the previous version are solved and additional features (e.g., creation of a status graph) are provided. It should be remarked that, given the fact that the application model is completely technology agnostic and can be used as argument for any version of the expanders, these bug fixes and additional features become available for all versions of all application models (only a re-expansion or "rejuvenation" is required). As a consequence, the figure contains multiple trapezoids that are put behind each other and the expander version represents a variability dimension (represented by the green bidirectional arrow).

Third, in the middle left of the figure, a set of *infrastructural options* are displayed by means of different rectangular blocks. These consist of global options (e.g., determining the build automation framework), presentation settings (determining the graphical user framework), business logic settings (determining the database used) and technical infrastructure (e.g., determining the background technology). For each of these infrastructural options, the modeler can choose out of a set of possibilities (e.g., different user interface frameworks for which the associated code can be generated), which will be used by the expanders as their *parameters*. That is, given a chosen application model version and expander version, different variants of boiler plate code can be generated, depending on the choices regarding the infrastructural options. As a consequence, the figure contains multiple infrastructural option sets (blocks) that are put behind each other and the infrastructural options represent a variability dimension (represented by the green bidirectional arrow).

Fourth, *craftings* ("custom code") can be applied to the generated source code. These craftings are represented in the lower left of the figure by means of red clouds as they enrich (are put upon) the earlier generated boiler plate code and can be harvested into a separate repository before regenerating the software application (after which they can be applied again). This includes extensions (e.g., additional classes added to the generated code base) as well as insertions (i.e., additional lines of code added between the foreseen anchors within the code). Craftings can have multiple versions throughout time (e.g., being updated or complemented) or concurrently (e.g., choosing between a more advanced or simplified version). These craftings should contain as little technology specific statements within their source code as possible (apart from the chosen background technology). Indeed, craftings referring to (for instance) a specific GUI framework will only be reusable as long as this particular GUI framework is selected during the generation of the application. In contrast, craftings performing certain validations but not containing any EJB specific statements will be able to be reused when applying other versions or choices regarding such framework. Craftings not dependent on the technology framework of a specific layer can be included in the "common" directory structure, whereas technology-dependent craftings need to reside in the directory structure specified for that technology (e.g., EJB for the logic layer, JPA for the data layer, Struts2 for the control layer). As a consequence, the figure contains multiple crafting planes that are put behind each other and the chosen set of craftings represents a variability dimension (represented by the green bidirectional arrow).

In summary, each part in Figure 3 with green bidirectional arrows is a variability dimension in an NST context. It is clear that talking about *the* "version" of an NST application (as is traditionally done for software systems) in such context becomes rather pointless. Indeed, the eventual software application (the grey puzzle at the bottom of the figure) is the result of a specific version of an application model, expander version, infrastructural options and set of craftings. Put differently, with M, E, I and C referring to the number of available application model versions, the number of expander versions, the number of infrastructural option combinations and crafting sets respectively, the total set of possible versions V of a particular NST application becomes equal to:

$$V = M \times E \times I \times C$$

Whereas the specific values of M and C are different for every single application, the values of E and I are dependent on the current state of the expanders. Remark that the number of infrastructural option combinations (I) is equally a product:

$$I = G \times P \times B \times T$$

Where G represents the number of available global option settings, P the number of available presentation settings, B the number of available business logic settings and T the number of available technical infrastructure settings. This general idea in terms of combinatorics corresponds to the overall goal of NST: enabling evolvability and variability by *leveraging the law of exponential variation gains* by means

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

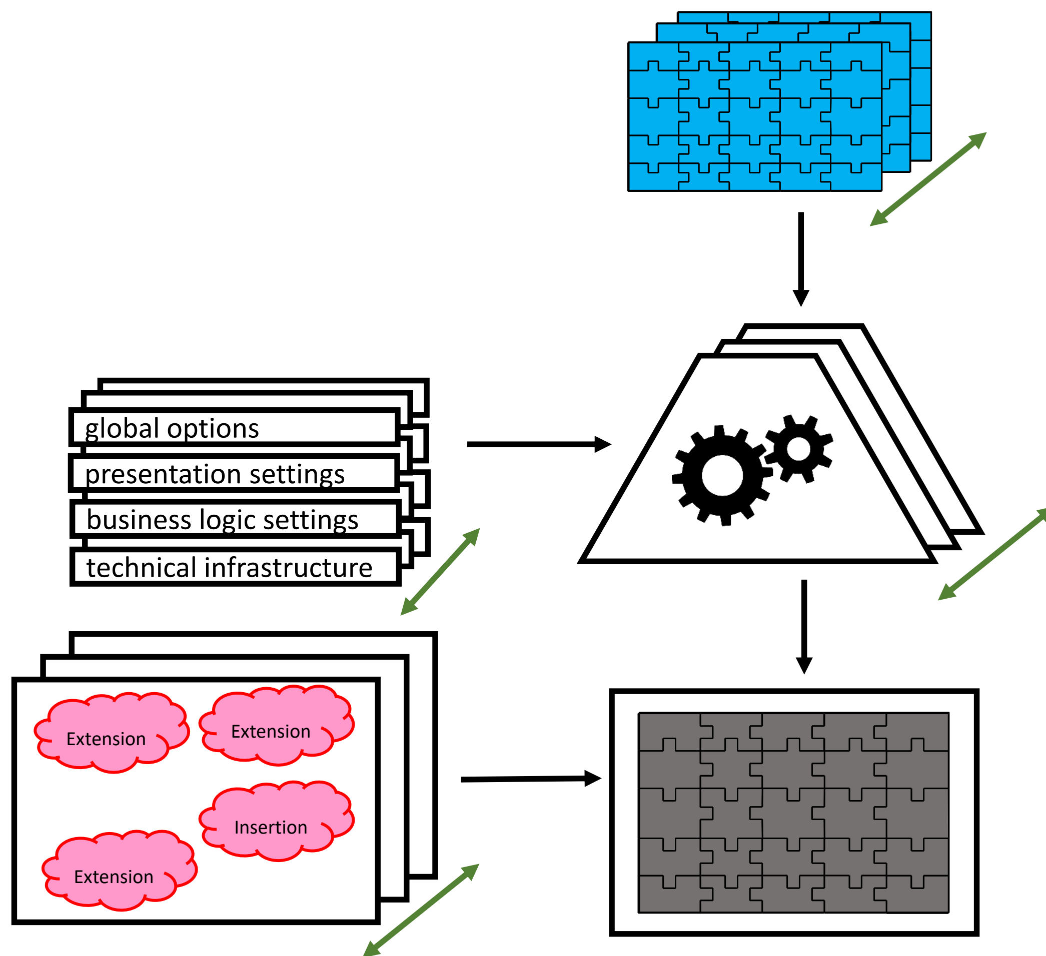

Figure 3. A graphical representation of four variability dimensions within a Normalized Systems application.

of the thorough decoupling of concerns and the facilitation of their recombination potential [5].

B. Project management

Performing software development projects in an environment where the chosen approach enables the variability dimensions as discussed in this paper, has some implications on how the project management in such context can be executed and, in particular, on the responsibilities of the analyst.

First, as NST is based upon the idea of realizing evolvable and adaptable applications, it seems logical that most NST projects are conducted in an iterative or agile way. More specifically, due to a visual modeling tool and supporting web application —allowing the definition of an NST model (specified in terms of data, action, flow, trigger and workflow elements), as well as the expansion and deployment of NST applications— analysts are able to create evolutionary prototypes. After some first examination of the universe of discussion, the analyst can thus make his interpretation of the main functional requirements which can immediately be incorporated in a working prototype and shown to, for instance, future end users. Based on their feedback, the analyst can then adapt his model, after which a new version of the prototype can be regenerated and demonstrated. Next to iterations related to the optimization of the model (still focusing on the same universe of discussion), the analyst can also enlarge the model by extending the universe of discussion covered by the application. That is, in initial stages one can opt to model (and further develop with craftings) only a part of an application and, later on, to extend the application towards other areas of the organization. This way of working is clearly related to the *model* variability dimension: the analyst only defines a model and the other dimensions (craftings, technologies and expanders) are at that point irrelevant and can be specified later on.

Once (a part of) the model has been defined, all basic functionality offered by the expanders is present within the generated application. In case additional functionality is required (i.e., not provided by the expanders), this can be added by developers within the provided anchors in the generated code or by additional files (e.g., classes). This code can be added and developed independently from the model (e.g., analysts can keep on working on the extension of the model while developers start adding additional crafting code) as the craftings can be harvested and reinjected from one model to the other (as long as the craftings do not become incompatible with the newly defined model, which could for instance be the case when programming the implementation of a custom finder method using an attribute that would be deleted later on). The analyst can inject these craftings into

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

his prototype as well, thereby validating whether the added code fits the customer requirements and crosscheck them with end users if required. Analysts can clearly see how specified customizations map onto crafting code. This can be helpful in assessing their complexity (e.g., by checking the size of code required) and maintaining the current state of the project (which customizations have been completed and which have not). This part of the project management is clearly related to the *crafting* variability dimension.

As the analyst verifies and inspects the customizations made by the developers, he might notice that some functionalities are not only relevant for the specific case at hand and might in fact prove their usefuless in other and future applications as well. At that point, certain functionalities within the craftings can be incorporated or "generalized" into the expanders. As each of the functionalities needs to adhere to the theorems of NST, needs to be (almost) free of bugs, etcetera this typically only happens after the functionality has been thoroughly tested in the context of multiple projects in such a way that sufficient experience in this matter has been gained. As of then, the functionality is removed from the application's crafting part and becomes available for all other applications as well. Therefore, this part of the project management is clearly related to the *expander* variability dimension.

Finally, the expanders can be used to employ various (combinations of) technology infrastructure. This is typically independent from the model (as it is mostly also not a responsibility of the analyst to take committing decisions on this area) and expander version. Clearly, the craftings need to be written in a certain technology or language which introduces a dependency and might imply adaptations to craftings in case a certain technological infrastructure option is chosen. For instance, when a certain GUI framework is chosen and craftings are added in this part, a future change of GUI framework might imply changes to the previous craftings (which need to be rewritten in the newly chosen GUI framework). In other cases however, technological infrastructure decisions can vary freely from code within the craftings, such as in cases where the chosen technology is irrelevant to the craftings (e.g., craftings in the view layer will not be impacted by changing the selected database) or when the craftings are all encapsulated from the changing technology (e.g., plain non-EJB specific Java code within anchors in a class using EJB annotations). Therefore, this part of the project management is clearly related to the *technology infrastructure* variability dimension.

VI. Conclusion

This paper presented one in-depth case study of an NST software application in an educational context and analyzed the different dimensions in which it could evolve. Additionally, three previously documented cases were reinterpreted using the same point of view. Based on this, four general variability dimensions were proposed.

This paper is believed to make several contributions. From a theoretical side, inductive reasoning based on our cases allowed the formulation and illustration of four variability dimensions, which might be the (or at least a subset of the) orthogonal dimensions along which a typical NST application can evolve. At the same time, these variability dimensions clarify that the concept of an overall application "version" is not applicable for NST applications as a specifically deployed application is the result of a combination of choices for each of the variability dimensions. For practitioners, this paper contributes to the set of case studies available on NST (as well as a reinterpretation from some previously documented cases), which might provide them with a better insight regarding the application potential of the theory in practice.

Next to these contributions, it is clear that this paper is also subject to a set of limitations. That is, we proposed the set of variability dimensions based on a limited set of case studies. Although the size, complexity and industry of the cases were different, their modest amount still limits the generalizability of our findings. Therefore, future research should be directed towards the analysis of additional cases, including information systems being even larger and more complex. These additional cases might confirm, and possibly extend, the variability dimensions proposed in this paper.

References

[1] P. De Bruyn, H. Mannaert, and P. Huysmans, "On the variability dimensions of normalized systems applications: Experiences from an educational case study," in Proceedings of the Tenth International Conference on Pervasive Patterns and Applications (PATTERNS) 2018, 2018, pp. 45–50.

[2] R. Agarwal and A. Tiwana, "Editorial—evolvable systems: Through the looking glass of IS," Information Systems Research, vol. 26, no. 3, 2015, pp. 473–479.

[3] 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, 2011, pp. 1210–1222, special Issue on Software Evolution, Adaptability and Variability.

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

[5] H. Mannaert, J. Verelst, and P. De Bruyn, Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design. Koppa, 2016.

[6] M. Op't Land, M. Krouwel, E. Van Dipten, and J. Verelst, "Exploring normalized systems potential for dutch mods agility: A proof of concept on flexibility, time-to-market, productivity and quality," in Proceedings of the 3rd Practice-driven Research on Enterprise Transformation (PRET) working conference, Luxemburg, Luxemburg, September 2011, pp. 110–121.

[7] G. Oorts, P. Huysmans, P. De Bruyn, H. Mannaert, J. Verelst, and A. Oost, "Building evolvable software using normalized systems theory: a case study," in Proceedings of the 47th annual Hawaii international conference on system sciences (HICSS), Waikoloa, Hawaii, USA, 2014, pp. 4760–4769.

[8] P. Huysmans, P. De Bruyn, G. Oorts, J. Verelst, D. van der Linden, and H. Mannaert, "Analyzing the evolvability of modular structures: a longitudinal normalized systems case study," in Proceedings of the Tenth International Conference on Software Engineering Advances (ICSEA), Barcelona, Spain, November 2015, pp. 319–325.

[9] P. Huysmans, J. Verelst, H. Mannaert, and A. Oost, "Integrating information systems using normalized systems theory: four case studies," in Proceedings of the 17th IEEE Conference on Business Informatics (CBI), Lisbon, Portugal, July 2015, pp. 173–180.

[10] P. De Bruyn, P. Huysmans, and J. Verelst, "Tailoring an analysis approach for developing evolvable software systems : experiences from three case studies," in Proceedings of the 18th IEEE Conference on Business Informatics (CBI), Paris, France, August-September 2016, pp. 208–217.

[11] M. Lehman, "Programs, life cycles, and laws of software evolution," in Proceedings of the IEEE, vol. 68, 1980, pp. 1060–1076.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org