FEDERAL UNIVERSITY OF BAHIA

INSTITUTE OF MATHEMATICS

DEPARTAMENT OF COMPUTER SCIENCE

Antonio de Almeida Souza Neto

Antares DSM: Visualization and Optimization Depencies on Design Structures Matrices (English Short Version)

Salvador

2008

ABSTRACT

Visualizing and understanding dependencies in complex, large-scale systems are challenging tasks, that if not managed, may hinder evolution and system validation. The Design Structure Matrix (DSM) promotes compact and intuitive visualization of a complex system as a whole, and supports the use of existings algorithms to optimize dependencies among elements with different goals. This work exploits the use of DSMs and proposes the **Quick Triangular Matrix**, a greedy partition algorithm to identify undesirable cycles. The Antares DSM tool is introduced, to support the visualization of DSMs and optimization of system dependencies by using the proposed algorithm. A simple case study is presented to illustrate the contributions and limitations of the proposed algorithm and tool.

Keywords: Design Structure Matrix, DSM, graphs, Antares, Quick Triangular Matrix, partitioning, identification of cycles.

LIST OF FIGURES

CONTENTS

1 INTRODUCTION

1.1 HISTORIC

From the ’70s, the amount of computing systems developed grew significantly. With the growth of amount, there was also an increase in complexity and diversity of systems developed, resulting in demand for the implementation of a software engineering discipline. The software engineering is the application of a systematic approach, disciplined, quantifiable for development, operation and maintenance of software (IEEE..., 1990), offering methods, techniques and tools for its construction, evolution and reuse.

One of the basic strategies to circumvent the complexity of any system is *divide and conquer*. A software system large and complex can be divided into smaller parts connected that exchanges information for a specific purpose; each part can be decomposed until a solution or implementation is provided. In a system Object-oriented, these shares may be Classes, methods or even packages. These shares will depend on each other for, together, provide the desired functionality.

Modularity of software is defined as the degree by which a program is composed of discrete components such as a change in a component has minimal impact on other components (IEEE..., 1990). Ideally, the modules of a system should be relatively independent, in order to reduce impacts of changes, facilitate understanding, reuse, and support for parallel development. Modules should be weakly coupled and have high internal cohesion, so that the complexity of the parties and dependencies between them does not exceed the complexity of the whole.

In a system large and complex, with hundreds of classes and methods and relationship between them, the control and the visualization of dependencies between modules are not trivial tasks.

Because of such difficulties, emerged as an alternative *Design Structure Matrix*, or DSM, which is a matrix of dependencies that represents elements that relate. Recently, the DSMs have won space, because shows itself as a model stripped down, easy and quick viewing of

information (DSMWEB, 2008).

1.2 OBJECTIVE

The objective of this work is to do a study on the DSMS and ways to optimize them in relation to the disposal of unwanted cycles. In especial, it has the specifics objectives: (1) The development of an greedy algorithm of partitioning, whose main purpose is the identification of such cycles, (2) its incorporation in tool support for use of DSMS, and (3) comparison of the algorithm proposed with algorithms deterministic existents, pointing his qualities and defects.

1.3 REASON

Since they were created, the DSMs have been shown quite effectives and are used for various purposes. In planning and management of projects, mainly in engineering, you can see a considerable reduction in total time of the project, optimizing the relationship of dependency between the activities described in the matrix, besides getting an easy, quick and intuitive view of the planning (PIERONI; NAVEIRO, 2005) and (MANZIONE; B.MELHADO, 2007).

The use of DSMs in the context of Software Engineering is still small. However, the consolidated use of the paradigm of objects to the development of systems software, in various fields of application, and the constant concern to control the dependencies between modules, have stimulated the growing use of DSMs in this context: managing dependencies between modules.

This type of modeling seems to be one of the best ways to identify relationships of dependency between sub-parts of a system, such as: classes, packages, modules or subsystems. The relationships between them can be described in the matrix of dependencies, in order to control and scalable viewing on implementation of structures more complex.

Apart from its ease of viewing and understanding of the software, you can apply algorithms based on graphs to the modeling software, to suggest the designer changes in dependencies between the components, in order to improve desirable characteristics specific to the software in question.

The partitioning of a matrix, specifically aims to make the matrix as close as possible to lower triangular matrix, that is, with all the elements below the main diagonal. Thus, if there are elements above the diagonal, these elements indicate the existence of cycles, as part of these. The method of partitioning of DSMs will be explained with details in the Chapter 3.

The completed partitioning of a dependencies matrix, with the identification of all cycles, on a deterministic, it is a problem NP-complete. This means that at the moment, all proposed algorithms are exponential.

The modern software is becoming largest and with more parts that are related. With this in a very large system, the time of partition of their corresponding matrix of dependencies can become unviable. The greedy algorithm of partitioning proposed in this work aims to help improve the system, partitioning and identifying cycles, even if not completely, with a polynomial algorithm, significantly faster than those proposed so far for the general case.

1.4 RELATED WORK

DSM not is only a model of processes to project management and modeling, but a tool analysis and improvement of modeling systems or processes. DSMs can be used to identify cycles in a model or identify groups, aims create packages. Many algorithms and methods of analysis can be founded. Gebala (1991) proposed two algorithms: “partitioning” and “tearing”. The partitioning algorithm reorganizes the sequence of the items of the matrix to maximize the availability and the flow of information of the elements. After partitioning, it is easier to identify cycles in the model. The tearing algorithm reorganizes the matrix forming blocks of elements coupled, that may well become packages, for example. Kusiak (1993) proposed a method of analysis qualitative of design process to identify topological patterns in the process (MORI et al., 1999).

Smith and Eppinger (1994) refined the concept of DSM. Browning (2001) reviewed the implementation of DSM for decomposition and integration problems, detailed the types of DSM and noted the new directions of this model, especially as one of the best contributions found in the literature. Banerjee, Carrillo and Paul (2007) did an analysis of computational complexity involved in the algorithms used in the manipulation of the matrix (MORAES, 2007).

In 1999, Mori, Ishii, Kondo and Ohtomi (MORI et al., 1999) proposed improvements in methods of optimizing DSMS, based in partitioning and tearing algorithms.

In addition to the various algorithms published, there are tools on the market as the `Lattix`[1] and `Intellij Idea`[2] which apply some of the existing algorithms to improve the modeling software.

DSMs are generic enough to be used in projects in various areas. Besides the area of

[1] www.lattix.com
[2] www.jetbrains.com/idea

Software Engineering, there are studies as examples in the area of building ships (PIERONI; NAVEIRO, 2005), construction (MANZIONE; B.MELHADO, 2007) and many others.

The use of DSMs involves operations such as partitioning and tearing seeking to obtain specific improvements for modeling projects with implementation of DSM.

2 CONCEPTS

2.1 GRAPHS AND DSMS

A graph is a mathematical model that represents elements with their relationships. A graph admits a visual representation where the elements are called vertices and may be represented by points, letters or components of free choice. Their relationship can be represented by lines or arcs and can be called by edges. Depending on the application, the edges may or may not have direction. If have direction, its edges can be represented by arrows, which attach a vertex to another (DIESTEL, 2000).

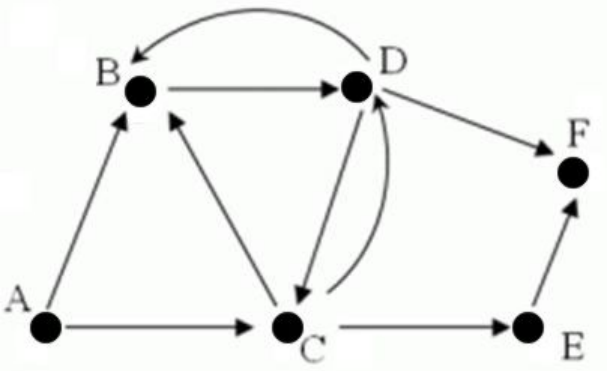

Figure 1: Example of graph

Matrices can represent graphs. Suppose N vertices, described in a square matrix M, NxN, where each vertex is represented by one row i, and also by a one column j. It is said that a vertex i depends on a vertex j, if exists, in the position of the matrix represented by ij (row i, column j), some value marked, numeric (indicating the degree of relationship) or a "X". This application can be used for describe all types of components that are connected, in any kind of modeling or planning. This type of matrix is called “Design Structure Matrix” (DSM).

2.2 PROBLEMS NP-COMPLETE

In theory of computational complexity, there are a set of problems called NP (Not Polynomial) and there are also a set of problems called P (polynomial) so that $P \subseteq NP$. Within all the NP, there is a subset of problems called NP-complete. You could say that the NP-complete problems are the most difficult problems of NP and most likely not part of the class of complexity P.

The NP-complete problems are classified as intractable as the algorithms suggested so far for their solution or have complex exponential or factorial, that very large entries to make its implementation impracticable. One of the strategies to solve some problems in such acceptable time, is the technique of greedy algorithms that seeks to resolve problems based on local choices, in the hope that this choice leads to the optimal solution overall. Such algorithms are usually simple and easy to implement, however, most of these algorithms do not always carry the issue to an optimal solution (CORMEN; LEISERSON; RIVEST, 1990).

Most of the algorithms used to solve problems in matrices of addictions have exponential complexity and lead to an optimal solution. For entries of small size, its results are acceptable, however, a matrix with many components, the analyst may have to wait hours just to see the outcome of the algorithm applied to the matrix.

In an attempt to improve the performance and time associated with any tool based on DSMs, a greedy algorithm polynomial of partitioning, focus of this work, was created and that will be addressed in the following chapter.

3 QUICK TRIANGULAR MATRIX

3.1 THE PROBLEM

The partitioning of a DSM is the process of manipulation, ie reordering of its rows and columns, so that the resulting matrix contains no brand of feedback, or is there any evidence to indicate cycle. The goal is to transform the DSM in a triangular matrix below (all above the diagonal are null). For complex systems, it is highly unlikely that the simple manipulation of rows and columns result in a lower triangular matrix (MORAES, 2007).

To realize the above the diagonal of a partitioned matrix, the analyst can see marks of feedback, or cycles, which typically are not desirable and correct them in the project, if possible.

Most of the algorithms used today have Partitioning complexity exponentially, making their implementation often impracticable for large matrices of dependencies. To circumvent this problem, the algorithm **Quick Triangular Matrix** was created, with the major objective minimize considerably the execution time of partitioning, without loss in its analysis.

Note that, as a greedy algorithm, **Quick Triangular Matrix** not always arrive at an optimum solution (In a matrix partitioned in that case may be elements that do not belong to cycles (no mark of "feedback") above the main diagonal); However, the likelihood of elements above the diagonal come from cycles will increase considerably, and therefore deserve a review, as shown by the results appear in Chapter 5.

3.2 THE ALGORITHM

The algorithm **Quick Triangular Matrix** is divided in two parts. The first part is a general sort of summation of the degree of dependency of rows and columns of the matrix. In the second part is made a refinement of the order of rows and columns, comparing elements in pairs, and replacing them if necessary. The description of the algorithm at high level is presented below, and his steps are illustrated through the matrix derived from the initial matrix shown in Figure 2.

Note that in the matrix example, to facilitate the understanding, all the dependencies have value 1 and the existing cycles are simple, although the algorithm supports greater degree of dependency, and also achieve “isolate” above the main diagonal elements that belong to cycles with more than 2 elements.

	1	2	3	4	5	6	7
1		1		1	1		1
2					1		
3	1	1			1		1
4	1	1	1		1	1	
5	1	1	1				
6		1	1		1		1
7	1				1		

Figure 2: Matrix example before to apply the Quick Triangular Matrix.

PART I

1. Mount 2 lists with the elements, sorted (Figure 3) using Quick Sort (HOARE, 1962) by criteria:

    1.1. The total value of degrees of dependence in the sense of suppliers (columns of matrix).

    1.2. The total value of degrees of dependence in the sense of dependents (rows of the matrix).

	1	2	3	4	5	6	7	
1		1		1	1		1	4
2					1			1
3	1	1			1		1	4
4	1	1	1		1	1		5
5	1	1	1					3
6		1	1		1		1	4
7	1				1			2
	4	5	3	1	6	1	3	

Lista de fornecedores: 4, 6, 3, 7, 1, 2, 5
Lista de dependentes: 2, 7, 5, 1, 3, 6, 4

Figure 3: Matrix example with the lists sorted of dependentes and suppliers

2. Create a new sort for the matrix removing of stack alternately, and in order, an element in the list 1.1 and placing at the end of the new sort of the matrix, and an element in the list 1.2 and placing at the beginning of the new sort of the matrix, until there are no more elements. Each element removed of a stack is also removed of the other list.

3. Mount this matrix with this sort in the rows and columns.

The Figure 4 displays the matrix obtained to the end of steps 1 through 3 of PART I. The matrix at that time already has most of its branches below the main diagonal.

	2	7	5	1	3	6	4
2			1				
7			1	1			
5	1			1	1		
1	1	1	1				1
3	1	1	1	1			
6	1	1	1		1		
4	1		1	1	1	1	

Figure 4: Matrix example after the end of Part I

PART II

4. Walking along the columns of backwards (or N to 1) and rows from bottom to top (or N to 1) seek dependencies above the diagonal (cell with a value greater than 0 in the matrix).

5. If you find dependence on (i, j), do square formed by all the cells between (i, j) and (j, i).

    5.1. Sum edges of the triangle above (elements (i,(i+1 until j)) and ((j+1 until j-1),j)) and the lower triangle (elements ((i+1 until j),i) and (j,(i+1 until j-1))), where i represents the dependent element found in the relationship (row) and j the provider of information element of the relationship (column).

    5.2. If the sum of the upper edge of the triangle is greater than the sum of the lower edge of the triangle, exchange (i, j) in rows and columns.

The Figure 5 shows an example where not will be made the exchange between the elements 1 and 4 and will be made the exchange between the elements 7 and 1.

After the application of the algorithm the DSM, that had 10 elements above the diagonal, now has only 5 (Figure 6).

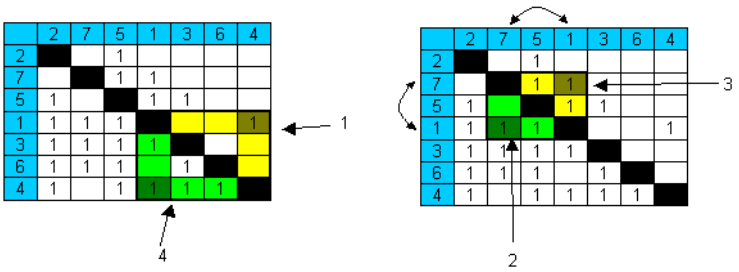

Figure 5: An example without exchange and other with exchange of elements

	2	1	5	7	3	6	4
2			1				
1	1		1	1			1
5	1	1			1		
7		1	1				
3	1	1	1	1			
6	1		1	1	1		
4	1	1	1		1	1	

Figure 6: Matrix example after aplication of the algorithm

You can observe that all elements left over above the diagonal of the matrix, in this case are simple cycles of $X \rightarrow Y \rightarrow X$. These elements must be reviewed by the analyst, to whether they are really needed in your project (Figure 7). In the case of models that contain dependencies with degree more than 1, the element of each cycle most likely to stay above the main diagonal is the element with a lower value, and therefore it is more easily be removed by the analyst. Note also that the algorithm maintains the original dependencies, it just changes the order of its elements, to facilitate the identification of cycles. The need for the cycle in each project is interpretative and therefore, the analyst and not the algorithm must make the decision about their removal.

	2	1	5	7	3	6	4
2			1				
1	1		1	1			1
5	1	1			1		
7		1	1				
3	1	1	1	1			
6	1		1	1	1		
4	1	1	1		1	1	

Figure 7: Matrix example displaying elements in cycles

4 ANTARES DSM

In the current scenario we can find a few tools to manipulate DSMs. Two examples are known: Lattix[1] and Intellij IDEA[2]. These tools are proprietary and closed source, and use of complex algorithms exponentially, which can raise a lot of time to implement inputs too great, making their use impracticable.

To meet this need, the tool **Antares DSM** has been created, a free tool under LGPL license (GNU Lesser General Public License[3]). This means that it can be downloaded at no cost and can have its code changed by other developers. Moreover, **Antares DSM** used as the partitioning algorithm, **Quick Triangular Matrix**, explained in Chapter 3, for better performance, as shown in Chapter 5.

The tool **Antares DSM** is disponible in *http://sourceforge.net/projects/antaresdsm/*.

4.1 DESCRIPTION OF TOOL

This section describes the basic features of Antares DSM: partitioning and identification of cycles. Examples of use of Antares DSM are shown in Chapter 6.

4.1.1 PARTITIONING

The partitioning of the matrix of dependencies in AntaresDSM uses the algorithm **Quick Triangular Matrix** through the menu item *DSM* → *Particionar*. **Antares DSM** open a new window (maintaining the previous) with the matrix partitioned (Figure 8).

Note that if the matrix contains no cycles, it should go in the triangular lower. If there are cycles, all the elements that are above the diagonal, should belong to any cycle. As the algorithm is "greedy" in some cases it can display above the diagonal elements that do not

[1] www.lattix.com
[2] www.jetbrains.com/idea
[3] More information about the license can be found at http://www.gnu.org/copyleft/lgpl.html

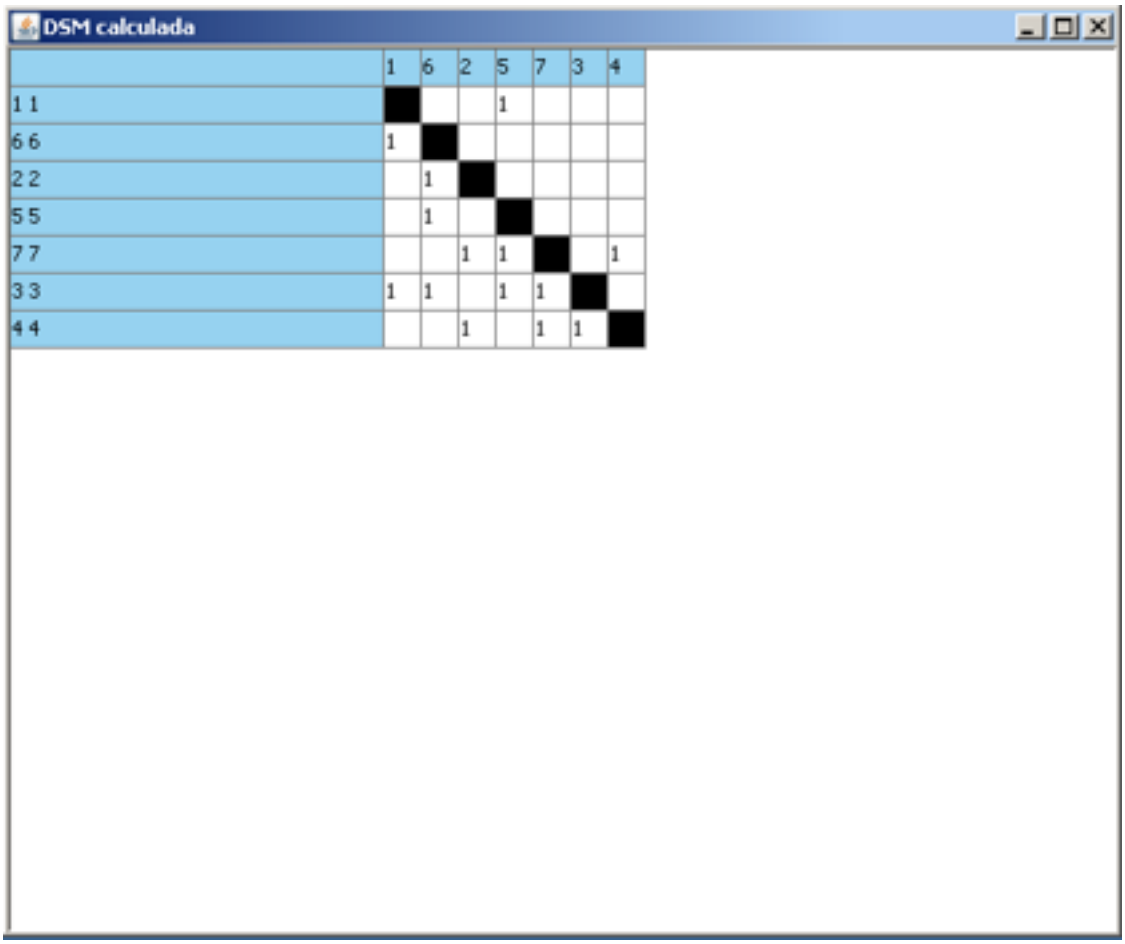

Figure 8: Antares DSM - Matrix partitioned by the algorithm Quick Triangular Matrix

belong to any cycle, since it does not test all possible options, although such cases are rare, as the experiments appear in Chapter 5. To ensure the identification of cycles, AntaresDSM supports another feature, shown below.

4.1.2 IDENTIFICATION OF CYCLES

With the partitioning complete, you can identify the cycles through the menu item *DSM* → *Identificar ciclos*. The results will appear in the same window described in the previous item. As result of this feature, the cells above the main diagonal belonging to cycles are painted in red (Figure 9).

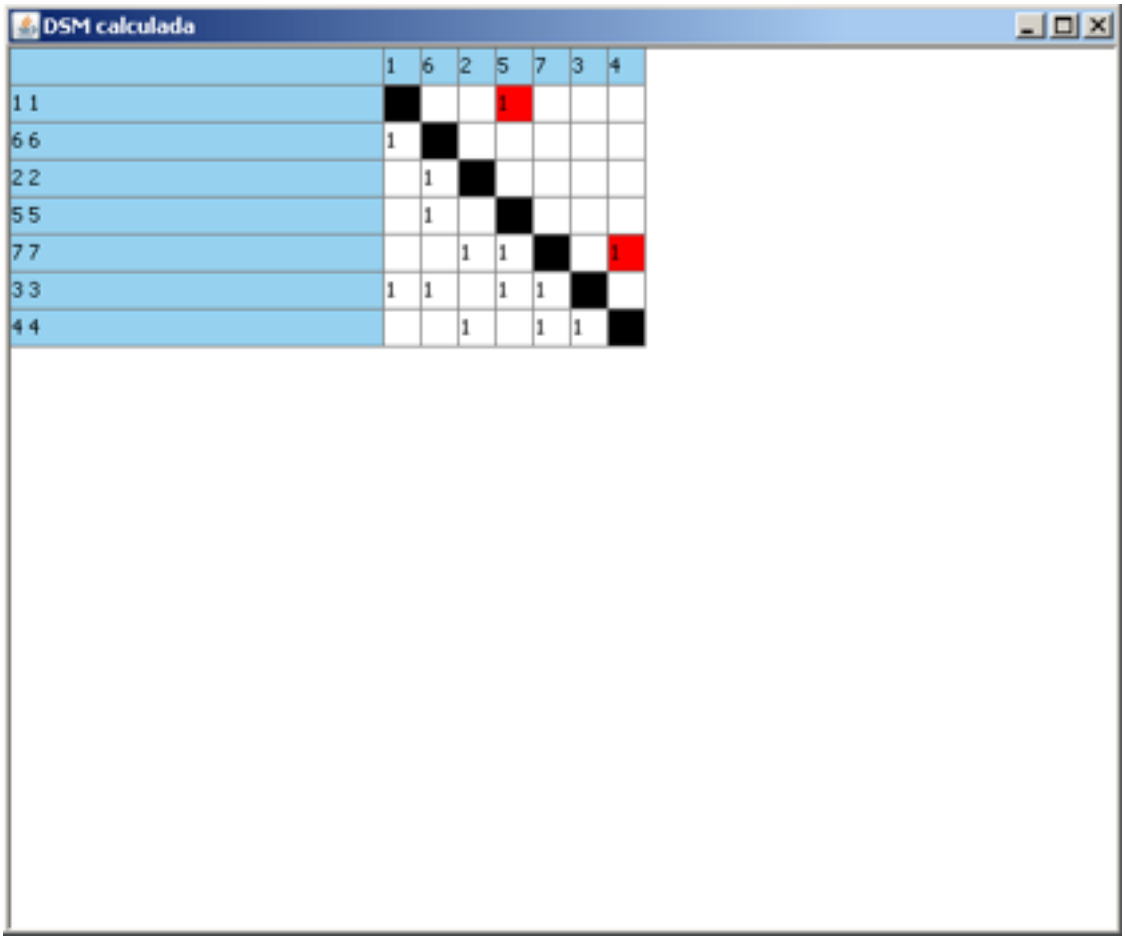

Figure 9: Antares DSM - Identification and visualization of cycles in DSM

The cells in red indicate to the analyst that the dependence must be reviewed and, if possible, removed. In case of multiple dependencies, ie degree greater than 1, the item of the cycle that is above the upper diagonal is the most "weak link" of the cycle, that is what has less value, and therefore what can be removed more easily.

The identification of cycles only after the partitioning greedy, guarantee a better perfor-

mance to the execution in order to identify cycles, because it is not necessary to test all the cycles of dependency matrix, and yes, only for those who remained above the diagonal. Thus, the identification of cycles complements the partitioning greedy by achieving identify cycles efficiently. Together the two algorithms achieve the optimal outcome for the proposed issue, since the algorithm of identification of cycles verifies if the elements above the diagonal belong to cycles traversing the paths of recursive way, to be found any cycle, or traveled all possible paths from that element.

This feature is also useful to validate the results of **Quick Triangular Matrix**, after its execution. If there are elements above the diagonal matrix in the upper partitioned that were not painted in red, then the algorithm **Quick Triangular Matrix** did not achieve optimal results.

5 EXPERIMENTAL RESULTS

This chapter aims to present some experimental results related to the algorithm **Quick Triangular Matrix**, implemented and used by the tool **Antares DSM**, and its comparison with the algorithm partitioning of the complexity exponential created by *David A. Gebala* and *Steven D. Eppinger* in 1991(GEBALA; EPPINGER, 1991), implementation in a *Microsoft Excel* macro.

For each algorithm, were created random matrices of 10x10, 50x50 e 100x100, with small population (probability of 10% of each cell is filled), medium population (probability of 50% of each cell is filled) and big population (probability of 90% of each cell is filled) each, and 10 tests for each combination of *size of the matrix X population size*, with the exception of combinations of matrices 100x100, where were made 5 tests for each. For the tests were computed the time of execution of each algorithm within its tool and whether or not to have great results, and calculated the mean and standard deviation for each case.

The tests were performed on a computer with 512MB of RAM and processor *Sempron* of 1,60GHz.

The test results are presented below.

Matriz 10 elementos pouco populosa (0,1) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,3	0,3	0,2	0,1	0,3	0,2	0,3	0,1	0,2	0,23	0,08232726
Resultado ótimo (S N)	S	N	S	N	S	S	S	N	S	S	70%	

Matriz 10 elementos pop média (0,5) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,1	0,1	0,2	0,1	0,1	0,2	0,2	0,2	0,2	0,17	0,067494856
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 10 elementos populosa (0,9) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,2	0,2	0,2	0,1	0,1	0,2	0,1	0,2	0,1	0,1	0,15	0,052704628
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 10 elementos	90%	0,183333	0,074663998

Matriz 50 elementos pouco populosa (0,1) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,1	0,2	0,3	0,3	0,3	0,3	0,2	0,3	0,2	0,25	0,070710678
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 50 elementos pop média (0,5) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,2	0,2	0,2	0,3	0,3	0,3	0,2	0,1	0,3	0,24	0,06992059
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 50 elementos populosa (0,9) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,2	0,3	0,2	0,3	0,2	0,2	0,4	0,2	0,2	0,3	0,25	0,070710678
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 50 elementos	100%	0,246667	0,068144539

Matriz 100 elementos pouco populosa (0,1) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,3	0,3	0,3	0,2						0,28	0,04472136
Resultado ótimo (S N)	S	S	S	S	S						100%	

Matriz 100 elementos pop média (0,5) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,5	0,3	0,4	0,2	0,3						0,34	0,114017543
Resultado ótimo (S N)	S	S	S	S	S						100%	

Matriz 100 elementos populosa (0,9) - Antares DSM - Quick Triangular Matrix

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,5	0,4	0,3	0,3	0,4						0,38	0,083666003
Resultado ótimo (S N)	S	S	S	S	S						100%	

Média c/ 100 elementos	100%	0,333333	0,089973541

Média Quick Triangular Matrix - Antares DSM	96%	0,2	0,092842985

Matriz 10 elementos pouco populosa (0,1) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,4	0,2	0,3	0,4	0,4	0,2	0,2	0,2	0,3	0,2	0,28	0,091893658
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 10 elementos pop média (0,5) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,4	0,3	0,3	0,3	0,2	0,2	0,3	0,4	0,2	0,2	0,28	0,078881064
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 10 elementos populosa (0,9) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	0,3	0,3	0,3	0,3	0,3	0,2	0,2	0,3	0,3	0,3	0,28	0,042163702
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 10 elementos	100%	0,28	0,071438423

Matriz 50 elementos pouco populosa (0,1) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	12,1	12,1	12	12	12,1	12,1	12,1	12,3	12,1	12,1	12,1	0,081649658
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 50 elementos pop média (0,5) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	13,8	13,7	13,9	13,9	13,7	13,7	13,9	13,8	13,7	13,8	13,79	0,087559504
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Matriz 50 elementos populosa (0,9) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	14,5	14,4	14,5	14,5	14,5	14,7	14,5	14,4	14,5	14,5	14,5	0,081649658
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 50 elementos	100%	13,463	1,0270424

Matriz 100 elementos pouco populosa (0,1) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	185	185	184	185	185						184,8	0,447213595
Resultado ótimo (S N)	S	S	S	S	S						100%	

Matriz 100 elementos pop média (0,5) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	210	210	209	210	209						209,6	0,547722558
Resultado ótimo (S N)	S	S	S	S	S						100%	

Matriz 100 elementos populosa (0,9) - Particionamento NP Completo

N. do experimento	1	2	3	4	5	6	7	8	9	10	Média	Desvio padrão
Tempo de execução (s)	220	219	220	220	219						219,6	0,547722558
Resultado ótimo (S N)	S	S	S	S	S	S	S	S	S	S	100%	

Média c/ 100 elementos	100%	204,67	15,15004322

Média Particionamento NP Completo	100%	46,4	80,14559551

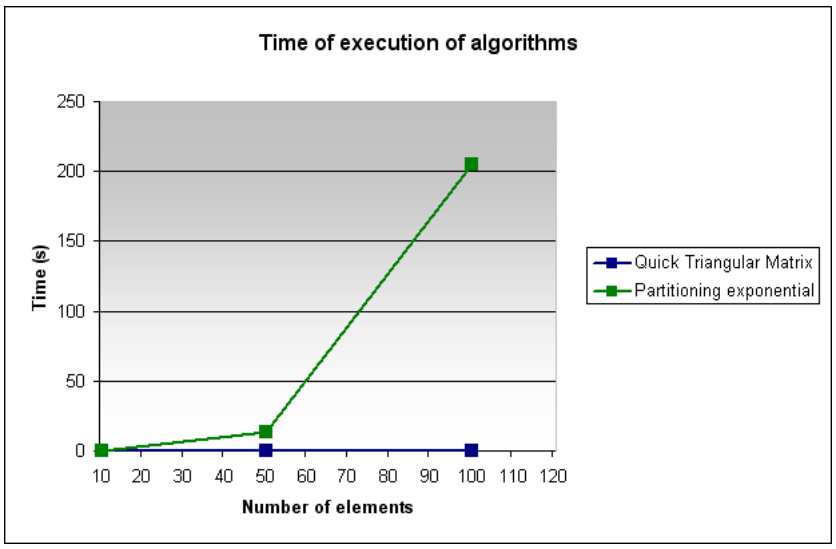

Figure 10: Time of execution of each algorithm

The Figure 10 presents a graphic interpretation of results in relation to the execution time of each algorithm. It is observed that for matrices with few elements, the difference in execution time is almost irrelevant, within the margin of error indicated by standard deviation; However, while the time of the algorithm **Quick Triangular Matrix** grows almost linearly, the time of the partitioning of *Gebala* and *Eppinger* grows exponentially, making the difference in time between the two algorithms increasing as the number of elements increases.

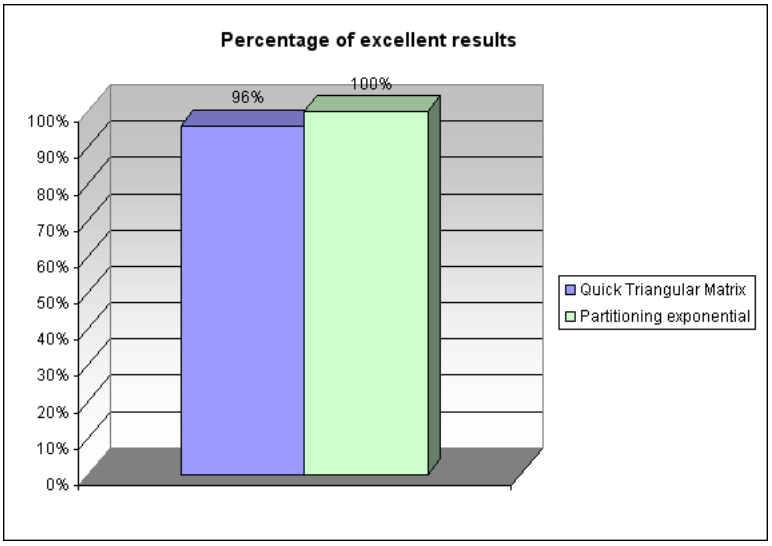

Figure 11: Percentage of excellent results in each algorithm

The Figure 11 presents a graphic interpretation of results in relation to the percentage of excellent results obtained with the implementation of each algorithm. In this aspect, the algorithm of *Gebala* and *Eppinger* proved to be better than the **Quick Triangular Matrix**, as expected, reaching the optimal outcome in 100% of the tests, versus 96% of **Quick Triangular Matrix**.

While the algorithm of *Gebala* and *Eppinger* tests all possibilities to make the lower triangular matrix, **Quick Triangular Matrix** does the ordering based on heuristics, in an attempt to find solution to the problem with the minimum possible trade. The techniques heuristics not provide the best solutions, but only valid solutions, closer together. However, in general, have better performances in relation to other techniques of exponential complexity.

6 CASE STUDY

6.1 OBJECTIVES

This case study aims to conduct a preliminary assessment of the benefits and limitations of using the DSM for viewing, understanding and optimization models as well of the tool **Antares DSM** and the algorithm partitioning **Quick Triangular Matrix**.

The proposal of the case study is to use modeling in an initial phase of project, and try to identify problems of unwanted cycles in this modeling using **Antares DSM**.

The tool should point to the analyst to cycles exist in the project, which, in turn, should review them and, if possible, modify the project.

6.2 CHOICE OF CASE

Although DSMs can be used in many professional areas, was chosen a case in the area of *Software Engineering*, sub-area of *Computer Science* for review.

The application being studied is a system of a pharmacy. This system was chosen because it is a simple and didactic. The Figure 12 presents a structural vision of the system by means of a diagram of classes in UML. The Unified Modeling Language (UML) (OMG, 2008) is a language for general purpose for specifying, visualization, construction and documentation of artifacts of software systems. Basically, the UML allows developers view the products of their work on standardized charts (WIKIPÉDIA, 2008).

6.3 ANALYSIS OF CASE

For the analysis of the case, was created in a file format standard input of the tool **Antares DSM** (Listings **??**), corresponding to the diagram of classes for the pharmacy system (Figure 12). Next, the matrix was generated and the flow pattern of the tool held (partitioning and

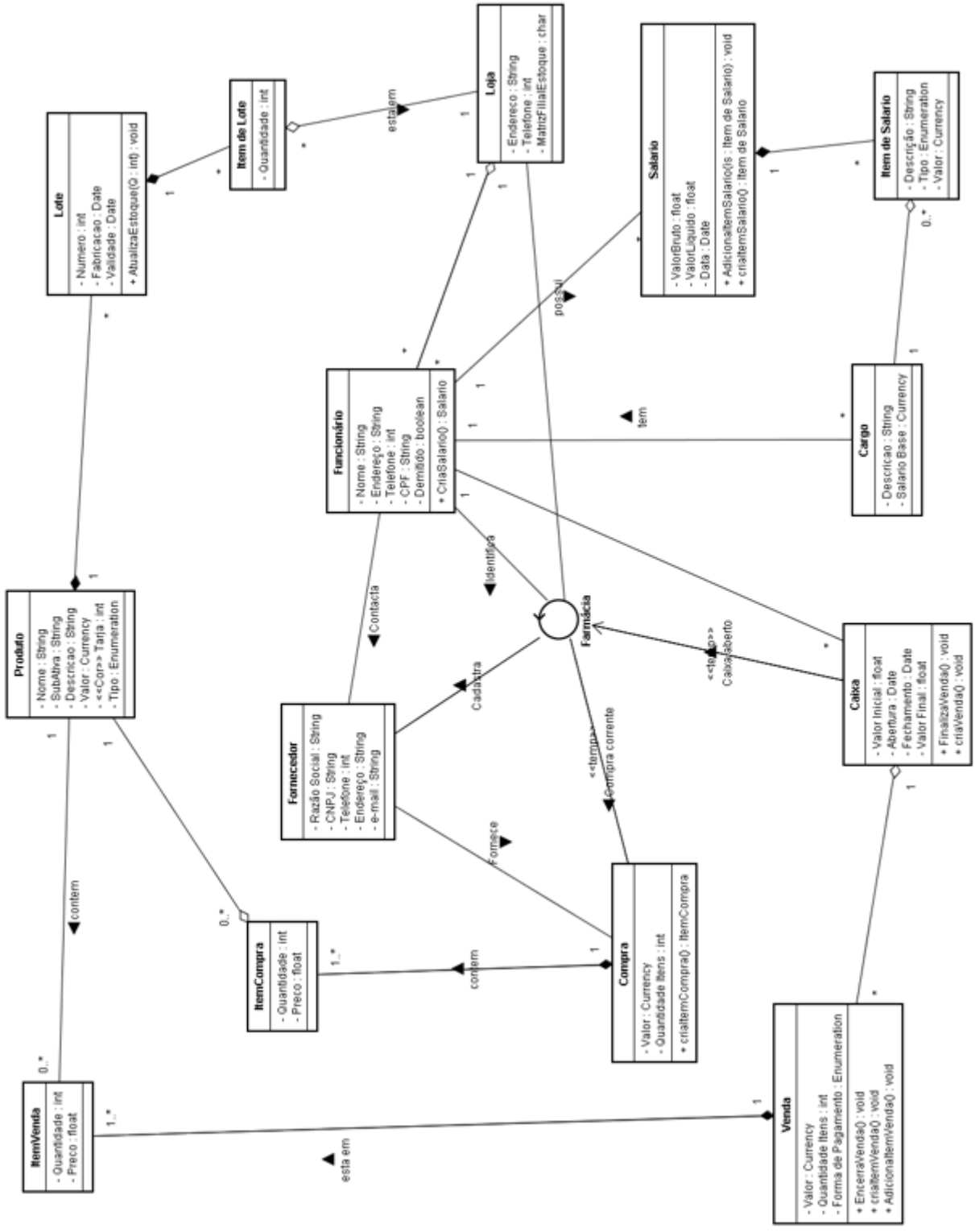

Figure 12: UML modeling initial system of pharmacy

identification of cycles).

6.3.1 VIEWING GRAPH AND DSM

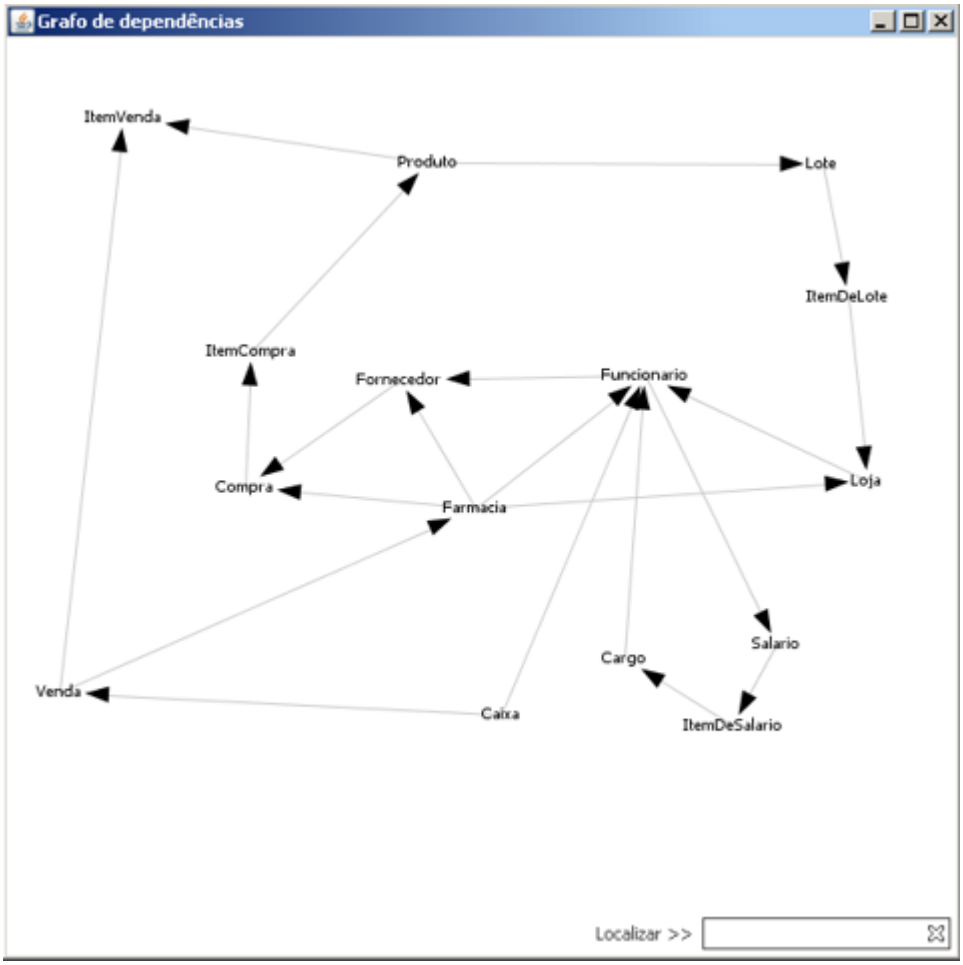

Figure 13: Graph of the system of pharmacy organized as in the diagram of classes

The Figures 13 and 14 show the graph and DSM corresponding to the class diagram of the system's pharmacy. Despite the DSM promotes an visualization compact and easy to understand the system as a whole, is not trivial the location of their cycles, even as a small system. In a larger system, it could sue a big time for the analyst to make this activity manually.

6.3.2 APPLICATION OF METHODS

After the initial display, were used the methods of partitioning and identification of cycles, respectively. The result of application of the methods is the matrix shown in Figure 15. These results show that the dependencies $4 \rightarrow 8$, $5 \rightarrow 3$, $13 \rightarrow 12$ and $15 \rightarrow 7$, painted in red, are the cycles that should be reassessed. Rearrange the graph, you can prove the existence of such

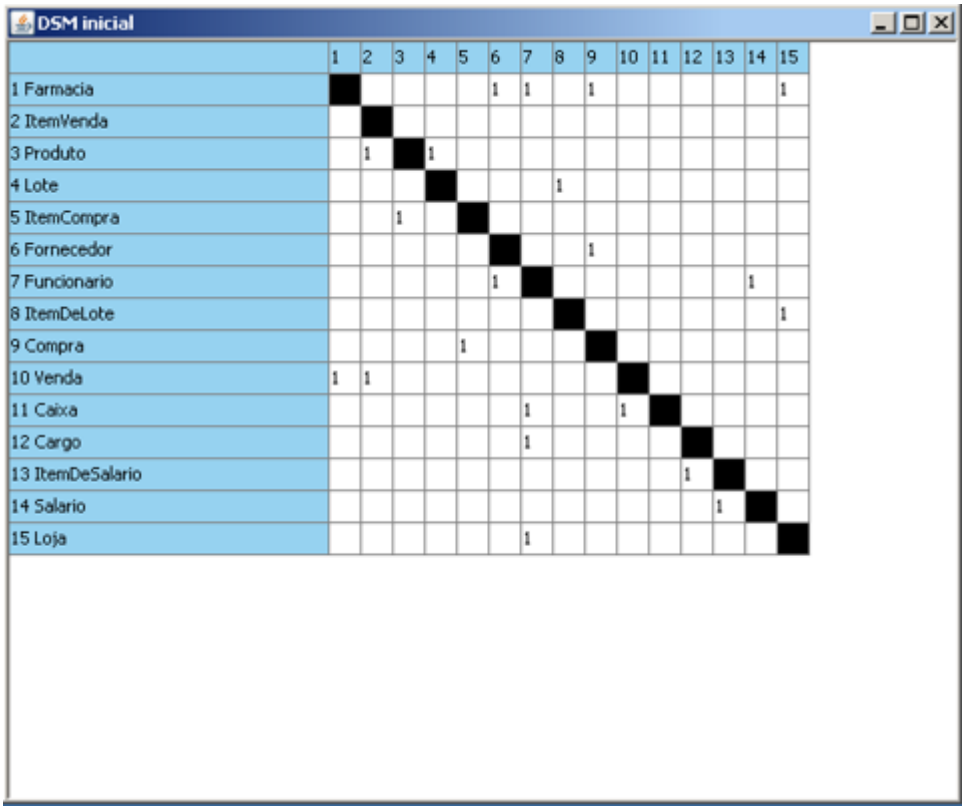

Figure 14: DSM system's pharmacy

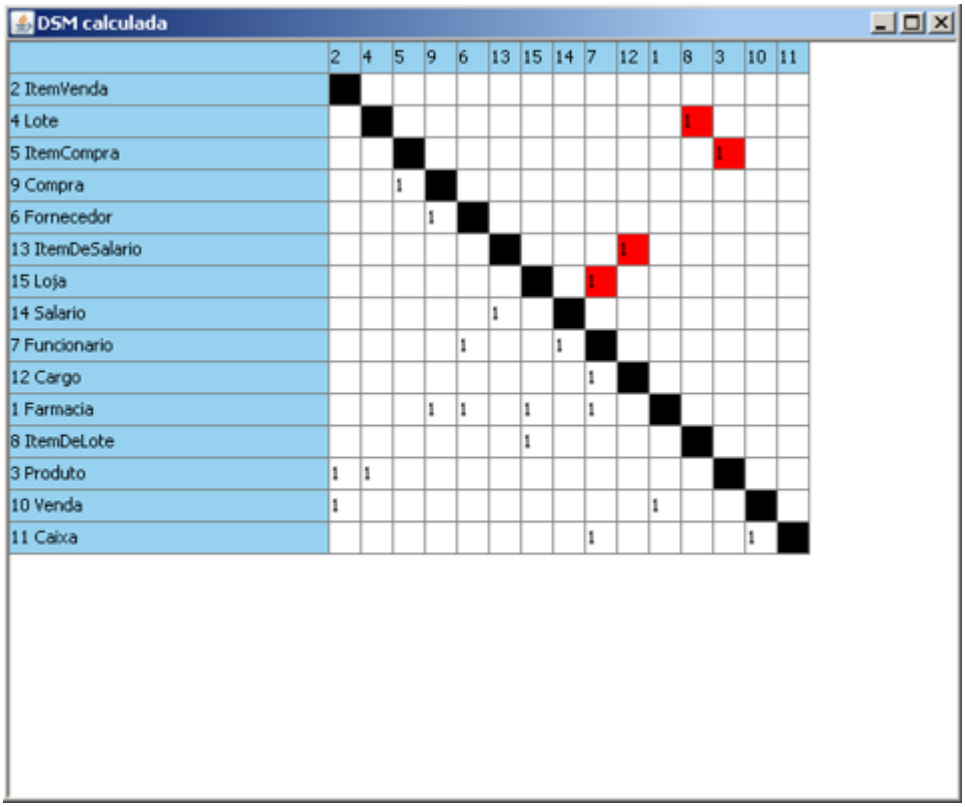

Figure 15: DSM system's pharmacy after partitioning and identification of cycles

cycles, as shown in Figure 16.

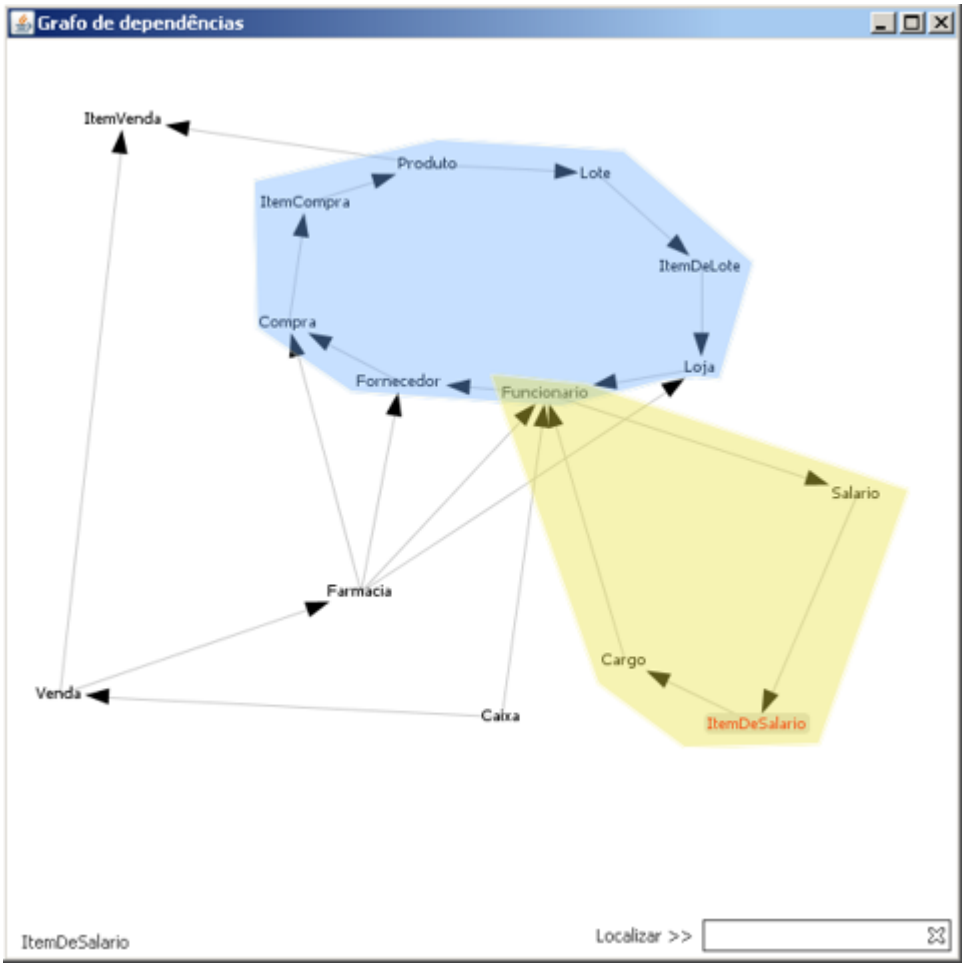

Figure 16: Graph of the system of pharmacy reorganized with an emphasis on the cycles

6.3.3 CHANGES IN THE MODEL

Based on previous results, the analyst should reassess the model shown in cycles. After the revaluation, were made the following changes:

- Removal of the relationship between the classes *Funcionario* and *Fornecedor*, and insertion of the relationship between *Compra* and *Funcionario*, as the employee contacts the official supplier within the real world, but in the system which is interesting in this case is doing an audit of purchases made by the employee.

- Reversing the direction of dependency *Fornecedor* $\rightarrow$ *Compra* to *Compra* $\rightarrow$ *Fornecedor*, on the assumption that the supplier does not provide the purchase within the vision of pharmacy, and yes, that the purchase is made from a supplier.

- Reversing the direction of dependency $ItemDeLote \rightarrow Loja$ to $Loja \rightarrow ItemDeLote$, as for the pharmacy is better to control the stock knowing what items of lot are in each store, than that store belongs to each item of lot.

- Reversing the direction of dependency $Cargo \rightarrow Funcionario$ to $Funcionario \rightarrow Cargo$. Despite a post have one or more employees, more relevant information in this case is whether the job of every employee.

- Reversing the direction of dependency $ItemDeSalario \rightarrow Cargo$ to $Cargo \rightarrow ItemDeSalario$. In this case there was a serious error of modeling, as an item of salary does not have a job, but rather a job that can be linked to items of salary, to set the initial salary for certain employee based on his job.

After the changes, was created a new class of UML diagram (Figure 17).

6.3.4 SUBSEQUENT CHECKS

After the changes described in the previous item, the analyst must update the matrix, saving it then. As the processes of modeling is iterative, any change in the modeling should lead to a change in the DSM, which in turn is to be partitioned again, and then submitted to the verification process of cycles. In this case study, after the changes of the model and the DSM, the matrix was partitioned again, including the application of the method of identifying cycles, and the result is shown in Figure 18.

Note that, as the algorithm **Quick Triangular Matrix** is greedy, this second iteration, it has not reached the optimum result, leaving elements that do not belong to cycles above the diagonal, but this “problem” was corrected with the method of identifying cycles, that painted any of the cells in red, showing the absence of cycles in modeling and the current success of the tool.

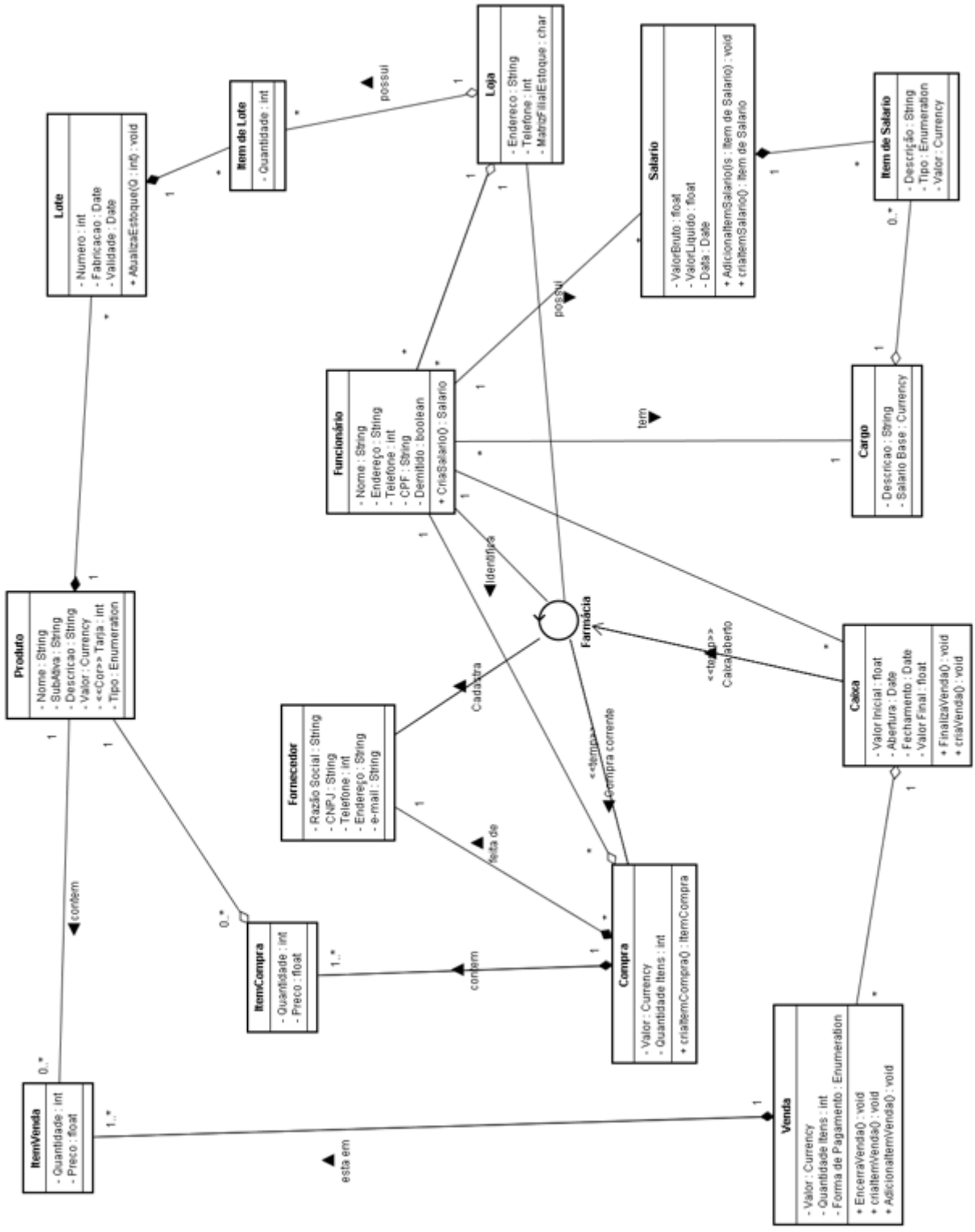

Figure 17: UML modeling system's pharmacy after the second round of improvements

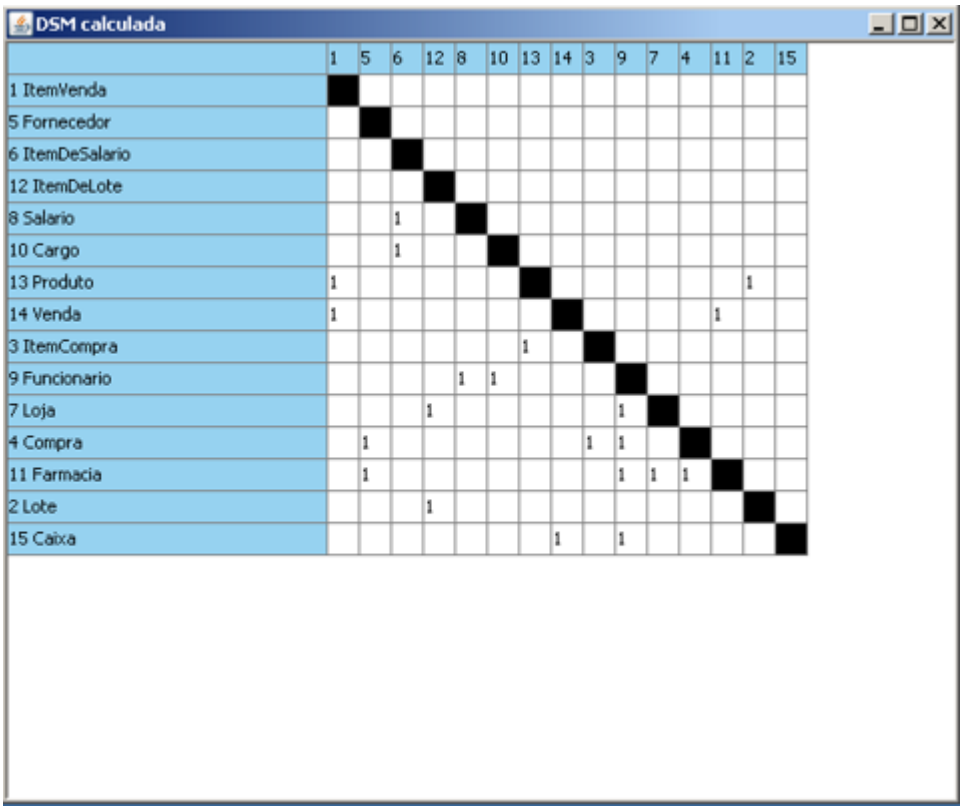

Figure 18: Second partitioning with identification of cycles of the DSM system's pharmacy

7 CONCLUSIONS

This work took as its starting point a study on DSMs, its concepts and practices associated in various fields. In particular, DSMs were studied in light of its increasing use in the area of Software Engineering, where DSMs can help analysts to improve their projects from a better understanding and management of the dependencies between artifacts and developers. From this study, some topics of interest were identified and led to the development of this work.

The main contributions of this work are:

1. **Creation of the algorithm Quick Triangular Matrix** - The use of DSMS for modeling software systems with potentially thousands of classes and methods, demand efficient algorithms to manipulate the matrices. As shown the comparison between the algorithms presented in Chapter 5, for applications with many elements, the waiting time for an identification of cycles in an algorithm of exponential complexity can make use of DSMS unviable. For these cases, the algorithm can be created an interesting alternative.

2. **Development of tool Antares DSM** - The creation of the tool was shown to be important because the algorithm Quick Triangular Matrix must be incorporated into a development environment for analysts and programmers who can use it, together with the design of DSMs. The tool also is an important contribution because it is free software.

3. **Study on DSMs** - The study on the DSMs in itself is a valuable contribution, by explaining their concepts and how they should be used with a simple case study in the area of Software Engineering, which may motivate others to explore and take advantage of DSMs from day to day.

The use of DSMS proved to be a compact shape and visual, easy to understand and manipulation, which leads to faster identification of dependencies between elements.

The algorithm **Quick Triangular Matrix** (embedded in the tool **Antares DSM**) proved to be very efficient with regard to performance, can be used mainly in models with many elements.

However, the studies show that the algorithm does not reach the optimum result in about 4% of cases, according to the experiments, problem that can be circumvented with the application together, of the algorithm for identification of cycles (also implemented in the tool **Antares DSM**).

The studies conducted have focused on the improvement of models through the identification of cycles, however, are not in all types of models that the cycles are undesirable. The analyst must observe before the need to remove cycles in your project. For other applications, there are other algorithms on which this work does not address, such as tearing, which serves, among other things, to set submodules or teams in the project.

REFERENCES

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. *Introduction to Algorithms*. [S.l.]: MIT Press and McGraw-Hill, 1990.

DIESTEL, R. *GraphTheory*. [S.l.]: Springer, 2000.

DSMWEB. 2008. Último acesso em 19 de Junho de 2008. Disponível em: <http://www.dsmweb.org>.

GEBALA, D. A.; EPPINGER, S. D. Methods for analysing design procedures. In: *ASME Design Theory and Methodology Conference*. [S.l.]: DETC, 1991. p. 227–233.

HOARE, C. Quicksort. *Computer Journal*, n. Vol. 5, p. 10–15, 1962.

IEEE Standard Glossary of Software Engineering Terminology. New York: Institute of Electrical and Electronics Engineers, 1990.

MANZIONE, L.; B.MELHADO, S. Porque os projetos atrasam? uma análise crítica da ineficácia do planejamento de projetos adotada no mercado imobiliário de são paulo. In: *III Encontro de Tecnologia da Informação e Comunicação na Construção Civil*. Porto Alegre - Rio Grande do Sul: [s.n.], 2007.

MORAES, N. A. *Compreensão e Visualização de Projetos Orientados a Objetos com Matriz de Dependências*. 2007. Universidade Federal da Bahia.

MORI, T. et al. Task planning for product development by strategic schedulling of design reviews. In: *1999 ASME Design Engineering Technical Conferences*. Las Vegas NV: DETC, 1999.

OMG. 2008. Último acesso em 19 de Junho de 2008. Disponível em: <http://www.uml.org/>.

PIERONI, E.; NAVEIRO, R. M. A design structure matrix (dsm) aplicada ao projeto de navios. In: *Anais do V Congresso Brasileiro de Gestão e Desenvolvimento do Produto*. Curitiba - Paraná: [s.n.], 2005.

WIKIPÉDIA. 2008. Último acesso em 8 de Junho de 2008. Disponível em: <http://pt.wikipedia.org/wiki>.