Computer Science and Information Technology 4(4): 147-156, 2016 http://www.hrpub.org
DOI: 10.13189/csit.2016.040403

Multiple Criteria Decision Making for the Structural Organization of Software Architecture

Sergey Orlov, Andrei Vishnyakov*

Transport and Telecommunication Institute, Latvia

Copyright©2016 by authors, all rights reserved. Authors agree that this article remains permanently open access under the terms of the Creative Commons Attribution License 4.0 International License.

Abstract Architectural decisions have a significant impact on the development process as well as on the quality of applied systems. On the other hand, it would be desirable to rely on mature solutions and proven experience when making such decisions. Partially this problem could be solved with the use of architectural patterns. Such solution for the same task can be implemented using different sets of patterns. As a result, there is a problem of choosing and evaluating the software architecture that is build using a number of patterns and that meets the system requirements. In this paper, the technique that allows selecting the optimal software architecture for applied software is proposed. This selection technique is reduced to the criteria importance theory problem. For applying it, we need to pick up a set of metrics that assess the characteristics of the software architecture. Next, we need to determine metrics scale and information about their importance. The results allow us making conclusions about usefulness of the proposed technique during architecture design phase for applied software.

Keywords Multicriteria Decision Analysis, Criteria Importance Theory, Decision Making, Software Architecture, Architectural Pattern, Architecture Metric

1. Introduction

The formation of architecture is the first and fundamental step in the software design process and provides the framework of a software system that can perform the full range of detailed requirements [1, 2].

Most of the existing techniques for constructing a software architecture are not well formalized and are usually not based on any mathematical theory [1]. Therefore, the problem of software architecture selection and analysis based on quantitative evaluation is very important. The analysis of an architecture enables early prediction of a system's qualities. In other words, it would be desirable to have a formalized technique that is based on mathematical theory, and which allows the user to analyse and make decisions when choosing a software architecture or its components.

A number of techniques have been proposed to assist software architects in making architecture decisions [3, 4]. There are several groups of such techniques, where some of them focused on architecture trade-off analysis, quality evaluation model analysis, performance optimization and some others well-known techniques [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15].

Some other studies propose the usage of the criteria of efficiency and the architecture efficiency metrics for quantitative evaluation of a software architecture structure [16, 17]. The disadvantage of this method is that the components of the architecture efficiency metrics are explicitly defined, and we cannot easily extend them to reflect the required software architecture features.

One of the key artefact obtained on the architecture design phase is the structural organization of the architecture, which can be represented using Unified Modeling Language (UML) or Architecture Description Language (ADL) [1, 2, 18].

In this paper, we consider exactly the structural organization of the architecture. In the initial stages of software development there were no standardized approaches for the development of the structural organization of software systems. Those that existed were too much generalized and not very accurate. In consequence of that the development of such systems it is required much more time and financial resources. Usually, highly skilled professionals were required for such development. To reduce the development costs the more formalized techniques began to emerge. This work resulted in the appearance of software design patterns [1]. The patterns later became more complex and some evolved to architectural patterns. Nowadays there are plenty of different design patterns. With the help of these patterns you can build a variety of different architecture alternative, and the question then is how to make a choice among these alternatives. To answer this question there have been proposed different approaches that allows to make such decisions. Unfortunately, most of existing techniques does not well examine architectures' structural organization.

Usually they based solely on expert evaluations [3, 4].

In this paper, we propose a technique that allows us to make architectural decisions when creating structure of a software architecture using set of architectural patterns. In other words, this technique allows us to choose the best structural organization of the software architecture that is build using the architectural patterns.

The proposed technique is based on so-called criteria importance theory [19, 20]. It allows decisions to be made when choosing a software architecture system from among several alternatives and lacks the disadvantages that exists with other methods.

1.1. Selection of Optimal Architecture

The software architecture is the structure of the system, which comprise software components, the externally visible properties of those components, and the relationships among them [1]. The software architecture is a complex design artefact.

It's a complicated task to make a validation of software architecture at early design stage and it's much easier to rely on tried and tested approaches for solving certain classes of problems. One of the approaches is to use architectural patterns. An architectural pattern, expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them [1, 2, 18]. The architectural patterns improve partitioning and promote design reuse by providing solutions to frequently recurring problems. They allow reducing a risk by reusing successful designs with known engineering attributes.

Creating a software architecture based on architectural patterns we need to bear in mind that different architectural patterns are focused on different areas and solve different problems. That is why they might be categorized in several groups. There are several approaches of architectural patterns classification [17, 18, 21, 22, 23, 24, 25]. Usually the software isn't limited to a single architectural pattern. It is often a combination of architectural patterns that make up a complete system. For instance, there might be SOA based architecture where some services are designed using layered or *N*-tier architecture approach [18, 22].

Software architectural patterns themselves won't help us to satisfy the requirement, to do so they need to be organized in structures; that is, we need to identify and connect elements that are derived from the selected design concepts. Such structure of architectural patterns created a reference architecture. Reference architecture is blueprints that provide an overall logical structure for particular types of applications. A reference architecture is a reference model mapped onto one or more architectural patterns [1, 18, 22, 25].

During the software architecture design stage there could be obtained several different reference architectures, built using architectural patterns. In this case, there is a problem of comparison and choose the best architecture that is best satisfying the software requirements. In this paper, we use a model based on criteria importance theory [19] for selecting the optimal software architecture.

To apply this technique, we need to perform a number of steps represented on Fig 1.

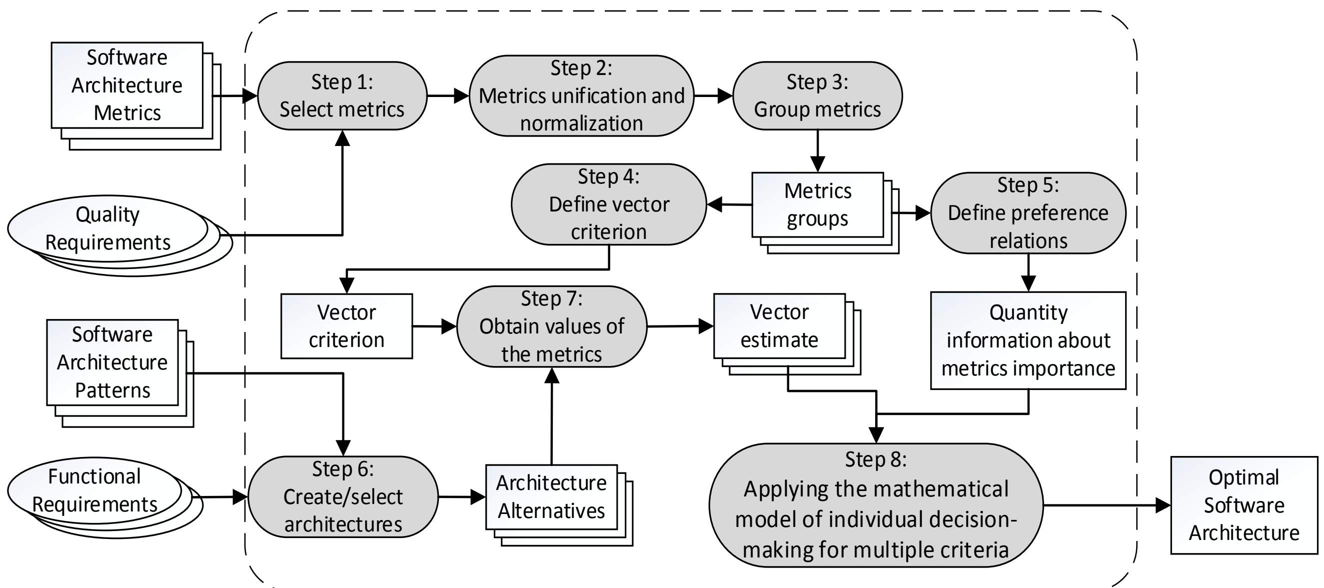

Figure 1. Selection of optimal architecture

The detailed process of applying the technique is as following:

1. Select software architecture metrics that assess the characteristics of the software architecture. In our case, we define our own metrics as the available architecture metrics are very limited and not easy to apply for component structure of a software architecture [2, 12, 16, 17, 26, 27];
2. Determine the metrics scale, metrics unification and normalization;
3. Group the metrics into several metric groups;
4. Determine vector criterion;
5. Define information regarding the metrics importance and define preference relations for the metric groups;
6. Identify and create potential software architecture candidates;
7. Obtain the software metrics values for the selected software architecture alternatives;
8. Select optimal software architecture by applying the mathematical model based on criteria importance theory.

For our case study we use the minimum number of required metrics. At the same time, we rely on the classical approach, which is adopted in software engineering theory for assessing the quality of project design, including size (complexity), coupling and cohesion [2]. Moreover, we consider the functional complexity that is measured using a modification of the well-known Function Point metric by Albrecht [28]. Also, we have a bit extended the number of metrics by the well-known quality characteristics from ISO/IEC 25023:2016 [29]. The resulting set of metrics can be slightly expanded, since it doesn't change the essence of the technique in general.

1.2. Model Definition

The mathematical model of individual decision-making for multiple criteria includes the following components [19, 20]:

- set of alternatives X;
- vector criterion f;
- preference and indifference relations of the decision maker (DM), which are denoted as P (preference) and I (indifference).

Each alternative x from the set of alternatives X is characterized by a number of criteria f_i, $i = 1, ..., m$, which are called *particular criteria*. The ordered set of such criteria forms a *vector criterion* $f = (f_1, \ldots, f_m)$. The *criterion* f_i is the function defined on X and taking its values from z_i, which is called a common scale (or number of assessments of such criterion). According to the standard software engineering terminology we call particular criteria metrics.

Assume that we have a number of different software architecture options. For example, the architecture of some transport system can be implemented based on a service-oriented pattern; an alternative architecture could be based on a Multitier pattern; another with the use of a micro services architectural pattern and so on. We define a set of such alternative software architectures as $X = \{X_i | i = 1, \ldots, n\}$. Let us assume that all alternative metrics are homogeneous, i.e. they are all measured using the same scale and have the same range defined as Z_0. Suppose that the number of scale gradations is finite, then: $Z_0 = \{1, \ldots, q\}$, where $q > 1$.

In other words, each metric f_i from the set of alternative software architectures X can take the values from the set of scale gradations Z_0. Assume that all estimates are expressed in numerical form and higher values are preferable to smaller ones. Thus, each software architecture alternative X_i is characterized by values $f_i(X_i)$ of every metric and forms its *vector estimate* $y = f(X_i) = (f_1(X_i), ..., f_m(X_i))$. Alternative software architectures are compared by comparing their vector estimates. The set of all possible vector estimates is defined as $Z = Z_0^m$.

In accordance with the importance of the software architectural characteristics, we split metrics on l groups:

metrics within the group are equally important;

metrics from different groups have different importance.

The fact that the metric f_i is *equally important* to metric f_j is denoted as $f_i \approx f_j$. A vector estimate that includes such metrics has indifferent preference: $y\ I\ y^{ij}$, where y^{ij} — vector estimate, which is obtained from vector y by replacing its components y_i and y_j.

The fact that metric f_i from one group is *more important* than metric f_j from another group is denoted as $f_i \succ f_j$. In that case, for the pair of vector estimates y and y^{ij}, the DM prefers the first one to the second one, and it is denoted as $y\ P^0 y^{ij}$. In addition to this, it is necessary to be able to quantitatively indicate for how many times the metrics from one group are more important than the metrics from the other group. To do so, we define the matrix of degrees of importance superiority $H = \|h_{ij}\|$, where $i, j = 1, ..., m$. When using it the preference relation is written as follows: $f_i \succ^{h^{ij}} f_j$, which means that metric f_i is h^{ij} times as important as metric f_j.

The set of relations as $f_i \approx f_j$ and $f_i \succ f_j$ for the metrics used forms the qualitative information about the metrics' importance Ω. On the contrary, the set of relations as $f_i \succ^{h^{ij}} f_j$ forms the quantitative information about the metrics' importance Θ.

1.3. Metric Selection

For the formation of the vector criteria, it is necessary to define a set of metrics that will be used. To select the metrics, we can use the technique of selection of the metric suite [26].

During the architecture design stage, the architectural patterns are treated as a "black box", so we could make some indirect measures of system characteristics. One of the most

suitable metric for such measures is function 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 [2, 26, 27, 28].

In addition to the complexity metrics, outer (Coupling) and inner (Cohesion) relations in architectural patterns can be measured. Coupling is a metrics that express the degree of data interconnection between modules. Low coupling is an indication of a well-designed system. Cohesion is a measure of how strongly the functionality inside a module is related. High cohesion of a module is a desirable trait [2, 27].

In addition to these metrics, we need to measure the architecture quality characteristics [27] such as complexity, responsibility, scalability, maturity, reusability, extensibility, replaceability, supportability, performability, interoperability and suitability.

For a comprehensive evaluation of the architecture, we combine the above listed metrics into three the following groups:

- Architecture size metrics;
- Architecture links metrics;
- Architecture's quality characteristics.

Since most of available software metrics are not applicable for the software architecture, we need to define such metrics first.

1.4. Architecture Size Metrics

For architecture size metrics, we consider the metrics that directly or indirectly measure the size of a software system.

1.4.1. Architecture Function Points (AFP)

One of the metrics which indirectly measure the software complexity is Function Points metric [2, 26, 28]. The input data for this metric are obtained using system requirements. Since the application of each architectural pattern has its own features, impacts and requires a different amount of effort, it is advisable to consider the software architecture impact for each separate component.

For Function Points metric we need to get Unadjusted Function Point count, which includes number of external inputs, external outputs, external inquiries, internal logical files, and external interface files [28]. Next, to determine the Degree of Influence we need to evaluate each of the 14 general system characteristics [28]. Examples of such general system characteristics include data communications, distributed data processing, performance, reusability, etc. The resulting degree of influence must be in the range of 0.65 to 1.35 [28]. After multiplying the Unadjusted Function Point count by the Degree of Influence we get the adjusted Function Point count.

For our calculations we must also consider the impact of patterns on the system's characteristics. Thus, the metric is defined as follows:

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

where

UFP – Unadjusted Function Point count;

CF_i – defined as follows:

$$CF_i = \begin{cases} 5, & if \ \ c_i \times F_i > 5; \\ round(c_i \times F_i), & otherwise, \end{cases} \quad (2)$$

where

F_i — degree of influence coefficient from *FP* metric;

c_i — influence of architectural pattern on the *i*-th system's characteristic.

To get the 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 actualizes a system characteristic;

5 — use of a pattern actualizes a system characteristic.

Next, these values are converted into c_i using the scale conversion rule presented in Table 1.

Table 1. Characteristics evaluation scale correspondence to c_i value

Score As	c_i
1	½
2	⅔
3	1
4	1½
5	2

1.5. Architecture Links Metrics

For software architecture links, we consider the strength of the relationship between the architectural components and inside each individual component. Roughly, this is similar to the relationship between classes in Object Oriented Programming, although here we evaluating software architecture components. For such evaluation, we define two new metrics, which are based on Software Package Metrics [30]. One of the metrics is used to measure the coupling, while the other is used to measure a cohesion.

1.5.1. Architecture Instability (AI)

Adaptation of the original instability metric [27, 30] for architectural functional components allows us to define the following instability of the architecture's component metric:

$$CI = \frac{C_e}{C_a + C_e}, \quad (3)$$

where

C_e — efferent coupling (outgoing dependencies). The number of classes inside this component that depend on classes outside this component. C_e is a form of the fan-in of a component.

C_a — afferent coupling (incoming dependencies). The number of classes outside this component that depend on

classes within this component. C_a is really the fan-out of a component.

If there are no outgoing dependencies, then *CI* will be zero and the component is stable. If there are no incoming dependencies, then *CI* will be one and the component is unstable.

Having the instability metric, let us define an Architecture Instability metric as follows:

$$AI = \frac{\sum_{i=1}^{N} CI_i}{N}, \quad (4)$$

where

CI_i — instability of the *i*-th architecture's component;

N — number of components of the architecture.

1.5.2. Architecture Relational Cohesion (ARC)

Adaptation of the Package Relational Cohesion metric [27, 30] for architectural functional components allows us to define the following relational cohesion of the architecture's component metric:

$$RC = \frac{R+1}{NPR}, \quad (5)$$

where

R — the number of relations between classes and interfaces in the component;

NPR — the number of possible relations between classes and interfaces in the component.

Having the relational cohesion metric for the architecture's component, we define the Architecture Relational Cohesion metric as follows:

$$ARC = \frac{\sum_{i=1}^{N} RC_i}{N}, \quad (6)$$

where

RC_i — relational cohesion of the *i*-th architecture's component;

N — number of components of the architecture.

Table 2. Architecture's quality characteristics

Name	Description	Measurement function
Complexity	What proportion of components in the software architecture have simple structure.	*X = A / B,* *A* — Number of simple components in the software architecture; *B* — Total number of components in the software architecture
Responsibility	What proportion of components in the software architecture have a single responsibility.	*X = A / B,* *A* — Number of components in the software architecture with single responsibility; *B* —Total number of components in the software architecture
Scalability	What proportion of components in the software architecture what could be easily scaled.	*X = A / B,* *A* — Number of components in the software architecture that could be scaled; *B* — Total number of components in the software architecture
Maturity	What proportion of components in the software architecture could be implemented with the reuse of known patterns.	*X = A / B,* *A* — Number of components in the software architecture that reuse any software patterns; *B* — Total number of components in the software architecture
Reusability	What proportion of components in the software architecture could be reused in the other architectures.	*X = A / B,* *A* — Number of components in the software architecture that can be reused; *B* — Total number of components in the software architecture
Extensibility	What proportion of components in the software architecture could be potentially extended.	*X = A / B,* *A* — Number of components in the software architecture that could be extended; *B* — Total number of components in the software architecture
Replaceability	What proportion of components in the software architecture could be easily replaced.	*X = A / B,* *A* — Number of components in the software architecture that could be easily replaced; *B* — Total number of components in the software architecture
Supportability	What proportion of components in the software architecture is well supported by frameworks and libraries.	*X = A / B,* *A* — Number of components in the software architecture that is well supported by frameworks and libraries; *B* — Total number of components in the software architecture
Performability	What proportion of components in the software architecture is well suitable to handle large amount of data.	*X = A / B,* *A* — Number of components in the software architecture that is well suitable to handle large amount of data; *B* — Total number of components in the software architecture
Interoperability	What proportion of components in the software architecture is easy to integrate with other systems.	*X = A / B,* *A* — Number of components in the software architecture that is easy to integrate with other systems; *B* — Total number of components in the software architecture
Suitability	What proportion of components in the software architecture is well fitted for the required domain.	*X = A / B,* *A* — Number of components in the software architecture that is well fitted for the required domain; *B* — Total number of components in the software architecture

1.6. Architecture's Quality Characteristics

For the third group of the metrics let us choose the metrics that evaluate the architecture's quality characteristics. Metrics from this group are based on the international standard "Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality" [27, 29]. The group includes the metrics listed in the table 2.

The scale for evaluation quality characteristics we define in range from 0 to 1. Where low value means that this characteristic is poorly supported by the considered architecture; high value of the characteristic indicates the opposite.

1.7. Uniformity and Normalization of Metrics

The selected metrics allow us to obtain a vector criterion for software architecture evaluation. Table 3 lists the metrics with their scale.

Table 3. Metrics and corresponding notation from vector criteria

Metric	Metric notation	Scale	Preferable value
Architecture Function Points	f_1	[1..+∞]	Low
Architecture Instability	f_2	[0..1]	Low
Architecture Relational Cohesion	f_3	[0..1]	High
Complexity	f_4	[0..1]	High
Responsibility	f_5	[0..1]	High
Scalability	f_6	[0..1]	High
Maturity	f_7	[0..1]	High
Reusability	f_8	[0..1]	High
Extensibility	f_9	[0..1]	High
Replaceability	f_{10}	[0..1]	High
Supportability	f_{11}	[0..1]	High
Performability	f_{12}	[0..1]	High
Interoperability	f_{13}	[0..1]	High
Suitability	f_{14}	[0..1]	High

As long as for two metrics the lower values are preferable, it is necessary to overcome this contradiction. To do that we replace the Architecture Function Points and Architecture Instability metrics by the alternatives which are listed below.

1.7.1. Inverted Architecture Function Points *(*AFP*')*

The Architecture Function Points metric must be normalized, and its value must be proportional to the preference. Therefore, we define the metric Inverted Architecture Function Points as:

$$AFP'_i = 1 - \frac{AFP_i}{AFP_{\max}}, \tag{7}$$

where

AFP_i — Architecture Function Points for the i-th architecture,

$AFP_{\max}$ — maximum value of AFP for compared architectures.

1.7.2. Architecture Stability *(*AS*)*

The original Architecture Instability metric is replaced by the Architecture Stability metric, which is determined as:

$$AS = 1 - AI, \tag{8}$$

where

AI — Architecture Instability for the architecture.

1.8. Scale

To apply the criteria importance theory model, we need to convert the linear scale for all metrics to an ordinal scale, where values are distributed from 1 to 10. Thus, all the metrics will have a common scale and higher values are preferable to lower ones.

To obtain the rank value we need to use the normalized value of the metric and convert it using the following definition:

$$r = ROUND(n \times (\text{Rank}_{MAX} - \text{Rank}_{MIN})) + \text{Rank}_{MIN}, \tag{9}$$

where

n – normalized value of the metric;

RankMax – maximum rank, in our case 10;

RankMin – minimum rank, in our case 1.

1.9. Preference Relations

Let us determine the preference relations for metric groups and individual metrics. First, we describe the relationship of the metrics in each group.

As mentioned above, metrics in each group has indifferent preference. Hence, it follows that for the second group the relations are:

$$f_2 \approx f_3. \tag{10}$$

Metrics from the third group are equally important as well:

$$f_4 \approx f_5 \approx f_6 \approx f_7 \approx f_8 \approx f_9 \approx f_{10} \approx f_{11} \approx f_{12} \approx f_{13} \approx f_{14}. \tag{11}$$

The next step is to determine the relationship between the three groups of metrics.

Since all metrics in every group are equally important, it's enough to determine only one relation between a metric from one group and any metric from another group.

First, let us define the relationship between the metrics from the first and second groups, which can be denoted as:

$$f_1 \succ f_2 . \quad (12)$$

Next, we define the relationship between the metrics from the second and third groups, as well as relations between metrics from the first and the third groups:

$$f_3 \succ f_4, \; f_1 \succ f_4 . \quad (13)$$

After the qualitative information about metrics importance is defined, we can determine for how many times the metrics from group one are more important than the metrics from group two, etc. To do so, we define the matrix of degrees of importance superiority *H*. With the expert evaluation, determine that the metrics from the group one twice more important than the metrics from the group two; and that the metrics from the group two twice more important than the metrics from the group three. Thus, preference relations take the following form:

$$f_1 \succ^2 f_2 \text{ and } f_3 \succ^2 f_4 . \quad (14)$$

The resulting relations forms the quantitative information about metrics importance Θ.

2. Case Study

To illustrate the example of usage of the proposed model, we conducted a case study for which we built several software architectures using different architectural patterns.

2.1. Selection of Optimal Architecture for Airline Business Case Study

Let us consider four different architectures for airline reservation system. The system should have information regarding planes, their departure and arrival times, source and destination. The system should have passenger details. The software should provide search options and checking for ticket availability. The user should be able to make and cancel a reservation. In addition, user should be able to check-in for the flight and select meal preferences. The domain model for that architecture is represented in Fig. 2.

From all possible solutions, we are considering three different approaches for creation of the architecture for such system. All these options are based on different architectural patterns that provides various capabilities and introduces specific requirements. The considered solutions include:

A_1 — Architecture that is based on a Service Oriented Architecture (SOA) architectural pattern;

A_2 — Architecture that is based on a Multitier architectural pattern;

A_3 — Architecture that is based on a Database-centric architectural pattern;

A_4 — Architecture that is based on a micro services architectural pattern.

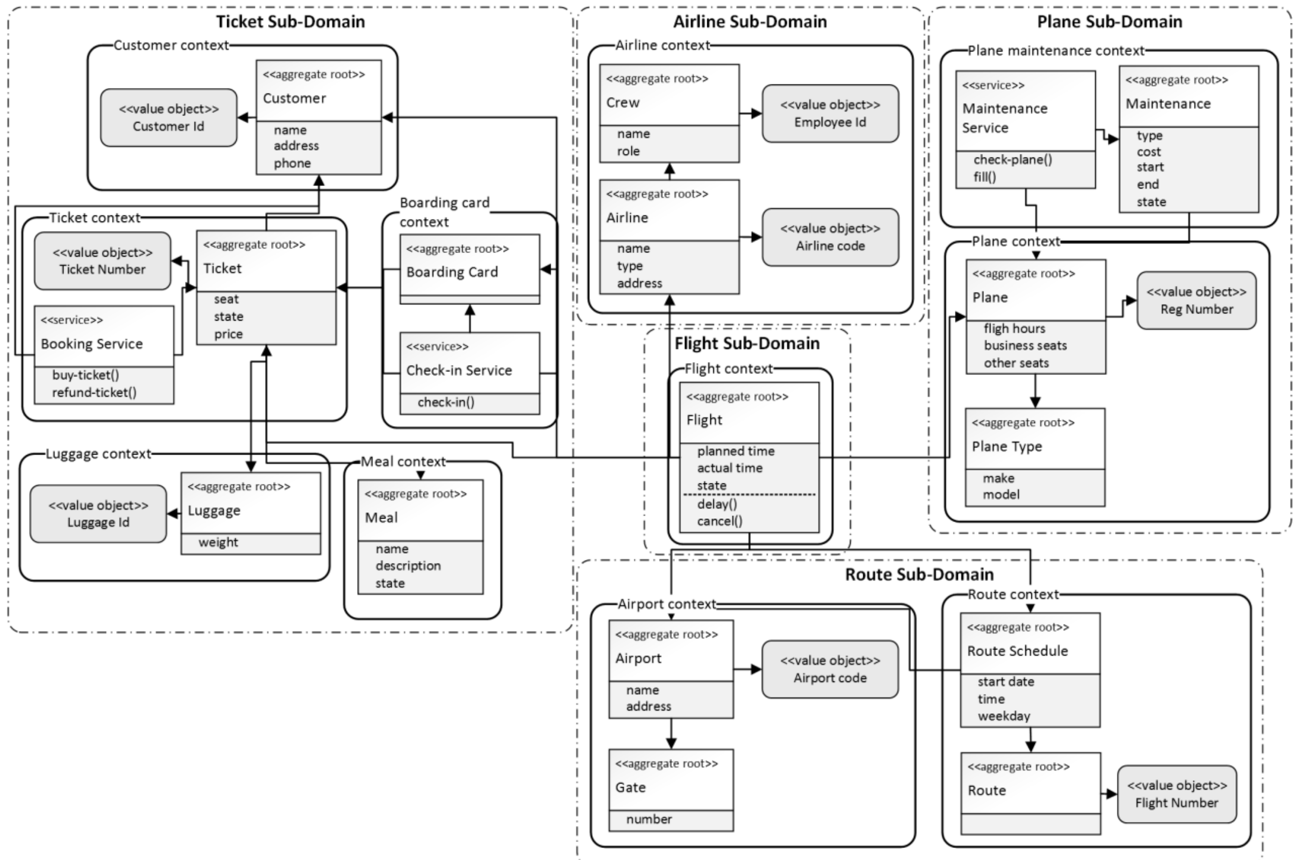

Figure 2. Domain model of airline system

2.2. Obtaining Values of the Metrics

It is necessary to obtain values for the selected metrics for the given software architectural solutions. Metrics f_1, f_2 and f_3 are evaluated based on the created software architectural models; and metrics f_4 – f_{14} are obtained using expert evaluation.

The obtained metric values are given in table 4.

Table 4. Obtained values of metrics

Notation	Metric	A_1	A_2	A_3	A_4
f_1	AFP	1200	1100	950	800
f_2	AI	0.42	0.38	0.48	0.45
f_3	ARC	0.69	0.52	0.48	0.63
f_4	Complexity	0.68	0.53	0.64	0.45
f_5	Responsibility	0.78	0.64	0.41	0.85
f_6	Scalability	0.73	0.65	0.38	0.45
f_7	Maturity	0.65	0.78	0.73	0.45
f_8	Reusability	0.73	0.61	0.48	0.65
f_9	Extensibility	0.84	0.64	0.53	0.75
f_{10}	Replaceability	0.77	0.67	0.44	0.71
f_{11}	Supportability	0.68	0.71	0.43	0.52
f_{12}	Performability	0.57	0.67	0.65	0.48
f_{13}	Interoperability	0.76	0.52	0.57	0.72
f_{14}	Suitability	0.62	0.67	0.74	0.65

It is advisable to convert the obtained values to a common ordinal scale. The metric values after conversion are shown in table 5.

Table 5. Metric values after conversion to common ordinal scale

Notation	Metric	A_1	A_2	A_3	A_4
f_1	AFP′	1	2	3	4
f_2	AS	6	7	6	6
f_3	ARC	8	6	5	7
f_4	Complexity	7	6	7	5
f_5	Responsibility	8	7	5	9
f_6	Scalability	8	7	4	5
f_7	Maturity	7	8	8	5
f_8	Reusability	8	6	5	7
f_9	Extensibility	9	7	6	8
f_{10}	Replaceability	8	7	5	7
f_{11}	Supportability	7	7	5	6
f_{12}	Performability	6	7	7	5
f_{13}	Interoperability	8	6	6	7
f_{14}	Suitability	7	7	8	7

2.3. The Solution Using Criteria Importance Theory

As a result, we have obtained four the following vector estimates:

$$y1 = (1, 6, 8, 7, 8, 8, 7, 8, 9, 8, 7, 6, 8, 7), \quad (15)$$

$$y2 = (2, 7, 6, 6, 7, 7, 8, 6, 7, 7, 7, 7, 6, 7), \quad (16)$$

$$y3 = (3, 6, 5, 7, 5, 4, 8, 5, 6, 5, 5, 7, 6, 8), \quad (17)$$

$$y4 = (4, 6, 7, 5, 9, 5, 5, 7, 8, 7, 6, 5, 7, 7). \quad (18)$$

In addition, the following set of indifferent preference is defined

$$f_2 \approx f_3, \quad (19)$$

$$f_4 \approx f_5 \approx f_6 \approx f_7 \approx f_8 \approx f_9 \approx f_{10} \approx f_{11} \approx f_{12} \approx f_{13} \approx f_{14}, \quad (20)$$

and two preference relations

$$f_1 \succ^2 f_2 \text{ and } f_3 \succ^2 f_4. \quad (21)$$

To transform the problem where the particular criteria from different groups are not equally important to the problem where all criteria are equally important, we need to create an N-model from criteria importance theory [19, 20]. For that purpose, the preference relations and number of the criteria in the group should be taken into consideration.

Given to preference rules we obtain the following N-model:

$$y^N = (f_1,f_1,f_1,f_1;f_2,f_2,f_3,f_3;f_4,f_5,f_6,f_7,f_8,f_9,f_{10},f_{11},f_{12},f_{13},f_{14}). \quad (22)$$

Thus we obtain the following N-estimates:

$$y^N{}_1 = (1, 1, 1, 1, 6, 6, 8, 8, 7, 8, 8, 7, 8, 9, 8, 7, 6, 8, 7), \quad (23)$$

$$y^N{}_2 = (2, 2, 2, 2, 7, 7, 6, 6, 6, 7, 7, 8, 6, 7, 7, 7, 7, 6, 7), \quad (24)$$

$$y^N{}_3 = (3, 3, 3, 3, 6, 6, 5, 5, 7, 5, 4, 8, 5, 6, 5, 5, 7, 6, 8), \quad (25)$$

$$y^N{}_4 = (4, 4, 4, 4, 6, 6, 7, 7, 5, 9, 5, 5, 7, 8, 7, 6, 5, 7, 7). \quad (26)$$

By substituting the above-listed data into the proposed decision-making model, we get the result that A_1 is non-dominated; A_2, A_3 and A_4 are dominated by A_1; A_2 and A_3 are dominated by A_4; A_3 is dominated by A_4.

3. Conclusions

The paper proposes a technique that allows selection of the optimal software architecture for applied software is proposed. This selection technique is reduced to the criteria importance theory problem.

We defined a model based on criteria importance theory that could be used to choose the optimal software architecture. To apply the model, picked up a set of metrics that assess the characteristics of a software architecture; determined the scale and information about the metrics importance.

For the formation of the vector criterion, we defined a set of metrics that includes metrics from three different groups, some of which asses the size, links and different quality characteristics of the software architecture.

To validate the proposed model, we conducted a case study for which we built three software architectures using different architectural patterns. For every software architecture alternative, the metrics values were evaluated and the corresponding vector criterions obtained. The result of the case study proved the correctness of the proposed model, which is based on the criteria importance theory. During the case study, we identified the optimal software architecture for the transport system with the specified requirements.

The results obtained indicate that the proposed technique is applicable for solving problems of the selection of optimal software architecture for applied systems.

As the next step of the research, we are planning to undertake extension of the proposed technique for case of hierarchical criteria.

REFERENCES

[1] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 3rd ed. New Jersey: Addison Wesley. 608 p, 2013.

[2] S. Orlov, Software Engineering. Technologies of Software Development: A Textbook for Universities, 5th ed. St. Petersburg: Piter. 641 p, 2015.

[3] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, I. Meedeniya, Software Architecture Optimization Methods: A Systematic Literature Review, IEEE Transactions on Software Engineering, v.39 n.5, p.658-683, May 2013

[4] D. Falessi, G. Cantone, R. Kazman, P. Kruchten, Decision-making techniques for software architecture design: A comparative survey, ACM Computing Surveys (CSUR), v.43 n.4, p.1-28, October 2011

[5] A. Lozano-Tello, A. Gomez-Perez, Baremo: how to choose the appropriate software component using the analytic hierarchy process. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, pp. 781–788 (2002)

[6] S. Vijayalakshmi, G. Zayaraz and V. Vijayalakshmi. 2010. Multicriteria Decision Analysis Method for Evaluation of Software Architectures, International Journal of Computer Applications, Vol. 1, No. 25, 22-27.

[7] L. Grunske, Identifying "good" architectural design alternatives with multi-objective optimization strategies, Proceedings of the 28th international conference on Software engineering, May 20-28, 2006, Shanghai, China

[8] R. Li, R. Etemaadi, M.T.M. Emmerich and M.R.V. Chaudron, An Evolutionary Multiobjective Optimization Approach to Component-Based Software Architecture Design, Proc. IEEE Congress Evolutionary Computation, pp. 432-439, 2011.

[9] O. Raiha, K. Koskimies and E. Makinen, Scenario-Based Genetic Synthesis of Software Architecture, Proc. Fourth Int'l Conf. Software Eng. Advances, pp. 437-445, 2009.

[10] C. Serban, A. Vescan and H.F. Pop,A New Component Selection Algorithm Based on Metrics and Fuzzy Clustering Analysis, Proc. Fourth Int'l Conf. Hybrid Artificial Intelligence Systems, pp. 621-628, 2009.

[11] H.A. Taboada, F. Baheranwala, D.W. Coit and N. Wattanapongsakorn, Practical Solutions for Multi-Objective Optimization: An Application to System Reliability Design Problems, Reliability Eng. &, System Safety, vol. 92, no. 3, pp. 314-322, 2007.

[12] A. Vescan, A Metrics-Based Evolutionary Approach for the Component Selection Problem, Proc. 11th Int'l Conf. Computer Modelling and Simulation, pp. 83-88, 2009.

[13] S.A. Wadekar and S.S. Gokhale, Exploring Cost and Reliability Tradeoffs in Architectural Alternatives Using a Genetic Algorithm, Proc. 10th Int'l Symp. Software Reliability Eng., pp. 104-114, 1999.

[14] A. Aleti, S. Bjornander, L. Grunske and I. Meedeniya, Archeopterix: An Extendable Tool for Architecture Optimization of AADL Models, Proc. ICSE 2009 Workshop Model-Based Methodologies for Pervasive and Embedded Software, pp. 61-71, 2009.

[15] S.S. Gokhale, Software Application Design Based on Architecture, Reliability and Cost, Proc. Int'l Symp. Computers and Comm., pp. 1098-1103, 2004.

[16] S. Orlov, A. Vishnyakov, Pattern-oriented decisions for logistics and transport software. Transport and Telecommunication, 11(4), 46–58, 2010.

[17] S. Orlov, A. Vishnyakov, Pattern-oriented architecture design of software for logistics and transport applications. Transport and Telecommunication, 15(1), 27–41, 2014.

[18] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture Volume 1: A System of Patterns. Willey, 1996. 476 p.

[19] V. Podinovski, Criteria importance theory. Mathematical Social Sciences. Vol. 27. P. 237–252, 1994.

[20] V. Podinovski, O. Podinovskaya, Criteria importance theory for decision making problems with a hierarchical criterion structure: Working paper WP7/2014/04. Moscow: Publishing House of the Higher School of Economics, 28 p, 2014.

[21] Microsoft Patterns & Practices Team. Microsoft Application Architecture Guide 2nd Edition (Patterns & Practices). Microsoft Press, 2009. 560 p.

[22] M. Fowler, Patterns of Enterprise Application Architecture. Boston: Addison Wesley. 560 p, 2002.

[23] G. Hohpe, B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston: Addison Wesley. 736 p, 2003.

[24] R. N. Taylor, N. Medvidovic, M. Dashofy, Software Architecture: Foundations, Theory, and Practice. New York: John Wiley and Sons. 736 p, 2009.

[25] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison Wesley. 320 p, 2003.

[26] A. Vishnyakov, S. Orlov, Software Architecture and Detailed

Design Evaluation, Procedia Computer Science Volume: 43 Pages: 41-52, 2015.

[27] N. Fenton, J. Bieman, Software Metrics: A Rigorous and Practical Approach, Third Edition. CRC Press. 617 p, 2014.

[28] International Function Point Users Group, Function Point Counting Practices Manual, International Function Point Users Group, Release 4.1, Westerville, Ohio, 1999. 335 p.

[29] ISO/IEC, ISO/IEC 25023:2016 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality. International Organization for Standardization, 47 p, 2016.

[30] R.C. Martin, Agile Software Development: Principles, Patterns and Practices. Prentice Hall. 552 p, 2003.