Towards Applying Normalized Systems Theory to Create Evolvable Enterprise Resource Planning Software

A Case Study

Ornchanok Chongsombut, Jan Verelst, Peter De Bruyn, Herwig Mannaert and Philip Huysmans
Department of Management Information Systems
University of Antwerp
Antwerp, Belgium
e-mail: {ornchanok.chongsombut, jan.verelst, perter.debruyn, herwig.mannaert, philip.huysmans} @uantwerp.be

Abstract— **Evolvability is widely considered to be an important concern for the design and development of software architectures, particularly in the area of information systems (IS). Current IS continue to struggle to provide evolvability, especially Enterprise Resource Planning (ERP) software. Complexity in ERP packages leads to a limit on changeable software. Normalized Systems (NS) theory has been proposed with the purpose of incorporate evolvability of IS. In this paper, an existing ERP package was subjected to reverse engineering in order to analyze and explore in terms of combinatorial effects (CEs), of which NS theory prescribes that they are harmful for evolvability. The results indicate that it is possible to redesign a data model for an existing ERP, adhering to NS theory. We also identified some issues and limitations with the current version of the ERP package.**

Keywords- Normalized Systems theory; evolvability; software architecture.

I. INTRODUCTION

IS have played an increasingly visible role over the past few years in improving the competitiveness of businesses. Organizations realize that ERP software is a crucial tool for organizational perfection because it enables flawless transactions and production runs, and can improve business performances and profitability through implementation of Business Intelligence [1][2]. An ERP system is a departmental integration software system that allows a company to have a unified enterprise view of the business and to manage enterprise data in one database [2][3]. ERP systems offer significant benefits to an enterprise through improving strategic planning and operational efficiency.

Notwithstanding the significant benefits, however, ERP systems have been criticized, since they are prone to extreme complexities and are often difficult to implement [4]. This complexity is mostly due to the fact that the system has to integrate all functions and data of a company. This contributes to development and maintenance costs, being important barriers to realize the potential gains which can be achieved through IS. Therefore, IS should exhibit a certain amount of simplicity to result in the anticipated gains.

Moreover, IS should be evolvable as well. The business environment is dramatically changing and the ability to easily change software therefore becomes crucial [7]. In this context, we consider software evolvability as the capability of software to be easily changed (i.e., with a reasonable effort). Current IS continue to struggle to provide such evolvability, especially ERP software. The significant complexity in these packages leads to a serious limit being placed on their ability to change.

In particular, evolvability can be considered as a criterion to evaluate and analyze the quality and usefulness of ERP packages for several reasons. First, the main objective of ERP is to support various organizations to achieve their business goals. Therefore, ERP should be adaptable to the specificities of organizations, for example, by means of configuration. Second, as stated before, business environments dramatically change. The evolvability of ERP systems becomes an increasingly crucial condition to enable changes for an enterprise as a whole and their increasing complexity.

The design of IS which are evolvable has been addressed in NS theory. For this purpose, the theory uses the stability concept (i.e., requiring that a bounded set of functional changes to the system should have a bounded impact within the system). The theory argues that CEs are the main obstruction to software evolvability. A CE occurs when the size of the impact of a change depends on the size of the information system [6]. In other words, NS states that the evolvability and flexibility is largely determined by (the absence of) CEs [8] and software without the CEs lead to evolvable software [5]. The theory proposes a set of theorems eliminating CEs.

The aim of this research is to identify CEs within existing ERP packages and to rebuild them based on NS theory. To this end, existing ERP packages are subjected to reverse engineering in order to explore the existence of CEs and their potential for other improvements. We describe how part of the existing ERP package is designed and developed based on the NS theory. Therefore, this paper tackles the evolvability of existing ERP software by designing a set of ERP guidelines to design and to develop existing ERP software according to the aforementioned theory. The main research question addressed by this paper is:

 ISBN: 978-1-61208-498-5

How to improve the existing ERP package based on NS theory?

Consequently, the proposed research mainly deals with the modularization of the data model of existing ERP packages. The modularity of an information system is important for the degree to which it exhibits evolvability [11]. The design science research approach, focusing on the creation and evaluation of IT artifacts and their surrounding organizational preconditions in order to solve organizational problems [7], is chosen as the methodology for the research.

This paper is structured as follows. Section II describes the essence of NS theory. Section III provides the design method of the paper. Afterwards, a partial analysis of the ERP application's data models is discussed in Section IV. Finally, Section V presents some final conclusions, limitations and suggestions for future research.

II. NORMALIZED SYSTEMS THEORY

NS theory is developed from the concept of stability which implies that a bounded set of input changes (i.e., changes in requirements) should result in bounded amount of output changes or impacts to the software (i.e., changes in software). In other words, stability demands that the impact of changes should only depend on the nature of the change itself. The size of the impact of changes should therefore not be related to the size of the system. If the size of the impact of a change is related to the overall size of the system, this is called a CE [9].

A CE is one of the biggest barriers to creating evolvable software according to NS. The theory states that the evolvability of software should be a characteristic embedded at the level of the software architecture. This implies that the software architecture should not only allow the realization of current requirements but also facilitate the incorporation of future ones. NS theory assumes an unlimited software evolution (i.e., ever growing software throughout time). This means that even the smallest change of which the impact is dependent on the size of the system (i.e., CE) will become troublesome over time and should therefore be removed. In fact, if the CEs occur mainly in software architectures, the software will become more difficult to cope with and the software's evolvability tents to increase.

A set of four theorems and five expandable elements has been suggested by NS theory to prevent CEs and to develop evolvable software.

The four theorems are the following [10][11][12][13]:

- Separation of Concerns: each change driver (concern) should be separated from other change drivers (concerns) in distinct modules;
- Data Version Transparency: the modification (insert, update, delete) of data entities should not impact other entities.
- Action Version Transparency: the modification (insert, update, delete) of data entities should not impact other entities.
- Separation of States: each step in a workflow should be separated from other steps in time by keeping state after every action or step.

Consequently, the systematic application of the theorems results in a very fine-grained modular structure in which each change driver has to become separated. Building software which systematically adheres to the NS theorems should result in software which is highly evolvable software. Moreover, the theory emphasizing the CEs identification will help to build IS that contain the smallest in amount of the CEs.

Furthermore, the NS theory provided evidence of the number of the CEs are the cause of a complex software and difficultness of software maintenance [10]. The NS theory is a modular structure that is free from the CEs. The CEs should not be present at compile time, deployment time, and run time in modular structures in order to constitute an achievement in this [10].

Software architectures without the CEs can be constructed as a set of highly structured instantiations and loosely coupled design patterns that are called elements [10]. For this reason, five elements have been proposed to facilitate the achievement of these aims. There are the NS elements:

- a data element, representing an encapsulated data entity with its set and get methods to access information in a version transparency way [14]. Then cross-cutting concerns should be added to the element in separate constructs;
- an action element, executing a core action or algorithm.
- a workflow element for the execution of the sequence of action elements;
- a trigger element, controlling the states and checking (time or status based) whether an action element has to be triggered;
- a connector element, providing the possibility for external systems to interact with the NS system without calling the elements in a stateless way.

III. APPLYING DESIGN SCIENCE RESEARCH

A. The Research Design

The conceptual framework of this research adheres to the IS research framework [15] shown in Fig. 1. The technology problem has been defined in the problem space: the requirement for ERP systems to exhibit evolvability. The research addresses technology which needs to achieve a higher degree of evolvability. The concepts of modularity and stability from NS theory will be applied as the foundation of the research. The NS theory will be used to create artifacts at the software level. The theory will also be used to develop a set of relevant measures or validation criteria in order to ensure that the proposed artifacts can be evaluated by appropriate evaluations. This should further support the rigor within the considered IS research. Finally,

 ISBN: 978-1-61208-498-5

as can be seem in the middle of the conceptual framework, we will explore and build an artifact which will be presented in the method term to construct increasingly evolvable in ERP systems. Also, we will create a working system (instantiation) and explore a suitable method to evaluate the artifact.

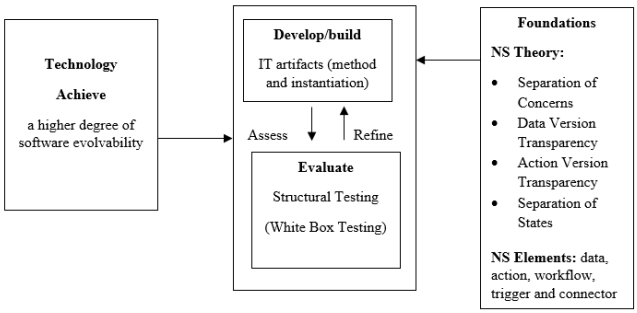

Figure 1. The research framework

Table I illustrates the appropriate evaluation methods which assess the utility, quality, and reliability of our designed artifact. Therefore, the evaluation activity is a crucial process of the research process [15]. Five evaluation methods have been proposed by [15]. In this research, we will combine several types of evaluation to ensure that our designed artifact is rigorously demonstrated via suitable evaluation methods:

TABLE I. THE RESEARCH DESIGN EVALUATION METHOD

The research design evaluation methods	Description	Application
testing evaluation	Executing coverage testing of some metric in the artifact implementation: Structural (White Box) Testing	Number of Change Impacts

B. Desing Science Research in the Research

The behavioral sciences and design science are two paradigms that characterize much of the research in the IS discipline, having a different purpose. Whereas behavioral science seeks to develop and verify theories that explain or predict human or organizational behavior, design science seeks to extend the human boundaries and organizational capabilities by creating new and innovative artifacts [15]. In the design science paradigm, artifacts are studied and created in order to solve a practical problem of general interest [15][16]. A practical problem is a gap between the current situation and a desirable situation that is observed in practice [16]. Therefore, this research was conducted using the design science methodology in order to address the research question.

Design science research includes six main activities according to [17] identify problem and motivation, define objectives of a solution, design and development artifact, demonstration, evaluation, and communication. The methodology allows the researcher to do the research in multiple research iterations to ensure and to improve the qualities of the artifact. The research design was aligned with this iterative process to end up with the research findings.

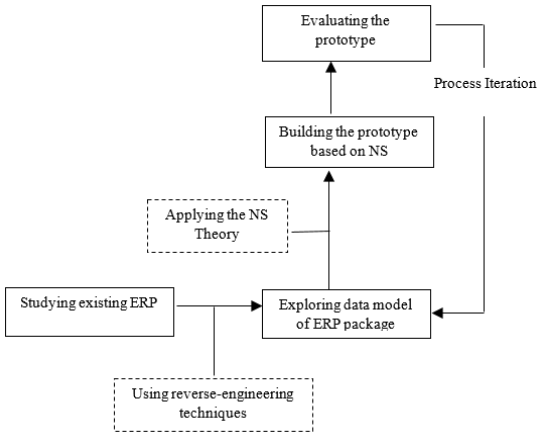

Figure 2. Design Science Process Model

An ERP application is a large and complicated system which often contains customized functions to fit the requirements of organizations. This is why such systems should be designed for evolvability so that maintenance costs remain under control. The paper aims to explore this issue. The processes of the research are demonstrated in Fig. 2. First, a research problem has been sketched by studying the architecture of an existing open source ERP package. Then we have defined a specific objective for the research which is to achieve a higher degree of ERP evolvability. We have solved the problem by using the reverse-engineering technique and applying it to an open source EPR system apart to see how it works. Second, CEs have been identified in the data model of the open source ERP package, which is an output of the reverse-engineering process. Third, we have redesigned the data model and built a prototype using the NS theory to improve the evolvability of the ERP software. The final stage of the design science process is white-box testing was used to evaluate the CEs of the redesigned data model and prototype that measure the number of change impacts.

IV. ANALYZING A PARTIAL OPEN SOURCE ERP MODULE: A SALES MODULE

In this section, we discuss some implications of using NS theory for developing evolvable ERPs in practice. An open source ERP, Odoo, was analyzed. This paper was due to several reasons. First, being open source, we have access to the source code and can analyze the architecture of the software in depth. Second, within the open source ERP market, Odoo is a very popular package. Furthermore, Odoo is an integrated suite of applications that includes modules for project management, billing, accounting, inventory management, manufacturing, and purchasing. However, we only focus on a partial module of the ERP package in this paper.

Fig. 3 illustrates a part of the data model of the sales module of the chosen ERP package which comprises three tables:

- res_partner for storing details of customer data
- sale_order for storing sales orders of customers
- sale_order_line for storing details of sales orders

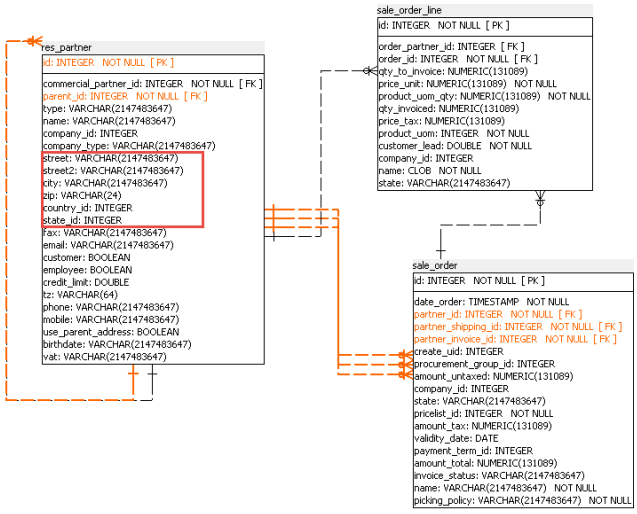

Figure 3. A partial data model of Sale module

Next, we analyzed the entity relationships of the data model. The res_partner table has a recursive relationship by using the parant_id field as a foreign key (FK). The parant_id filed save data to show partners is belonging to which a partner. For instance, Fig. 4 describes that id 46 is a parent of data of id 47, 48, 46. What's more, all of them are the same person.

The sales order table comprises the following foreign keys: partner_id, partner_shipping_id and partner_inovoic_id to join with the res_partner table as presented in Fig. 3. From sales order, employees can know where they have to send invoices and products to their customers.

id	name	parent_id	street	street2	city	zip	state_id	country_id	type
49		46	23 Korning		Antwerp	2000	61	21	contact
48		46	999 Oudan		Antwerp	2000	61	21	delivery
47		46	55 Prinesee		Antwerp	2000	61	21	invoice
46	Ornchanok		23 Korning		Antwerp	2000	61	21	contact

Figure 4. The example of res_partner data

A. *Implementation of the Evolvable ERP Application with regard to NS Fundamentals*

Previously, we described the existing ERP data model from our case study. In this model, only one address is used and it is incorporated in the res_partner table. However, the model can be redesigned in other ways as well. For instance, the sales order data might not want to use only invoice address and shipping address but also other address types. For instance, the organization might want to use different addresses for invoicing and shipment. Moreover, customers might want the company to send sales invoices to more than one address. Initially, we aim to examine how the high evolvable ERP software is designed. Then we have proposed on alternative design explanation for evolvable ERP software development.

Normally, practical requirement of Sale module should be able to serve multiple customer addresses. We developed new approach to meeting the requirements of the evolvability of ERP software.

In order to attain the objective, the following data elements are defined. Address data have been separated from partner data in order to support the requirement. According to the previous model, the tables have been split up into six tables as illustrated in Fig. 5.

- Partner for storing only general data of customers such as name, birthday, status of partner (active, inactive), etc.
- Address_type for storing types of address. Hence, the changing requirement for address type can be supported.
- Address for saving all addresses' details of partners.
- Sale_order_address_line to save details of customers' sale order addresses.
- Sale_order to represent sale order data of customers.
- Sale_order_line to keep sale order details.

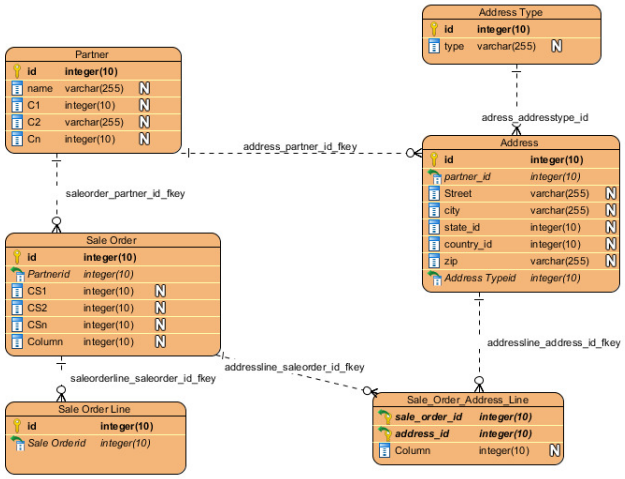

Figure 5. A redesinged data model of Sale module

In this model, address fields are divided into two tables: address and address type. Furthermore, the sale_order_address_line table is created to support the idea about multiple address in sales order data. Furthermore, the foreign keys partner_shipping_id and partner_invocie_id, in the sale_order table were removed.

Consequently, it was made possible to incorporate the new requirement of having different addresses. For example, if organizations can have other types of addresses by adding a new address type into the address_type table. When they want to record more types of customer address in sale order, they can only add more address data details into sale_order_address_line.

In NS theory, the objective of the theory is to design software in an evolvable way. In case a new version of a data model is designed, a new skeleton of the application can be generated by applying the expanders on the defined model [13].

 ISBN: 978-1-61208-498-5

In this study, a prototype of the new designed data model has been created using these NS expanders. Accordingly, the prototype has been developed without the CEs to increase evolvability of software.

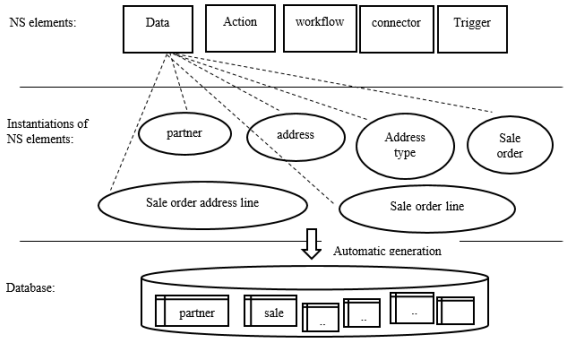

Figure 6. The prototype structure based on NS elements

According to NS theory, an evolvable application could be implemented by using a set of five elements which are already free of many CEs. For the case study that we describe, six data element instances have been generated as illustrated in Fig. 6. For each of these data elements, a set of cross-cutting concerns (persistency, security, graphical user interface, etc.) are automatically generated as well.

V. DISCUSSION

In this paper, we discussed how an existing open source ERP package can be analyzed based on NS theory. The paper raises some significant points. First, designing data models and developing a software application should be done in a modular way to enable the reusability of that software. For example, by using address and address type entities in a sales module. Second, the software should have highly cohesive modules so that the impact of a user requirement change on the actual application, remains limited. In the case, we separated the address data of a partner into two parts: address and address type entities. This represents an attempt to decrease the amount of effect of changes and increase software maintainability. Lastly, this study confirms that NS software will have less ripple effects in its system. For example, adding an address type into the prototype. To record additional address data, a user should now only add the data of the new address type. The change does not affect the partner, sale order and address elements.

This paper made a considerable contribution towards presenting the advantages of developing and maintaining software as stated by NS theory. These advantages can normally be commonly observed in amount of software development life cycle time. The new designed data model and prototype, which are designed using the NS theory, have the smaller number of change impacts. Furthermore, the software prototype can be created in a few days. Additionally, this paper contributes to the redesigning approach of building evolvable ERP software.

The limitations of our exploratory study need to be acknowledged. First, we only analyzed a partial data model of one ERP package. We could not perform reverse-engineering and explore commercial ERP software packages such as SAP, Oracle, etc. As part of future research, analyzing and rebuilding all modules of an existing ERP software package based on NS theory can be considered.

REFERENCES

[1] L. Shaul and D. Tauber, “Critical success factors in enterprise resource planning systems: Review of the last decade,” ACM Comput. Surv., 45(4): pp. 1-39, 2013.

[2] K. K. Hong, and Y. G. Kim, “The critical success factors for ERP implementation: an organizational fit perspective. Information & Management,” 40(1): pp. 25-40, 2002.

[3] E. J. Umble, R. R. Haft, and M. M. Umble, “Enterprise resource planning: Implementation procedures and critical success factors,” European Journal of Operational Research, 146(2): pp. 241-257, 2003.

[4] Y. Xue, H. Liang, W. R. Boulton, and C. A. Snyder, “ERP implementation failures in China: Case studies with implications for ERP vendors,” International Journal of Production Economics, 97(3): pp. 279-295, 2005.

[5] G. Oorts, K. Ahmadpour, H. Mannaert, J. Verelst, and A. Oost, “Easily Evolving Software Using Normalized System Theory A Case Study,” The Ninth International Conference on Software Engineering Advances, pp. 322-327, 2014.

[6] K. Ven, D. V. Nuffel, P. Huysmans, D. Bellens, and H. Mannaert, “Experiences with the automatic discovery of violations to the normalized systems design theorems,” International journal on advances in software, 4:1/2(2011): pp. 46-60, 2011.

[7] P. De Bruyn, “Generalizing Normalized Systems Theory: Towards a Foundational Theory for Enterprise Engineering,” in Faculty of Applied Economics. 2014, University of Antwerp: University of Antwerp.

[8] J. Verelst, A. R. Silva, H. Mannaert, D. A. Ferreira, and P. Huysmans, “Identifying Combinatorial Effects in Requirements Engineering,” in Advances in Enterprise Engineering VII: Third Enterprise Engineering Working Conference, EEWC 2013, Luxembourg, Proceedings, H.A. Proper, D. Aveiro, and K. Gaaloul, Editors. 2013, Springer Berlin Heidelberg: Berlin, Heidelberg. pp. 88-102, May 13-14, 2013.

[9] P. De Bruyn, P. Huysmans, G. Oorts, D. van Nuffel, H. Mannaert, J. Verelst, and A. Oost, “Incorporating design knowledge into the software development process using normalized systems theory,” International journal on advances in software, 6(1-2): pp. 181-195, 2013.

[10] J. Verelst, H. Mannaert, and P. Huysmans, "IT isn't different after all: Implications of Normalized Systems for the Industrialization of Software Development,” IEEE International Conference on Business Informatics, pp. 357-357, 2013.

[11] H. Mannaert, J. Verelst, and K. Ven, “Towards evolvable software architectures based on systems theoretic stability. Software: Practice and Experience,” 42(1): pp. 89-116, 2012.

[12] D. V. Nuffel, “Towards Designing Modular and Evolvable Business Process,” in Department of Applied Economics. 2011, University of Antwerp.

[13] G. Oorts, P Huysmans, P. De Bruyn, H. Mannaert, J. Verelst, and A. Oost, “Building Evolvable Software Using Normalized Systems Theory: A Case Study,” 2014 47th Hawaii International Conference on System Sciences, pp. 4760-4769, 2014.

 ISBN: 978-1-61208-498-5

[14] K. D. Ven, D. V. Nuffel, P. Huysmans, D. Bellens, and H. Mannaert, “Experiences with the automatic discovery of violations to the normalized systems design theorems,” International journal on advances in software, 4(1/2): pp. 46-60, 2011.

[15] Alan Hevner, “Design Research in Information System Theory and Pratice,” 2010: Springer.

[16] Paul Johannesson “An Introduction to Design Science,” 2014: Springer.

[17] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research.,” J. Manage. Inf. Syst., 24(3): pp. 45-77, 2007.

 ISBN: 978-1-61208-498-5