Conceptual Integrity of Software Systems: Architecture, Abstraction and Algebra

Iaakov Exman
Software Engineering Department
The Jerusalem College of Engineering – JCE - Azrieli
Jerusalem, Israel
iaakov@jce.ac.il

***Abstract*— Conceptual Integrity has been claimed to be the essence of high-quality software system design. On the other hand, it has been a rather elusive attribute of software systems, challenging various attempts of a clear-cut characterization. This paper evolves in this direction by two means: first, by analysis and clarification of open issues in architecture and abstraction terms; second, by pointing out to a mathematical formulation in algebraic terms. This paper also serves as a broad introduction to discussions on "Conceptual Integrity of Software Systems".**

Keywords: Conceptual Integrity; software system design; architecture; abstraction; algebra; Linear Software Models; Modularity Matrix; Conceptual lattice; plausibility criteria; design constraints.

I. INTRODUCTION

Frederick Brooks [2], [3], based upon his extensive experience with system development, in particular the first families of OS/360 operating systems, proposed that Conceptual Integrity is essential for high-quality software system design. It takes some time to assimilate this idea, but even after reading about it once and again and having second and third thoughts, Conceptual Integrity remains attractive, but a quite elusive notion.

The first task of this paper is to introduce the notion, its attractiveness and why it is still elusive. Then, we argue in favor of mathematical formalization, like in any respectable science. The overall purpose of this paper is to try to open and discuss deep issues on "Conceptual Integrity of Software Systems" and to point out to a formal solution to these issues.

A. The Notion of Conceptual Integrity

Software system design and development, as it is common practice nowadays, gives a superficial impression of unrestricted flexibility, where everything is allowed and nothing is forbidden. Often practitioners think that software design and development is an art, rather than a science. People exposing this opinion may express distaste for suggestions of mathematical description of the processes involved.

The feeling of unrestricted flexibility and its artistic connotation is acquired from the first experiences with computer programs that one writes. Except for an apparent arbitrariness of the programming language syntax, one has full confidence in the choice of the preferred programming construct – this is the unrestricted flexibility – and the program is supposed to run for sure. Then, happens the inevitable bug, demanding a subtle correction – this is the artistic connotation.

A widespread rational response to the *artistic flexibility* is to introduce methodologies. If one is methodical, then the price of software design and development should decrease. This is the first encounter with elusiveness (cf. the word "Mythical" in the title of the earlier book by Brooks). Any experienced software developer knows that methodologies are not the panacea that they promise to be. They fail, sometimes miserably.

Then, Conceptual Integrity, by Brooks, enters the stage. This is a deeper response to the software system development problem. We explain the notion in literally *architecture* terms, exactly as in Brooks' books, while presenting open issues in section III.

B. Paper Organization

The remaining of the paper is organized as follows. Section II refers to related work. Section III introduces open issues of Conceptual Integrity in a broad context. Section IV focuses on architecture principles. Section V turns to abstraction principles. Section VI summarizes algebraic principles of software design. Section VII concludes with a discussion.

II. RELATED WORK

A. Conceptual Integrity Outside Software Systems

This paper refers to Architecture of buildings in general, following Brooks' metaphors used in his books to illustrate notions of Conceptual Integrity. In this context, we refer here to a few architecture-related modeling techniques and ontologies.

DeLuca and co-authors [5] describe a generic formalism for semantic modeling of architectural elements, which compose buildings of historic interest in classical architecture. Doerr [8] reviews ontologies for cultural heritage; he emphasizes physical objects in archeology, including architecture. Quattrini et al. [22] describe modern computer-based techniques for semantically-aware 3D modeling of architecture.

DOI: 10.18293/SEKE2017-206

B. *Conceptual Integrity Within Software Systems*

Conceptual Integrity is closely related to efforts regarding various development phases of software systems. These efforts are among others: requirements engineering, conceptual modeling, integrity verification, software system design, and software architecture planning. The relevant literature is very extensive, and we provide just a few representative pointers.

Jackson and co-authors [6],[17],[18] analyzed widely used software systems, such as Git, in terms of Conceptual Integrity, suggesting design improvements.

Insfran et al. [16] refer to conceptual modeling based on requirements engineering. For these authors conceptual modeling is an UML object-oriented approach rather than based on conceptual integrity. Nevertheless, there are similarities between these approaches, as discussed later on.

Cabot and Teniente [4] refer to integrity checking of UML/OCL conceptual schemas. Integrity here means software state conditions that must be satisfied. Sometimes it specifically refers to avoidance of critical system malfunction. Conceptual schemas are basic relations between concepts within the general knowledge required by any information system (see also e.g. the book by Olive [21]).

Kazman and Carriere [19] reconstruct a software system architecture using *conceptual integrity* as a guideline. Their goal is to achieve a restricted number of components connected in regular ways, with internally consistent functionality.

C. *Mathematical Conceptual Integrity of Software*

In this work we shall mainly refer to the Modularity Matrix [9],[10],[13] which is based upon linear algebra. Other matrices have been used for modular design. For instance, the Design Structure Matrix (DSM) is an integral part of the 'Design Rules' by Baldwin and Clark [1]. It has been applied for various kinds of systems, including software systems. DSM design quality is estimated by an external economic theory superimposed on the DSM matrix.

Conceptual lattices, analyzed within Formal Concept Analysis (FCA) were introduced in Wille [24]. A generic review of its mathematical foundations is given by Ganter and Wille [14]. Conceptual Lattices have been shown to be equivalent to Modularity Matrices (e.g. Exman and Speicher [12]), linking the algebraic characteristics to conceptual ones.

III. OPEN ISSUES: ARCHITECTURE, ABSTRACTION, ALGEBRA

We now consider the open issues of Conceptual Integrity in a wide context. We start by referring to it in architecture terms. Then we consider abstraction and algebraic formulations.

A. *Open Issues: Architecture*

In Brooks' book "The Mythical Man-Month" [2] in front of the fourth chapter opening (page 41) there is a photo of the interior of the Reims cathedral, planned by Jean D'Orbais. An annotated sketch of the cathedral is seen in Fig. 1 in this paper. It has an imposing huge height relative to humans, as usual for medieval gothic cathedrals, implying the difficulty to actually build it.

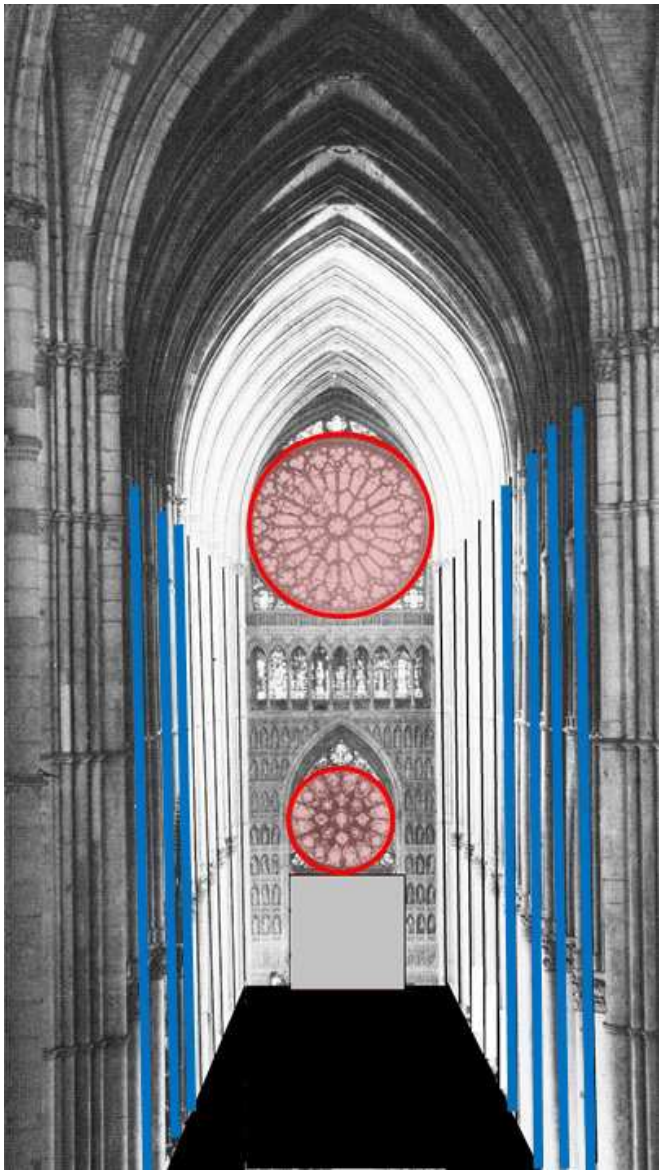

Figure 1. Reims Cathedral Interior: sketchy annotations – one can see the perfect symmetry of its elements: two rows of very high parallel columns (annotated by vertical blue color line segments), two stained glass rose windows (annotated by a pink background within a red circle), and above the columns the parallel gothic arches.

Nonetheless, despite the incredible weight of its stones, it displays elegance, coherence, and symmetry of its component forms: e.g. long repetitive rows of identical very high columns and two symmetrical stained glass rose windows above the altar region.

Another example is the renaissance cathedral of Firenze, whose symmetric dome with repetitive elements was designed by Brunelleschi. It is seen in Fig. 6-4 (page 75) in "The Design of Design" [3] book by Brooks.

Thus, Conceptual Integrity in classical, medieval and renaissance *architecture* means that the overall structure is immediately recognized by its repetitive and symmetric components' consistency, and is very attractive by its esthetics.

Figure 2. Bilbao Museum by Frank Gehry: sketchy annotations – one perceives the intentional total absence of symmetry of its elements. It displays one very high column (annotated by two vertical blue linear segment bounds) in its entrance; compare its height with that of human visitors, which are pointed out by the (dark blue) arrow. The overall asymmetric structure should resemble a ship contour (annotated by red linear segments, of various sizes and non-parallel directions). The outer metal structures reflect light by fish-like scales.

The Guggenheim Museum in Bilbao, designed by the architect Frank Gehry, has been metaphorically referred to as a modern cathedral, due to its enormous size, in particular the huge height of its atrium [15]. Fig. 2 is an annotated sketch of it. Moreover, it seems to be very consistent, by the similarity of materials and forms, to resemble a ship – since Bilbao is a port. In fact, it was designed with the support of the Digital Project (see e.g. [7]), which itself is based upon CATIA a sophisticated software system used to plan modern aircraft.

On the other hand, nothing is repetitive or symmetric in the Bilbao Museum. There are no two identical components in the Museum. We thought that we had the keys to the notion of Integrity, but, modern *architecture* breaks down the older keys. This is our second encounter with elusiveness. We are left with the open issue:

Open Issue #1 – Conceptual Integrity in Architecture
Despite symmetries being fundamental to physics – i.e. they correspond to conservation principles – and being important in classical architecture, symmetries are not ubiquitous in modern architecture.
What is the generalization of symmetry that is common to Conceptual Integrity in all kinds of architecture?

Paradoxically one still could say that the Bilbao Museum displays an internal Conceptual Integrity – the ship outline – and even to the apparently dissimilar buildings of the city of Bilbao. Moreover, the outer titanium metal sub-structures reflect light like fish scales! But how is ship and fish related concepts? This is discussed in the next sub-section on Abstraction.

B. Open Issues: Abstraction

We shall refer to abstraction in two senses: first, geometric or image abstraction; second, conceptual abstraction.

Regarding geometric abstraction, we mean that any considerations of Integrity are not taken with respect to the actual buildings referred to in the previous architecture sub-section. Neither real stone blocks, nor titanium metal surfaces are necessary to perceive symmetry or the "ship" shape.

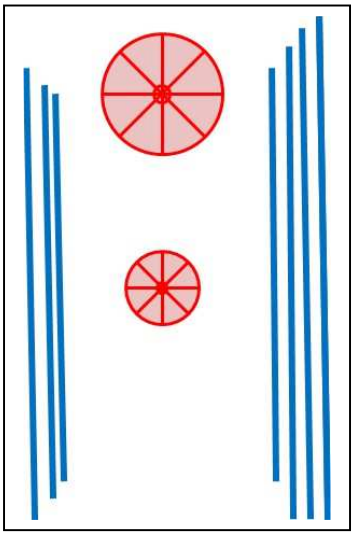

Figure 3. Reims Cathedral Geometric Abstraction: only sketchy annotations – this figure abstracts Fig. 1 to just the schematic representation of the rows of parallel columns (in blue) and the two stained glass rose windows (in red).

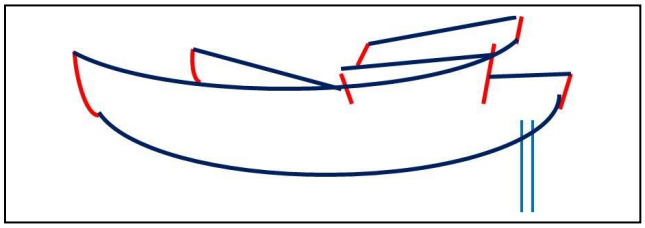

Figure 4. Bilbao Museum Geometric Abstraction: only sketchy annotations – this figure abstracts Fig. 2 to just the schematic representation of the exactly copied contour segments (in red) and the column bounds two vertical (light blue) segments. The contour segments were in addition linked by round (dark blue) lines, to facilitate our perception of the overall "ship" form.

One performs quite complex abstraction operations in the human brain, which are simulated by computer image processing – noise elimination, segmentation, and object recognition – to extract just the lines needed to infer either symmetry as in Fig. 3 or a "ship" shape as in Fig. 4.

Conceptual abstraction is a further step beyond geometric abstraction. In the cathedral case the relevant concepts are "*columns*" and "*stained glass rose window*", together with the even more abstract concept of "*symmetry*". The next step in conceptual abstraction is the usage of architecture domain ontologies (see e.g. [8]). In the Bilbao Museum case, relevant concepts could be "*port*" of Bilbao, "*ship*" and "*fish*". These could be found in Maritime domain ontologies (see e.g. [23]). But, the conceptual jump from these specific concepts to a Museum of modern art architecture is far from obvious. Then:

Open Issue #2 – Abstraction Conceptual Integrity
Abstraction Conceptual Integrity seems to be intrinsically associative irrespective of the relevant domain.
How to formally capture the associative character of abstraction Conceptual Integrity?

C. Open Issues: Algebra

Let us temporarily take for granted that Algebra is the mathematical basis of our formal approach to Conceptual Integrity – an approach extensively justified elsewhere (see e.g. [10], [11]), with respective arguments presented later on in this paper. In order to apply algebra to concepts we do not try to get an answer to Open Issue #2. Instead, we assume that the associative process of choosing sets of abstraction concepts – as discussed in the previous sub-section – is done by human software engineers, without any current attempt to formalize this process.

The algebraic phase of Conceptual Integrity starts with the chosen sets of concepts that humans find necessary to build a desired software system. We have called the specific ontology relevant to a desired software system as "*application ontology*". One can easily see that application ontologies have a striking similarity to UML class diagrams. Ontology concepts correspond to classes, and relationships among classes are of three kinds: inheritance (a sub-class is a type of class), composition (wholes are made of parts) and association (say usage of the functionality of another class). Thus, we deal with classes and their functionalities, and their respective generalizations.

What is lacking in the above picture? The answer is: we lack criteria to verify that the choice of concepts is reasonable. This is the role of algebra. Thus:

Open Issue #3 – Algebraic Conceptual Integrity
Assume that humans choose the needed concepts relevant to a desired software system, by a black-box algorithm.
What are the criteria to verify that the concepts choice indeed comply with an idea of Conceptual Integrity for the desired software system?

IV. Architecture Principles

The answer to the Open Issue #3 starts with the understanding that in any domain there are constraints limiting solutions. We start as in the previous section with principles of architecture.

A. Structure: Buildings and Software

The basic constraints obeyed by any building, either in antiquity or in modern times are first and foremost the laws of static – the branch of classical mechanics (itself a field of physics) dealing with structures which have no motion. One should have columns and beams to support structures of a building, a bridge over a bay or tunnels below a river. These columns and beams have a material composition, say concrete or steel, and suitable sizes and forms. Modern constructions have additional constraints, such as heating, acoustics, distribution of electricity and water tubing.

Software structures, just as buildings, are hierarchical with classes and patterns – as sets of classes – being the relevant structural units. The relevant constraints are enforced by the above mentioned relationships already found in the application ontologies – viz. inheritance, composition and association.

B. Behavior: Flying and Running Systems

Physical systems displaying motion, such as airplanes, autonomous cars, robots, helicopters, boats and drones (unmanned aircraft systems) may fly, walk, navigate, etc. In addition to structural constraints, they have kinematic constraints limiting their motion in terms of speed and possible trajectories.

Software systems when running also have constraints on their speeds, communication and memory usage. These are certainly affected by their structural constraints. With the increasing usage of autonomous systems, with embedded software, the software itself is also affected by the constraints of the containing physical systems.

C. Modularity

Modularity is an important consideration for any system, as it facilitates development, building and understanding of systems. Modularity is achieved by simplistic repetitive reuse of the same units again and again – such as the stones in the columns of the Reims cathedral, and wheels in mobile vehicles.

Modularity maybe also achieved by more sophisticated means than strict repetition. The titanium metallic surfaces of the Bilbao Museum are each of them unique, but they are generated by a software system which reuses the same technology to obtain different shapes.

Modularity is the result of complying with constraints, partially relaxing them as far as possible while minimizing undesirable effects. This is true in particular for software systems.

V. Abstraction Principles

In this section we deal with abstraction principles. Abstraction here is understood as conceptual abstraction for system design. We further focus only on embedded software or purely software system design.

The abstraction principles express the necessary constraints which limit design to modular solutions. Concomitantly these provide the criteria to verify that design solutions are optimal under these constraints. In terms of the abstract concepts, this is the meaning of *Conceptual Integrity*.

Conceptual abstraction principles were first formulated by Brooks in his books [3]. Here we succinctly review these principles. Note that all the three principles are formulated as ***negative*** expressions, i.e. "**Do not…**", which are effectively constraining design and meaning.

A. Abstract Propriety

Brooks concisely formulates propriety as: "***Do not introduce what is immaterial***".

He explains this principle by an example of propriety in computers, viz. the representation of zero in twos-complement notation, which obeys the constraint of not attaching a sign to zero.

In contrast, signed-magnitude and ones-complement representations do attach a sign to zero, which is extraneous and inconsistent with the original meaning of the "zero" concept. The consequence of these representations is addition

of artificial rules needed to characterize the behavior of zero in arithmetic operations.

B. Abstract Orthogonality

Brooks concisely formulates orthogonality as: "***Do not link what is independent***".

He explains this principle by a basic example of orthogonality in computers, viz. the operation of a (software or embedded) alarm clock. Two functionalities of such a clock are lighting and alarm. Orthogonality means that these two functionalities obey the constraint of actually being independent.

In contrast, if the alarm would operate only when the clock is illuminated, it would violate orthogonality, since one is artificially linking two unrelated functionalities.

C. Abstract Generality

Brooks concisely formulates generality as: "***Do not restrict what is inherent***".

He explains this principle by a surprising example of generality of an operation in a processor, viz. the operation "*restart*". Its original purpose was to restart a process after an interruption. But the design generality enabled its use as returning from a subroutine. The obeyed constraint here is avoiding incidental restriction.

In contrast, if it were strictly allowed only for the original purpose, one would clutter the design by introducing another almost overlapping concept for the second kind of usage.

VI. Algebra: Principles of Software Design

The power of a mathematical formalism is to express the abstract constraints in a precise way, enabling calculations upon the representation of a given software system design – e.g. of eigenvectors [11],[20] to obtain software modules –, and the verification whether the design comply with the imposed constraints. Moreover, one can improve the software design, based upon the verification results.

The choice of linear algebra structures, such as the Modularity Matrix and its equivalents, is justified by the following reasons:

- ***Expressivity*** – algebraic structures are expressive enough to represent the wide variability of software structures;
- ***Constraints Nature*** – it is very natural to formulate Brooks' abstraction constraints in algebraic terms, as will be seen in the following sub-sections;
- ***Conceptual Meaning*** – since each Modularity Matrix of a software system is equivalent to its corresponding Conceptual Lattice, the Conceptual meanings are immediately taken into account.

A. Algebraic Propriety

The algebraic translation of Brooks' Propriety is based upon the following ideas [10]. Instead of sets of classes (the concepts), one works with vectors of classes. Structors are generalized classes to all levels of the hierarchical software system. Structors provide functionalities. Thus, one also has functional vectors.

In order to avoid immaterial additions, i.e. the design goal is to minimize the representation of a software system, the algebraic ***Propriety*** constraint is: ***linear independence*** of all the structor vectors and of all the functional vectors in the Modularity Matrix of the given software system. Any dependent vector is superfluous and discarded.

B. Algebraic Orthogonality

The algebraic translation of Brooks' Orthogonality is even more immediate. One literally uses exactly the same word orthogonality, but now with the exact algebraic meaning of "zero scalar product of a pair of vectors".

The algebraic ***Orthogonality*** constraint is: ***orthogonality*** of all structor vectors and all functional vectors inside a module to the respective vectors in all other modules in the Modularity Matrix of the given software system. Thus, different software modules actually deal with different concerns, literally having different conceptual meanings.

C. Algebraic Generality

The algebraic translation of Brooks' Generality is that since the definition of a structor or its functionalities are not strictly repeated in another location of the same software system, one still can re-use these structors and their functionalities elsewhere in the same system by means of composition or by means of aspect-oriented design.

The algebraic ***Generality*** constraint is: ***generality*** once again minimizes the number of appearances of structor vectors and functional vectors in various hierarchical levels of the Modularity Matrix of the given software system. On the other hand, in an aspect-oriented design fashion, one puts the general structors/functionals in a high enough hierarchical level, to be accessible from anywhere in the system.

VII. Discussion

We here summarize the important points of this paper. Brooks' used architecture of cathedrals to justify his Conceptual Integrity principles. Indeed the architecture of buildings provides fruitful metaphors in our context, as we have seen that one obtains valuable concepts from the building abstractions. But, even more importantly, they lead to open issues challenging simplistic notions of Conceptual Integrity.

A. Eliciting Concepts

Concept elicitation from system stakeholders, using system sketches, drafts or early models, is an activity related to requirements engineering. It demands high human capabilities and in this paper this activity was left to the human engineers. We currently give up any effort to mathematical formalize this activity. It may well be that we shall return to this basic issue in future work.

We deliberately assumed that one starts the Conceptual Integrity analysis from an initial list of concepts – translated

into structor and functional vectors. This initial list may change along the design and development processes.

B. *Criteria for Integrity*

Brooks' principles actually are constraints on the software system design solutions. Once this is understood, one can clearly express algebraic equivalent formulations of the same principles.

The propriety and orthogonality constraints received a very natural translation to linear algebra notions. The generality principle seems to deserve further investigation.

C. *Main Contribution*

The main contribution of this paper is a set of open issues to be discussed within a Conceptual Integrity forum, and the clear understanding of its principles, as mathematical constraints on the design solution for a software system.

REFERENCES

[1] C.Y. Baldwin and K.B. Clark, *Design Rules*, Vol. I. The Power of Modularity, MIT Press, Cambridge, MA, USA, 2000.

[2] F.P. Brooks, *The Mythical Man-Month – Essays in Software Engineering* – Anniversary Edition, Addison-Wesley, Boston, MA, USA, 1995.

[3] F.P. Brooks, *The Design of Design: Essays from a Computer Scientist*, Addison-Wesley, Boston, MA, USA, 2010.

[4] J. Cabot and E. Teniente, "Incremental Integrity Checking of UML/OCL Conceptual Schemas", J. Systems and Software, Vol. 82, pp. 1459-1478, Sept. 2009. DOI: http://doi.org/10.1016/j.jss.2009.03.009

[5] L. De Luca, P. Veron and M. Florenzano, "A generic formalism for the semantic modeling and representation of architectural elements", Visual Computer, Feb. 2007. DOI: http://dx.doi.org/10.1007/s00371-006-0092-5

[6] S.P. De Rosso and D. Jackson, "What's Wrong with Git? A Conceptual Design Analysis", in Proc. of Onward! Conference, pp. 37-51, ACM, 2013. DOI: http://dx.doi.org/10.1145/2509578.2509584.

[7] Digital Project, https://en.wikipedia.org/wiki/Digital_Project

[8] M. Doerr, "Ontologies for Cultural Heritage", in Staab and Studer (eds.) Handbook on Ontologies, Springer-Verlag, Berlin, 2009, DOI: http://dx/doi/org/10.1007/978-3-540-92673-3

[9] I. Exman, "Linear Software Models", Proc. 1st SEMAT Workshop on a General Theory of Software Engineering, KTH Royal Institute of Technology, Stockholm, Sweden, 2012. http://semat.org/wp-content/uploads/2012/10/GTSE_2012_Proceedings.pdf. video of this paper in the web site: http://www.youtube.com/watch?v=EJfzArH8-ls.

[10] I. Exman, "Linear Software Models: Standard Modularity Highlights Residual Coupling", Int. Journal of Software Engineering and Knowledge Engineering, Vol. 24, pp. 183-210, 2014. DOI: 10.1142/S0218194014500089.

[11] I. Exman, "Linear Software Models: Decoupled Modules from Modularity Matrix Eigenvectors", Int. Journal of Software Engineering and Knowledge Engineering, Vol. 25, pp. 1395-1426, 2015. DOI: http://dx.doi.org/10.1142/S0218194015500308

[12] I. Exman and D. Speicher, "Linear Software Models: Equivalence of the Modularity Matrix to its Modularity Lattice", in Proc. 10th ICSOFT'2015 Int. Conference on Software Technology, pp. 109-116, ScitePress, Portugal, 2015. DOI: 10.5220/0005557701090116

[13] I. Exman, "Linear Software Models: An Algebraic Theory of Software Composition", in Proc. 28th Int. Conf. on Software Engineering and Knowledge Engineering, Keynote Abstract, KSI Research, Redwood City, CA, USA, 2016.

[14] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*, Springer-Verlag, Berlin, Germany, 1998.

[15] Guggenheim Museum in Bilbao by Frank Gehry, https://en.wikipedia.org/wiki/Guggenheim_Museum_Bilbao

[16] E. Insfran, O. Pastor and R. Wieringa, "Requirements Engineering-Based Conceptual Modeling", Requirements Eng. Vol. 7, pp. 61-72, June 2002, DOI: http://dx/doi/org/10.1007/s007660200005

[17] D. Jackson, "Conceptual Design of Software: A Research Agenda", CSAIL Technical Report, MIT-CSAIL-TR-2013-020, 2013. URL: http://dspace.mit.edu/bitstream/handle/1721.1/79826/MIT-CSAIL-TR-2013-020.pdf?sequence=2

[18] D. Jackson, "Towards a Theory of Conceptual Design for Software", in Proc. Onward! 2015 ACM Int. Symposium on New Ideas, New Paradigms and Reflections on Programming and Software, pp. 282-296, 2015. DOI: 10.1145/2814228.2814248.

[19] R. Kazman and S.J. Carriere, "Playing Detective: Reconstructing Software Architecture from Available Evidence." Technical Report CMU/SEI-97-TR-010, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 1997.

[20] U. von Luxburg, "A Tutorial on Spectral Clustering", *Statistics and Computing*, 17 (4), pp. 395-416, 2007. DOI: 10.1007/s11222-007-9033-z

[21] A. Olive, *Conceptual Modeling of Information Systems*, Springer-Verlag, Berlin, 2007.

[22] R. Quattrini, E.S. Malinverni, P. Clini, R. Nespeca and E. Orlietti, "From TLS to HBIM. High Quality Semantically-aware 3D Modeling of Complex Architecture", in 3D Virtual Reconstruction and Visualization of Complex Architectures, pp. 367-374, Avila, Spain, Feb. 2015. DOI: http://dx/doi/org/10.5194/isprsarchives-XL-5-W4-367-2015

[23] G. Santipantakis, K.I.Kotis and G.A. Vouros, "Ontology-Based Data Integration for Event Recognition in the Maritime Domain", WIMS '15, July 2015, Larnaca, Cyprus, DOI: http://dx.doi.org/10.1145/2797115.2797133

[24] R. Wille, "Restructuring lattice theory: an approach based on hierarchies of concepts" In: I. Rival (ed.): *Ordered Sets*, pp. 445–470, Reidel, Dordrecht-Boston, 1982.