Proceedings of the 13th International Conference "Reliability and Statistics in Transportation and Communication" (RelStat'13), 16–19 October 2013, Riga, Latvia, p. 340–351. ISBN 978-9984-818-58-0
Transport and Telecommunication Institute, Lomonosova 1, LV-1019, Riga, Latvia

MULTI-TIER ARCHITECTURE OPTIMISATION FOR LOGISTICS AND TRANSPORT SOFTWARE

Sergey Orlov, Andrei Vishnyakov

Transport and Telecommunication Institute
Lomonosova str.1, Riga, LV-1019, Latvia
E-mail: Orlovs.S@tsi.lv, andrei.vishnyakov@gmail.com

At the moment there are no well-proven methods for software architecture quality evaluation. The same apply for logistic and transportation systems. Although there are design approaches, which rely on proved and tested practices. One of such approaches is to reuse patterns, which express a fundamental structural organization of software systems and its behaviour.

In this paper the technique that allows selecting and evaluating suite of architectural patterns is proposed. It can be used for logistics and transportation software, which is constructed using Multi-tier architecture. The technique allows us to consistently evaluate the impact of specific patterns to software characteristics with a given functionality. Effectiveness and efficiency of the described method is confirmed by a case study.

Keywords: multi-tier architecture, pattern, functional points, coupling and cohesion, logistics and transport software, optimisation, decision

1. Introduction

It's important to construct architecture for logistics and transportation software properly and with the use of best practices, so we need to pay a lot of attention when building software architecture for such systems.

Architectural design provides an understanding of the system organization. Also it creates a framework for the proper representation of a system. The creation of architecture is the first and fundamental step in the software designing. It creates software system representation base that satisfies to the full range of detailed requirements [1, 2, 3].

As long as there is no effective method for the architecture building we should rely on used techniques as well as past experience in that area. One of the common approaches is to use architectural patterns for creation of the software architecture.

Architectural patterns organize the essence of architecture which was used in various software systems. Today the patterns are widely used during the software development process. They help to reuse the knowledge and best practice [4, 5, 6].

Architectural patterns can be seen as a generalized description of best practice. The patterns were tested and proven in a variety of systems and environments, as a result of that, the architectural pattern describe the system organization which has been successful in previous systems.

It's obvious what we need to have some technique that allows selecting the optimal suite of patterns from a number of patterns, also such selection should take into account the specific requirements for the logistics and transport system.

2. Selection of an Optimal Patterns Suite

To select an optimal patterns suite we need to define a model, choose a set of patterns and examine their impact on system characteristics.

2.1. Model definition for selection of an optimal patterns suite

Let's assume that there is a set of patterns which can be separated into groups according to their corresponded functionality. Also we know numerical values of system characteristics which depend on used patterns for the given system. So we need to develop a model which helps us to determine the optimal suite of patterns for logistic and transport system with a given functionality.

Suppose that there is a set of input pattern groups — $\{P_i | i = 1,..., g\}$.

Each group can have different number of patterns, so we can define it as follows:

$$\{P_{ij} \,|\, i = 1,...,g; j = 1,...,m_j\},$$

where P_{ij} — i-th pattern from group j, m_j — number of pattern in group j which is a variable number.

Let's assume that from some groups we aren't obligated to select a pattern (this is due to the fact that the selection of some patterns can exclude a whole group of patterns). On the other hand, we can select several patterns from some of the groups.

Thus the input data for our model makes a complete set of patterns for each group, and such set can be represented as a multiset.

For simplicity, we reduce the multiset to a uniform set of patterns $\{P_1,...,P_n\}$ where we use special restrictions for partitioning to the groups.

At the output, the model with the specified constrains should select the optimal combination of patterns which should be used for software development.

The produced restrictions should exclude those combinations of patterns that are logically inconsistent or interchangeable. In addition, some restriction should allow selection of multiple patterns from specified group of patterns.

The objective function for finding the optimal suite of pattern defined as follows:

$$W = f(P_1) \times x_1 + f(P_2) \times x_2 + \ldots + f\left(P_n\right) \times x_n \to min\,,$$

where:

$f(P_i)$ — function which reflects a numerical changes of the system characteristics depending on used pattern P_i;

x_i — variable which indicates the usage of the *i-th* pattern.

It's obviously that the integrality constrain should be applied for a given variable x_i:

$$x_i = \{0,\ 1 \,|\, i = 1,2,...,n\},$$

where n — number of patterns in the one dimensional set which were transformed from the original multiset of patterns.

To indicate the fact that we can select only one pattern from the group, let's introduce the following restriction:

$$\sum_{i=start}^{i=end} x_i = 1,$$

where

start, end — the start and end indices of patterns in a group.

To take in to account that the selection of the *j-th* pattern excludes patterns from a different group, we use the following restrictions:

$$x_j + \sum_{i=start}^{i=end} x_i = 1.$$

If we can select any number of patterns from the group we specify the following constrains:

$$\sum_{i=start}^{i=end} x_i \le (end - start + 1).$$

On the base of the mentioned definitions and assumptions, we obtain the classical integer programming problem where we need to find the optimal solution.

We should pay special attention for choosing the function $f(P_i)$. Such selection should be based on the requirements for a software system.

2.2. The choice of patterns for Multi-tier architecture

According to the statistics on architecture types used for transportation and logistics systems, which are represented in the global ISBSG database, most of these systems are based on client-server architecture model [7]. Nowadays the most used subtype of such architecture is Multi-tier architecture.

Therefore, for our case study we select a set of patterns used for transportation and logistics systems' Multi-tier architectures.

There are patterns that can be divided into the following groups:

- **Domain Logic Patterns**. *Transaction Script*, *Domain Model*, *Table Module*, *Service Layer*.
- **Data Source Architectural Patterns**. *Table Data Gateway*, *Row Data Gateway*, *Active Record*, *Data Mapper*.
- **Object-Relational Behavioural Patterns**. *Unit of Work*, *Identity Map, Lazy Load*.
- **Web Presentation Patterns**. *Model View Controller*, *Page Controller*, *Front Controller*, *Template View*, *Transform View*, *Two-Step View*, *Application Controller*.
- **Distribution Patterns**. *Remote Facade*, *Data Transfer Object*.
- **Offline Concurrency Patterns**. *Optimistic Offline Lock*, *Pessimistic Offline Lock*, *Coarse Grained Lock*, *Implicit Lock*.

Fowler in [6] indicates the steps how to select a pattern from multiple groups taking into account the requirements for the software. The problem of his approach is that it isn't formalized enough. Also he indicates the relationship between groups of patterns, for example, it's allowed to choice only one pattern from multiple groups, etc. This selection technique can be represented as follows:

1. Initially we have to select the base pattern for *Domain Layer* implementation; such pattern should be selected from *Domain Logic Patterns* group.
2. Next, we need to select a pattern for *Data Source Layer* implementation (it should be selected from Data *Source Architectural Patterns* group). This choice also depends on the first step (for example, when we select *Domain Model* on the first step we can choose only *Data Mapper* on this step). Together with patterns from *Data Source Layer* we can use *Object-Relational Behavioural Patterns*, *Concurrency Patterns* and some other groups of the patterns.
3. In the final step we do select a pattern from *Presentation Layer* (from *Web Presentation Patterns* group).
4. Furthermore, in addition to the selected patterns, we select other patterns from the remaining groups.

As a result of this technique application, we have obtained the list of patterns listed in Table 1 which we use in our case study.

Table 1. List of patterns used for the case study

Group of patterns	Pattern	Pattern's notation
Domain Logic Patterns	Transaction Script	P_{11}
	Domain Model	P_{12}
	Table Module	P_{13}
	Service Layer	P_{14}
Data Source Architectural Patterns	Table Data Gateway	P_{21}
	Row Data Gateway	P_{22}
	Active Record	P_{23}
	Data Mapper	P_{24}
Web Presentation Patterns	Model View Controller	P_{31}
	Page Controller	P_{32}
	Template View	P_{33}
	Application Controller	P_{34}
Distribution Patterns	Remote Facade	P_{41}
	Data Transfer Object	P_{42}
Offline Concurrency Patterns	Optimistic Offline Lock	P_{51}
	Pessimistic Offline Lock	P_{52}
	Coarse Grained Lock	P_{53}
	Implicit Lock	P_{54}

2.3. The patterns usage restrictions

When we build the model we should take into account the following corresponding constraints:

- restrictions which are applied on pattern groups;
- restrictions applied on patterns compatibility.

For the considered patterns we have the following limitations:

1. We can choose only one pattern from the first three groups as well as from *Offline Concurrency Patterns*;
2. We can choose any patterns from the remaining groups (each pattern can be selected only one time);

3. If *Transaction Script* is selected from the first group we can choose *Table Data Gateway* or *Row Data Gateway* from the second group;
4. If *Table Module* is selected from the first group we are allowed to choose only *Table Data Gateway* from the second one;
5. If *Domain Model* is selected from the first group we can choose *Active Record* or *Data Mapper* from the second group.

2.4. Mathematical model building

Applying the above results we obtain the following objective function:

$$W = f(P_{11}) \times x_1 + f(P_{12}) \times x_2 + f(P_{13}) \times x_3 + f(P_{14}) \times x_4 + f(P_{21}) \times x_5 + f(P_{22}) \times x_6 + f(P_{23}) \times x_7 +$$
$$+ f(P_{24}) \times x_8 + f(P_{31}) \times x_9 + f(P_{32}) \times x_{10} + f(P_{33}) \times x_{11} + f(P_{34}) \times x_{12} + f(P_{41}) \times x_{13} + f(P_{42}) \times x_{14} +$$
$$+ f(P_{51}) \times x_{15} + f(P_{52}) \times x_{16} + f(P_{53}) \times x_{17} + f(P_{54}) \times x_{18} \to min$$

with the following restrictions which came from the patterns usage limitations:

a) We can choose only one pattern from the first three groups:

$$x_1 + x_2 + x_3 + x_4 = 1,$$
$$x_5 + x_6 + x_7 + x_8 = 1,$$
$$x_9 + x_{10} + x_{11} + x_{12} = 1,$$
$$x_{15} + x_{16} + x_{17} + x_{18} = 1.$$

b) We can choose any patterns from the remaining groups (or not to choose a pattern at all):

$$x_{13} \le 1,$$
$$x_{14} \le 1,$$
$$x_{13} + x_{14} \le 2.$$

c) If *Transaction Script* is selected from the first group we can choose *Table Data Gateway* or *Row Data Gateway* from the second group:

$$x_1 + x_7 + x_8 \le 1.$$

d) If *Table Module* is selected from the first group we are allowed to choose only *Table Data Gateway* from the second one:

$$x_3 + x_6 + x_7 + x_8 \le 1.$$

e) If *Domain Model* is selected from the first group we can choose *Active Record* or *Data Mapper* from the second group:

$$x_2 + x_5 + x_6 \le 1.$$

2.5. Selecting of $f(P_i)$ function

Using the above listed patterns we need to build a model for selecting the optimal suite of patterns; where the requirement for the software should be considered. For doing so we must determine the patterns impact on specific system characteristics. This means that we need to define the function $f(P_i)$.

During the architecture design stage we can operate the system requirements as well as make indirect measures of some system characteristics, so one of the most suitable metric for consideration is functional point (*FP*) metric, which indirectly measures software and the cost of its development. The value of this metric reflects the functional complexity of the product [1, 8]. In addition to the complexity metric, inner (cohesion) and outer (coupling) relations should be measured [1].

The selection of a pattern affects the overall system characteristics. Therefore, it is necessary that the metric for such system also reflects this influence. In our case the metric should reflect a change of *FP* metric, coupling and cohesion when we use a specific pattern.

In order to combine these three metrics let's use criterion of efficiency described in the publication [9]. As long as the calculation of the proposed metrics for coupling and cohesion is quite complicated we replace these metrics with alternatives which are supported by many tools for metric calculation. For

example, we can use Coupling between Object Classes (*CBO*) and Lack of Cohesion of Methods (*LCOM*) metrics from Chidamber & Kemerer's metric suite [1, 10].

CBO and *LCOM* metrics are calculated for specific classes, but we need to evaluate the entire system. So it's necessary to make these metrics applicable for a group of classes. We define Coupling between Object Classes Factor (*CBOF*) and Lack of Cohesion of Methods Factor (*LCOMF*) metrics which could be used in our criterion of efficiency.

CBOF metric is defined as the arithmetic mean of the normalized values of *CBO* in the system (the value of this factor varies from 0 to 1):

$$CBOF = \frac{\sum_{i=1}^{N} \begin{Bmatrix} CBO_i, & if\ CBO_i < T_{CBO} \\ T_{CBO}, & else \end{Bmatrix}}{T_{CBO} \times N},$$

where

CBO — Coupling Between Object metric from Chidamber & Kemerer's metric suite;

T_{CBO} — threshold which cut down very large values of *CBO*. Such limitation is necessary as the theoretical value of *CBO* may vary indefinitely;

N — number of classes in the system.

The definition of *LCOMF* metric is similar, i.e. *LCOMF* defined as the arithmetic mean of the normalized values of *LCOM* in the system:

$$LCOMF = \frac{\sum_{i=1}^{N} \begin{Bmatrix} LCOM_i, & if\ LCOM_i < T_{LCOM} \\ T_{LCOM}, & else \end{Bmatrix}}{T_{LCOM} \times N},$$

where

LCOM — Lack Of Cohesion metric from Chidamber & Kemerer'c metric suite;

T_{LCOM} — threshold which cut down very large values of *LCOM*. Such limitation is necessary as the theoretical value of *LCOM* may vary indefinitely;

N — number of classes in the system.

Thus a metric of *original architecture efficiency K* defined as:

$$K = \frac{\alpha_1 \times FP}{(1 - \alpha_2 \times CBOF) \times (1 - \alpha_3 \times LCOMF)},$$

where

α_1, α_2, α_3 – weight coefficients of efficiency indicators;

FP — the value of functional points;

CBOF – the value of Coupling between Object Classes Factor;

LCOMF – the values of Lack of Cohesion of Methods Factor.

Based on listed above, our function which reflects numerical changes of the system characteristics depending on used pattern P_i defined as follows:

$$f(P_i) = \frac{K'_{P_i}}{K},$$

where

K — the metric of architecture efficiency;

K'_{Pi} — metric of partial *pattern-architecture efficiency* (if pattern P_i is used for software development).

Therefore metric of partial *pattern-architecture efficiency K'* defined as:

$$K'_{Pi} = \frac{\alpha_1 \times FP'_{Pi}}{(1 - \alpha_2 \times CBOF'_{Pi}) \times (1 - \alpha_3 \times LCOMF'_{Pi})},$$

where

FP'_{Pi} — the value of functional points if pattern P_i is used for software development;
$CBOF'_{Pi}$ – the value of *CBOF* if pattern P_i is used for software development;
$LCOMF'_{Pi}$ – the value of *LCOMF* if pattern P_i is used for software development.
FP' is a modification of the original *FP* and it is calculated as follows:

$$FP' = UFP \times \left(0.65 + 0.01 \times \sum_{i=1}^{14} CF_i \right) + P_{FP},$$

where

UFP – Unadjusted Function Point count;
P_{FP} – the value of functional points for specified pattern implementation;
and CF_i defined as follows:

$$CF_i = \begin{cases} 5, & if\ c_i \times F_i > 5; \\ round(c_i \times F_i), & otherwise, \end{cases}$$

where

CF_i — adjusted degree of influence coefficient which corresponds to F_i used in original *FP*;
c_i — pattern influence on *i*-th system's characteristic.

For getting c_i values, first, we need to evaluate a characteristic using the following scale:

- 1 — use of a pattern reduces the significance of a system characteristic;
- 2 — use of a pattern slightly reduces the significance of a system characteristic;
- 3 — no influence;
- 4 — use of a pattern slightly actualises a system characteristic;
- 5 — use of a pattern actualises a system characteristic (i.e. we must pay more attention to this characteristic when applying this pattern).

Next, these values are converted into c_i using scale conversion rule presented in Table 2.

Table 2. Characteristic's evaluation scale correspondence to c_i value

Score As	c_i
1	½
2	⅔
3	1
4	1½
5	2

CBOF' metric is modification of *CBOF* and it is defined as:

$$CBOF' = CBOF + \alpha \times CBOF \times \left(c_{Pi} - 3 \right),$$

where

CBOF – the original value of Coupling between Object Classes Factor;
α – weight coefficient;
c_{Pi} – pattern influence on *CBOF* which is evaluated using scale similar to c_i and varies from 1 to 5.

LCOMF' metric is defined as:

$$LCOMF' = LCOMF + \alpha \times LCOMF \times \left(c_{Pi} - 3 \right),$$

where

LCOMF – the original value of Lack of Cohesion of Methods Factor;
α – weight coefficient;
c_{Pi} – pattern influence on *LCOMF* which is evaluated using scale similar to c_i and varies from 1 to 5.

2.6. Obtaining values of indicators and coefficients

Prior the case study it's necessary to obtain the values of several indicators and coefficients. In our case, most of these values were obtained empirically.

Based on our requirements let's define the weight coefficient vector for metric of efficiency with the help of an expert evaluation. The weight coefficients of efficiency indicators for the functional points, coupling and cohesion factors are defined as:

$\alpha_1 = 0.5,\ \alpha_2 = 0.3,\ \alpha_3 = 0.2.$

So, functional points have the greatest weighting coefficient of efficiency indicator and *LCOMF* the least.

For obtaining *FP′* values we need to get the patterns influence coefficients. The Table 3 represents the patterns influence coefficients c_i with the given requirements. These figures are empirical and intended to demonstrate the proposed technique, so these values might be not optimal. To obtain more precise values of the coefficients, the values should be calibrated on a number of projects. These values also might be different for the software of the other domains, i.e. not transportation or logistics. In addition, the values might vary for systems with other requirements.

Table 3. Values of pattern influence coefficients c_i used in *FP* metric

Group of patterns	**Pattern**	**Pattern's notation**	c_1	c_2	c_3	c_5	c_8	c_9	c_{10}	c_{13}	c_{14}
Domain Logic Patterns	Transaction Script	P_{11}	3	4	1	2	4	2	5	4	5
	Domain Model	P_{12}	2	2	4	3	1	4	1	2	1
	Table Module	P_{13}	3	4	3	3	3	3	4	3	4
	Service Layer	P_{14}	2	1	3	2	2	3	2	2	2
Data Source Architectural Patterns	Table Data Gateway	P_{21}	2	3	2	3	3	2	4	3	3
	Row Data Gateway	P_{22}	3	4	2	2	4	3	5	4	4
	Active Record	P_{23}	3	3	3	3	3	3	2	2	3
	Data Mapper	P_{24}	2	2	4	2	2	4	1	2	2
Web Presentation Patterns	Model View Controller	P_{31}	4	2	3	2	3	3	2	2	3
	Page Controller	P_{32}	2	3	3	3	2	2	3	2	2
	Template View	P_{33}	4	3	2	2	4	3	3	2	2
	Application Controller	P_{34}	2	4	2	4	3	2	4	3	3
Distribution Patterns	Remote Facade	P_{41}	4	3	3	2	3	3	2	3	3
	Data Transfer Object	P_{42}	4	2	3	3	3	3	3	2	2
Offline Concurrency Patterns	Optimistic Offline Lock	P_{51}	4	2	2	2	4	3	3	3	3
	Pessimistic Offline Lock	P_{52}	3	3	4	4	2	3	3	3	3
	Coarse Grained Lock	P_{53}	3	3	3	3	3	3	3	3	3
	Implicit Lock	P_{54}	2	3	4	4	2	4	3	3	3

The coefficients c_i listed in the table have the following meanings:

- c_1 – pattern influence coefficient on system characteristic "Data Communications";
- c_2 – pattern influence coefficient on system characteristic "Distributed Data Processing";
- c_3 – pattern influence coefficient on system characteristic "Performance";
- c_5 – pattern influence coefficient on system characteristic "Transaction Rate";
- c_8 – pattern influence coefficient on system characteristic "Online Update";
- c_9 – pattern influence coefficient on system characteristic "Complex Processing";
- c_{10} – pattern influence coefficient on system characteristic "Reusability";
- c_{13} – pattern influence coefficient on system characteristic "Multiple Sites";
- c_{14} – pattern influence coefficient on system characteristic "Facilitate Change".

We consider only these system characteristics, since the described patterns do not affect other characteristics of the system, i.e. the pattern influence for them is 3.

To obtain the values of *FP′* we also need to determine the value of *FP* which is required for each pattern implementation. The empirically estimated values are shown in Table 4.

In addition to this we also need to evaluate the patterns impact on *CBOF* and *LCOM* values, so we need to obtain values of pattern influence coefficients c_{CBOF} and c_{LCOMF}. The values of pattern influence coefficients are listed in Table 4.

Table 4. Values of *FP* required for the patterns implementation and pattern influence coefficients

Pattern's notation	*FP*	c_{CBOF}	c_{LCOMF}
P_{11}	0	4	4
P_{12}	20	2	1
P_{13}	10	3	3
P_{14}	30	2	2
P_{21}	0	3	3
P_{22}	0	4	4
P_{23}	10	3	3
P_{24}	20	2	4
P_{31}	20	4	2
P_{32}	10	3	4
P_{33}	10	3	2
P_{34}	0	2	3
P_{41}	10	4	3
P_{42}	10	3	3
P_{51}	20	3	3
P_{52}	10	3	3
P_{53}	30	3	3
P_{54}	0	3	3

With the help of expert evaluation define the weighting coefficient used in for *CBOF* and *LCOMF* metrics as:

$\alpha = 0.1$.

After analysing all the values of *CBO* and *LCOM* for the considered programs the threshold values for metrics *CBOF* and *LCOMF* defined as:

$T_{CBO} = 100$;
$T_{LCOM} = 1000$.

3. Case Study

For the proposed technique validation let's perform a series of experiments using real logistics and transportation software systems.

3.1. Selection of logistics and transportation systems for case study

Based on the statistics on architecture types used for transportation and logistics systems in the global ISBSG database we can see that the majority of these applications based on multi-tier architecture.

For the case study we selected open source applications which are the most representative according to our requirements. We take into account the application domain (only logistics and transportation systems), popularity (number of download, ratio), commercial support, etc. As a result, the systems showed in Table 5 were selected.

Table 5. Considered software systems

	Name	URL	Programming language
1	Dolibarr ERP&CRM	http://www.dolibarr.org/	PHP
2	ERPNext	https://erpnext.com/	Python
3	Bookyt	http://bookyt.ch/	Ruby
4	koalixcrm	http://www.koalix.org/	Python
5	Vtiger CRM	https://www.vtiger.com/crm/	PHP
6	Openbravo ERP	http://www.openbravo.com/	Java
7	ADempiere ERP	http://www.adempiere.com/	Java
8	GO Gestionale Open	http://www.gestionaleopen.org/	Delphi
9	Libertya ERP	http://www.libertya.org/	Java
10	favesERP	http://www.faves-erp.com/	PHP

In addition we considered a simulation model for decision-making and risk analysis when releasing a new product on the market [11]. The model is developed in AnyLogic simulation tool using

Java programming language. As long as architecture of this simulation model is different from the client-server, we will assume that we have to reengineer the program using client-server architecture (since this is requirement for our technique). Thus, we considered eleven logistics and transportation software systems.

3.2. Obtaining values of the metrics

As long as we take already existing products we use the conversion table to obtain *FP* values from *LOC* measures [1]. The obtained *FP* values from *LOC* measures showed in Table 6.

Table 6. The values of *LOC* and *FP* metrics

	Name	***LOC***	***FP***
1	Dolibarr ERP&CRM	371947	11623
2	ERPNext	59959	2855
3	Bookyt	11902	566
4	koalixcrm	5166	246
5	Vtiger CRM	284272	8883
6	Openbravo ERP	637924	7974
7	ADempiere ERP	1138181	14227
8	GO Gestionale Open	739622	25504
9	Libertya ERP	1217125	15214
10	favesERP	744134	23254
11	Simulation Model	17548	219

The presence of program source code allows us to obtain the values of *CBO* and *LCOM* for all available classes, which are used to obtain values of *CBOF* and *LCOMF* for each system (Table 7).

Table 7. The obtained values of *CBOF* and *LCOMF*

	Name	***CBOF***	***LCOMF***
1	Dolibarr ERP&CRM	0.10	0.38
2	ERPNext	0.05	0.10
3	Bookyt	0.06	0.23
4	koalixcrm	0.07	0.17
5	Vtiger CRM	0.12	0.61
6	Openbravo ERP	0.08	0.13
7	ADempiere ERP	0.07	0.22
8	GO Gestionale Open	0.02	0.43
9	Libertya ERP	0.09	0.10
10	favesERP	0.10	0.53
11	Simulation Model	0.02	0.16

System characteristics for *FP* metric were determined based on the requirements and existing implementation of the software systems (Table 8).

Table 8. The values of the system characteristics used for *FP* metric

	Dolibarr ERP& CRM	**ERPNext**	**Bookyt**	**koalixcrm**	**Vtiger CRM**	**Openbravo ERP**	**ADempiere ERP**	**GO Gestionale Open**	**Libertya ERP**	**favesERP**	**Simulation Model**
F_1	2	3	5	3	2	3	4	3	3	2	1
F_2	1	2	1	2	3	3	2	2	4	1	0
F_3	2	5	2	2	2	3	4	3	3	1	2
F_4	3	4	2	3	2	3	3	2	4	3	1
F_5	2	3	2	2	3	4	2	3	3	2	0
F_6	2	3	3	2	4	3	2	4	3	3	1
F_7	3	4	2	3	2	4	3	3	4	3	2
F_8	2	2	4	3	3	3	3	4	2	3	3
F_9	1	4	2	2	1	3	4	2	3	1	5
F_{10}	4	3	4	4	3	5	2	3	4	3	2
F_{11}	2	3	2	3	3	3	3	3	4	3	1
F_{12}	2	2	3	3	2	4	4	3	3	3	2
F_{13}	3	2	3	4	3	4	4	3	5	3	1
F_{14}	2	2	4	3	2	3	2	3	3	2	3

3.3. The solution of integer programming problems

Having value of *UFP*, F_i, *CBOF* and *LCOMF* we can obtain the objective functions for each considered system.

For example, the obtained objectives function for Dolibarr ERP/CRM expressed as:

$$W = 1.06x_1 + 0.902x_2 + 1.032x_3 + 0.907x_4 + 0.99x_5 + 1.08x_6 + 0.98x_7 + 0.969x_8 + 0.961x_9 + 0.987x_{10} +$$
$$+ 0.953x_{11} + 1.003x_{12} + 0.998x_{13} + 0.99x_{14} + 1.001x_{15} + 1.011x_{16} + 1.002x_{17} + 1.01x_{18} \rightarrow min.$$

Once the objective function is defined as well as all restrictions, we can find the optimal solution for this integer programming problem. The optimal solution for Dolibarr ERP&CRM system is formed by the following values of the variables: $x_2 = x_8 = x_{11} = x_{13} = x_{14} = x_{15} = 1$. Thus, the optimal pattern suite for our system consists of the following patterns: P_{12}, P_{24}, P_{33}, P_{41}, P_{42}, P_{51}.

In the same way we obtained optimal pattern suites for other systems (Table 9).

Table 9. Optimal pattern suites

Group	Dolibarr ERP&CRM	ERPNext	Bookyt	koalixcrm	Vtiger CRM	Openbravo ERP	ADempiere ERP	GO Gestionale Open	Libertya ERP	favesERP	Simulation Model
1	P_{12}	P_{14}	P_{12}	P_{12}	P_{14}	P_{14}	P_{14}	P_{12}	P_{14}	P_{12}	P_{11}
2	P_{24}	P_{24}	P_{23}	P_{23}	P_{24}	P_{21}	P_{24}	P_{23}	P_{24}	P_{23}	P_{21}
3	P_{33}	P_{32}	P_{33}	P_{33}	P_{31}	P_{31}	P_{31}	P_{33}	P_{32}	P_{31}	P_{34}
4	P_{41}	-	P_{41}	-	P_{41}	P_{41}	P_{41}	-	-	-	-
	P_{42}	P_{42}	P_{42}	P_{42}	P_{42}	P_{42}	P_{42}	P_{42}	P_{42}	P_{42}	P_{42}
5	P_{51}	P_{51}	P_{54}	P_{54}	P_{51}	P_{53}	P_{51}	P_{51}	P_{51}	P_{53}	P_{54}

From these results we can conclude that the most frequently used patterns from the first group are Domain Model and Service Layer. From the second: Active Record and Data Mapper, which is logical since there are restrictions on pattern usage combinations from the first and second groups. In the third group the most frequently used patterns are Model View Controller and Template View. Results showed that the use of Data Transfer Object pattern is reasonable for the most of cases; and Remote Facade pattern is noticeably less used. The most preferred pattern for considered systems from the last group is Optimistic Offline Lock.

3.4. Results and analysis

For the optimal pattern suites (which are obtained as a result of solving integer programming problems) for considered software systems we can obtain pattern-architecture efficiency metric *K′*. Before that we need to find average values of CF_i, P_{FP}, c_{CBOF} and c_{LCOMF}. The results of obtained pattern-architecture efficiency metric *K′* for considered systems are shown in Table 10.

Table 10. The optimal pattern suite impact on pattern-architecture efficiency metric

	Name	*K*	*K′*	Δ*K*=*K*-*K′*	Δ*K*%
1	Dolibarr ERP&CRM	6310	6092	218	3.45%
2	ERPNext	1513	1487	26	1.72%
3	Bookyt	336	356	-20	-5.95%
4	koalixcrm	141	157	-16	-11.35%
5	Vtiger CRM	4625	4495	130	2.81%
6	Openbravo ERP	4334	4346	-12	-0.28%
7	ADempiere ERP	7532	7305	227	3.01%
8	GO Gestionale Open	15992	15086	906	5.67%
9	Libertya ERP	8417	8015	402	4.78%
10	favesERP	14107	13466	641	4.54%
11	Simulation Model	115	118	-3	-2.61%

Figure 1 shows the relation of original architecture efficiency (*K*) and pattern-architecture efficiency metric (*K′*) for all considered systems. Ideally the value of pattern-architecture efficiency metric (*K′*) must be lower than original architecture efficiency (*K*), since the values of coupling and cohesion should became better and the value of *FP* should be reduced.

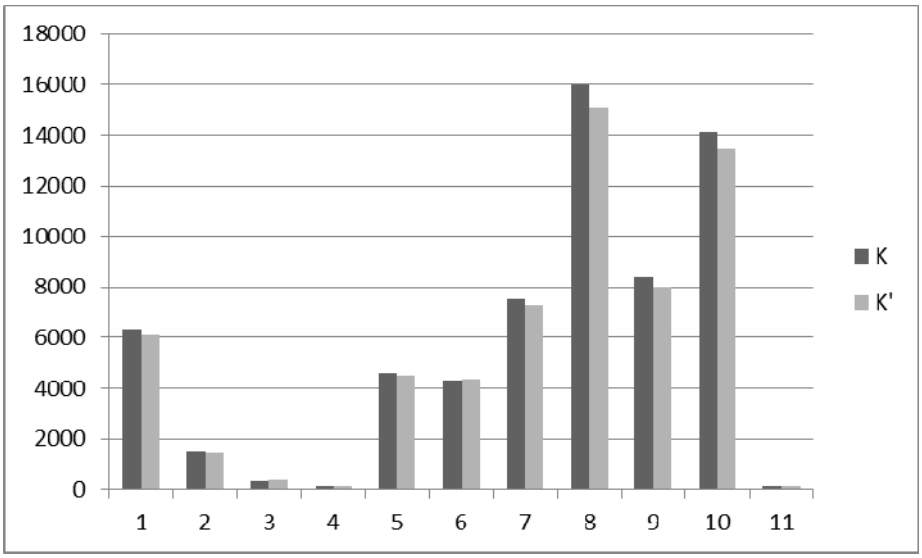

Figure 1. The values of architecture efficiency K and K' for the considered systems

Percentage difference between values of the original architecture efficiency and pattern-architecture efficiency metric is illustrated on Figure 2.

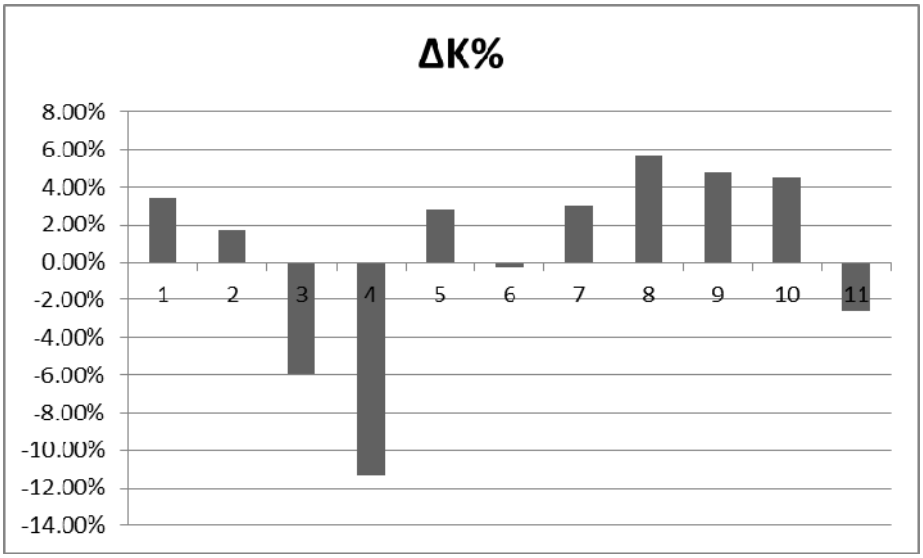

Figure 2. The changes of the architecture efficiency metric in percent

We can see that for lager system the architecture efficiency in improved by 2-5%, in contrast, it's deteriorated for small systems.

The results indicate that the use of optimal pattern suite isn't appropriate for each considered system. It can be noted that for systems where the value of *FP* is smaller than 1000 (i.e. small and medium software systems) the application of the proposed technique doesn't improve the architecture efficiency. The reason is quite simple: the pattern implementation itself requires reasonable effort which is costly for small systems, so such usage for small systems is overkill. For software system Openbravo ERP (number 6) we can conclude that the original figures were already close to the optimal. In addition, it is also possible that expert estimates of some indicators and coefficients aren't calibrated reasonably well.

4. Conclusions

In this paper the technique that allows selecting the optimal suite of architectural patterns for logistics and transportation software is proposed. This selection technique is reduced to the classical problem of integer programming where the optimal solution should be found.

As long as the most of the modern logistics and transportation systems are based on Multi-tier architecture, we've considered a set of patterns that are suitable for its creation. The proposed technique is applied for this set of architectural patterns.

Pattern-architecture efficiency metric is used to measure patterns' numerical impact on a system. This metric is based on functional point (*FP*) metric, which indirectly measures the functional complexity of software. In addition to the complexity metric, inner (cohesion) and outer (coupling) relations are taken into account.

For the case study we've selected eleven logistics and transportation software systems. The objective functions are defined for the case study as well as constrains on the use of specific architectural patterns. The resulting solution reflects the optimal suites of architectural patterns that are suitable for the development of the systems with the specified requirements.

The quantitative study is given to evaluate the changes in the architectural decisions efficiency by applying the selected suite of patterns. The analysis of case study results allowed determining the appropriateness of this technique application, which is dependent on the functional size of the software system.

According to that, the results indicate that the proposed technique is applicable for solving problems of optimal architectural patterns' suite selection when we construct architecture for the large-scale logistics and transportation systems.

References

1. Orlov, S., Tsilker, B. (2012). *Software Engineering: A Textbook for Universities,* 4th Ed. SPb.: Piter. 608 p. (In Russian)
2. Bass, L., Clements, P., Kazman, R. (2003). *Software Architecture in Practice*, 2nd Ed. Addison Wesley. 560 p.
3. Pressman, R.S. (2010). *Software Engineering: A Practitioner's Approach*. 7th Ed. McGraw-Hill. 895 p.
4. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal M. (1996). *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Willey. 476 p.
5. Martin, R.C. (2003). *Agile Software Development: Principles, Patterns and Practices*. Prentice Hall. 552 p.
6. Fowler, M. (2002). *Patterns of Enterprise Application Architecture.* Addison Wesley. 560 p.
7. Sergeev, V., Grigoriev, M., Uvarov, S. (2008). *Logistics. Information systems and technology.* Moscow: Alpha-Press. 608 p. (In Russian)
8. International Function Point Users Group. (1999). *Function Point Counting Practices Manual*, International Function Point Users Group, Release 4.1. Westerville, Ohio. 335 p.
9. Orlov, S. Vishnyakov, A. (2010). *Pattern-oriented decisions for logistics and transport software*. Transport and Telecommunication, 11(4), 46–58.
10. Chidamber, S.R., Kemerer, C.F. (1994). *A Metrics Suite for Object Oriented Design*. IEEE Transactions on Software Engineering, 20(6), 476-493.
11. Faingloz, L., Savrasov, M., Fyodorov, S., Leibenzon, B., Sadovnikova, Y. (2009). Use of simulation modelling for support of decision making for the purpose of introducing new product to the market. In Proceedings of the Conference 'Reliability and Statistics in Transportation and Communication (RelStat'09)', October 21–24, 2009 (pp. 279-286). Riga, Latvia: TTI.