A Software System is Greater than its Modules' Sum: Providers & Consumers' Modularity Matrix (TSE)

Iaakov Exman and Harel Wallach
Software Engineering Department
The Jerusalem College of Engineering – JCE - Azrieli
Jerusalem, Israel
iaakov@jce.ac.il, harel.wallach@gmail.com

***Abstract*— Modularity Matrices and their Laplacians enable finding software system modules by a rigorous algebraic procedure. However, Modularity Matrices have up to now focused mainly on structors as providers of functionals. This paper takes a broader view at the software system as a whole. The Software System Modularity Matrix, besides displaying *provider* relationships, also describes which structors *consume* functionals provided by other structors. This broader view improves software system design in two ways. First, consumer relationships set realistic expectations for consumer numbers and roles. Second, the Software System Modularity Matrix generates standard design criteria for interacting providers and consumers. This standard System matrix obeys linear independence of its constituent vectors, and block-diagonality of its recognizable modules. The novelty consists of modules being composed into a whole working Software System by means of a limited number of consumers playing the role of module *connectors*. Modules and their connectors are formally obtained by the same spectral method applied to the respective Laplacian, which obtained provider matrices. This is illustrated by case studies.**

Keywords: Linear Software Models; Spectral Software Design; Modularity Matrix; Laplacian Matrix; Providers; Consumers; Modules; Connectors; Linear Independence; Block-Diagonality.

I. INTRODUCTION

Linear Software Models represent each abstraction level of a given Software System by means of a Modularity Matrix [8], [10] or its corresponding Laplacian Matrix [13], [14]. These models enable rigorous software design of any given system:

- ***Standard Matrix*** – is defined as an exact criterion to compare different proposed designs;
- ***Matrices highlight the need of redesign*** – pinpointing eventual coupling problem locations.

A Modularity Matrix is made of column *structors* – a generalization of classes in object-oriented programming languages – and row *functionals* – a generalization of class methods. A matrix element is 1-valued if its structor provides the respective functional. Otherwise, the element is zero-valued.

DOI: 10.18293/SEKE2019-003

The meaning of a structor as a *provider* of a functional is e.g. a class containing the declaration/definition of a method, usable by other classes. Another structor using such a functional is called a *consumer* (e.g. a class calls a method of another class).

This paper's goal is to propose and deal with the broader sense of Modularity Matrix, and its corresponding Laplacian, displaying not only providers, but also matrix elements standing for consumers. This extended Modularity Matrix is a more complete representation of Software from the system perspective and is of practical interest for software design.

This Introduction concisely reviews concepts of Modularity and Laplacian matrices.

A. Software Modularity

A central problem to be solved by software engineering is the hierarchical composition of a software system from sub-systems, down to software architecture units, typically classes, considered indivisible by the designer.

Solving the software system composition problem involves software modularity. Applying Linear Software Models, one designs one or more Modularity Matrices, obtaining modules by spectral methods. One then compares their quality with a square and block-diagonal standard matrix, resolving eventual coupling problems, highlighted by the matrices.

A simple example of a schematic Modularity Matrix with providers only is shown in Fig. 1. It has five structors and five functionals. It displays three modules, the blocks along the diagonal. It is a standard Modularity Matrix as it does not have any outlier, a 1-valued matrix element outside the modules.

B. Modularity Matrices and their Laplacians

A Laplacian Matrix is easily generated from a Modularity Matrix in two steps:

- ***Extract a bipartite graph*** – with a structors' vertex set and a functionals' vertex set having edges corresponding to 1-valued matrix elements of the Modularity Matrix;
- ***Generate the Laplacian Matrix*** – from the bipartite graph, according to equation (1):

$$L = D - A \qquad (1)$$

where ***L*** is the Laplacian matrix, ***D*** is the Degree matrix of the graph vertices and ***A*** is the Adjacency matrix of vertex pairs.

	S1	S2	S3	S4	S5
F1	1	1			
F2	0	1	All zeros		
F3			1		
F4	All zeros			1	1
F5				0	1

Figure 1. Schematic Providers Modularity Matrix – It has 5 structors (S1 to S5), 5 functionals (F1 to F5) and three modules, the (blue) blocks along the diagonal: upper-left and lower-right with 2*2 size and middle block of size 1*1. It is a standard matrix as it does not have outliers (1-valued elements outside modules). (All figures are in color online).

A bipartite graph, obtained from the Modularity Matrix in Fig. 1, is shown in Fig. 2.

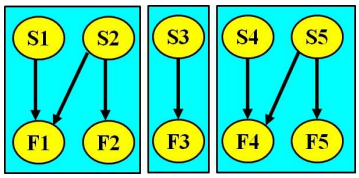

Figure 2. Bipartite Graph from Modularity Matrix in Fig. 1 – It has two vertex sets: the upper set of structors (S1 to S5), and the lower set of functionals (F1 to F5). A bipartite graph only has edges linking vertices in different sets. Arrows pointing down mean that structors provide functionals. The (blue) rectangles separate vertices belonging to given connected components (the modules).

A schematic Laplacian Matrix, generated from the bipartite graph in Fig. 2, is shown in Fig. 3.

	F1	F2	F3	F4	F5	S1	S2	S3	S4	S5
F1	2					-1	-1			
F2		1				0	-1			
F3			1					-1		
F4				2					-1	-1
F5					1				0	-1
S1	-1	0				1				
S2	-1	-1					2			
S3			-1					1		
S4				-1	0				1	
S5				-1	-1					2

Figure 3. Schematic Providers Laplacian Matrix – This Laplacian is generated from the bipartite graph in Fig. 2. By equation (1) its diagonal is ***D*** the Degree matrix (in green) showing the degrees of each vertex of the Bipartite graph. The upper-right quadrant (and its reflection in the lower-left quadrant) is the negative of ***A*** the graph Adjacency matrix, which is identical to the Modularity Matrix.

C. Paper Organization

The rest of the paper is organized as follows. Section II mentions related work. Section III introduces Consumer Matrices to be included in the whole System Matrices, which is done in Section IV. Section V illustrates the provider and consumer matrices by means of a server case study. Section VI concludes the paper with a discussion.

II. RELATED WORK

Here one finds a concise review of the extensive literature about Modularity approaches, by spectral and other methods. Also shortly reviewed are some references to consumers.

A. Linear Software Models

Linear Software Models have been developed by Exman and collaborators (e.g. [8], [9]) as a rigorous theory to solve the hierarchical software system composition problem from sub-systems. Linear Software Models are based on linear algebra operations and theorems. One assumes that all structors should be mutually linearly independent and also all functionals are linearly independent, an assumption motivated by minimization of the number of structors and functionals needed to build the system. Given this assumption, a linear algebra theorem demands that the Modularity Matrix be square. This is not a trivial result for software systems; it demands some effort to understand the theorem's rationale and implications.

Moreover, if sub-sets of structors/functionals are disjoint to other sub-sets, a second theorem states that these sub-sets can be rearranged into a block-diagonal matrix. These diagonal blocks are recognized as the modules in that software system level (for detailed proofs and examples see the work by Exman [10] and references therein).

A given software system modularization may display undesirable provider outliers coupling between modules. A procedure to compare different designs of the same software system, and to improve design is given by spectral methods as described in [11]. The Perron-Frobenius theorem (see e.g. Gantmacher [17]) is central for the Modularity Matrix theory.

Exman and Sakhnini [13], [14] have shown how to generate a Laplacian Matrix from the Modularity Matrix. The Laplacian matrix obtains the same modules as the Modularity Matrix, by similar spectral methods. The Fiedler theorem [1], [15] is central for the Laplacian theory. The so-called Fiedler eigenvector fits the lowest non-zero eigenvalue of the Laplacian Matrix. It allows locating outliers and splitting of too sparse software modules.

B. Alternative Approaches to Modularity

There exist a variety of techniques applying matrices for modularity analysis. For instance, Baldwin and Clark describe a Design Structure Matrix (DSM) in their "Design Rules" book [2]. DSM has been applied to many systems, including software engineering, see e.g. Cai and Sullivan [5].

Conceptual lattices, another algebraic structure relevant to software design, were introduced by Wille in 1982 [21] as part of Formal Concept Analysis (FCA). They have been used for

software system design e.g. by Siff and Reps [19] and by Exman and Speicher [12].

Alternative clustering techniques to obtain software modules are found e.g. in Shtern and Tzerpos [18].

C. Theoretical Approaches to Consumers

There have been modelling systems in the literature representing provider and consumer interactions. Yau and Caglayan [22] use Petri Nets to design distributed software systems. One of their examples is a producer-consumer system.

Clark et al. [6] describe experiences with PEPA (Performance Evaluation Process Algebra) modelling tools. In particular they refer to Producer-Consumer relations.

Browning [4] suggests that system modelers often build two DSM matrices, one for information supplier and another for the consumer, similar to our providers/consumers pairs of matrices.

III. The Nature of Consumer Matrices

Consumer matrices display Structors and Functionals consumed by the referred Structors. This section describes assumptions needed to generate Consumer matrices.

A. Consumer Matrices Shape and Size

Consumer matrices are not by themselves the aim of this paper. The goal of consumer matrices, jointly with provider matrices, is a more complete description of a software system, showing the interactions between the given sets of Structors and Functionals, from a system perspective.

Given this goal, we assume that Structors and Functionals of the consumer matrices are identical to those of the provider matrices. Thus, a consumer matrix fitting a standard providers' matrix is also square. We emphasize that the reason for consumer matrices being square is essentially different from the provider matrices. As stated in section II standard provider matrices are square by algebraic considerations. Consumer matrices are square just to enable unification of providers and consumers into a single system matrix, as described below.

We often refer to sub-systems, to allow for the possibility that consumer matrices leave outside, e.g. service functionality, obtained from external libraries. Alternatively, functionals provided by our sub-system may be consumed by other sub-systems, not included in our provider/consumer matrices.

Modularity and Laplacian matrices are tools to solve software design problems resulting from coupling interactions between different architectural units – structors and their functionals – in a given hierarchical level. Functionals provided and consumed by the same Structor do not appear in either of the provider/consumer matrices, as these are not interactions between different structors at that level.

B. Theoretical Properties of Consumer Matrices

Given the above assumptions on Consumer Matrices, we state easily verifiable theoretical properties.

Property 1 – ***Complementarity to Provider Matrices.***

Since Consumer Matrices have exactly the same Structors and Functionals as the Provider Matrices, and the matrices do not display Functionals provided and consumed by the same Structor, consumer Matrices are complementary to Provider matrices of the same sub-system. In other words, there is no overlap of non-zero matrix elements of the consumer matrix with non-zero matrix elements of the provider matrix.

Property 2 – ***Consumer Matrices may have empty (totally zero-valued) columns or rows, while Provider matrices cannot***.

This may happen since the Sub-system Under Design (SUD) may interact with other external sub-systems or libraries not represented in the matrices of the SUD. For example, if an SUD Structor in the Provider matrix provides a Functional consumed only by external sub-systems, the respective SUD Consumer matrix will have an ***empty row*** corresponding to the Functional consumed externally. Another example, if an SUD Structor appearing in the Provider matrix does not consume any Functional, the respective SUD Consumer matrix will have an ***empty column*** corresponding to the referred Structor. Provider matrices cannot have empty columns or empty rows, since they display only Structors actually providing Functionals.

Property 3 – ***Consumer Matrices per se generally neither display linear independence of their Structors/Functionals, nor have block-diagonal modules, in contrast with Provider matrices.***

This happens, since besides the empty columns/rows already mentioned in the previous property, it may be that a single given Structor consumes several Functionals originating identical rows. In other words, the rank of a Consumer Matrix is generally less than its size would permit. This does not occur with Provider matrices as already mentioned in section II.

C. An Example of Consumer Matrix

Now we reveal that the provider modularity matrix in Fig. 1 refers to the Command Design Pattern code in CSharp found in [7]. We use the same Structors and Functionals to show the respective Consumer matrix (in Fig. 4). The Command Design Pattern serves as an introductory running example, and as a first case study, in this and in the next section.

Structors → / Functionals ↓		ICommand	Concrete File Command	File Operation Invoker	IFile Operator	Concrete File Operator
		S1	S2	S3	S4	S5
Execute	F1			1		
Create Command	F2					
InvokeAll	F3					
File operations	F4		1			
Create Operator	F5					

Figure 4. Command Design Pattern, Consumers only Modularity Matrix – This consumers matrix fits the Providers Modularity Matrix in Fig. 1. Both matrices have the same Structors (S1,…,S5) and the same Functionals (F1,…,F5), and both comply with the Properties enumerated in sub-section B. The consumers matrix has just two 1-valued matrix elements (green hatched background), respectively (S3, F1) and (S2,F4) and is much sparser than the providers matrix.

IV. System Modularity Matrices: Providers and Consumers

This section finally deals with whole system modularity matrices including providers and consumers. The same algebraic techniques, previously used to identify provider-only matrix modules, are applied for the whole system matrices. This is done here for the Command Design Pattern Laplacian matrix.

A. System Weighted Modularity Matrices

We obtain the System Modularity Matrix by straightforward superposition of the provider matrix with the consumer matrix in a single overall matrix. This is possible as, by *Property 1* above, there is no overlap between non-zero matrix elements of these two matrices. But simple superposition would imply loss of "*direction*" information, i.e. whether a Functional is provided or consumed by a given Structor. To avoid this ambiguity one assigns a different weight to each direction: a functional *provided* by a structor is assigned a weight of "2" and a functional *consumed* by a structor is assigned a weight of "1".

In this context, it is important to state that Fiedler [15] has extended the algebraic connectivity properties of Laplacians to those for weighted edge graphs (see e.g. de Abreu [1]).

A System Weighted Modularity Matrix for the Command Design Pattern is show in Fig. 5, combining the provider matrix of Fig. 1 with the consumer matrix of Fig. 4.

Structors → / Functionals ↓		ICommand	Concrete File Command	File Operation Invoker	IFile Operator	Concrete File Operator
		S1	S2	S3	S4	S5
Execute	F1	2	2	1		
Create Command	F2	0	2			
InvokeAll	F3			2		
File operations	F4		1		2	2
Create Operator	F5				0	2

Figure 5. Command Design Pattern, System Weighted Modularity Matrix – This matrix is obtained by superposition of the Consumers Matrix in Fig. 4 with weights of "1", upon the Providers Matrix (blue modules) in Fig. 1 with weights of "2", to distinguish the consumers from the providers *direction*.

The weighted bipartite graph obtained from the System Modularity Matrix in Fig. 5, is shown in Fig. 6.

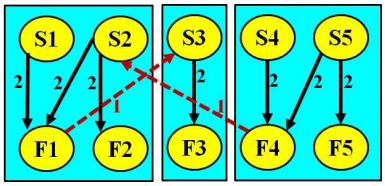

Figure 6. Command Pattern Weighted Bipartite Graph from Modularity Matrix in Fig. 5 – It has two vertex sets: the upper set of structors (S1 to S5), and the lower set of functionals (F1 to F5). Structors providing functionals are shown by (black) arrows pointing down with weight=2. Structors consuming functionals are shown by (red) arrows pointing up with weight=1. The (blue) rectangles denote providers' connected components (within the providers' modules). Consumer arrows are *connectors* between providers' modules.

B. Generation of the Weighted Laplacian

In order to identify the whole system modules including providers and consumers, we obtain from the Weighted Bipartite Graph (Fig. 6), the Weighted Laplacian Matrix in Fig. 7.

C. Connector Discovery from the Weighted Laplacian

As a last step towards the modules of the whole Command Pattern system, including both providers and consumers, we apply the same algebraic spectral method previously used (in [14]) for the providers-only Laplacian. It consists of:

a) *Calculate eigenvalues and eigenvectors* – of the Laplacian Matrix;
b) *Obtain Modules from eigenvectors* – whose eigenvalues are zero-valued;
c) *Discover Module connectors by splitting modules* – using the Fiedler eigenvector.

	F1	F2	F3	F4	F5	S1	S2	S3	S4	S5
F1	5					-2	-2	-1		
F2		2				0	-2			
F3			2					-2		
F4				5			-1		-2	-2
F5					2				0	-2
S1	-2	0				2				
S2	-2	-2		-1			5			
S3	-1		-2					3		
S4				-2	0				2	
S5				-2	-2					4

Figure 7. Command Pattern Weighted Laplacian Matrix from bipartite graph in Fig. 6 – It weights (by 2) Laplacian providers in Fig. 3, and adds the consumer elements, with negative weight=1 and a hatched (green) background. Diagonal degrees are changed to guarantee that all rows and columns sum to zero.

Laplacian Matrix Eigenvalues

Eigenvalues are shown in Fig. 8. The only zero-valued eigenvalue is the sixth one: it shows *Modules* in the Laplacian by the fitting eigenvector. The lowest eigenvalue closer to zero is the seventh one and is the *Fiedler* eigenvector, which allows further splitting of the Module.

#1	2	3	4	5	6	7	8	9	10
8.15	7.11	5.00	4.27	4.00	0.00	0.24	0.67	1.35	1.22
Type	→				Modules	Fiedler			

Figure 8. Command Pattern Weighted Laplacian Matrix eigenvalues – these are shown in the middle row of the figure.

Laplacian Matrix Eigenvectors

Eigenvectors' in Fig. 9 fit the Fig. 8 eigenvalues: the 2nd row from the top *Modules* eigenvector has all equal elements, implying one big whole system module; the 3rd row *Fiedler* eigenvector splits the whole system into two modules, by its element signs. Negative signs cluster (F1, F2, F3, S1, S2, S3) into one module and positive signs the vertices (F4, F5, S4, S5) into another module. In this first splitting iteration, the single structor module (F3, S3) seen in Fig. 5 and in Fig. 6, is left inside the 1st module. The 2nd Fiedler vector splitting iteration, in the bottom row of Fig. 9, finally separates the smaller module (F3, S3), obtaining all the three modules in this system.

Laplacian eigenvectors obtain only modules (either directly or by Fiedler vector splitting), as modules are mathematically "connected components" [20] of the graph.

Consumers, as external "connectors", are the remaining positive elements of the Modularity Matrix *by exclusion*, after the modules were directly characterized. For instance, in the 1st splitting iteration the consumer (F1,S3) is left inside the 1st module, while the consumer (F4,S2) is outside both modules. In the 2nd splitting iteration which obtains all modules, also obtains by exclusion both external connectors.

F1	F2	F3	F4	F5	S1	S2	S3	S4	S5
0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32
-0.20	-0.06	-0.47	+0.28	+0.43	-0.23	-0.05	-0.41	+0.32	+0.38
0.12	0.47	-0.64	0.00	0.00	0.17	0.34	-0.47	0.00	0.00

Figure 9. Command Pattern Weighted Laplacian eigenvectors – the top row has vertex indices; the 2nd row from top has all *Modules* vector elements equal 0.32; the 3rd row *Fiedler* different sign elements split the system into two modules 3*3 (negative, blue) and 2*2 (positive, yellow); the bottom 2nd *Fiedler* iteration splits the previous biggest module into 2*2 (positive, green) and 1*1 (negative).

Connectors Discovery by Splitting Modules

The final iteration from the Laplacian eigenvectors is displayed in the System Modularity Matrix, with the referred modules enclosed within dashed rectangles, as seen in Fig. 10.

Structors → / Functionals ↓		ICommand	Concrete File Command	File Operation Invoker	IFile Operator	Concrete File Operator
		S1	S2	S3	S4	S5
Execute	F1	2	2	1		
Create Command	F2	0	2			
InvokeAll	F3			2		
File operations	F4		1		2	2
Create Operator	F5				0	2

Figure 10. Command Design Pattern, System Weighted Modularity Matrix with Connectors – This matrix shows the provider modules as the result of the Laplacian eigenvectors, seen as delimited by the dashed (black) rectangles. These are the upper-left and lower-right modules of 2*2 size and the middle 1*1 module. The consumer (F1,S3) links the upper-left and middle modules. The upper-left module is also linked to the lower-right module by the consumer (F4,S2). Iterative splitting by the Fiedler vector obtains these two connectors.

The conclusion from this Command Pattern example is that consumers are *connectors* linking provider modules. This is seen in the bipartite graph (in Fig. 6), and corroborated by the partition by the Laplacian eigenvectors (in Fig. 10).

V. Case Study: An Asynchronous Server System

This case study is a larger system from the Boost library written in C++, viz. an Asynchronous Echo Server System [3]. The calculation steps were the same as in the Command Pattern example. Here are shown only the important steps' results.

A. Provider and Consumer Modularity Matrices

The Providers Modularity Matrix is strictly diagonal (Fig. 11). The Consumers' Modularity Matrix in Fig. 12 is very sparse and barely understood. Consumption is concentrated in "control" structors, viz. Main, Server and Session.

Structors → / Functionals ↓		Main	Io-context	Server	Acceptor	Endpoint	Socket	Session	Buffer
		S1	S2	S3	S4	S5	S6	S7	S8
main	F1	1							
run	F2		1						
server-constructor	F3			1					
async-accept	F4				1				
v4(), port	F5					1			
read, write, is-open, close	F6						1		
start, do-read, do-write	F7							1	
to-stream	F8								1

Figure 11. Asynchronous Server System – Providers-only strictly diagonal Modularity Matrix.

B. Spectral Approach to System Modules

In order to obtain System Modules, we apply the spectral method as done previously with the Command Design Pattern. One superposes the providers and consumers in a single weighted Modularity Matrix, and obtains the bipartite graph. Then one generates its Laplacian Matrix and calculate its eigenvalues and eigenvectors, as shown in Fig. 13 and Fig.14.

Structors → / Functionals ↓		Main	Io-context	Server	Acceptor	Endpoint	Socket	Session	Buffer
		S1	S2	S3	S4	S5	S6	S7	S8
main	F1								
run	F2	1							
server-constructor	F3	1							
async-accept	F4			1					
v4(), port	F5			1					
read, write, is-open, close	F6							1	
start, do-read, do-write	F7			1					
to-stream	F8							1	

Figure 12. Asynchronous Server System Consumers only Modularity Matrix – – This consumers matrix fits the Providers Modularity Matrix in Fig. 11. Both matrices have the same Structors (S1,…,S8) and the same Functionals (F1,…,F8), and both comply with the Properties enumerated in sub-section III B. The non-zero consumers' matrix elements are marked (in green hatched background).

#1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
7.10	6.09	5.73	4.00	4.24	4.38	1.30	0.81	0.00	0.11	0.23	4.56	2.00	0.43	4.56	0.43
Type	→							Module	Fiedler						

Figure 13. Asynchronous Server System Laplacian Eigenvalues – The eigenvalue #9 is the only one zero-valued, implying one *Module* in this system. Eigenvalue #10 is the *Fiedler* vector allowing splitting of this overall module.

The Modules and the Fiedler eigenvectors are shown in Fig. 14. The Fiedler eigenvalue #10 splits the whole system into two modules of 5*5 and 3*3 sizes. The next iteration Fiedler vector (the lowest row) further splits the 5*5 module into two smaller modules of sizes 2*2 and 3*3 sizes.

F1	F2	F3	F4	F5	F6	F7	F8	S1	S2	S3	S4	S5	S6	S7	S8
0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
0.28	0.38	0.04	0.06	0.06	-0.34	-0.24	-0.34	0.29	0.36	0.12	0.05	0.05	-0.32	-0.15	-0.32
-0.25	-0.48	0.17	0.33	0.33	0.00	0.00	0.00	-0.28	-0.43	0.03	0.29	0.29	0.00	0.00	0.00

Figure 14. Asynchronous Server System Laplacian Eigenvectors – From top row to bottom: the 1st row shows the vertices (on yellow); the 2nd row is the eigenvector fitting the single module eigenvalue #9; the 3rd row is the Fiedler #10 eigenvector splitting the 2nd row eigenvector into two modules according to the positive and negative signs; the lowest row is the next iteration Fiedler eigenvector, splitting the larger (green) module in the 3rd row eigenvector into two smaller modules of sizes 2*2 and 3*3 (light blue and orange background).

C. Connector Discovery in the System Matrix

Modules are discovered by looking at the eigenvectors of the Laplacian. These modules are the upper-left (F1,F2,S1,S2), the middle (F3,F4,F5,S3,S4,S5) and the lower-right (F6,F7,F8,S6,S7,S8) as shown in Fig. 15.

External consumers, the "connectors" between the three referred modules, viz. (S1, F3) and (S3, F7) are discovered by exclusion (i.e. outliers), as the remaining positive matrix elements of the System Modularity Matrix outside the three modules (Fig. 15). An interesting observation for this system is the existence of consumers as "internal connectors", inside the three modules, where each consumer reasonably links a pair of provider structors.

Structors → / Functionals ↓		Main	Io-context	Server	Acceptor	Endpoint	Socket	Session	Buffer
		S1	S2	S3	S4	S5	S6	S7	S8
main	F1	2							
run	F2	1	2						
server-constructor	F3	1		2					
async-accept	F4			1	2				
v4(), port	F5			1		2			
read, write, is-open, close	F6						2	1	
start, do-read, do-write	F7			1				2	
to-stream	F8							1	2

Figure 15. Asynchronous Server System Modules – The three modules enclose the original Provider modules and respective internal connectors. The external connectors are found in the elements **(S1,F3)** linking the upper-left and middle modules, and **(S3,F7)** linking the middle and lower-right modules.

VI. Discussion

A. Interpretation of System Matrix Results

A Consumer Matrix by itself is rather perplexing: it is very sparse and not easily interpreted. Consumer Matrices do not obey any apparent algebraic rules such as structors or functionals linear independence, or module block-diagonality, as for Provider Matrix. There are no obvious correctness criteria for Consumer Matrices by themselves.

When one superposes a Consumer Matrix upon its Provider Matrix the picture suddenly clarifies: **consumers are "*connectors*"** between pairs of provider modules. One could say that the Software System is greater than the sum of its modules, due to the interactions of the external consumer *connectors* with provider modules.

There are two slightly different situations with the case studies in this paper. The Command Design Pattern of the providers-only Modularity Matrix has the same number of three modules (Fig. 1) as the System Modularity Matrix with connectors (Fig. 10). The two connectors are external to the three modules and actually connect pairs of modules.

In the Asynchronous Server case study the providers-only Modularity Matrix (Fig. 11) is strictly diagonal and has 8 single-structor modules. In the System Modularity Matrix with connectors (Fig. 15) the providers-only modules were re-configured into three larger modules internally linked by connectors. In addition there are two external connectors. It is still true that connectors – both internal and external – link pairs of modules. The internal connectors link the original providers-only modules. The external connectors link the re-configured system modules (containing both providers and consumers, playing the internal connectors role).

We are led to the following conjecture as a summary of our currently empirical findings:

Conjecture: *Software System Modularity Connectors*

The *Minimal Number of System Module* ***Connectors*** is equal to the number of System Provider Modules minus one, i.e. each System module is connected to at least one other System module by a consumer Connector.

B. System Benefits for Software Design

First of all, it has been clear, before this work, that providers-only Modularity Matrix, the corresponding bipartite graph and its Laplacian Matrix, were incomplete descriptions of a software system. The addition of the consumers certainly improves the ability to judge the overall system design quality.

Once consumers are interpreted as "*connectors*" of provider modules, there are clear expectations on *consumers*' quantity and matrix element locations for their software system role.

Moreover, there are two System Matrix correctness criteria:

a- ***Algebraic*** – the providers and consumers joint modules, excluding the external connectors, obey linear independence and block-diagonality;

b- ***Semantic*** – system modules are semantically sound (e.g. the Fig. 15 lower-right module clusters *read*, *write*, and *stream* belong to the same category of messaging functionals).

C. Future Work

The paper's results, in particular the Software System Modularity Connectors conjecture, deserve formal proofs and extensive verification for a variety of software systems. These will be presented in an expanded version of this paper.

Although Fiedler (see e.g. [1]) extended the Laplacian spectral properties validity to weighted graphs, we need to investigate the specific weights' influence on modules calculations done with the Laplacian matrix.

D. Main Contribution

The main contribution of this paper is the introduction of Consumers in the software System Overall Modularity Matrix, and in the corresponding Laplacian Matrix, in the role of connectors between provider modules.

We have thereby shown that the same Linear Software Models that have been applied to providers-only matrices, is a generic algebraic theory of software composition, applicable to the overall system, including consumers.

ACKNOWLEDGMENT

The authors are grateful to the anonymous reviewers for incisive comments that helped to improve this version of the paper.

REFERENCES

[1] N.M.M. de Abreu, "Old and new results on algebraic connectivity of graphs", Linear Algebra and its Applications, 423, pp. 53-73, 2007. DOI: https://doi.org/10.1016/j.laa.2006.08.017.

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

[3] Boost libraries, asio c++11 examples, Christopher M. Kohlhoff. URL: https://www.boost.org/doc/libs/1_66_0/doc/html/boost_asio/example/cpp11/echo/async_tcp_echo_server.cpp

[4] T.Y. Browning, "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions", IEEE Trans. Eng. Management, Vol. 48, pp. 292-306, 2001.

[5] Y. Cai and K.J. Sullivan, "Modularity Analysis of Logical Design Models", in *Proc. 21st IEEE/ACM Int. Conf. Automated Software Eng. ASE'06*, pp. 91-102, Tokyo, Japan, 2006.

[6] G. Clark, S. Gilmore, J. Hillston and N. Thomas, "Experiences with the PEPA performance modelling tools", IEE Proceedings, Software, vol. 146, no. 1, pp. 11-19, 1999. DOI: https://doi.org/10.1049/ip-sen:19990149

[7] CsharpDesignPatterns, by Jason de Oliveira, 2017. URL: https://csharpdesignpatterns.codeplex.com/SourceControl/latest#DesignPatterns/DesignPatterns/DesignPatterns.csproj

[8] I. Exman, "Linear Software Models", Extended Abstract, in I. Jacobson, M. Goedicke and P. Johnson (eds.), *GTSE 2012, SEMAT Workshop on General Theory of Software Engineering*, pp. 23-24, KTH Royal Institute of Technology, Stockholm, Sweden, 2012. Video: http://www.youtube.com/watch?v=EJfzArH8-ls

[9] I. Exman, "Linear Software Models are Theoretical Standards of Modularity", in J. Cordeiro, S. Hammoudi, and M. van Sinderen (eds.): *ICSOFT 2012, Revised selected papers, CCIS*, Vol. 411, pp. 203–217, Springer-Verlag, Berlin, Germany, 2013. DOI: 10.1007/978-3-642-45404-2_14

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

[11] I. Exman, "Linear Software Models: Decoupled Modules from Modularity Eigenvectors", Int. Journal on Software Engineering and Knowledge Engineering, vol. 25, pp. 1395-1426, October 2015. DOI: 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 and R. Sakhnini, "Linear Software Models: Modularity Analysis by the Laplacian Matrix", in Proc. 11th ICSOFT'2016 Int. Conference on Software Technology, Volume 2, pp. 100-108, ScitePress, Portugal, 2016. DOI: 10.5220/0005985601000108

[14] I. Exman and R. Sakhnini, "Linear Software Models: Bipartite Isomorphism between Laplacian Eigenvectors and Modularity Matrix Eigenvectors", Int. Journal of Software Engineering and Knowledge Engineering, Vol. 28, No 7, pp. 897-935, 2018. DOI: http://dx.doi.org/10.1142/S0218194018400107

[15] M. Fiedler, "Algebraic Connectivity of Graphs", *Czech. Math. J.*, Vol. 23, (2) 298-305, 1973.

[16] E. Gamma, R. Helm, , R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, MA, 1995.

[17] F.R. Gantmacher, The Theory of Matrices, Volume Two, Chelsea Publishing Co., New York, NY, USA, 1959. Chapter XIII, page 53, Available in the Web (out of copyright): https://archive.org/details/theoryofmatrices00gant.

[18] M. Shtern and V. Tzerpos, "Clustering Methodologies for Software Engineering", in *Advances in Software Engineering*, vol. 2012, Article ID 792024, 2012. DOI: 10.1155/2012/792024

[19] M. Siff and T. Reps, "Identifying modules via concept analysis", IEEE Trans. Software Engineering, Vol. 25, (6), pp. 749-768, 1999. DOI: 10.1109/32.824377

[20] R. Todd and E.W. Weisstein, "Connected Component", Wolfram, MathWorld.

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

[22] S.S. Yau and M.U. Caglayan, Distributed software system design representation using modified Petri Nets, *IEEE Trans. Software Engineering*, Vol. SE-9, pp 733-745, 1983.