Toward Evolvable Document Management for Study Programs Based on Modular Aggregation Patterns

Gilles Oorts and Herwig Mannaert

Normalized Systems Institute
University of Antwerp
Antwerp, Belgium
Email: gilles.oorts,herwig.mannaert@uantwerp.be

Ilke Franquet

Unit for Innovation and Quality Assurance in Education
Faculty of Applied Economics
University of Antwerp, Belgium
Email: ilke.franquet@uantwerp.be

***Abstract*—Despite technological and operational business advances over the past decades, organizations are still required to draft and manage documents. Although a lot of these documents have taken an electronic form, their structure is in essence still the same as their analogue and physical predecessors. In this paper, we present a different view of documents as we imagine them as modular structures. Based on the patterns of the artifacts they describe, documents can be modularized and decomposed into fine-grained text modules. This leads to easier maintenance of the text modules as they offer a clear aggregate structure and any information is stored in only one text module. This enables re-usability and allows for a greater versatility of the information stored in the text modules. One can generate several new types of documents with different purposes as to the ones imaginable at this moment. All of this enables the creation of truly evolvable documents according to the Normalized Systems theory.**

Keywords–Normalized Systems theory; Modularity; Document Management; Prototype; Evolvable Documents; Modular Documents; Text Modules.

I. INTRODUCTION

Despite technological and operational business advances over the past decades, organizations are still required to draft and manage documents. These documents can take a plethora of forms, such as books, spreadsheets, slide decks, manuals, legal contracts, emails, reports, etcetera. Although a lot of these documents have taken an electronic form, their structure is in essence still the same as their analogue and physical predecessors. Invoices are often just printed and sent by mail, after which they are opened and scanned by the receiving organization. Or instead of printing and handing out new operational procedures, they are often just exported as a pdf-file and saved on a server.

Despite the endless opportunities the revolution in Information Technologies (IT) brought along, most efforts in document management were limited to just digitizing documents, i.e., transforming them from analogue to digital form as monolithic blocks. In this paper, we show how this view of static documents that are a mere representation of their analogue predecessors is out-of-date. Instead, we will present a view of multidimensional and ever-changing documents, based on the insights from modularity and Normalized Systems reasoning. The practical implications of this view will be discussed based on a case study of a document management system for study program documentation.

In Section II, we will first demonstrate the need for variability and evolvability in documents. Next, we will show how to achieve these document characteristics using the principles of modularity and evolvability based on Normalized Systems theory in Section III. To illustrate this approach we first introduce the case of study programs in Section IV before discussing a prototype of such a document system in Section V.

II. THE NEED FOR EVOLVABILITY AND VARIABILITY OF DOCUMENTS

Documents are rarely invariant artifacts. In todays competitive business environment, companies need to be able to adapt to changing requirements of customers, government, competitors, suppliers, substitute products or services, and newcomers to the market [1]. These changes also require adaptations to the documents used in the organization. As these documents are managed in a digital way and can be easily edited by multiple people throughout time, they are changed more frequently and have several concurrent variants. In terms of *change* over time, consider for instance the following change events:

- a new legislation may require companies to add additional safety measures to their operational guidelines in order to avoid oil leaks on drill platforms;
- a software or product manual may need to be updated because a new version with added functionality or fixed bugs was designed and is put into production;
- an audit report may need to be updated with new information about the audited objects or new auditing criteria;

These are just a few examples of business changes that require adaptations of documentation. Enumerating a full list of change events that require documentation changes in contemporary businesses is impossible, as they are countless. For this reason, documents need to be designed to be changed with ease -to be *evolvable*- from the start. This will be discussed in the next sections of this paper.

The continuous change of documents also contributes to the creation of *variability* in documents. Adaptations in documents do not necessarily lead to the deletion of the previous document, as both versions might need to exist. Consider for instance the following possible variants [2]:

 ISBN: 978-1-61208-534-0

- a similar slide deck on a subject may be created for a one day seminar to a management audience, a one week course for developers, a full-fledged course for undergraduate students;
- a product manual may be drafted in different languages, several product variants (standard – professional – deluxe) may contain a partly overlapping set of production parts requiring similar yet different manuals, etcetera;
- similar, but slightly different, legal documents (contracts) may be drafted for different clients purchasing the same service (based on the same contract template), etcetera;

These are of course just a few examples of how different versions of a document can arise. To manage the concurrent and consequential document variants, most companies use so-called Document Management Systems (DMS). To the best of our knowledge, these systems store the documents at the "document" level. As we will discuss in this paper, we propose a solution to store and manage documents at more fine-grained modular levels, enabling the creation of evolvable and reusable documents.

III. Modular and Evolvable Documents

The concept of modularity has proven to be a very successful as a design principle in various settings. It has been cited to be very useful in product, system and organizational design [3][4].

Based on these insights, it was demonstrated how systems such as accountancy, business processes and enterprises can be regarded as modular systems in previous work [5][6][7]. This research shows that applying the modularity principle to systems entails benefits in the design, maintenance and support to the system.

We are convinced that documents can be considered to be clear examples of modular structures. Take for instance these examples [2] :

- A book or a report typically consists of a set of *chapters*. Each of these chapters will contain a set of *sections*, subsections, subsubsections, and so on. Each of these (sub)sections can then contain *paragraphs* with the actual text, tables and/or figures;
- A product manual will contain guidance *sections* regarding the different product parts and/or functionalities;
- A legal document may contain different *parts*, within each part different *clauses*, and each clause may contain different *paragraphs*.

All of these document parts can be considered to be modules. In our approach, we define a module as a part of the system that is used or activated separately. Once a part of the system cannot be used or activated as such, it is considered to be on a sub-modular level.

Modularity is however but a prerequisite in obtaining adaptive documents. For documents to easily assimilate changes over time, they need to exhibit evolvability. Based on the modularity concept, *Normalized Systems (NS) theory* was proposed to achieve such modular evolvability. Although originally defined for software architectures, its applicability and value in other domains (e.g., organizational design, business processes, accountancy) quickly became clear [5][6][7].

To obtain flexible systems that can easily evolvable over time, NS theory states that so-called *combinatorial effects* should be eliminated. These effects occur when changes to a modular structure are dependent on the size of the system they are applied to [8]. This means the impact of the change does not solely depend on the nature of change itself. Assuming systems become more complex over time, combinatorial effects would therefore become ever bigger barriers to change. As such, it is clear how combinatorial effects should be avoided if systems need to be changed easily (i.e., be evolvable).

To obtain evolvability, NS theory proposes four *theorems*, two of which are of importance in this paper [8]:

- *Separation of Concerns*, stating that each change driver (concern) should be separated from other concerns. This closely relates to the concept of cohesion;
- *Version Transparency*, stating that modules should be updatable without impacting any linked modules;

In practice, the consistent application of these theorems results in a very fine-grained modular structure.

The theory also defines *cross-cutting concerns*. This concept is often used in information technology and refers to functionality or concerns that cut right across the functional structure of a system. These cross-cutting concerns should also be encapsulated to exhibit any form of evolvability. As we will illustrate in this paper, this is not self-evident as the functionality of these concerns are embedded deep down within systems.

An important cross-cutting concern in documents is a mechanism for "relative" embedding of text parts in the hierarchical structure of overarching documents. This means one should be able to include a text module on several hierarchical levels in a document without this inclusion causing any changes in the text module. As such, a text module can be variably used as a chapter, section, subsection, etc. without any changes to the module. Preliminary research shows there are several other cross-cutting concerns for documents, such as for example typesetting (layout), language, target audience, etc.

Besides the cross-cutting concerns resulting form the nature of documents, there are also concerns specific to the underlying artifact(s) described in the document. These are mostly cross-cutting concerns that stem from content or descriptions of the artifact(s). Take for example technical documents describing the machines used in the production process of a manufacturer. These documents will contain machine specifications, operating instructions, power requirements, maintenance instructions, etc. This are necessary subjects needed in the description of every machine and can as such be defined as cross-cutting concerns according to the previous stated definition.

Based on these concepts of modularity and evolvability based on Normalized Systems Theory, a prototype was built to manage the documents of the study programs at the faculty of Applied Economics at the University of Antwerp.

 ISBN: 978-1-61208-534-0

IV. Study Program Design at the Faculty of Applied Economics

Before we can study program documentation, we first need to take a look at the underlying artifacts. The study programs at the Faculty of Applied Economics at the University of Antwerp were recently redesigned to be modular and evolvable. Naturally, an evolvable study program design enables all related documents to be adaptable as well. Furthermore, the well-defined modular structure of the study programs allows for new possibilities in generating related supporting documents.

The new study program design was formulated to include learning-teaching tracks and sub-tracks. As such, additional levels of modularity were added to the existing 258 courses offered in five distinct bachelor study programs and seven study programs at a master level. As proposed in previous work [9], this leads to the study program design shown in Table I. Each of the 258 courses belongs to one main (sub)track, but can be connected to other (sub)tracks as it may contain subject matters belonging to several (sub)tracks.

Besides the addition of two new modular levels in the study program design, considerable efforts were put into defining cross-cutting concerns that manifest themselves in the courses taught in the faculty. In total, 10 cross-cutting concerns were identified specific to the studied artifacts (i.e., study programs). These concerns are a short content description, regular content description, internationalization, blended learning, assignments, ethical awareness, sustainability, social impact, learning outcomes and teaching method(s). These cross-cutting concerns represent important aspects of a study programs, and therefore its underlying learning-teaching tracks and courses. In Figure 1, some of these cross-cutting concerns are presented on the vertical axis. On the horizontal axis, the learning-teaching tracks and sub-tracks are listed, each with the included courses. Besides allowing to check the presence of certain concerns in courses and learning-teaching tracks, this matrix shows the extensive modular design of documents describing the courses and learning-teaching tracks. How we design and generate documents to support this modular and evolvable study program design will be discussed in the next section.

TABLE I. OVERVIEW OF THE LEARNING-TEACHING TRACKS AND SUB-TRACKS

Learning-teaching track	Sub-track
General economics	Fundamentals
	Policy
Business economics	Accountancy
	European and international business
	Finance
	Marketing
	Strategy and organization
	Transport and logistics
Engineering	Fundamentals
	Sustainable technology
	Supply chains and operations
Information systems	Fundamentals
	Engineering and architecture
	Governance and audit
Quantitative methods	Mathematics
	Statistics
Practice	Apprenticeship and internship
	Summer school
Broadening areas of study	Social sciences
	Jurisprudence
Business communication	English
	French
	German
	Spanish
Projects and dissertations	Bachelor project
	Master dissertation
	Master integration project

V. A Prototype for Generating Study Program Documents

A. Decomposing Documents into Text Modules

The new modular and evolvable design of the study programs allowed for a similar redesign of documents describing the study programs. Therefore the existing content describing the courses was looked at and modularized to allow the generation of different kind of documents. Previously, most content on study programs was contained in course descriptions that were published on the faculty's website. From this descriptions, text modules with similar content were identified. In total, 10 types of text modules were recognized. These are the content cross-cutting concerns mentioned in the previous section and include for example a short content description, internationalization, etcetera. Combined, these 10 types of text modules allow for a complete representation of the courses. And as learning-teaching tracks and study programs are considered to be mere compositions of courses according to modularity reasoning, the text modules can be used to represent these parent artifacts as well. Taking into account the total number of 258 courses and 10 content cross-cutting concerns, the modularization of the course descriptions resulted in a total of $258 * 10 = 2580$ text modules. These represent all aspects of the courses, learning-teaching tacks and study programs of the faculty.

Although this amount of text modules seems cumbersome to achieve and maintain, this fine-grained decomposition actually simplifies several aspects of document management. First, this imposed separation of concerns creates structure across all course descriptions. This gives professors (who are responsible for the content of the text modules) something to hold on to in describing their courses. Furthermore it is easier to retrieve certain information, as the text modules are in separate files. This also allows for easier maintenance of the information. But by far the biggest advantage of the decomposition is the endless possible document types that can be generated with the decomposed course descriptions. The information included in the decomposed text modules allows for the generation of a vast variety of documents with different purposes. This system for example allows one to generate documents containing a short description of all courses in a study program. But the system can also generate a document listing all courses or learning-teaching tracks using a specific teaching method. Furthermore, if students were to be added, the system would allow to generate a document detailing all sustainability or social impact aspects a student has encountered in his study program. Or how much hands-on experience he has gained due to assignments or case studies. As such, it allows to draw up student-specific diploma's with one click of a button. In general, the decomposition thus allows for a versatile use of document modules and allows the definition of new types of documents with new purposes.

B. Relative Sectioning

As mentioned before, an important aspect of allowing text modules to be re-usable is to implement relative sectioning.

 ISBN: 978-1-61208-534-0

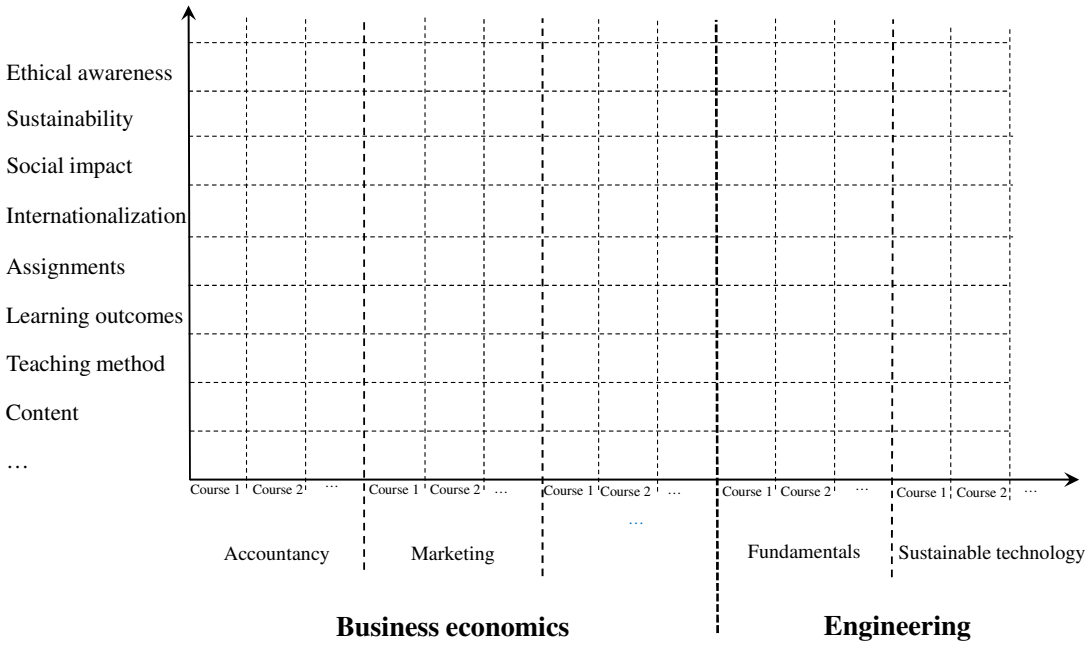

Figure 1. The cross-cutting concern presence in learning-teaching tracks and courses

To implement this prototype, the LaTeX document preparation system was used. This was done because it allows the hierarchical inclusion of sub-files (i.e., text modules) and allows the layout cross-cutting concern to be handled in a separate layout file. LaTeX however does not provide a system for relative sectioning out of the box. The hierarchical structure of sections (i.e., whether something is a chapter, section, subsection, subsubsection, etcetera) should be hard coded within .tex files and therefore limits the potential for text modules to be freely combined into final documents which might use the same text excerpts at different levels within their own document hierarchy. To overcome this problem, a LaTeX style file needed to be used that provided the functionality of relative sectioning [10]. This allows the prototype to generate a LaTeX structure file, of which the first part is shown in Figure 2. In this file, text modules are imported via the `\input{}` command. The names included in this command are the files that should be part of the generated document. More importantly, the `\leveldown` and `\levelup` commands are automatically added by the prototype whenever the next text module of the document should be added on a lower or higher level. As such, the basic text modules exist of solely a title (included within the `\dynsection{}` that is provided by the custom style file) and the content of the module.

```
\input{input/"Content description"}
\leveldown
\input{input/"Business Economics"}
\leveldown
\input{input/"Business Economics_Accountancy_content "}
\input{input/"Business Economics_European and International Law_content "}
\input{input/"Business Economics_Finance_content "}
\input{input/"Business Economics_Marketing_content "}
\input{input/"Business Economics_Strategy and Organisation_content "}
\input{input/"Business Economics_Transport and Logistics_content "}
\levelup
\levelup
\input{input/"Assignments"}
\leveldown
\input{input/"Business Economics"}
\leveldown
\input{input/"Business Economics_Accountancy_assignments"}
\input{input/"Business Economics_European and International Law_assignments "}
\input{input/"Business Economics_Finance_assignments "}
\input{input/"Business Economics_Marketing_assignments "}
\input{input/"Business Economics_Strategy and Organisation_assignments "}
\input{input/"Business Economics_Transport and Logistics_assignments "}
\levelup
\levelup
…
```

Figure 2. The LaTeX Structure File generated by the prototype

C. Practical implementation

The practical implementation of the prototype was done in Java. A graphical user interface (GUI) was developed that allows documents to be developed as easily and efficiently as possible. The result of this effort is shown in Figure 3. In this screen, the user can enter the document title (which will also be the file name) and create up to three document levels. This is done by first selecting the content of a level, which can be cross-cutting concerns, learning-teaching tracks, sub-tracks or courses. When this selection is made, the user must specify whether he wants one single instance of content on that level, multiple instances or all of them (by using the "Select all" button). In this way, one or multiple levels can be defined.

Once the user made his selection, he can press the "Generate" button to start the process of generating the document he specified. At this point, the system will generate two LaTeX files. First, a "Structure" file is created, which is partially shown in Figure 2. This file is procedurally generated based on the selections the user made. It takes into account the amount of document levels and amount of instance selections on each level.

Next, a "Generator" file is created by the system, of which an example is shown in Figure 4. This file contains code needed for LaTeX to generate a PDF version of the designed

 ISBN: 978-1-61208-534-0

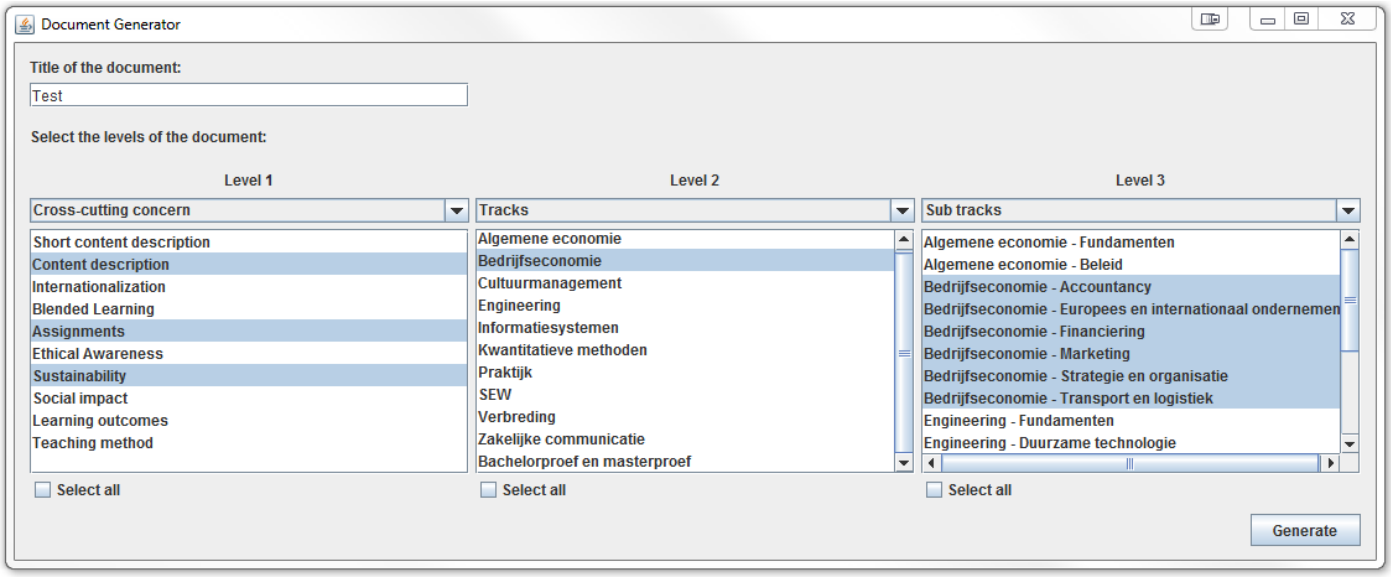

Figure 3. The Graphical Interface of the Prototype

```
\documentclass[a4paper]{report}

\usepackage{blindtext}
\makeatletter
\newif\ifusedot
\usedottrue

\usepackage{relsec}
\setcounter{secnumdepth}{3}
\setcounter{tocdepth}{3}

\title{Test}
\makeatother

\begin{document}
\maketitle'
\usedotfalse

\include{Test_StructureFile}

\end{document}
```

Figure 4. The LaTeX Generator File generated by the prototype

document. Apart from the LaTeX -specific code, this file simply contains the `\include{}` command to refer to the structure file and as such the structure and text modules defined in that file. This shows how the structure of the document is also clearly separated from the generation implementation, according to the separation of concerns principle.

Once these two files have been created, the system simply uses the LaTeX document generation functionality to generate a PDF file of the document.

D. Document Versatility, Variability, and Evolvability

Having described the prototype, we can now illustrate the possibilities of *document versatility* this system provides. Let's explain this in numbers. As mentioned previously, the faculty offers 12 study programs (5 Bachelor and 7 Master programs). For each study program, one can generate a document containing three document levels. Abstracting from the courses to make things easier, there are 3 possible selections for the first document level (i.e., cross-cutting concerns, learning-teaching tracks and sub-tracks). This means there are only two possible selections for the second level (the two remaining ones), and two possible selection for the final level (i.e., either choosing the remaining selection or not including a third level). This totals up to 12 possible selections for the document levels. Considering either including or not including the 10 cross-cutting concerns, the amount of combinations adds up to $2^{10} = 1024$ possibilities. Multiplying the 12 study programs, 12 possible document level selections and 1024 possible combinations of 10 cross-cutting concerns inclusions gives us a total of 147,456 possible document combinations that can be generated based on the 2,580 defined text modules.

If the approximate 3,000 students of the faculty were to be included in the system, the document versatility would increase exponentially. Let's assume of all students, there are 1,000 unique versions of study programs, which is a cautious estimate considering the amount of courses students can choose in some study programs. Substituting the 12 study programs by 1,000 study program versions in the previous multiplication results in 12,288,000 possible document combinations. This example clearly shows the combination potential of decomposing course descriptions into fine-grained text modules.

The decomposition in text modules also allows more fine-grained version control to manage the *variability* in all types of documents that can be generated. If version control is managed on a text module level, changes can be tracked more specifically. Each version of a text module can be archived based on their moment of change, allowing the generation

 ISBN: 978-1-61208-534-0

of documents according to specific time specifications. One important application of this version control system is for example the re-generation of a student diploma after it has been lost. It may have been a few years since the student graduated, so courses and study programs will have changed. Yet it is important for a university to be able to generate the diploma with the correct descriptions of the version of the courses the student took. This example shows the importance of tracking changes on a fine-grained modular level.

The implementation of modular text modules also shows the importance of eliminating combinatorial effects to achieve *evolvability*. A change in the description of a course needs to be made in only one of the 2580 files/text modules. By creating a script that regenerates all documents in which this module is included, this change is easily applied to all documents it is included in. As such, combinatorial effects are avoided and the system generates evolvable documents.

VI. Conclusion

In this paper, we presented an alternative to the view of static and monolithic documents. By applying the concept of modularity and decomposing documents into text modules, several advantages can be achieved. First, modularity leads to easier to maintain text modules. This because the modules show a clear structure and specific information is stored in only one module that is easily recognized. Second, the text modules enable a greater versatility: new types of documents can be composed by combining text modules in new ways. As such new types of documents can be created with new goals and purposes. This is shown in the paper by calculating the number of possible document combinations. And finally, the systematic decomposition of modules allows for the elimination of combinatorial effects to create evolvable documents. These advantages of modular document design are clarified in the paper by the description of system to generate documents containing study program information.

In future research, other cases will be studied to corroborate the findings of the case discussed in this paper. Furthermore, the theoretical basis of modularity and evolvability of documents will be solidified.

References

[1] M. E. Porter, “Strategy and the Internet.” Harvard Business Review, vol. 79, no. 3, 2001, pp. 62–78, 164.

[2] 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.

[3] C. Y. Baldwin and K. B. Clark, Design Rules: The Power of Modularity Volume 1. Cambridge, MA, USA: MIT Press, 1999.

[4] D. Campagnolo and A. Camuffo, “The Concept of Modularity in Management Studies: A Literature Review.” International Journal of Management Reviews, vol. 12, no. 3, 2010, pp. 259–283.

[5] D. Van Nuffel, “Towards Designing Modular and Evolvable Business Processes,” Ph.D. dissertation, University of Antwerp, 2011.

[6] P. Huysmans, “On the Feasibility of Normalized Enterprises: Applying Normalized Systems Theory to the High-Level Design of Enterprises,” Ph.D. dissertation, University of Antwerp, 2011.

[7] E. Vanhoof, P. Huysmans, W. Aerts, and J. Verelst, Advances in Enterprise Engineering VIII: EEWC 2014. Springer International Publishing, 2014, ch. Evaluating, pp. 76–90.

[8] H. Mannaert, J. Verelst, and K. Ven, “Towards evolvable software architectures based on systems theoretic stability,” Software: Practice and Experience, vol. 42, no. 1, 2012, pp. 89–116.

[9] G. Oorts, H. Mannaert, P. De Bruyn, and I. Franquet, On the evolvable and traceable design of (under)graduate education programs. Springer International Publishing, 2016, vol. 252.

[10] C. Leichsenring, “Relsec style file,” 2013. [Online]. Available: https://github.com/mudd1/relsec/blob/master/relsec.sty [Accessed: 31-01-2017]

 ISBN: 978-1-61208-534-0