Applying Normalized Systems Theory in a Data Integration Solution, a Case Report.

Hans Nouwens[1](✉), Edzo A. Botjes[2], and Christiaan Balke[1]

[1] Sogeti Nederland, Vianen, the Netherlands
{Hans.Nouwens,Christiaan.Balke}@sogeti.com
[2] Xebia - Security, Hilversum, the Netherlands
ebotjes@xebia.com

Abstract. Enterprise Application Integration (EAI) becomes more important and non-evident when confronted with the growth of complexity. The evolvability of the EAI solution itself decreases over time due to Lehman's law. This case report describes and evaluates the architecture (model and principles), governance, solution design, and realisation, which are based on the Normalized Systems Theory. There are indications that the realisation resulted in a mitigation of the effects of the Lehman's law, hence improving the evolvability of the created EAI solution.

Keywords: normalized systems · software architecture · case report · enterprise application integration

1 Introduction

When implementing information systems, the main focus is on fast realisation and deployment. Preventing complexity of integration between the information systems is not the main priority [5,15].

The integration of applications, the exchanging of data between enterprise applications, also known as Enterprise Application Integration (EAI) [10,3,5], becomes more important and non-evident when confronted with the growth of complexity of the application integration [5]. This often results in EAI becoming distinct from application development with its own software used for performing the task of integration [5]. There are various methods to integrate applications [5]. Each methods has an unique complexity in their solution and an impact of the complexity of the application that is integrated with the whole [5,7].

Lehman's law of increasing complexity [9] states: *"As an evolving program is constantly changed, it complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it"* [12, par. 6.2.2]. Normalized Systems Theory (NST) proposes a set of theorems/ principles to counter Lehman's law including the previously described complexity of EAI. NST argues that reduction of Combinatorial Effects (CE) will reduce the complexity of software. Disconnecting the relation between the size of the application and the impact of a change [1,15], will at least maintain a stable cost of change in a growing

system. The theorems of NST are described in two books [11,12] and various articles [13,14,4]. Description of NST implemented in software is limited available [1,16,15] as is description on integrating applications using NST [5].

We want to add our experience with applying NST in a real life environment to the available body of knowledge in the form of a case report following the STARR format (Situation, Task, Approach, Result, Reflection).

2 Situation

The case report subject is a Dutch university of applied science. The business application landscape of this educational organisation evolved with little guidance (organic). This resulted in many custom applications providing overlapping functions and supporting undocumented information flows. Migrating this complicated and interwoven application landscape to industry standard SaaS applications proved to be very time consuming, holding the organisation hostage in its current situation.

Previous attempts to describe an architecture failed, mainly because of their generic approach and lack of a clear vision on how to implement an evolvable design.

3 Task

To help in this migration from custom applications to standard SaaS, in 2017 the first author was requested to write an architecture. An architecture that guided a design for an evolvable data integration solution based on a hub-and-spoke pattern with a central data storage hub.

One of the clients requirements was to make the design and solution able to adapt to a changing environment, to be evolvable. Attention to potential change drivers and possible combinatorial effects turned out to be one of the main drivers behind the architecture and the solution designs.

1. The solution needs to support: multiple data sources and data targets, multiple vendors, multiple communication patterns (pub/sub, etc), multiple security patterns (JSON Web Tokens, IP-Allow-List, etc), multiple connection techniques (ODBC, REST/SOAP API, (s)FTP, http(s), e-mail, SAMBA/NFS, etc), multiple file types (JSON, CSV, XLS(X), etc) and multiple (cloud) networks (on-premise, Private-cloud, Azure, AWS, GCP, etc).
2. The solution needs to support multiple frequencies of delivering data such as a 24-hour bulk upload or small messages, based on events in a source application.
3. The solution needs to support quality rules to prevent unqualified source data to be distributed.
4. The solution needs to support filtering to ensure a data minimisation policy.

4 Approach

This case report contains our translation of the NST theorems [12, par. 12.2] into an architecture for an EAI solution. In our experience, in the Dutch educational sector the integration bus pattern is commonly used to realise EAI, especially with of-the-shelf and Software-as-a-Service (SaaS) applications. This decision is mainly based on the general conviction that the application of an ESB reduces the number of connections between application from N(N-1)/2 to N as described in [12, p. 275, fig. 12.3].

However, we have not seen an application of the NST in this problem domain in this sector.

For our evaluation and reflection, we try to answer this question:

Does our architecture, applying our interpretation of the Normalized Systems Theory, improve the evolvability of the created EAI solution?

Author one fulfilled the role of architect, Author two fulfilled the role of architect at other educational organisations with similar requirements, and not directly involved at the case report subject. The third author was analyst and engineer in realising and operating the solution during two years. The numbers in this case report include the period when the engineer was involved as the consulting architect when there was a limited involvement of the main architect.

Next to a number of personal observations and evaluations, we retrieved the perceived effort to create new versions of the connections between the ESB and the application. For this we consulted the previously hired developer, and the developers, analyst and product owner currently working at the organisation.

If we were successful in preventing complexity, in the context of Lehmans law, we should be able to see that the effort of changing a connection does not increase together with the overall size of the system, the total number of connections.

5 Result

The Integration Framework (IFW) is (1) a technical solution that integrates multiple systems and data sources and (2) a framework of processes, controls and agreements. The IFW system architecture describes the components, the way of organising their relations and the principles guiding its design and evolution.

5.1 IFW system architecture

The IFW system consists out of the following three types of components: ‘source pipelines‘, a ‘data-hub‘ component and ‘target pipelines‘ (see figure 1).

The ‘source pipelines‘ are responsible for the optimal ingestion of information from the designated source systems and maintain integrity and actuality of the

information in the 'data storage'. For each source system an instance of a source pipeline is created. The 'data-hub' is responsible for the availability of the information. There is only one instance of the data-hub. The 'target pipelines' are responsible for providing data to systems that are not designated as the source for this data. For each target system an instance of a target pipeline is created. Together this creates the pattern of a ESB, plus a central data persistence component.

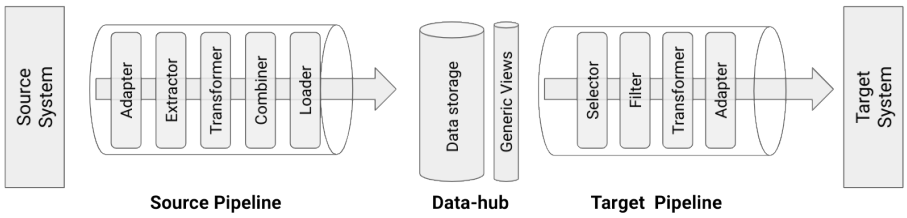

Fig. 1. The Integration Framework (IFW) system model

The IFW system consists of the following sub-systems:

1. *Source systems* are connected to the IFW system to deliver part of it data elements. The connection to the source systems is part of the scope of the IFW, the source system itself is not. Each source system is connected to a dedicated source pipeline.
2. *Source pipelines* consist of the following sequence of modules:
   (a) The *Adapters* separate the connection specifics of the source system from the standards used within a pipeline. The adapter is the only sub-system of the IFW system that directly interacts with the source system. It handles functions like an IP-Allow-List, SSL-certificates, API-tokens, accounts and passwords, file-transfers et cetera. A typical implementation is based on an API gateway service. If the adapter receives individual messages, it will buffer them in a table.
   (b) The *Extractor* retrieves the data from the result table of the Adapter module. It converts technical formats (for example: JSON, XML, CSV) to strong typed database records and values. The technical formats are dictated by the source system. Type conversion may trigger a technical error status and notifications. The errors in the extractor are limited to technical data integrity, job integrity and connection errors.
   The extractor is responsible for storing a backup of the retrieved data (files). The backup is used to restart the pipeline. Since every step can be run independent it is possible that the extractor collects multiple sets of retrieved data that accumulated while waiting to be processed by the transformer step.
   (c) The *Transformer* replaces values in the new data by updating it with reference tables from the meta system. Field quality rules check for allowed

values. Column quality rules count rows. Row counts can be compared to a range of expected number of rows.

(d) The *Combiner* has a read-only database connection to the data-hub. It changes the data model from based on the source system, to the shared data model of the data-hub (also known as ODS[3]). It looks in the data-hub at existing values and relational keys and creates new unique values to be inserted. An unique key from the source system is maintained in the data-hub to resolve future updates.

(e) The *Loader* creates or updates data in the data-hub. This is the only IFW sub-system that updates or creates data in the data-hub.

3. The *Data-hub* contains the data that is to be distributed to target systems. It has no historic record of previous versions like a data warehouse. The data warehouse(s) of the organisation are considered as target system and source system of the data-hub. The data-hub contains the most recent state of the data-elements from the source systems.

   (a) The *canonical data model* (CDM) of the data-hub must be able to answer many questions by target systems, hence its independent, canonical model.

   (b) The *technical data model* (TDM) of the data-hub is the materialisation of the Canonical Data Model [18].
   The data-hub *can* consist of multiple types of data-storage solutions, with each their own Technical Data Model, to ensure fulfilling all the data requirements of the target systems. A single data-storage solution would be in a 3NF[4].
   Examples of multiple data-storage solutions are (1) an on-premise SQL master database with an SQL slave database in the cloud and (2) an SQL database in combination with a Data-lake solution in the cloud.

   (c) The *Generic Views* (GV) function as a facade between the data-storage in the data-hub and the target pipelines [8]. The complexity of the database model in the data-hub is hidden by the usage of views [2]. The views are to be re-used by multiple outgoing pipelines. Generic Views can not re-use other Generic Views as this would create a unwanted dependency. Views are created to deliver grouped data on the level of a complete business object[5].

4. The *Target pipelines* are each created for each target system. They consist of the following sequence of modules:

   (a) The *Selector* combines several generic views from the data-hub. A subset of the columns is selected to deliver only the required and allowed data for this target system, enabling data minimisation policies.

[3] The term ODS is commonly used as Operational Data Store in relation to the data-provisioning of a Data Warehouse (DWH) [6]. Currently it is more common to use the term data-hub in the scope of data-provisioning to operational applications within an organisation.

[4] See https://en.wikipedia.org/wiki/Third_normal_form

[5] An architectural name to describe a collection of attributes that together have a meaning to business people, also known as information.

(b) The *Filter* filters a subset of the rows, again enabling data minimisation policies. Changes on both the column and row filters are explicitly under governance of the privacy officer[6] (CPO) and security officer[7] (CISO) [17]. Their policies determine the classification of the target system connected to this instance of the pipeline [17]. Hence the data that is allowed to be delivered to the target system. The list of attributes to be delivered will be recorded in the data-delivery register, enabling GDPR[8] compliance and processes.
(c) The *Transformer* again transforms from one data model (in this case the CDM via the TDM) to the data model of the target system. It can also apply changes to the data using the reference tables.
(d) The *Adapter*; the equivalent of the Adapter in the incoming pipeline. It has the same dependency, but now to the target system. There are three types of adapters present in the IFW solution: (1) a generic adapter using API technology and services that resemble the Logical Data Model, (2) a generic adapter using API technology and services that are specified in a domain standard (for example: open banking api, open education api) and (3) system specific adapter.

5. The *Target systems* receive data from the IFW solution. For every data-element in the data-hub CDM there is a source system defined. Every other system is a potential target system of this data-element. When a system needs data that is not part of their assigned ownership, the required data-elements are retrieved via a target pipeline.
6. Meta-data
   Each of the modules maintains its own status in a globally available table. The status "is_enabled" in the Loader module enables testing of the complete source pipeline and executing the quality rules, feeding back results based on production data without actually changing the data-hub and feeding to target systems. This is a remediation of privacy concerns during testing.
   Severity levels of a failing quality rule will be logged. The next module will use this to determine if it is allowed to start.
   Log messages will be used to automatically report about any data elements that failed, to the responsible owner of the source system. It enables quality improvement of the reused information (not data) on an organisational level.

5.2 IFW governance architecture

The second part of the IFW is the governance processes and agreements.

Creating a pipeline starts based on a new target system that needs data. If the required data is already available in the data-hub, only a target pipeline will be created. If the required data is not available in the data-hub, the team looks for a new existing source system to be connected with a new source pipeline.

[6] See https://en.wikipedia.org/wiki/Chief_privacy_officer
[7] See https://en.wikipedia.org/wiki/Chief_information_security_officer
[8] See https://eur-lex.europa.eu/eli/reg/2016/679/oj

For each pipeline the Information Manager (IM) signs an internal agreement with the functional owner of the connected system. This agreement describes the responsibilities of the stakeholders, the data quality requirements, privacy limitations and includes a list of the data elements. The data ownership is delegated from the functional owner of the source system to the IM. The IM signs a agreement with the functional owners of the target systems (see Fig. 2). Compared to the classic data-delivery agreements (spanning from source to target), this decoupling aims to simplify the negotiations. The explicit use of the agreements aims to increase the security, privacy and data quality awareness.

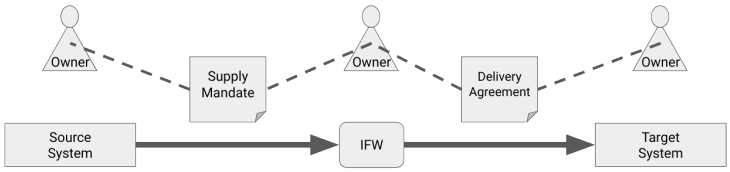

Fig. 2. Two separate relations and respective agreements.

5.3 Application of Normalized Systems Theory

The following customised interpretation of the Normalized Systems Theory (NST) [12,4,14] was given to the designers:

The agility and changeability and hence the maintainability of a composite system is determined by the difficulty of changing the entire system. The focus here is on preventing ripple (combinatorial) effects as an effect of changing a sub-system.

1. **Separation of concerns:** a change must not have an impact on more than one sub-system, this is also called functional loose coupling. Each sub-system has only one responsibility.
2. **Data Version Transparency:** the version of an data element type is included in the data element meta-data. Applications that process information only use the version that they know.
3. **Action Version Transparency:** the version of an action element is included in the action element. Applications invoke an action element only with a version that they know.
4. **Separation of States:** after each action in a series, the intermediate result is saved. Every sub-system is a real black box. Other sub-systems do not need to know what the inside of another system looks like. They can ask the sub-system what it does and what information can be delivered or requested.

Designers are given this set of architecture principles:

1. **Principle: Minimise dependencies**
   Rationale: Systems change, thus the integration system like the IFW will change forever. These changes will be problematic if there are many dependencies.
   Implications: It can be cumbersome to prevent dependencies. This investment will pay for itself in the long run.
2. **Principle: business objects are recorded in the Canonical Data Model.**
   Rationale: Business objects consist of a considerable number of attributes, rarely supplied by a single source system. In the Canonical Data Model (CDM), the business objects with an unambiguous meaning are not related to the model from a source system.
   Implications:
   (a) Business objects and their attributes are described in the CDM.
   (b) Attributes of a business object can be provided by multiple source systems.
   (c) The CDM is based on industry, general and market standards.
   (d) The CDM contains only structured and machine readable fields.
3. **Principle: Everything has a version.**
   Rationale: Explicit granting and simultaneous use of different versions gives more flexibility because systems and links do not have to be changed simultaneously when changing another system or link.
   Implications:
   (a) All elements, building blocks, steps, tables, views et cetera have a version.
   (b) Every update of a building block or step with a functional difference gives a new version. Other building blocks never automatically use the newer version.
   (c) Versions exist side by side and are used simultaneously.
   (d) All links (system calls, web services, FTP folders, references to tables, etc.) explicitly refer to a version of an implementation.
   (e) The (version) naming standard is applied and maintained.
   (f) Implementations of new versions of elements are subject to a formal change process.
4. **Principle: All prescribed building blocks always exist and are exclusively automated.**
   Rationale: Only if all building blocks in the IFW are designed and realized, there is independence from all building blocks in the chain. Manual processing steps are not allowed because this creates a dependency on an (interpretation of a) person. The only meaning of the IFW is integration.
   Implications:
   (a) There is no building block that has presentation functions. This is reserved for portals and user interfaces.
   (b) There is no building block that realises business logic and business processes. This is a task of the target or source systems or a special system for automated business processes.

(c) There is no building block that realises reporting and analysis functions. This is a task for target systems such as the data warehouse (DWH) or Management Information System (MIS)
(d) Messages that lead to an error situation in the chain or in the target systems (both technical and functional) must be corrected by an adjustment in the source.
(e) There is no functionality available for manually creating, changing or deleting messages other than in the source.

5.4 Solution realisation

Based on an explicit requirements by the client, the design is created using Microsoft components. Hence the adaptors being based on the Azure API management services and the databases implemented by a Microsoft SQL server. These databases are on-premise because they mainly connect to on-premise systems. The IFW modules are implemented as SSIS packages[9], usually created by a BI developer. However, the architect and designer did choose for the use of BIML[10], a XML based language that is used to generate the SSIS packages. The BIML scripts make use of meta-data in tables in the SQL server and maintained with a Microsoft Access GUI. The quality rules are also generated during design time. This generation is comparable to a pattern expansion process. In runtime, a copied subset of the meta-data is used as reference tables (validation of allowed values and replacement of values) and thresholds for the quality rules.

6 Reflection

We collected and grouped the following observations on the realised IFW at this organisation:

1. Applied architecture principles in respect to NST
   (a) Principles 1 and 2 are translations of avoiding combinatorial effects requirement and the separations of concerns theorem.
   (b) Principle 3 translates the data and action version transparancy theorem.
   (c) Principle 4 is based on anticipated changes of adding functionality. The chosen tactic is to include all possible building blocks from the start, even if they initially do nothing. Adding functionality later is limited to enabling functions in the empty building blocks, not changing the structure and relations.
   (d) There is no principle that translates the separation of states theorem. This is part of the IFW architecture model that describes the building blocks, the way they handle data locally and handle state using shared meta data.
2. Applied architecture principles in respect to building the solution

[9] See https://en.wikipedia.org/wiki/SQL_Server_Integration_Services
[10] See https://en.wikipedia.org/wiki/Business_Intelligence_Markup_Language

(a) Although the solution designers and developers were given freedom to interpret the principles themselves, they frequently requested guidance by an architect.
(b) Instead of the term ”preventing combinatorial effects” the team used the term ”blocking domino effects”.

3. Solution design of module size
   (a) We created expander scripts to generate modules as described in the system structure (see Fig. 1). The size of these atomic elements of the system is too large. This results in some repetition of the sub-functions within a pipeline module and in combinatorial effects on the lowest (SSIS package) level. Improved examples for the smallest elements could be:
      i. reading the status from the meta-data and deciding if the module is allowed to start
      ii. application of quality rules
      iii. replacing a value
      iv. copy a value (database table to database table)
      v. (re)creating result tables
      vi. writing status or logging
4. Applied solution in respect to the data-hub
   (a) A 3NF for the data-hub is cumbersome to design. There have been discussions to apply a different data model form such as a snow flake. A supporting argument would be that this form is easier to generate based on meta-data. This is not tested nor implemented as it would impact the design of then existing loader modules. This is already a combinatorial effect in the current architecture and design. However, the generic views module as described in the design will act according to the Separation of States theorem, effectively stopping the combinatorial effects to the outgoing pipelines. This separation allows for a incremental update of the data-hub.
   (b) The architecture describes the data-hub and the 3NF as a Canonical data model (CDM). During design we recognised that the generic views of the data-hub are the CDM. The 3NF database design should be seen as part of the black box of the data-hub.
   (c) The use of meta-data, both during design time and run time is not optimal. On the one hand using the same database tables in development and production is out of the question. On the other hand, making copies introduces duplicates and unwanted complexity.
5. Applied solution in respect to adaptors
   (a) Recognised combinatory effects: Adaptors highly depend on the technical implementation of the source system. Views are dependent on the data-hub design. During the design phase these effects are recognised. Detailed measures are taken to ensure the effect is only related to the next module. This is preventing further domino effects. The combiner and loader both depend on the (canonical and technical) data model of the data-hub. This cannot be prevented as the combiner converts the incoming data

to the model of the data-hub, and the loader inserts and update data within the constraints and design of the data-hub data model.

6. Governance
   (a) Getting the responsible business owners to commit to the proposed agreements (mandate and usage) turned out to be a hassle. It did cost a lot of explaining to convince them of the necessity of the agreements. It did also take some time to find the right level of details for the description of the data elements.
   (b) the ability to separate the creation and deployment of source pipelines from target pipelines is an improvement. In the applied agile way of working the individual backlog items turned out to be close to a module in a pipe line. Several tasks within this item could be worked on by different developers. This increased the understanding on the progress of the development.

6.1 Validating the observations

To validate our observations, we interviewed four people who where also directly involved in the design and development of the system; The product owner, the development manager, the senior designer/developer, and a developer/sysadmin.

At the start of each interview we briefly explained Lehman's law and the goal of our paper and interview. We presented our observations 2a, 2b, 3a, 4a, 4b, 4c, 5a, 6a, and 6b for validation in the form of "Can you agree that …?" Observations 1a..d were skipped because these explain the mapping between the NS theorems and the architecture principles and therefore not suitable for validation by the interviewees. We added a question to retrieve insight in the perception of the evolvability of the system: "7: Now that the system has about 20 pipelines, compared to the beginning when the system had two pipelines. What is your global impression: Has the effort of making a comparable change stayed the same?".

For each question the answers where limited to the following options: "not applicable" (for instance when the interviewee was not involved in that topic and did not know an answer), and "strongly disagree", "disagree", "agree", or "strongly agree". By leaving out the option for a neutral answer, we forced the interviewees to make a explicit choice. All interviews were done by author one as interviewer, carefully using comparable wording.

Based on the summed scores, the interviewees confirmed all except observations 3a and 4a (see table 1).

Regarding observation 3a, about the module size in the solution design, that it should have been designed smaller; The product owner and development manager disagreed. The senior designer/developer strongly disagreed, even suggested they should be larger. The developer agreed.

In respect to observation 4a, about 3NF being cumbersome to design; The two managers both strongly agreed, giving arguments that it took a lot of time, effort and thus money. The two developers both disagreed, giving arguments

Table 1. Results of four validation interviews

		n.a.	strongly disagree	disagree	agree	strongly agree	
observation	2a	0	0	0	3	1	
observation	2b	0	0	0	2	2	
observation	3a	0	1	2	0	1	--> not confirmed
observation	4a	0	0	2	0	2	--> not confirmed
observation	4b	0	0	0	4	0	
observation	4c	2	0	0	2	0	
observation	5a	0	0	0	1	3	
observation	6a	0	0	0	1	3	
observation	6b	0	0	0	1	3	
added question	7	0	0	0	2	2	

that the 3NF was not easy but do-able and most of all necessary to be able to deliver all possible combinations towards outgoing pipelines.

The added question (7) about the effort related to the system size was agreed upon by all four interviewees.

7 Conclusion

This paper reports about an application of Normalized Systems Theory (NST) in the context of a data integration solution. A summary is given of our interpretation of NST, as given by the architect to the designers of the integration software. This case report is part of the practitioners double learning loop.

In this case, the application of the Normalized Systems Theory helped the practitioners to improve the adaptability of the data integration solution. In our view it is definitely better but not perfect.

As the design team was unfamiliar with NST, the use of the NST theorems within the context of an architecture model and architecture principles, indicated the need for continuous decision-making support. Using the term ”blocking domino effects” improved comprehension of the team’s design challenge.

The SSIS-packages, part of the Microsoft SQL suite, can be generated using BIML. This case report indicates that this combination can be used as a Normalized Systems Theory expander. However, the optimal module size is disputable.

Based on our validated observations we can state; There are indications that this IFW design, and architecture principles, our interpretation of NST, disconnect the relation between the size of the application and the impact of a change. Thus mitigating the effects of Lehman’s Law and improving the evolvability of the created EAI solution.

8 Future Research

Based on the experience at this specific organisation, the IFW architecture was applied and realised at other Dutch educational organisations. Each with different technologies and various implementation strategies. Future research could be collecting these individual use cases and combine it with a post implementation questionnaire to compare the perceived (added) value of the IFW architecture and the NST application.

Research could be done to find the smallest element to generate using the BIML scripts.

Research could be done to automate the generation of the Technical Data Model of the data-hub. This would save a significant amount of time and effort.

Acknowledgements

We would like to thank the involved Dutch educational organisations in supporting the collaborative open discussion on the topic of enterprise application integration, and willing to exchange the knowledge between their organisations. We also would like to thank our (former) colleagues at Sogeti for reflecting on our concept architecture descriptions and adding their views and experiences.

We see the creation of IFW as an example of combining academic research, the operational application of this knowledge in production and sharing the experience back to the academic field.

References

1. Chongsombut, O., Verelst, J., De Bruyn, P., Mannaert, H., Huysmans, P.: Towards applying normalized systems theory to create evolvable enterprise resource planning software: a case study. In: The Eleventh International Conference on Software Engineering Advances (ICSEA). vol. 11, pp. 172–177. IARA, Rome, Italy (2016), https://hdl.handle.net/10067/1352400151162165141
2. Dayal, U., Hwang, H.Y.: View definition and generalization for database integration in a multidatabase system. IEEE Transactions on Software Engineering **SE-10**(6), 628–645 (1984), https://doi.org/10.1109/TSE.1984.5010292
3. Erasala, N., Yen, D.C., Rajkumar, T.: Enterprise application integration in the electronic commerce world. Computer Standards & Interfaces **25**(2), 69–82 (2003), https://doi.org/10.1016/S0920-5489(02)00106-X
4. Huysmans, P., Oorts, G., De Bruyn, P., Mannaert, H., Verelst, J.: Positioning the normalized systems theory in a design theory framework. In: International Symposium on Business Modeling and Software Design. pp. 43–63. Springer (2012), https://doi.org/10.1007/978-3-642-37478-4_3
5. Huysmans, P., Verelst, J., Mannaert, H., Oost, A.: Integrating information systems using normalized systems theory: Four case studies. In: 2015 IEEE 17th Conference on Business Informatics. vol. 1, pp. 173–180. IEEE (2015), https://doi.org/10.1109/CBI.2015.43

6. Inmon, W.H., Linstedt, D.: 3.5 - the operational data store. In: Inmon, W.H., Linstedt, D. (eds.) Data Architecture: a Primer for the Data Scientist, pp. 121 – 126. Kaufmann, Morgan, Boston, USA (2015), https://doi.org/10.1016/B978-0-12-802044-9.00019-2
7. Irani, Z., Themistocleous, M., Love, P.E.: The impact of enterprise application integration on information system lifecycles. Information & Management **41**(2), 177–187 (2003), https://doi.org/10.1016/S0378-7206(03)00046-6
8. Ku, C.S., Marlowe, T.J., Budanskaya, T., Kang, P.: Software engineering design patterns for relational databases. In: Proc. International Conference on Software Engineering Research and Practice. vol. II, pp. 340–346. SERP '07, CSREA Press, Las Vegas, Nevada, USA (2007), https://www.researchgate.net/publication/221611033
9. Lehman, M.M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE **68**(9), 1060–1076 (1980), https://doi.org/10.1109/PROC.1981.12005
10. Linthicum, D.S.: Enterprise application integration. Addison-Wesley information technology series, Addison-Wesley Professional, Boston, USA (2000), http://www.worldcat.org/oclc/890643434
11. Mannaert, H., Verelst, J.: Normalized systems: re-creating information technology based on laws for software evolvability. Koppa, Kermt, Hasselt, Belgium (2009), http://www.worldcat.org/oclc/1073467550
12. Mannaert, H., Verelst, J., De Bruyn, P.: Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design. NSI-Press powered by Koppa, Kermt, Hasselt, Belgium (2016), http://www.worldcat.org/oclc/1050060943
13. Mannaert, H., Verelst, J., Ven, K.: Towards evolvable software architectures based on systems theoretic stability. Software: Practice and Experience **42**(1), 89–116 (2011), https://doi.org/10.1002/spe.1051
14. Mannaert, H., Verelst, J., Ven, K.: The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability. Science of Computer Programming **76**(12), 1210–1222 (2011), https://doi.org/10.1016/j.scico.2010.11.009, special Issue on Software Evolution, Adaptability and Variability
15. Oorts, G., Ahmadpour, K., Mannaert, H., Verelst, J., Oost, A.: Easily evolving software using normalized system theory-a case study. In: ICSEA 2014 : The Ninth International Conference on Software Engineering Advances. pp. 322–327 (2014), https://www.researchgate.net/publication/283089164
16. Oorts, G., Huysmans, P., De Bruyn, P., Mannaert, H., Verelst, J., Oost, A.: Building evolvable software using normalized systems theory: A case study. In: 2014 47th Hawaii International Conference on System Sciences. pp. 4760–4769. IEEE (2014), https://doi.org/10.1109/HICSS.2014.585
17. Papelard, T., Bobbert, Y., Berlijn, D.: Critical Success Factors for Effective Business Information Security. Uitgeverij Dialoog, Zaltbommel, The Netherlands (2018), http://www.worldcat.org/oclc/1088899915
18. Saltor, F., Castellanos, M., García-Solaco, M.: Suitability of datamodels as canonical models for federated databases. ACM Sigmod Record **20**(4), 44–48 (1991), https://doi.org/10.1145/141356.141377