Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 43 (2015) 41 – 52

Procedia Computer Science

ICTE in Regional Development, December 2014, Valmiera, Latvia

Software Architecture and Detailed Design Evaluation

Andrei Vishnyakov[a]*, Sergey Orlov[a]

[a]*Transport and Telecommunication Institute, Lomonosova str.1, Riga, LV-1019, Latvia*

Abstract

Software design and estimation play the key role for software development process. Different methods are used for architecture design and detailed design evaluation. For architectural design stage a technique that allows selecting and evaluating suite of architectural patterns is proposed. It allows us to consistently evaluate the impact of specific patterns to software characteristics with a given functionality. Also the criterion of efficiency metric is proposed which helps us to evaluate architectural patterns for specified software. During detailed design stage we are interested in the selection of the optimal metric suits which takes into account the characteristics of required system. The proposed technique contains a number a steps where at each step a specific criterion should be used to make a selection from the available metric suites. In the end we can perform the selected metric suite improvement.

Peer-review under responsibility of the Sociotechnical Systems Engineering Institute of Vidzeme University of Applied Sciences

Keywords: Architectural patterns; Architecture efficiency; Metric suite selection; Functional points; Coupling and cohesion

1. Introduction

Software design and estimation play the key role for software development process. The mistakes which are introduced at this development stage are expensive and can lead to the project failure. Therefore it is necessary to put reasonable effort while evaluating artefacts created on this step. The software design stage consists of two steps where the first step is a software architecture design, and the following step is a detailed design. Each of these steps is important, therefore there must be some techniques allowing quality evaluation on each step. Also we need to have some techniques which allow making a design decisions based on some numerical metrics.

* Corresponding author.
E-mail address: andrei.vishnyakov@gmail.com

Peer-review under responsibility of the Sociotechnical Systems Engineering Institute of Vidzeme University of Applied Sciences
doi:10.1016/j.procs.2014.12.007

This paper describes four the following techniques that help to improve the quality during software design:

- Usage of the criterion of efficiency for choosing architectural patterns;
- Selection and evaluation of an architectural patterns suite;
- Selection of an metric suite for detailed design stage;
- Metric suite optimisation.

2. Architecture patterns quality estimation

As long as software architecture design and estimation play the key role for software development process we need to have some metrics for architecture quality estimation. Unfortunately, at the moment there are no any effective methods for the software architecture quality estimation. Despite this we can evaluate the software architecture from the structural organization point of view, so it's obviously that the major part of architectures use some set of the common architectural patterns. The architectural patterns express a fundamental structural organization of software systems and software behaviour. They influences to its base characteristics as well. Such usage of the proven and tested approaches allows increasing the software quality and reducing a budget and potential risks.

2.1. Software architecture and patterns

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. 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[2]. It's a complicated task to make a validation at early design stage and it's much easier to rely on tried and tested approaches for solving certain classes of problems. So 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. Different architectural patterns are focused on some areas and that is why they might be categorized in several groups. There are several approaches of architectural patterns classification. 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 Service Oriented Architecture (SOA) pattern based architecture where some services are designed using layered or Multi-tier architecture approach.

2.2. Criterion of efficiency for choosing architectural patterns

At the architecture design stage the designer may choose from a certain number of available architectural patterns. There are several groups of patterns, which are separated by the object domain. Some architectural patterns from one group can be used as part of other patterns. For example, Presentation tier from Multi-tier architecture can be implemented using MVC Pattern. Another example, some services in SOA might be implemented using Multi-tier architecture. It's necessary to have a tool that indicates the optimal architectural pattern in the specified group or even in different groups, where such is possible. For such purposes the criterion of efficiency can be used, which allows to select the optimal solution from proposed alternatives. The efficiency indicators should be selected for criterion of efficiency construction as well as ratio between them, which determines the type of criterion of efficiency[3].

During the architecture design stage the architectural patterns are treated as a "black box", therefore the complexity metrics should be based on indirect measures instead of directs ones. Functional point (FP) metric is an example of such complexity metrics. It measures the amount of functionality in a system[4]. In addition to the complexity metrics, outer (Coupling) and inner (Cohesion) relations in architectural patterns can be measured. Coupling is a metrics which 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[4]. It is obvious that the rate of Coupling should be minimized and the high value of Cohesion is desirable. Since the value of FP is proportional to the functional complexity, the greater value is more desirable. According to the preceding, the generalized architecture metric has the following representation:

$$P = \frac{a_1 \times FP + a_2 \times Cohesion}{a_3 \times Coupling} \quad (1)$$

Hence the criterion of architecture efficiency is formulated as follows:

$$P = \frac{a_1 \times FP + a_2 \times Cohesion}{a_3 \times Coupling} \to \max, \quad (2)$$

where a_1, a_2, a_3 – weight coefficients of efficiency indicators.

It is obvious that these efficiency indicators have different degrees of importance; therefore it's necessary to take into account their priority. The priority of efficiency indicators can be taken into account with the help of weigh coefficient vector (a_1, a_2, ..., a_n). The weight coefficient of efficiency indicator can be selected by expert evaluation and the value for a1 should be determining carefully, since wrongly selected value may suppress the values of Coupling and Cohesion. The optimal combination of the weight coefficient vector differs depending on architecture.

As long as efficiency indicators have different scale it is necessary to standardize them. To do so we can reduce it to single dimension using the following equation[3]:

$$\overline{A_i} = \frac{A_i}{A_{\max_i}}. \quad (3)$$

So the criterion of architecture efficiency is modified as follows:

$$P = \frac{a_1 \times \overline{FP} + a_2 \times \overline{Cohesion}}{a_3 \times \overline{Coupling}} \to \max. \quad (4)$$

The proposed metric allows us making a conclusion regarding the usage of some architecture patterns during architecture design stage. A case study with this metric usage could be found in another research[5].

3. Selection of an optimal patterns suite

To improve the quality of the architecture we can use the technique that allows selecting and evaluating suite of architectural patterns. The technique allows us to consistently evaluate the impact of specific patterns to software characteristics with a given functionality. To select an optimal patterns suite we need to define a model, choose a set of patterns and examine their impact on system characteristics.

3.1. 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 considered 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:

$$\{P_{ij} \,|\, i = 1,...,g; j = 1,...,m_j\}, \tag{5}$$

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:

$$W = f(P_1) \times x_1 + f(P_2) \times x_2 + \ldots + f\left(P_n\right) \times x_n \to min, \tag{6}$$

where:

$f_{(Pi)}$ — 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:

$$x_i = \{0,\ 1 \,|\, i = 1,2,...,n\}, \tag{7}$$

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:

$$\sum_{i=start}^{i=end} x_i = 1, \tag{8}$$

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:

$$x_j + \sum_{i=start}^{i=end} x_i = 1. \tag{9}$$

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

$$\sum_{i=start}^{i=end} x_i \le (end - start + 1). \tag{10}$$

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.

3.2. The choice of patterns for Multi-tier architecture

If we consider Multi-tier architecture there are number of patterns that can be divided into multiple groups. Fowler indicates the steps how to select a pattern from multiple groups taking into account the requirements for the software[6]. This selection technique consists of few steps. Initially we have to select the base pattern for *Domain Layer* implementation. Next, we need to select a pattern for *Data Source Layer* implementation which depends on the first step. In the final step we do select a pattern from *Presentation Layer*. Also we can select other patterns from the remaining groups.

For this technique application we have selected the list of the following patterns which we use for the model building:

- *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 Façade (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}).

When we build the model we should take into account the corresponding constraints related with restrictions that are applied on pattern groups and compatibility.

3.3. 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:

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

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

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

$$x_{13} \le 1,$$

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

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

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

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

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

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

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

3.4. 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[4,7]. In addition to the complexity metric, inner (cohesion) and outer (coupling) relations should be measured[4]. 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[5]. 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[4,8]. *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):

$$CBOF = \frac{\sum_{i=1}^{N} \begin{Bmatrix} CBO_i, & if\ CBO_i < T_{CBO} \\ T_{CBO}, & else \end{Bmatrix}}{T_{CBO} \times N}, \qquad (11)$$

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:

$$LCOMF = \frac{\sum_{i=1}^{N} \begin{Bmatrix} LCOM_i, & if\ LCOM_i < T_{LCOM} \\ T_{LCOM}, & else \end{Bmatrix}}{T_{LCOM} \times N}, \tag{12}$$

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:

$$K = \frac{\alpha_1 \times FP}{(1-\alpha_2 \times CBOF) \times (1-\alpha_3 \times LCOMF)}, \tag{13}$$

where

- $\alpha_1, \alpha_2, \alpha_3$ – 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:

$$f(P_i) = \frac{K'_{P_i}}{K}, \tag{14}$$

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:

$$K'_{Pi} = \frac{\alpha_1 \times FP'_{Pi}}{(1-\alpha_2 \times CBOF'_{Pi}) \times (1-\alpha_3 \times LCOMF'_{Pi})}, \tag{15}$$

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:

$$FP' = UFP \times \left(0.65 + 0.01 \times \sum_{i=1}^{14} CF_i \right) + P_{FP}, \tag{16}$$

where

- *UFP* – Unadjusted Function Point count;
- *PFP* – the value of functional points for specified pattern implementation;

and CF_i defined as follows:

$$CF_i = \begin{cases} 5, & if\ c_i \times F_i > 5; \\ round(c_i \times F_i), & otherwise, \end{cases} \tag{17}$$

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, we need to evaluate a characteristic using the following scale and convert them to c_i:

- 1 — use of a pattern reduces the significance of a system characteristic (c_i = ½);
- 2 — use of a pattern slightly reduces the significance of a system characteristic (c_i = ⅔);
- 3 — no influence (c_i = 1);
- 4 — use of a pattern slightly actualizes a system characteristic (c_i = 1½);
- 5 — use of a pattern actualizes a system characteristic (i.e. we must pay more attention to this characteristic when applying this pattern) (c_i = 2).

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

$$CBOF' = CBOF + \alpha \times CBOF \times \left(c_{Pi} - 3 \right), \tag{18}$$

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:

$$LCOMF' = LCOMF + \alpha \times LCOMF \times \left(c_{Pi} - 3 \right), \tag{19}$$

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.

3.5. Obtaining values

When we get the values of all necessary indicators and coefficients we can select optimal pattern suite (which are obtained as a result of solving integer programming problems) for considered software systems. Also we can obtain pattern-architecture efficiency metric which is used to measure patterns' numerical impact on a system. A detailed case study where this technique is used could be found in another research[9].

4. Selection of a metric suite

For software quality evaluation at the design stage different metric suites could be used, including object-oriented and post-object-oriented design metrics (see Fig. 1). The total number of different metrics is extremely large, and they have different characteristics, as well as offer different approaches to measuring the quality of the systems. Metrics which are used to evaluate complex characteristics of systems and integrated into the groups called metric suites. Examples of such metric suites include widely used metric suited like Chidamber & Kemerer's metrics suite[4,8], Lorenz and Kidd metrics suite[4], Metrics for Object Oriented Design proposed by Fernando Brito e Abreu[4].

During the initial stage of development, it's necessary to have a technique that allows selecting a metric suite among of different suites. Also such technique should help to optimise the selected metric suite. In addition, such a choice should take into account the specific requirements for the system. To choose the optimal metric suite it's necessary to use a technique that contains a number of steps and based on some criteria.

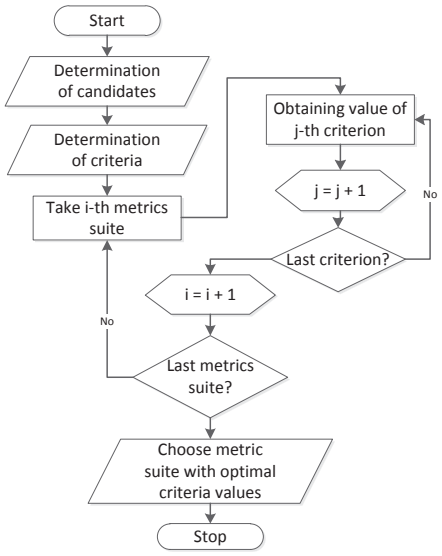

Fig. 1. Flowchart representation of metric suite selection algorithm.

The algorithm for this technique consists of the following steps (see Fig. 1): Determination of metric suites; Determination of criteria; Obtaining values of the criteria; Choice of the metric suite with the optimal values of the criteria.

4.1. Determination of criteria

The evaluation and comparison of several metric suites isn't an easy task. Therefore the criteria, which take into consideration the requirements for the metric suite, should simplify the problem. The selection of the criteria depends on many factors, and every solution might have different criteria. In general, such criteria selection is made by software engineers and should be based on some expert evaluations. For a criterion preference rule the maximization is used, where each criterion includes a combination of indicators. As long as not every indicator can be quantified or it might have different scales, we use a scale of correspondence for its evaluation. This scale has a range from 0 to 2, where 0 — no correspondence, 1 — average level of correspondence, 2 — full correspondence. The value of every indicator is chosen empirically. The weighting coefficients could be used in addition to preference rule. Weight coefficients should be chosen according to the specified requirements for the system.

According to the preceding, our criterion preference rule is defined as:

$$K = \sum_{i=1}^{n} \alpha_i \cdot C_i \to \max \tag{20}$$

where:

- $\alpha_1, \alpha_2, \ldots, \alpha_n$, – weigh coefficient of preference indicators which usually meets the following conditions:

$$0 \le \alpha_i < 1, \sum_{i=1}^{n} \alpha_i = 1;$$

- C_i ($i = 1, 2, \ldots, n$) – preference indicators which meets the conditions presented below:

$$\forall C_i \in R, \; R = \{0, \quad 1, \quad 2\}.$$

4.2. Determination of indicators and weight coefficients

For the metric suits evaluation we can use different indicators. For example, we can use indicators like usability, availability of tools, metric suite completeness, metric redundancy, theory base availability, etc. Also it's required to determine a weight coefficient for each indicator. Weight coefficients could be determined using expert evaluation.

4.3. Obtaining values

After we selected appropriate indicators and obtained values for them we can select most suitable metric suite using criterion of preference. A detailed case study where this technique is used could be found in another research[10].

5. Metric suite optimisation

After the metric suite is chosen it's necessary to perform its improvement. To do so each metric in the suite should be considered separately, analogous metrics should be determined and compared against the original metric. After such comparisons, the decision should be made regarding the original metric replacement with the analogous one. After these steps we obtain the modified metric suite which is optimal according to the requirements (see Fig. 2). An example of such metric optimization could be LCOM metric selection from Chidamber & Kemerer's metrics suite as there are plenty of different alternatives including LCOM1, LCOM2, LCOM4, tight class cohesion (TCC), loose class cohesion (LCC), information-flow-based cohesion (ICH), etc[4,10].

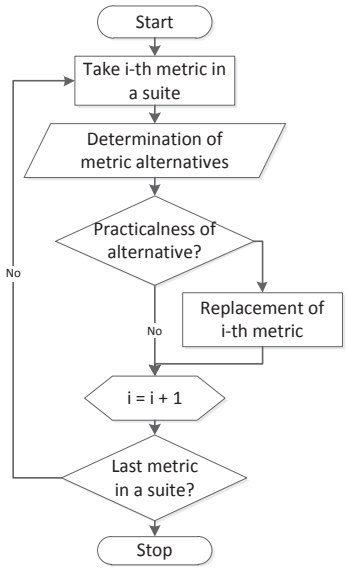

Fig. 2. Flowchart representation of metric suite optimisation algorithm.

The algorithm for this technique consists of the following steps (see Fig. 2): Determination of criteria; Take *i*-th metric; Determination of metric alternatives; Obtaining criteria values for every alternative; Replacing the metric with the alternative if alternative's criteria values are more optimal; If it isn't the last metric in the suite then take the next metric (return to step 2).

5.1. Determination of criteria

The criterion preference rule for metric suite optimization is similar to the criterion for metric suite selection and defines as:

$$K = \sum_{i=1}^{n} \alpha_i \cdot C_i \to \max \quad , \qquad (21)$$

where:

- $\alpha_1, \alpha_2, \ldots, \alpha_n$ – weigh coefficient of preference indicators;
- C_i ($i = 1, 2, \ldots, n$) – preference indicators.

5.2. Determination of indicators and weight coefficients

For a metric suite optimization indicators should be chosen based on the requirements for the considered systems as well as the metric requirements for such systems. We are also using the same scale of correspondence for its evaluation as for metric suite selection technique. For the metric suits optimisation we can use different indicators. For example, usability, availability of tools, correlation, etc. Weight coefficients for the indicators could be determined using expert evaluation.

5.3. Obtaining values

After appropriate indicators selection and obtained values for them the values of the criteria should indicate the most appropriate metric according to the requirements. After the comparisons, the decision should be made

regarding the original metric replacement with the analogous one. In the end we obtain the modified metric suite which is optimal according to the requirements. A detailed case study where this technique is used could be found in another research[10].

6. Conclusion

In this paper various techniques are proposed which helps to improve the quality of the developed software on architecture and detailed design stages. For architecture design stage we could use the criterion of efficiency which allows evaluating software architecture in general. The criterion of efficiency is based on such metrics as FP, Coupling and Cohesion. It allows making a conclusion regarding the usage of some architectural patterns for the considered software. Also on the architectural design stage we could use technique which allows making decisions regarding the patterns selection. It reduces the pattern selection task to the classical problem of integer programming where the optimal solution should be found. As an example, the model for Multi-tier architecture is created together with the set of patterns that are suitable for its creation. For the model creation we selected a function which reflects numerical changes of the system characteristics depending on a used pattern. This function is based on criterion of architecture efficiency metric. Criterion of architecture efficiency metric is used to measure patterns' numerical impact on a system. This metric is based on functional point 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. At the detailed design stage we have a large number of different metric suites; therefore we need ability to select the optimal one. In this paper the technique that allows making a decision regarding metric suite selection for considered software is proposed. The technique consists of several steps where at each step a specific criterion should be used to make a selection from the available metric suites. There may be used several different criteria indicators for a metric suite like completeness, usability, availability of tools, etc. Such indicators should be changed in accordance with the required characteristics of considered systems. As long as the metric suite is chosen it's necessary to perform its improvement. To do so each metric in the suite should be considered separately, analogous metrics should be determined and compared against the original metric. After the comparisons, the decision should be made regarding the original metric replacement with the analogous one. In the end we obtain the modified metric suite which is optimal according to the requirements.

Thus, the paper proposed four different techniques which help to improve the quality during architecture and detailed design stages.

References

1. Bass, L. Clements, P. Kazman, R. Software Architecture in Practice, 2nd Edition. Addison Wesley 2003. 560 p.
2. Buschmann, F. Meunier, R. Rohnert, H. Sommerlad, P. Stal M. Pattern-Oriented Software Architecture Volume 1: A System of Patterns. Willey, 1996. 476 p.
3. Orlov. S.A., Tsilker, B.J. Organization of computers and systems: textbook for high schools, 2nd ed. SPb.: Peter, 2010. 688 p. (In Russian)
4. Orlov, S. Tsilker, B. Software Engineering: A Textbook for Universities, 4rd ed. SPb.: Piter, 2012. 608 p. (In Russian)
5. Orlov, S. Vishnyakov, A. Pattern-oriented decisions for logistics and transport software, Transport and Telecommunication, Vol. 11, No 4, 2010, pp. 46–58.
6. Fowler, M., Patterns of Enterprise Application Architecture. Addison Wesley, 2002. 560 p.
7. International Function Point Users Group, Function Point Counting Practices Manual, International Function Point Users Group, Release 4.1, Westerville, Ohio, 1999. 335 p.
8. Chidamber, S. R., Kemerer, C. F. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, v.20 n.6, 1994. pp. 476-493.
9. Orlov, S. Vishnyakov, A. Multi-tier architecture optimisation for logistics and transport software, In: Reliability and Statistics in Transportation and Communication (RelStat'13), Riga, Latvia, October 16–19, 2013. Riga: TTI. pp. 340-351.
10. Orlov, S. Vishnyakov, A. Software-engineering measurement for logistics and transport systems, Transport and Telecommunication, Vol. 12, No 3, 2011, pp. 41–53.