Method and tool of detecting software architecture patterns in the process of computer systems development

Vasyl Yatsyshyn[1,†], Oleh Pastukh[1,†], Victoria Kukharska[1,†], Andriy Palamar[1,†] and Serhii Kulikov[1,†]

[1] *Ternopil Ivan Puluj National Technical University, Ruska, 56, Ternopil, 46001, Ukraine*

Abstract

In this paper a method and a tool for automatic detection of software architecture patterns are proposed. The main idea of the developed method is to automatically identify design patterns using artificial intelligence approach to software requirements and meta description of reuse software component. The main value of the method is to increase accuracy of design pattern detection in the process of computer systems development and to avoid developers subjective solutions.

This has been achieved by implementing a neural network with dynamic structure, which works out with attributes of software requirements and architecture patterns' meta description.

The tool of detecting software architecture patterns is created like a web-service with C# and help to automate the process of proposed method.

Keywords

detection, pattern, neural network, software, computer system

1. Introduction

Modern computer and information systems are characterized by a high level of functional complexity and the processing of a large amount of diverse and complexly structured data. In its turn, it requires from the developers of computer systems introducing new intelligent and effective methods and tools of developing both hardware and software components of the system. At the same time, formalized methods and CASE-tools must be integrated into the general process of developing the system, in particular its software component.

The most complex and time-consuming processes for any system are the identification, formulation and tracing (communication) of requirements at the stages of the life cycle. This is due to the fact that the requirements are the "foundation" for the further development of the system.

[1]*ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems, October 23-25, 2024, Ternopil, Ukraine, Opole, Poland*

[*] Corresponding author.

[†] These authors contributed equally.

vyatcyshyn@gmail.com (V. Yatsyshyn); oleg.pastuh@gmail.com (O. Pastukh); victoria_kukharska@tntu.edu.ua (V. Kukharska); palamar.andrij@gmail.com (A. Palamar); kulikov.serhiy@gmail.com (S. Kulikov)

0000-0002-5517-6359 (V. Yatsyshyn); 0000-0002-0080-7053 (O. Pastukh); 0000-0003-4763-0387 (V. Kukharska); 0000-0003-2162-9011 (A. Palamar); 0009-0003-3742-3568 (S. Kulikov)

From the point of view of the software product quality which is felt by the end user of the system, it is advisable to use the recommendations of standards, in particular, ISO/IEC 25010 and ISO/IEC 14598 in the development process.

The quality characteristics of the software component of the computer system and the processes of their provision are defined by the given standards. At the same time, there are still a number of processes that require automation. This applies to the collection and storage of requirements, the classification of attributes by quality characteristics, and a number of others [1].

The next important stage of software development is designing the system architecture. Architecture can be presented at different levels, starting from conceptual one and ending with the structure of software modules.

To provide the reflection of the requirements defined at the first stage of software system development at the architecture design stage it is worth using the approach proposed in [1, 2]. With this approach, system requirements from the end user's point of view are presented in the form of a quality model in use of the standard [3], and architecture requirements are presented in the form of the same standard external quality model.

The procedure of quality requirements display in external quality requirement using is developed and given in [4]. However, for increasing the efficiency of the process of developing software systems, it is necessary to additionally develop a method and tool for automatic detection and optimal selection of an architectural template for the implementation of a requirement or its part. Therefore, the article proposes a method and tool based on a neural network approach for detection optimal architectural patterns in the implementation of software components of computer systems.

2. Analysis of the software architecture design process features

Software architecture is an important aspect of a quality software product that affects the program's performance and its life cycle [5].

Software architecture is the structure of a program or computer system which contains software components, externally visible properties of these components, as well as the relationship between them [5]. This term also refers to the documentation of software architecture. Documenting the software architecture simplifies communication between interested parties and allows to capture high-level system design decisions made at the early design stages and gives the possibility to reuse design components and patterns in other projects.

K. Lavrishcheva, P. Andon, T. Korotun, G. Koval, O. Kharchenko, I. Sommerville, E. Braude, M. McCall, B. Boehm and other scientists devoted their scientific papers to researching the processes of designing the software systems architecture, their optimization, as well as methods and tools of ensuring and controlling the quality of software systems at the life cycle stages.

An important contribution to the development and application of software design patterns was made by such IT giants as Google, Microsoft, Hewlett Packard, Amazon and others.

Among the important tasks that are solved both by scientists and private companies it is worth noting the tasks related to determining and ensuring the balance between the cost, quality and duration of software development. In other words, setting and solving problems are aimed at achieving an acceptable level of quality of software systems with minimal expenditure of time, labor and financial resources. Although many of these problems have been solved at the moment, the problems in the field of designing software systems, namely, the process that directly affects the the final product quality, still remain incompletely solved.

Taking into account the fact that considerable practical experience has been accumulated in the field of software engineering, it is economically and technically expedient to reuse components that have shown the effectiveness and quality of their application in the software products operation. While building integration platforms such as Onlizer, component reuse is especially important.

At the current stage of software engineering and computer engineering, so many different design patterns with similar functions have been developed that it is quite difficult to choose the optimal one among them. This is due to the fact that the procedures for the optimal selection of architectural patterns are still weakly formalized, they are based on the empirical judgments of system architects and do not ensure the complete implementation of requirements for software systems. And this does not make it possible to make an optimal decision regarding the choice of a design pattern.

Therefore, the development of a method and tool of automated detection and selection of the optimal architectural pattern are the urgent tasks in the software architecture design as a component of a more complex computer system. This will minimize the subjective influence of experts or system architects and ensure the full realization of the customer's needs in the software systems properties.

3. Method of detecting patterns of software architecture based on a neural network approach

The pattern detection process is proposed to be implemented on the automatic analysis of requirements models and design patterns models (architectural patterns). Models of requirements and patterns can be formed by an expert based on development goals, specifics of the development process, or accepted agreements. For example, the requirements model can be based on the recommendations of the standard [3] and on the internal quality model, and the pattern model can use the classification presented in [7].

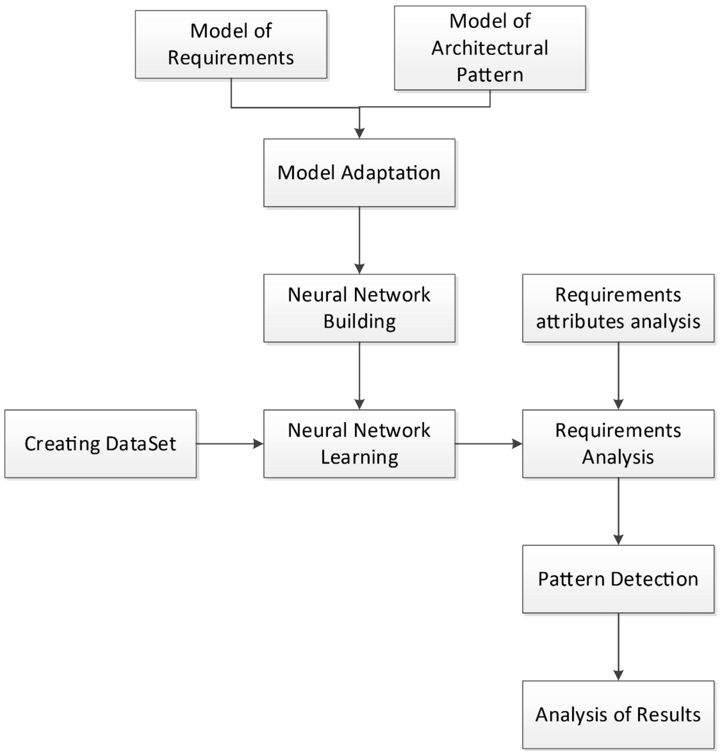


Figure 1: Conceptual diagram of the design pattern detection process.

Since the existing standards and approaches to the formation of requirements models are of a recommendatory and general nature, the obtained models require adaptation for use in the specific conditions of the PS development process, which are determined by the subject area, software type and development goals.

The requirement model is a collection of attributes forming a set $R\{A_1,\dots,A_n\}$where the attribute value is determined according to the metric scale for the given attribute.

The scale for attribute values can be of any type but its projection on the interval scale $S_{Ai} \to S_{\int ¿\{S_1,\dots,S_k\}, S_i \in [0;1]¿}$ must exist. The set of software requirements forms the matrix R:

$$R=\begin{bmatrix} A_{11} & \dots & A_{1n} \\ \dots & \dots & \dots \\ A_{m1} & \dots & A_{mn} \end{bmatrix}, \tag{1}$$

where $A_{ij} \in S_{A_{ij}}$ is an attribute of R_i requirement.

Respectively the architectural pattern design model is a collection of attributes forming the set $P\{A_1^{'},\dots,A_k^{'}\}$. Regarding the scale of metric values of the attributes of the design pattern model, a similar requirement of the existence of a projection on the interval scale is put forward: $S_{\int ¿ : S_{A_i^{'}} \to S_{\int ¿¿}¿}$ A set of pattern models forms a matrix P:

$$P=\begin{bmatrix} A_{11}^{'} & \dots & A_{1k}^{'} \\ \dots & \dots & \dots \\ A_{t1}^{'} & \dots & A_{tk}^{'} \end{bmatrix} \tag{2}$$

where $A'_{ij} \in S_{A'_{ij}}$ is an attribute of the design pattern P_i.

After adapting the models for the specific requirements of the software development process the mapping of matrix elements R and P to the $S_{\int ¿.¿}$ scale is formed:

$$R_{\int ¿} = \begin{bmatrix} I_{11} & \dots & I_{1n} \\ \dots & \dots & \dots \\ I_{m1} & \dots & I_{mn} \end{bmatrix} ¿ \tag{3}$$

where I_{ij} is the mapping of the A_{ij} attribute to the $S_{\int ¿¿}$ scale:

$$P_{\int ¿} = \begin{bmatrix} F_{11} & \dots & F_{1k} \\ \dots & \dots & \dots \\ F_{tk} & \dots & F_{tk} \end{bmatrix} ¿ \tag{4}$$

where F_{ij} is the mapping of the P_{ij} attribute to the $S_{\int ¿¿}$ scale.

The main component of the process of detecting design patterns is a multi-level neural network which is schematically shown in Figure 2. The dimensionality of the neural network is determined by the dimensionality of the matrices R and P.

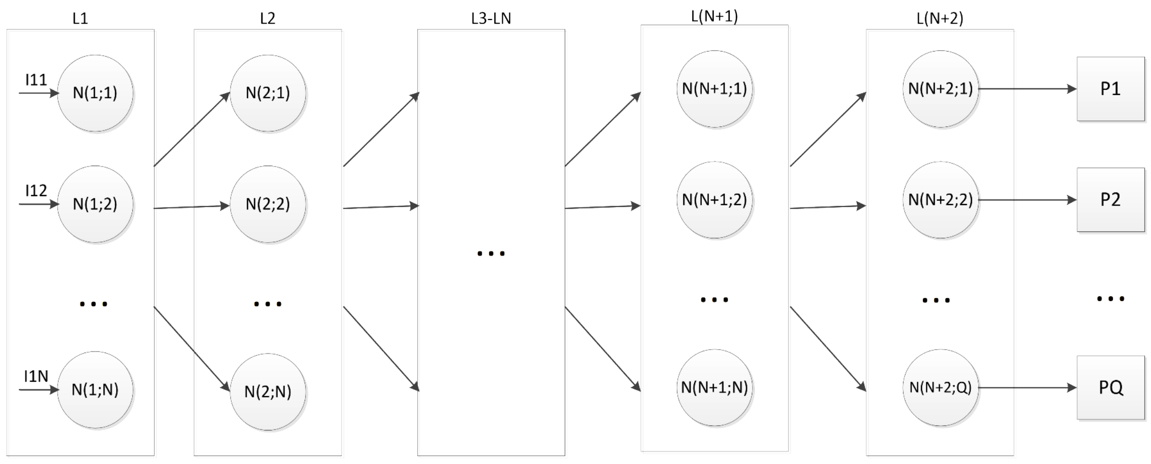


Figure 2: The structure of the neural network of the requirements model analyzer.

The L_1 input layer contains artificial neurons with a linear transform function, which prevents the input signal from degenerating. Values from the rows of the matrix $R_{\int ¿¿}$ are applied to the input of neurons of the L_1 level.

The neuron layer L_2 contains artificial neurons with a sigmoid activation function and is the first level of the hidden layer of the neural network.

Neurons of levels $L_3 - L_N$ are artificial neurons with a sigmoid activation function.

The L_{N+1} *l*evel is composed of artificial neurons with a sigmoid activation function and is the last level of the hidden layer.

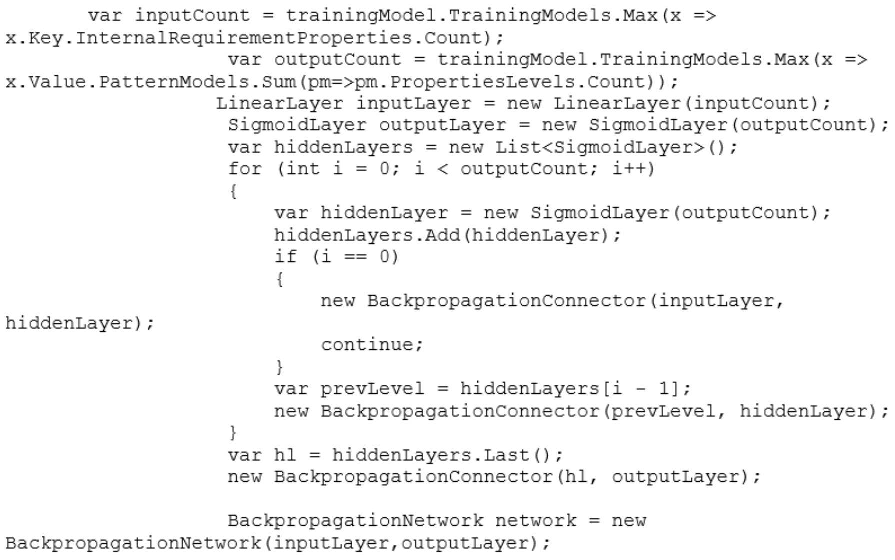
The L_{N+2} level is composed of artificial neurons with a sigmoid activation function and is the initial level of the neural network. The values of the signals at the output of the L_{N+2} layer form the probable values of the model attributes of the design patterns P.

Based on the analysis of the values of the resulting vector at the output of the neural network using the simple moving average method, the probability of the Pi pattern entering the set of requirements R is formed.

4. Software implementation of a neural network for detecting architectural patterns

The NeuronDotNet library was used for the software implementation of the neural network which allows using the C# language flexibly construct various neural networks.

The NeuralNetworkBuilder module containing the static method BuildAndTrainNetwork has been developed for the formation and training of a neural network. In the process of forming a neural network, input and output levels of neurons and neurons of hidden levels are constructed and connections between them are established, as shown in Figure 3.



```
       var inputCount = trainingModel.TrainingModels.Max(x =>
x.Key.InternalRequirementProperties.Count);
                   var outputCount = trainingModel.TrainingModels.Max(x =>
x.Value.PatternModels.Sum(pm=>pm.PropertiesLevels.Count));
                  LinearLayer inputLayer = new LinearLayer(inputCount);
                   SigmoidLayer outputLayer = new SigmoidLayer(outputCount);
                   var hiddenLayers = new List<SigmoidLayer>();
                   for (int i = 0; i < outputCount; i++)
                   {
                       var hiddenLayer = new SigmoidLayer(outputCount);
                       hiddenLayers.Add(hiddenLayer);
                       if (i == 0)
                       {
                           new BackpropagationConnector(inputLayer,
hiddenLayer);
                           continue;
                       }
                       var prevLevel = hiddenLayers[i - 1];
                       new BackpropagationConnector(prevLevel, hiddenLayer);
                   }
                   var hl = hiddenLayers.Last();
                   new BackpropagationConnector(hl, outputLayer);

                   BackpropagationNetwork network = new
BackpropagationNetwork(inputLayer,outputLayer);
```

Figure 3: Listing of a neural network construction.

A pre-formed set of requirements and detected design patterns is used for training the neural network (the text of the requirements and the meta-description of the architectural patterns are used in agreement with the Onlizer company), the training is conducted without a teacher. A fragment of the neural network training listing is given in Figure 4.

```
    TrainingSet ts = new TrainingSet(inputCount, outputCount);
                foreach (var tm in trainingModel.TrainingModels)
                {
                    var inputVector =
                        tm.Key.InternalRequirementProperties.OrderBy(x =>
x.InternalID).Select(x => x.Value).ToArray();
                    var outputVector =
                        tm.Value.PatternModels.OrderBy(x =>
x.PatternModel.InternalID)
                            .SelectMany(x => x.PropertiesLevels.OrderBy(kvp
=> kvp.Key).ToList()).Select(x => x.Value)
                            .ToArray();
                    var sample = new TrainingSample(inputVector,
outputVector);
                    ts.Add(sample);
                }
                network.Learn(ts,trainingModel.TrainingCyclesCount);
```

Figure 4: Listing of neural network training.

After training the neural network can be used to find patterns using the Run method the code of which is shown in Figure 5. The input of the method is given by is a set of attribute values of the requirements model.

```
        public double[] Run(InternalRequirement inputRequirement)
            {
                var input =
inputRequirement.InternalRequirementProperties.OrderBy(x =>
x.InternalID).Select(x => x.Value).ToArray();
                var output = _network.Run(input);
                return output;
            }
```

Figure 5: Listing of finding patterns using a neural network.

The resulting vector of values is analyzed using a simple moving average method to find the likely design pattern and is shown in Figure 6.

```
        public PatternModel ResolvePattern(double[] valuesVector)
            {
                var deltas = new double[_patternsList.Count];

                var startIndex = 0;
                for (int i = 0; i < _patternsList.Count; i++)
                {
                    var pattern = _patternsList[i];

                    var patternVector =
pattern.PatternModelProperties.OrderBy(x=>x.InternalID).Select(x =>
x.Value).ToArray();
                    var targetVector =
valuesVector.Skip(startIndex).Take(patternVector.Length).ToArray();
                    startIndex += targetVector.Length;
                    for (int j = 0; j < patternVector.Length; j++)
                    {
                        patternVector[j] = Math.Abs(patternVector[j] -
targetVector[j]);
                    }
                    deltas[i] =
patternVector.Sum()/(double)patternVector.Length;
                }

                var minDelta = deltas.Min();
                return _patternsList[Array.IndexOf(deltas, minDelta)];
            }
```

Figure 6: Listing of analysis of the resulting vector and pattern finding.

The proposed conceptual model and software implementation for detecting design patterns based on requirements models allows to change flexibly approaches to the formation of requirements models and templates, adjust the requirements analysis parameters by changing the configuration of the neural network or its training conditions. In addition, the use of a neural network as the main component of the analyzer allows to ensure the progressiveness of the search parameters, increase the accuracy of detection and automate the process of design templates searching.

5. A tool for automated detection of software architectural patterns

The user interface of the software system for choosing an architectural pattern should provide convenient entry, editing and viewing of information, be intuitive, and ensure the correctness of data entry. To implement the algorithm of the program, it is proposed to use a multi-window interface with main and subsidiary forms. UML graphical notation was chosen for system analysis and design. Figure 7 shows a use case diagram with an administrator actor.

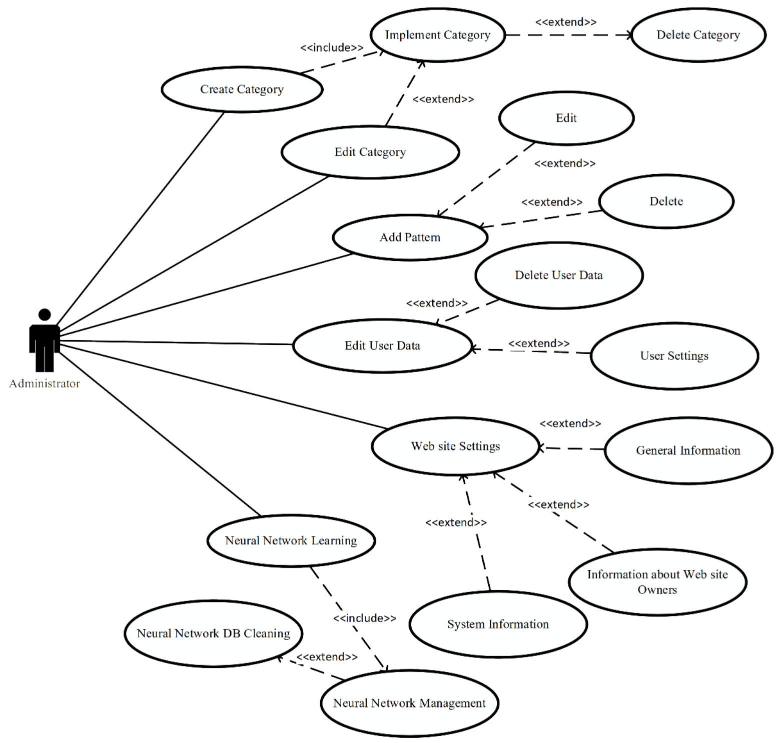


Figure 7: Use Case Diagram "Administrator".

Figure 8 provides a diagram of the use cases for users of the software system architecture design pattern selection system.

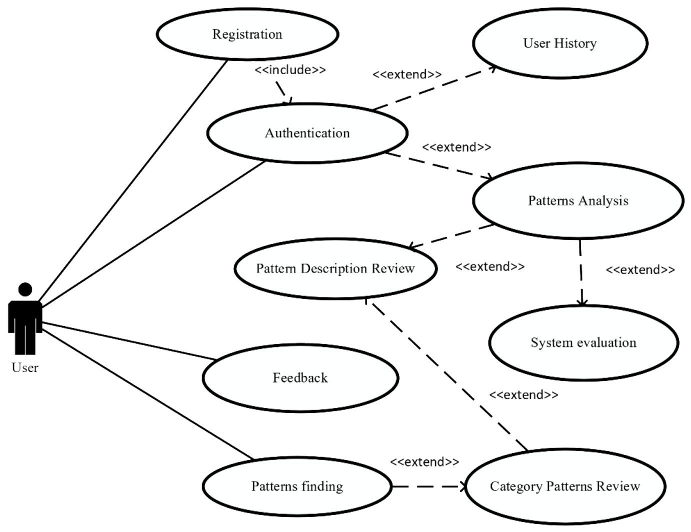


Figure 8: Use case diagram "End user".

The template catalog is a part of the website database that stores all information about architectural design patterns. The catalog has a hierarchical, tree-like structure that will reflect the way templates are classified. The directory structure directly depends on which templates are available on the site.

Figure 9 shows a sequence diagram of actions when working with the system for selecting architectural patterns of software systems.

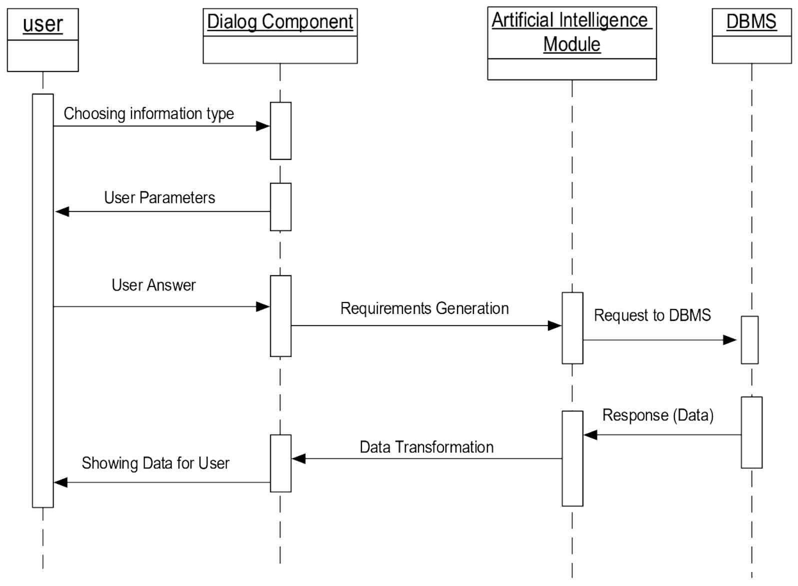


Figure 9: Sequence diagram.

The sequence diagram shows 4 main components (objects): user, dialog component, intelligent module, DBMS.

- the user selects the type of information and answers the questions of the dialog component;
- the dialog component forms parameters that must be obtained from the user and passes the data to the administrator;
- after that the dialog component forms requirements for the intelligent module which in turn transforms the data to be issued to the dialog component;
- the intelligent module generates requests to the DBMS;
- DBMS sends data to the intelligent module to make recommendations based on the collected statistical data which are displayed to the user.

As a result of the analysis of the subject area, the following classes are defined for the implementation of the design pattern detection automation tool: user, web page, web server, intelligent component, database and interface - user interface. In Figure 10 the class structure of the design pattern detector according to the established software requirements is presented.

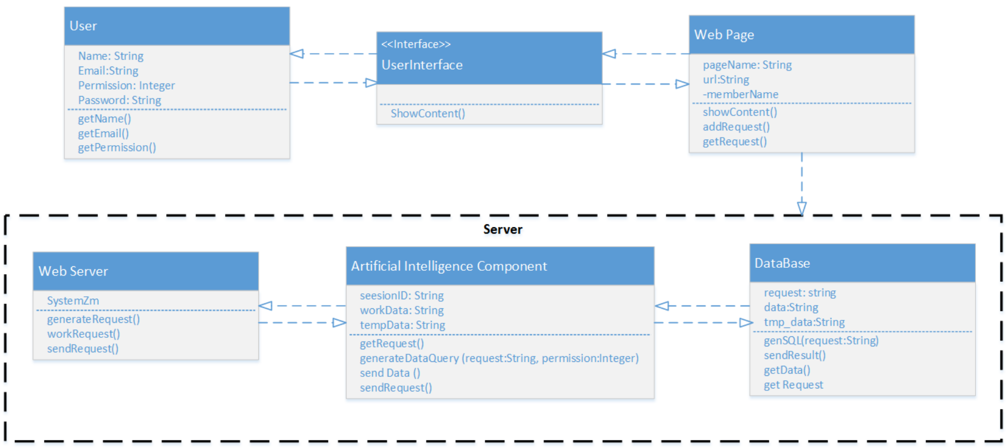


Figure 10: Class diagram.

It is worth paying attention to the presence of an interface. This is a type of class that does not contain attributes but interacts with other classes through it.

The activity diagram for the template selection system is shown in Figure 11.

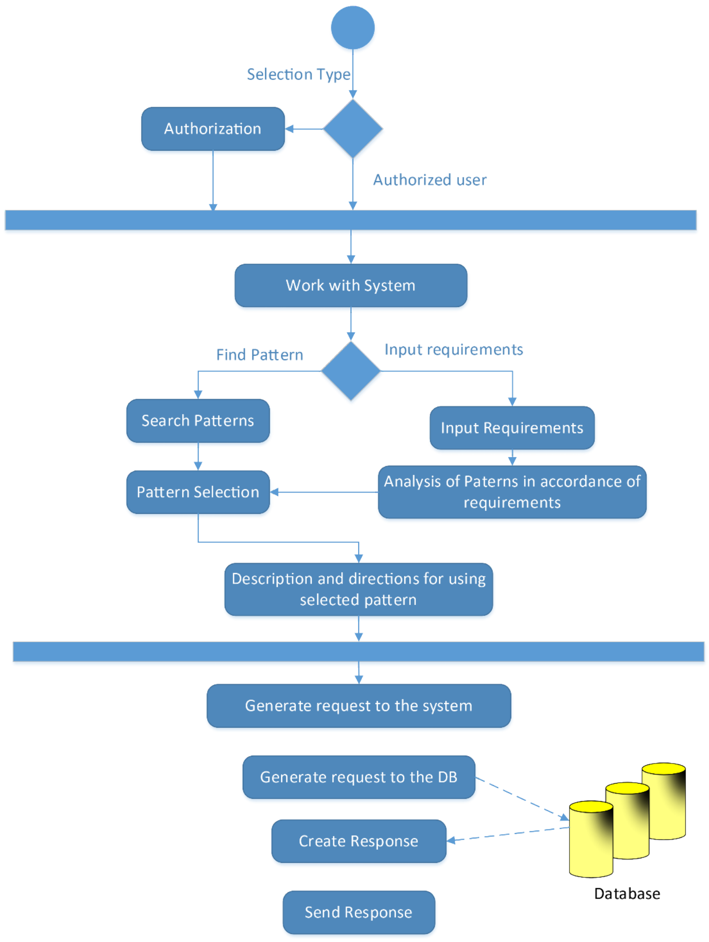


Figure 11: Activity diagram.

As can be seen from Figure 11, after choosing a category of templates, the principle of working with the tool for automated selection of design templates is the same: forming a query - selecting data from the database – outputting the result. This diagram shows the steps that are performed before and after the system accesses the database.

The user, regardless of whether he is authorized or not, must first choose which operation he is going to perform, set the search criteria immediately or view the catalog of template categories.

According to the selected actions, a request is generated to the system. The intelligent component processes the request and transfers control to another component which forms a request to the database (DB) based on the received parameters.

DBMS executes the query and returns the result in the form of data which is formatted in a way that is convenient for the end user.

For authorized users, it is allowed to use the template selection system in accordance with the requirements set for the software. An unauthorized user must specify the analysis criteria.

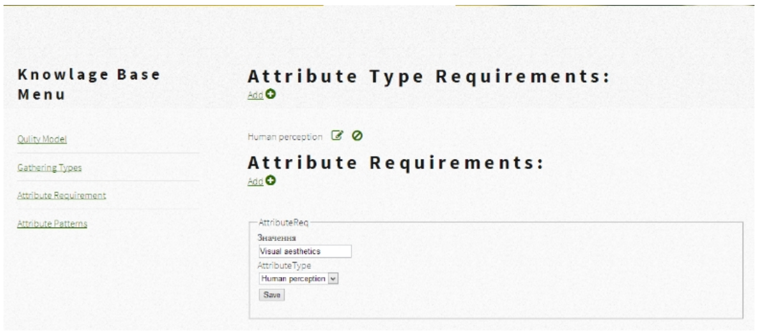
The developed software design pattern detector prototype is positioned as a web service, so the main hardware is a web server and a database server.

The web server contains the following artifacts:

- web-interface which is an ordinary web page responsible for the tool's dialogue with the user;
- intelligent component that is responsible for the correct interpretation of requests from the user, issuance and data processing;
- database interface forms a request to DBMS based on the criteria obtained from the intelligent component.

The DBMS artifact is the MongoDB management system. This component is responsible for storing and accessing data.

An example of the formation of software requirement attributes is presented in Figure 12.


Knowlage Base Menu

Qulity Model
Gathering Types
Attribute Requirement
Attribute Patterns

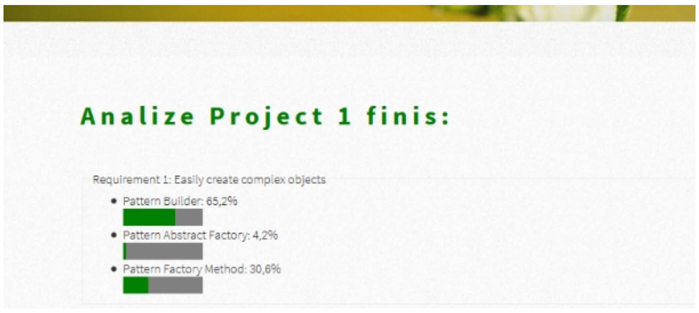
Attribute Type Requirements:
Add
Human perception

Attribute Requirements:
Add

AttributeReq
Значення
Visual aesthetics
AttributeType
Human perception
Save

Figure 12: Formation of software requirement attributes.

The result returned by the intelligent component for the set of requirements formulated for the project named "Project1" is shown in Figure 13.


Analize Project 1 finis:

Requirement 1: Easily create complex objects
- Pattern Builder: 65,2%
- Pattern Abstract Factory: 4,2%
- Pattern Factory Method: 30,6%

Figure 13: The result of choosing the optimal design pattern for the project.

Data from the data set provided by the Onlizer company as part of cooperation and conducting joint scientific research were used for experimental research.

As a result, as can be seen from Figure 9, the distribution of probabilities regarding the optimal choice of software architecture design patterns is obtained.

Conclusion

The article proposes a method and a tool for detecting and optimally choosing software architecture design patterns depending on the requirements for the system. The requirements for the software system are presented in the form of quality models of the ISO/IEC 25010 standard. The needs of end users are reflected in the structure of the quality in use model, and the requirements for the system architecture are in the form of an external quality model, according to the procedures proposed in [1].

To implement the method of detection and optimal selection of design patterns, a neural network was built and implemented using the C# language, which takes into account the requirements for the system and the meta description of existing design patterns.

The tool for detection and optimally selecting design patterns of software architecture of computer systems makes it possible to ensure the efficiency of management of the system development process, and due to automation it provides high productivity and accuracy of decision-making regarding the use of reusable components.

References

[1] Kharchenko O., Yatsyshyn V. Rozrobka ta keruvannya vymohamy do prohramnoho zabezpechennya na osnovi modeli yakosti. Visnyk TDTU. 2009. Tom 14. №1. S. 201-207.

[2] Yatsyshyn V, Kharchenko A, Galay I. The Method of Quality Management Software. Proceeding of the VIIth International Conference "Perspective technologies and methods in MEMS design" 11-14 May 2011 - Polyana, Ukraine: Publishing House Vezha&Co. 2011.- p. 228-230.

[3] ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

[4] ISO/IEC 14598: Information Technology – Software product evaluation. Parts 1 to 6: 1999 -2001, International Organization for Standardization, Geneva, 1999-2001.

[5] Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Pearson. 2017. 420 p.

[6] Clean Architecture Solution Template. URL: https://github.com/jasontaylordev/CleanArchitecture.

[7] Yatsyshyn V., Pastukh O., Zharovskyi R., Shabliy N. Software tool for productivity metrics measure of relational database management system. Mathematical Modeling. No 1 (48). 2023. P. 7-17.

[8] Pastukh O., Yatsyshyn V. Development of software for neuromarketing based on artificial intelligence and data science using high-performance computing and parallel programming technologies. Scientific Journal of TNTU. Tern.: TNTU, 2024. Vol 113. No 1. P. 143–149.