Software Design using Genetic Quality Components Search

Evgeny Nikulchev[1], Dmitry Ilin[2], Aleksander Gusev[3]
MIREA – Russian Technological University
Russia

***Abstract*—The paper presents a software design methodology based on computational experiments for effective selection of software component set. The selection of components is performed with respect to the numerical quality criteria evaluated in the reproducible experiments with various sets of components in the virtual infrastructure simulating the operating conditions of a software system being developed. To reduce the number of experiments with unpromising sets of components the genetic algorithm is applied. For representing the sets of components in the form of natural genotypes, the encoding mapping is introduced, reverse mapping is used to decipher the genotype. In the first step of the technique, the genetic algorithm creates an initial population of random genotypes that are converted into the assessed sets of software components. The paper shows the application of the proposed methodology to find the effective choice of Node.js components. For this purpose, a MATLAB program of genetic search and experimental scenario for a virtual machine running Ubuntu 16.04 LTS operating system were developed. To guarantee the proper reproduction of the experimental conditions, the Vagrant and Ansible configuration tools were used to create the virtual environment of the experiment.**

Keywords*—*Software design; selection of software components set; numerical quality criteria evaluated; genetic algorithm

I. Introduction

Effective selection of software components based on assessments of the quality of service criteria [1] is becoming increasingly important problem [2] in connection with the spread of the framework approach to software development. This paper considers the problem in the context of highly loaded distributed client-server information systems (IS) implemented in the JavaScript.

The framework is a template of architectural solution. It allows the developer to unify the process of developing an IS based on a combination of the constant part of the IS (framework), which does not vary from configuration to configuration, and connected components that are compatible with the constant part. The JavaScript framework is a framework written in the JavaScript language that allows programmers to manipulate a set of compatible components (libraries) to solve a problem. The framework differs from the JavaScript library in the control flow: the library is always called by its parent code, while the framework defines the overall architecture of the IS and calls certain components to implement the functionality defined by the developer.

The research was performed as part of the state task of the Ministry of science and height education of Russian Federation, project 25.13253.2018 / 12.1 "Development of the technological concept of the Data Center for Interdisciplinary Research in Education"

The aim of this work is creating and experimentally testing a technique for effectively selecting the software components for the framework based on experimental evaluations of quality criteria.

The article consists of six sections. The first is Introduction. In the second section the review of the related works is presented. In the third section the problem is formulated. The fourth section describes the methods of genetic search and the configuration of the experimental stand. The fifth section provides the results of the genetic search. The sixth section discusses the results. The seventh section concludes the article.

II. Related Works

Solved with the help of JavaScript in recent years, the variety of tasks has led to the emergence of hundreds of frameworks. They can be divided into two groups. (1) Universal one (for example, Node.js), which allows the developers to use JavaScript for writing the server part of a web application as a general-purpose language with the ability to interact with I/O devices. (2) Frameworks for writing browser-based (front-end) applications running on the user's side, such as the followings: Angular.js, Angular (it is written in TypeScript, which is a backwards-compatible JavaScript modification), Vue.js, React.js and lots of others.

To support the optimal choice of the framework, various techniques were proposed earlier. Those techniques allowed the developers to assess the compliance of the framework with the general needs of the developer for a given set of components using the performance benchmark results [3] or expert evaluations [4]. However, an urgent task is creation of a methodology for selecting the efficient set of software components for the specified framework to provide the guaranteed quality of service [5] (QoS) and the efficiency of operation under given conditions. Those conditions include the specific development environment, computing infrastructure, computational loads during normal and peak operation, etc.

The basic element of such a technique should be the procedure for conducting reproducible computational experiments to assess the quality of the functioning of software components. Unlike the automated software testing [6], the experimental assessment of the quality of the functioning would provide more flexible approach to select the components for the IS even if there were no errors reported but the performance of the IS could be increased with effective selection of the components. By automating this procedure, it is generally necessary to solve the problem of reducing the

number of iterations of software components in order to obtain the effective set. The solution can be found using genetic algorithms [7]. They have proven themselves to be useful in assessing the problem of multicriteria optimization in the field of software development: evolutionary software development cost estimation [8]; optimization of computing resources utilization [9, 10]; generating optimal test data sets [11]; evaluating [12] software reliability; optimization of software partitioning into modules [13]; prioritizing client requirements for software development [14]; software refactoring [15]; in solving problems of project management [16] and human resources allocation [17]; in other tasks, including those related to the development of cloud web – services with QoS-aware resource allocation and dynamic web-service composition [18–20].

III. Model and Research Methods

Let us consider n functional features q_i, $i=\overline{1,n}$, which should be implemented in the IS and t different configurations $\omega^k, k=\overline{1,t}$ of the virtual infrastructure, simulating the IS operating conditions; M is the set of all the software components available for the research, each of which implements at least one of the features q_i. The subset of alternative software components from M, suitable for implementing the feature q_i is denoted as $m_i, i=\overline{1,n}$. Let us consider the situation when there exist p technology stacks $s^j, j=\overline{1,p}$ i.e. such sets of software components, in which for every feature q_i, $i=\overline{1,n}$ there exists at least one software component from M. Let us denote the set of all possible stacks as S. We introduce then the set of f experimentally evaluated partial quality criteria $r_\xi^{k,j}$, $\xi=\overline{1,f}$, values of which belong to the space R^{f}. Thus,

$$\forall\omega^k : s^j \to R^{k,j} \in \mathrm{R}^{\mathrm{f}},$$

$$R^{k,j} = (r_1^{k,j}, r_2^{k,j}, \ldots, r_\xi^{k,j}, \ldots, r_f^{k,j})^T, k=\overline{1,t}, j=\overline{1,p},$$

where $r_\xi^{k,j}, \xi=\overline{1,f}, k=\overline{1,t}, j=\overline{1,p}$ are the values of experimentally evaluated partial quality criteria for the configuration ω^k of the virtual infrastructure and the stack s^j being evaluated.

Let us introduce the integral quality criteria for the IS:

$$\Psi\left(\omega^k, s^j\right) = \sum_{\xi=1}^{f} w_\xi \tilde{r}_\xi^{k,j}, \tag{1}$$

where $\tilde{r}_\xi^{k,j}$ are the normalized values of partial quality criteria $r_\xi^{k,j}$; $\xi=\overline{1,f}$; $w_{\xi,}$ are the weights of the partial criteria. Herewith $\sum_{\xi=1}^{f} w_\xi = 1$.

The problem of the effective choice of software components based on the experimental evaluation of the quality of operation (see Fig. 1) for the chosen configuration of the virtual infrastructure ω^k consists in the choice of the technology stack s^* satisfying the following condition:

$$s^* = \underset{s^j, j=\overline{1,p}}{\operatorname{argmin}} \Psi\left(\omega^k, s^j\right). \tag{2}$$

Using the above introduced approach let us consider the case of selecting Node.js components.

Table I shows the set of functional features and components that implement those features in a computational experiment. Thus, $n=10$, $p=216$.

The considered configuration $\omega^k, k=t=1$ of the virtual infrastructure is specified in Table II.

The evaluation of the quality of operation is performed with respect to the $f=14$ partial quality criteria defined in Table III.

The weighting factors for the criteria are $w_2 = w_{11} = 0.08$; $w_\xi = 0.07 (\xi=1, \xi=\overline{3,10}, \xi=12)$, setting the target QoS.

When conducting the experiment, $r_\xi^{k,j}$ ($\xi=\overline{1,14}$, $k=\overline{1,t}, j=\overline{1,p}$) are normalized with respect to their maximum values in the experiment and take their values in the segment [0; 1].

The task is selecting the stack s^* of Node.js components, solving the problem (2).

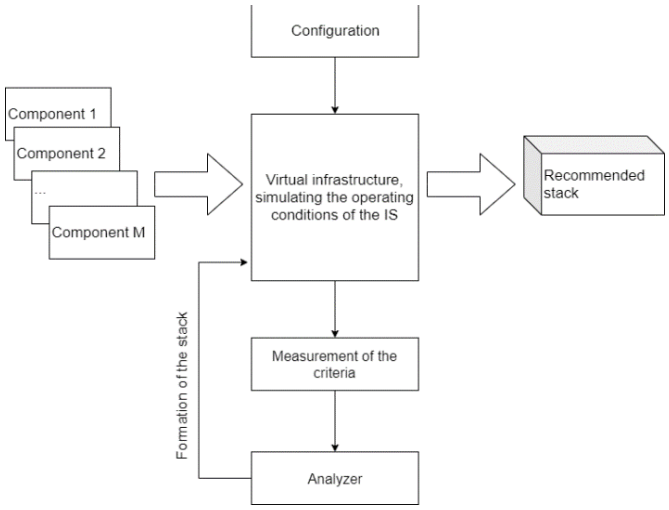

Fig. 1. The Effective Choice of Software Components based on the Experimental Evaluation of the Quality of Operation.

TABLE. I. LIST OF FUNCTIONAL FEATURES AND COMPONENTS

q_i	Name of the functional feature	Alternative components	Description
q_1	Filter	Lodash Underscore	Checks all the elements of an array against some condition and returns an array of elements for which the check gave "True"
q_2	First	Lodash Underscore	Returns the first element of an array
q_3	FsRead	Fs–extra Fs	Reads data from a file
q_4	FsReaddir	Fs–extra	Reads the contents of a directory
q_5	FsReaddirRecursive	Recursive–readdir	Recursively reads the contents of a directory
q_6	HashMD5	Hasha md5 Ts–md5	Calculates the MD5 hash for the specified data set
q_7	Map	Lodash Underscore JavaScript language tools	Applies the specified function to all the elements of the array, thereby returning a new array consisting of the transformed elements
q_8	PathResolve	Path	Generates the full path to the file or directory based on the specified array of path elements
q_9	StringReplace	JavaScript language tools	Finds and replaces a substring in the string passed
q_{10}	ZipCompress	Adm–zip Jszip Zipit	Performs archiving of the transferred file array and returns the generated Zip – archive

TABLE. II. VIRTUAL INFRASTRUCTURE CONFIGURATION $\omega^k (k = 1)$

Num.	Parameter	Value
1	CPU	Intel® Core ™ i7–7700
2	Number of cores	4
3	Number of logical processors	8
4	Clock frequency	3.60 GHz
5	RAM	12.0 GB
5	Host operating system	Ubuntu 16.04 LTS
6	Vagrant version	2.2.4
7	Node.js version	10.15.3
8	Virtual machine parameters	2 CPU cores 2.0 GB RAM Ubuntu 16.04 LTS
9	Provisioning software	Ansible
10	File exchange tools for the virtual machine	NFS–server + BindFS inside the virtual machine
11	Additional system software	– git – make – htop – iotop – rsync – node–gyp

TABLE. III. PARTIAL QUALITY CRITERIA ($k = 1, j = \overline{1, p}$)

Notation	Criterion	Unit
$r_1^{k,j}$	The microprocessor operating time spent on the initialization of the experiment	ms
$r_2^{k,j}$	The operating time of the microprocessor spent on the execution of system functions during the initialization of the experiment	ms
$r_3^{k,j}$	The increase in the Resident Set Size noted after the completion of the initialization of the experiment (including heap, code segment and stack)	byte
$r_4^{k,j}$	The increase in the heap size, marked upon completion of the initialization of the experiment	byte
$r_5^{k,j}$	The increase in the volume of the used heap, marked upon completion of the initialization of the experiment	byte
$r_6^{k,j}$	The increase in the amount of RAM used by C++ objects associated with JavaScript objects, marked after the experiment has been initialized	byte
$r_7^{k,j}$	The real time spent on the initialization of the experiment	ns
$r_8^{k,j}$	The microprocessor operating time spent on the experiment	ms
$r_9^{k,j}$	The microprocessor operating time spent on the execution of system functions during the experiment	ms
$r_{10}^{k,j}$	The increase in the Resident Set Size noted at the end of the experiment (including heap, code segment and stack)	byte
$r_{11}^{k,j}$	The increase in the heap size, marked at the end of the experiment	byte
$r_{12}^{k,j}$	The increase in the amount of the used heap noted at the end of the experiment	byte
$r_{13}^{k,j}$	The increase in the amount of RAM used by C ++ objects associated with JavaScript objects, marked upon completion of the experiment	byte
$r_{14}^{k,j}$	Real time spent on the experiment	ns

IV. EXPERIMENTAL METHODOLOGY

The automated methodology for selecting an effective set of software components involves the use of a genetic algorithm to generate and experimentally evaluate stacks of technologies (see Fig. 2).

The integration of the components of the stack is implemented using a functional approach, which is the most convenient way to combine various sets of software components. Each function which is being called during the experiment is a kind of software interface that is implemented using one of the stack components. Since components, as a rule, provide tools that go beyond a single function, they can be used to implement several functions. The use of a single component for performing a variety of tasks in the general case is preferable, since it reduces the amount of RAM needed by the IS.

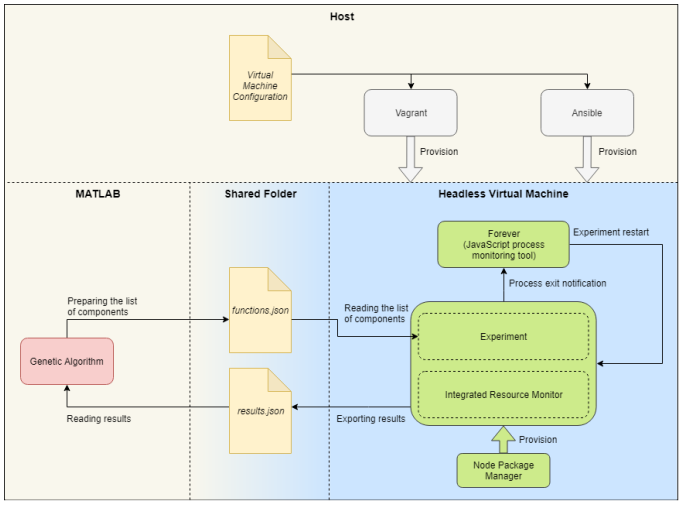

Fig. 2. Experimental Methodology.

At the initialization stage, the functions, which define the basic settings of the components, are called. Each of them forms a new anonymous function at the output, which has exclusive access to the component with the specified settings. Next, anonymous functions are placed in a single namespace with the names listed in Table III. To increase the reliability of the results obtained, the component cache (also known as Node.js module cache) is cleared before initialization.

After the initialization, the execution phase of the experimental algorithm begins. For research purposes, the following experimental algorithm is used:

1) Form the path to the directory with a set of subdirectories.
2) Read the list of subdirectories.
3) Exclude hidden subdirectories.
4) Form the path for each directory.
5) Do the followings for each path:
a) Read all the list of files recursively.
b) Read and load into the RAM all the files.
c) Create a Zip-archive in the RAM.
d) Calculate the MD5-hash for the created archive.

After the initialization procedure and the execution of the experimental algorithm are done, a json file results.json is generated. It contains the source data for the calculation of the integral criterion (1). This data is obtained through the interface of the "process" object of Node.js.

The genetic search configuration is specified in Table IV.

For the numerical representation of stacks, the encoding mapping is introduced as $C: S \to \Lambda \subseteq \mathrm{N}^n$. Thus, to each stack s^j, $j = \overline{1,p}$, which is called a phenotype, the natural set $\zeta^j = C(s^j)$, $j = \overline{1,p}$, which is called a genotype, will correspond. The genetic algorithm treats genotypes as

$$\zeta_g^h = \left(\alpha_{1_g}^h \ldots \alpha_{n_g}^h\right), h = \overline{1,|\Theta_g|},$$

TABLE. IV. GENETIC SEARCH CONFIGURATION

Num.	Parameter	Value
1	MATLAB version	R2018a
2	Integer constraints	All the genes are integer-valued
3	Selection operator	Tournament selection [21]
4	Mutation operator	Extended power mutation [21]
5	Crossover operator	Laplace crossover [21]
6	Probability of mutation P_M	0.01
7	Probability of crossover P_K	0.8
8	Elite count	1
9	Population size	20
10	Max generations	100
11	Max stall generations	10
12	Function tolerance	0.01

where each $\alpha_{i_g}^h$ takes its values in the range from 1 to $|m_i|$, with respect to the sequence number of the selected alternative component from m_i; Θ_g is a set of genotypes (population of individuals), which belong to the gth generation, $H = |\Theta_g|, H << p$. The inverse mapping $C^{-1}: \Lambda \to S$ converts the genotype of a stack into its corresponding phenotype. Considering the above introduced notation, the initial problem (2) with the use of the genetic algorithm transforms into the following problem:

$$s_G^* = \underset{s_G^h, h=\overline{1,|\Theta_G|}}{\operatorname{argmin}} \Psi\left(\omega^k, s_G^h\right), \tag{3}$$

where G is the last population of individuals before the genetic algorithm stops.

Thereby, the algorithm of genetic search for the solution of problem (3) consists of the following steps:

1) Create the initial population: assign $g = 1$; generate N random genotypes constituting the initial population $\Theta_1 = \{\zeta_1^1, \zeta_1^2, \ldots, \zeta_1^H\}$, get the corresponding choice of the software components $s_1^j = C^{-1}(\zeta_1^j)$ for each ζ_1^h, perform the computational experiment and calculate the value vector of the integral criterion (1) for each individual in the population $\mu = (\mu_1, \mu_2, \ldots, \mu_H)$, $\mu_i = \Psi(\omega^k, s_1^j,)$; set $\mu_{min} = \min_{j=\overline{1,H}} \mu_j$.
2) Start creating the next generation: assign $\kappa = 1$.
3) Select the first parent: assign $g = g + 1$; using the specified selection method, choose $\zeta_g^{'\kappa}$ individual as the first parent.
4) Crossing-over: using the specified selection method, choose $\zeta_g^{''\kappa}$ individual as the second parent. With the probability P_K, cross over the parents $\zeta_g^{'\kappa}$ and $\zeta_g^{''\kappa}$ using the specified crossing-over operator. Mark the result (the child) as $\zeta_g^{'\kappa}$.
5) Mutation: with the probability P_M, act on the individual $\zeta_g^{'\kappa}$ with the specified mutation operator.
6) Create the next child: assign $\kappa = \kappa + 1$; if $\kappa = H$ then go to step 7, else go to step 3.

7) Select the elite individual: from the population Θ_g, the individual ζ_g^i with the lowest value of the quality criterion $\mu_i = \min_{j=\overline{1,H}} \mu_j$ is selected.

8) Complete creating next generation: create the population $\Theta_{g+1} = \{\zeta'^1_g, \zeta'^2_g, \ldots, \zeta'^H_g\}$ of individuals selected earlier; for each ζ'^{κ}_g get the corresponding choice of the software components $s'^j_g = C^{-1}(\zeta'^j_g)$, perform the computational experiment and compute the value vector of integral criterion (1) $\mu' = (\mu'_1, \mu'_2, \ldots, \mu'_H)$, $\mu'_i = \Psi(\omega^k, s'^j_g)$; set $\mu_{min} = \min_{j=\overline{1,H}} \mu_j$.

9) Stop condition check: if no termination condition is met then go to step 3, else issue the solution corresponding to the $\mu_{\min}$ as the answer and terminate the genetic search.

In the general case, plenty of methods including tournament selection, roulette wheel method, ranking method, uniform ranking, sigma – clipping, and modifications of these methods can be used as a selection operator. During the crossing-over process, a new individual is created by exchanging subsets of parameters between two parents. The mutation operator changes the genotype of an individual in a predetermined way.

However, when solving integer-constrained problems such as the one considered in this paper, the special set of genetic operators is used to produce the integer-valued genes. Those operators were described in detail in [20].

The algorithm can be stopped in the following cases: it reaches the limit number of generations, upon reaching the limit number of stall generations (the best fitness value among such consecutive generations does not change), when the change of the average fitness for a number of consecutive generations is less than the established threshold, at the request of the user, in other cases defined by the developer.

To implement the genetic algorithm in solving the problem (3) the ga library from the MATLAB Global Optimization Toolbox was used. The experiment was carried out on a virtual machine running Ubuntu 16.04 LTS, in which the Node.js 10.15.3 runtime was deployed. Creating a virtual machine is carried out using the Vagrant virtual development environment configuration tool. Ubuntu 16.04 LTS, in which the MATLAB R2018a system was installed, was also used as the Host operating system. It should be noted that the use of the virtual machine helps to ensure the reproducibility of the computational experiment, to avoid unrecorded changes in parameters, to reduce the influence of other random factors on the results of the experiment.

When the virtual machine starts, the reading of the functions.json is performed. That file is generated by the genetic search program. The functions.json file is a json representation of the framework configuration under study $s_i = C^{-1}(\zeta_i)$ which defines a set of alternative Node.js components for the experiment. The experiment consists of two main stages: the initialization of the components and the execution of the experimental algorithm with those components.

V. Results

As a result of the implementation of 11 generations of genetic search, the solution was found for the problem (3), corresponding to the value of the integral criterion of 0.242436. The average value of the integral criterion in the terminal generation was 0.284875. Genetic search took 71 seconds and ended when the specified threshold of convergence of the algorithm was reached (Function tolerance). Experimental measurements for the terminal generation of genetic search are presented in the Table V for the criteria $r_1^{k,j} \ldots r_6^{k,j} (k=1)$ and in the Table VI for the criteria $r_7 \ldots r_{10}^{k,j}, r_{12} \ldots r_{14}^{k,j}, \Psi(k=1)$. The r_{11} criterion is equal to zero for all the individuals in the terminal generation. The effective choice of components s_G^* is identified in the Table VII.

The graph of genetic search, reflecting the solution process for the problem of minimizing the integral quality criterion Ψ, is shown in the Fig. 3.

It should be noted that the slight "oscillation" in the genetic search graph was due to the unavoidable measurement noise caused by small variations of the real-time execution of the same process by the machine.

TABLE. V. Experimental Measurements for the Terminal Generation of the Genetic Search for the Criteria $r_1^{k,j} \ldots r_6^{k,j} (k=1)$

j	$r_1^{k,j}$	$r_2^{k,j}$	$r_3^{k,j}$	$r_4^{k,j}$	$r_5^{k,j}$	$r_6^{k,j}$
1	0.24	0	0.0754	0.6816	0.071	0.0873
2	0.24	0	0.0782	0.5243	0.0519	0.0672
3	0.28	0	0.1036	0.6291	0.0666	0.0807
4	0.2	0.4	0.1036	0.4719	0.055	0.0667
5	0.12	0.8	0.1032	0.5243	0.0565	0.0668
6	0.24	0	0.2392	0.6332	0.2689	0.1717
7	0.12	0.8	0.0766	0.5243	0.0584	0.0664
8	0.08	1.2	0.0782	0.5767	0.0564	0.0754
9	0.2	0.4	0.2388	0.6332	0.2689	0.1854
10	0.24	0	0.0766	0.6291	0.0586	0.0665
11	0.16	0.8	0.0774	0.5767	0.057	0.0674
12	0.2	0	0.1061	0.5767	0.0675	0.0892
13	0.16	0.4	0.1303	0.5243	0.0918	0.0754
14	0.16	0.8	0.1298	0.5243	0.0817	0.0667
15	0.16	1.2	0.2654	0.7381	0.3004	0.1714
16	0.2	0	0.0762	0.5243	0.0657	0.0748
17	0.24	0	0.1016	0.6291	0.0683	0.0829
18	0.2	0	0.1303	0.6291	0.0679	0.0748
19	0.2	0	0.1032	0.4719	0.0634	0.0745
20	0.2	0	0.1032	0.5243	0.0831	0.0668

Normalized, rounded to 4 decimal places, experimental measurements of the partial quality criteria in the terminal generation of genetic search. The measurements corresponding to the effective solution are **highlighted**

TABLE. VI. EXPERIMENTAL MEASUREMENTS FOR THE TERMINAL GENERATION OF THE GENETIC SEARCH FOR THE CRITERIA $r_7 ... r_{10}^{k,j}, r_{12} ... r_{14}^{k,j}, \Psi(k=1)$

j	$r_7^{k,j}$	$r_8^{k,j}$	$r_9^{k,j}$	$r_{10}^{k,j}$	$r_{12}^{k,j}$	$r_{13}^{k,j}$	$r_{14}^{k,j}$	Ψ
1	0.4283	0	0.4	0.2621	0.1709	0.96	0.4549	0.2682
2	0.5067	0.4	0	0.2376	0.1678	0.96	0.457	0.2583
3	0.4705	0.4	0	0	0.1726	0.96	0.4222	0.251
4	0.4299	0.8	0	0	0.1699	0.96	0.6203	0.3034
5	0.5538	0	0	0	0.1699	0.96	0.4606	0.2751
6	0.4578	0	0.4	0	0.162	0.96	0.4051	0.2757
7	0.4251	0.4	0	0.2376	0.1675	0.96	0.4323	0.3068
8	0.5054	0.4	0	0	0.1673	0.96	0.4534	0.3307
9	0.4913	0.4	0	0.2335	0.1599	0.96	0.4067	0.3244
10	0.3863	0.4	0	0.2417	0.4296	0.96	0.4274	0.2741
11	0.5028	0.4	0	0.2335	0.1696	0.96	0.4512	0.3199
12	0.4072	0.4	0	0.2294	0.1696	0.96	0.44	0.2552
13	0.4091	0.4	0	0	0.1703	0.96	0.4133	0.2654
14	0.5337	0.4	0	0	0.1711	0.96	0.469	0.3087
15	0.4411	0	0.4	0.2458	0.1596	0.96	0.4016	0.393
16	0.4049	0.4	0	0.2376	0.1728	0.96	0.4222	0.2477
17	0.4867	0.4	0	0.512	0.4318	0.96	0.4596	0.306
18	0.4159	0	0.4	0	0.1701	0.96	0.4152	0.2424
19	0.3574	0.4	0	0.2376	0.1742	0.96	0.482	0.2467
20	0.5173	0.4	0	0	0.1671	0.96	0.4737	0.2447

Normalized, rounded to 4 decimal places, experimental measurements of the partial quality criteria in the terminal generation of genetic search. The measurements corresponding to the effective solution are highlighted

However, as the genetic search proceeds, the genotype of the best choice begins to predominate from generation to generation (this can be seen as decreasing average value of Ψ) and the best choice is identified as the number of experiments attributable to the best genotype increases, which eliminates random factors in the assessment of this genotype.

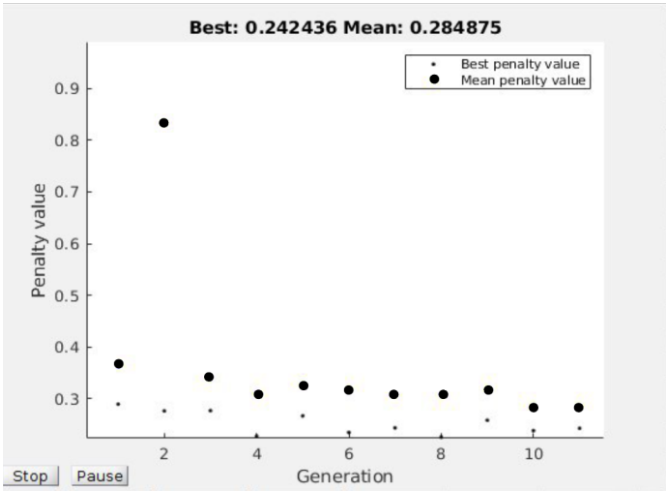

Fig. 3. Graph of Genetic Search. Penalty Value is Equal to the Ψ for the Specific Stack being Assessed in the Experiment. Best Penalty Value is the Minimal Ψ in the Generation. Mean Penalty Value is the Average Ψ in the Generation.

TABLE. VII. THE EFFECTIVE CHOICE OF COMPONENTS s_G^*

Genotype	**Phenotype**	
	Functional feature	**Component**
[2 3 2 1 1 1 2 2 1 1]	Filter	Underscore
	Map	Underscore
	First	Underscore
	PathResolve	JavaScript language tools
	StringReplace	JavaScript language tools
	ZipCompress	Adm–zip
	HashMD5	Md5
	FsRead	JavaScript language tools
	FsReaddir	Fs–extra
	FsReaddirRecursive	Recursive–readdir

VI. DISCUSSION

In the process of the development of the digital economy, the majority of data collection and exchange services are implemented using digital platforms and web portals. Those service range from public and municipal services and banking platforms to home control systems for household appliances. An important task of such systems development is to ensure effective interaction of components within the software system. Modern component-oriented development environments and interaction technologies provide a set of development tools, significant in number and approximately the same in functionality. Alternative technologies can bring different values of performance indicators depending on the functional features of the software system.

The paper deals with the framework architectural approach to the construction of software systems. A framework is a common form of template software structure that allows the developer to unify the software development process by combining the permanent piece of software (the framework) that does not change from configuration to configuration, and some plug-in components that are compatible with the permanent part. A component is a piece of software that has a specific interface and explicit context dependencies. Thus, the software system is created by selecting the appropriate components for the corresponding framework, while it is possible to use alternative sets of components that have a similar interface to implement similar functionality.

When there is a choice from a variety of components, the task of quality assessment is particularly important for the development and operation of software systems in the given conditions. The results of numerical evaluation of different variants of program interaction can be the basis for the formalization and finding the solution of the problem of choosing an effective set from a variety of alternatives.

In accordance with ISO/IEC 25041:2014, measurement procedure should be able to provide measurement to the quality characteristics of software. It should ensure that the measurements are made with the sufficient accuracy to determine the criteria and make the necessary comparisons. In the developed method, the physical execution time of the invariant algorithm of the experiment are measured with an

accuracy of 10–9 seconds, the measurements of the amount of memory occupied are carried out with an accuracy of 1 Byte, the measurement of processor time spent on the execution of the experimental algorithm are carried out with an accuracy of 10^{-6} sec.

VII. CONCLUSIONS

A methodology of effective selection of software components based on experimental estimates of the criteria and the genetic algorithm was created. The methodology was experimentally approved in the task of effective selection of Node.js components to implement a specific set of functional features in accordance with the specified quality criteria. The configuration and parameters of the experimental stand as well as the parameters of the genetic algorithm were presented in the paper. The integral quality criterion was formulated, which allows to consider the contribution of a set of 14 experimentally evaluated partial quality criteria to the overall assessment of the effectiveness of the choice of software components. Experimental estimates of the quality criteria for the terminal generation of genetic search were also given in the paper. The effective selection of the software components was identified.

Future work will be aimed at evaluating the relative importance and the mutual influence of the partial quality criteria to reduce the dimension of the criteria space as well as considering some security criteria when choosing the set of software components.

REFERENCES

[1] Lun L., X. Chi and H. Xu, "Coverage criteria for component path-oriented in software architecture", Engineering Letters, vol. 27, no. 1, pp. 40-52, 2019.

[2] S. Gerasimou, R. Calinescu and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering", Automated Software Engineering, vol. 25, issue 4, pp. 785-831, 2018.

[3] A., Gizas S. Christodoulou and T. Papatheodorou, "Comparative evaluation of javascript frameworks," In WWW'12 Companion Proceedings of the 21st International Conference on World Wide Web, 2012, pp. 513-514.

[4] J. Ferreira, "A javascript framework comparison based on benchmarking software metrics and environment configuration," Masters dissertation, DIT, 2018. [Online]. Available: https://arrow.dit.ie/cgi/viewcontent.cgi?article=1142&context=scschcomdis.

[5] P. Kolyasnikov, E. Nikulchev, I. Silakov, D. Ilin and A. Gusev, "Experimental evaluation of the virtual environment efficiency for distributed software development," International Journal of Advanced Computer Science and Applications, vol. 10, no. 5, pp. 309-316, 2019.

[6] B. M. Basok, V. N. Zakharov and S. L. Frenkel, "Iterative approach to increasing quality of programs testing," Russian Technological Journal, vol. 5, no. 4, pp. 43-12, 2017.

[7] L. D. Chambers, "Practical handbook of genetic algorithms: Complex coding systems". CRC press, 2001.

[8] S. Bilgaiyan, K. Aditya, S. Mishra and M. Das, "Chaos-based modified morphological genetic algorithm for software development cost estimation," in Advances in Intelligent Systems and Computing, vol. 710, pp. 31-40, 2010.

[9] L. R. Still and L. S. Indrusiak, "Memory-Aware genetic algorithms for task mapping on hard real-time networks-on-chip," in 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Cambridge, 2018, pp. 601-608.

[10] L. Liu, M. Zhang, R. Buyya and Q. Fan, "Deadline constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," Concurrency and Computation: Practice and Experience, vol. 29, no. 5, p. e3942, 2017.

[11] D. B. Mishra, R. Mishra, A. A. Acharya and K .N. Das, "Test data generation for mutation testing using genetic algorithm," in Advances in Intelligent Systems and Computing, vol 817, pp. 857-867, 2019.

[12] R. Jain and A. Sharma, 'Assessing software reliability using genetic algorithms,' The Journal of Engineering Research, vol. 16, no. 1, pp. 11-17, 2019.

[13] A. C. Kumari, K. Srinivas and M. P. Gupta, "Software module clustering using a hyper–heuristic based multi–objective genetic algorithm," in 2013 3rd IEEE International Advance Computing Conference (IACC), Ghaziabad, 2013, pp. 813-818.

[14] H. Ahuja, Sujata, U. Batra, "Performance Enhancement in Requirement Prioritization by Using Least-Squares-Based Random Genetic Algorithm," in Studies in Computational Intelligence, vol. 713, pp. 251-263, 2017.

[15] A. Ouni, M. Kessentini, H. Sahraoui and M. S. Hamdi, "The use of development history in software refactoring using a multi–objective evolutionary algorithm," in 15th annual conference on Genetic and evolutionary computation (GECCO '13), 2013, pp. 1461-1468.

[16] S. Kaiafa and A. P. Chassiakos, "A genetic algorithm for optimal resource-driven project scheduling," Procedia Engineering, vol. 123, pp. 260-267, 2015.

[17] W. Almadhoun and M. Hamdan, "Optimizing the Self-Organizing Team Size Using a Genetic Algorithm in Agile Practices," Journal of Intelligent Systems, 2018. Available: https://doi.org/10.1515/jisys-2018-0085.

[18] P. Devarasetty and S. Reddy, "Genetic algorithm for quality of service based resource allocation in cloud computing," in Evolutionary Intelligence, pp. 1-7, 2019. Available: https://doi.org/10.1007/s12065-019-00233-6.

[19] C. Jatoth, G. R. Gangadharan, U. Fiore and R. Buyya, "QoS-aware Big service composition using MapReduce based evolutionary algorithm with guided mutation," Future Generation Computer Systems, vol. 86, pp. 1008-1018, 2018.

[20] C. Jatoth, G. R. Gangadharan and R. Buyya, "Optimal Fitness Aware Cloud Service Composition using an Adaptive Genotypes Evolution based Genetic Algorithm,", Future Generation Computer Systems, vol. 94, pp. 185-198, 2019.

[21] K. Deep, K. P. Singh, M. L. Kansal and C. Mohan, "A real coded genetic algorithm for solving integer and mixed integer optimization problems," Applied Mathematics and Computation, vol. 212, no. 2, 505-518, 2009.