Effective Selection of Software Components Based on Experimental Evaluations of Quality of Operation

Alexander Gusev, Dmitry Ilin, Pavel Kolyasnikov and Evgeny Nikulchev, *Member, IAENG*

***Abstract* — The paper shows an experimental approach to the selection of a set of software components based on computational experiments simulating the desired operating conditions of the software system being developed. A mathematical model is constructed, aimed at an effective selection of components from the available alternative options. The model and process of components selection are introduced and applied to the case of selecting Node.js components for the development of the Digital Psychological Tools for Conducting the Large-Scale Psychological Research. The aim of the platform development is to facilitate the countrywide simultaneous online psychological surveys in schools in the conditions of unstable Internet connection and the large variety of desktop and mobile client devices, running different operating systems and browsers. The module, which development is considered in the paper, should provide the functionality for archiving and checksum verification of the survey forms and graphical data. With the experimental approach proposed in the paper, the effective set of components was identified based on evaluations of 14 quality of operation indicators. To simulate the desired operating conditions and to guarantee the reproducibility of the experiments, the virtual infrastructure was configured, and the genetic algorithm was applied to reduce the number of experiments with the unpromising sets of software components. The application of the genetic algorithm led to the reproducible results of components selection after 220 experiments instead of 1080 experiments needed by the exhaustive search algorithm. The suggested approach can be widely used for effective selection of software components for distributed systems operating in the given conditions at the stage of their development.**

***Index Terms* — quality of systems and programs, software systems development, frameworks, genetic algorithm, evolutionary computation, computational experiments.**

Manuscript received June 18, 2019; revised February 18, 2020.

This work was supported in the Ministry of Science and Higher Education of the Russian Federation, project 8.2321.2017 and grant of RFBR 17-29-02198.

Alexander Gusev, is with the MIREA – Russian Technological University, 78 Vernadsky Avenue, Moscow 119454, Russia (e-mail: alexandrgsv@gmail.com).

Dmitry Ilin, is with the MIREA – Russian Technological University, 78 Vernadsky Avenue, Moscow 119454, Russia (e-mail: i@dmitryilin.com).

Pavel Kolyasnikov, is with the Russian Academy of Education, 8 Pogodinskaya Street, Moscow 119121, Russia (e-mail: pavelkolyasnikov@gmail.com).

Evgeny Nikulchev, is the professor of the chair of Systems Control and Modelling of MIREA – Russian Technological University, 78 Vernadsky Avenue, Moscow 119454, Russia (corresponding author, phone no. +74957705012, e-mail: nikulchev@mirea.ru).

I. Introduction

In the integrated development and design of distributed highly loaded systems the question of proper technological solutions selection always arises [1]. Moreover, technologies are often already implemented in several components [2]. Thus, the software designer should select the components as best as possible. The basis for the selection can be formulated as a set of parameters and criteria [3]. But it is necessary to answer the question how to evaluate the value of those parameters [4]. Although expert assessments and load test reports for the single components are widely used, they may be irrelevant to the specific operating conditions the software system made of those interacting components is intended for (infrastructure, number of users, communication channels, operating systems, computing resources, other interacting components). Thus, the main task is to formulate a set of parameters and methods for their assessment for the effective selection of components considering their interaction and the desired operating conditions of the software system being designed.

Ineffective selection of software components can lead to one of them blocking access of other components to the shared resources, such as a database. Another example is choosing the least resource-intensive set of interacting software components when client hardware resources are limited [5]. This is especially important when developing Internet platforms and decentralized control systems for mobile agents. Also, the effective selection of components is necessary to ensure the guaranteed data delivery over the limited or unstable channel through archiving, checksum verification, encryption. Many other examples can be given in which the selection of components is the crucial decision to provide the required quality of operation of the software system.

For the popular software development frameworks, for example Node.js, the number of available components is measured in millions, while lots of them implement similar functionality. For instance, the Node.js components Lodash and Underscore provide 114 similar functions. Thus, the preference of a component should be based on an experimental study of the quality of its operation in a stack together with the components implementing the rest of the functionality of the software system to ensure the components do not block each other's access to the shared resources, do not consume too much of resources, guarantee

the data delivery over the unstable channel and fulfill all the other requirements to the quality of operation.

In the previous decades, a significant amount of research was devoted to the development of an optimal modular architecture for the component-oriented programming [6], including quality assessment of modular software architecture [7], increasing the productivity of modular software by various methods, including clustering methods [8], genetic algorithms [9] and other evolutionary algorithms [10]. These studies were aimed at a priori optimization of the component-oriented software architecture in terms of the structural connectivity of the modules, preliminary expert assessments of the functional completeness of the selected components. The modern spread of the framework-based architecture requires the new approach for numerical measurements of the quality of software products, i.e. the degree to which the product meets the stated and implied needs when used under specified conditions.

Considering the exhaustive nature of the problem, as well as the resource consumption and time duration of the experiments to evaluate the quality of operation for each component selection, it is advisable to use evolutionary algorithms that are highly convergent and reduce the number of experiments with unpromising sets of software components. Evolutionary algorithms have proven themselves in the problems of optimizing the modular structure of software, as well as in other tasks of multi-criteria optimization in the field of software development, i.e. optimization of software development efforts [11], optimizing the allocation of computing resources [12], generating optimal test data sets [13], evaluation [14] and increasing [15] of software reliability, optimizing the software module clustering [16], release planning [17], separating the implementation of functionality into software and hardware parts [18], refactoring [19], in developing web services with dynamic selection of components [20].

The aim of this article is to offer an experimental approach to the effective selection of software components based on evaluations of the quality of operation in a virtual infrastructure that simulates the operating conditions of the software system being developed and allows the software developer to identify the most effective stack for the given operational conditions.

The proposed approach is illustrated with the case of development of the Digital Psychological Tools (DigitalPsyTools.ru) for Conducting the Large-Scale Psychological Research at the Russian Academy of Education [21, 22].

The article consists of seven sections. The first is Introduction. In the second section the model of selection is presented. The third section describes the use of the model in software engineering for the specific case of design of the Digital Platform for Conducting the Large-Scale Psychological Research. The fourth section presents the experimental methodology. The fifth section provides the results of the effective selection of components for the Digital Platform. The sixth section discusses the results of the implementation of the proposed approach. The seventh section concludes the article.

II. Model

At the first stage of constructing the model of components selection the n functional requirements q_i, $i=\overline{1,n}$ to the software system should be identified as well as t different configurations $\omega^k, k=\overline{1,t}$ of the virtual infrastructure, representing the set of desired operating conditions of the software system. The software developer identifies then the set of M software components available for the research. Each component should implement at least one of the requirements q_i. and may be provided by various third-party providers. The subset of alternative software components from M, capable of implementing the requirement q_i, is denoted as $m_i, i=\overline{1,n}$. The sets of software components, in which for every functional requirement q_i, $i=\overline{1,n}$ there exists at least one software component from M, are defined as stacks $s^j, j=\overline{1,p}$. S is the set of all the possible stacks. To evaluate the quality of operation for a stack, the f experimentally evaluated partial quality indicators $r_\xi^{k,j}$, $\xi=\overline{1,f}$ are introduced. Their values belong to the space $\mathbb{R}^{\mathrm{f}}$. Thus,

$$\forall \omega^k : s^j \to R^{k,j} \in \mathbb{R}^{\mathrm{f}},$$
$$R^{k,j} = (r_1^{k,j}, r_2^{k,j}, \ldots, r_\xi^{k,j}, \ldots, r_f^{k,j})^T, k=\overline{1,t}, j=\overline{1,p},$$

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 indicators for the configuration ω^k of the virtual infrastructure and the stack s^j being evaluated.

The integral quality indicator for the stack is defined as:

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

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

The process of the effective selection of software components based on the experimental evaluation of the quality of operation (see Figure 1) for the chosen configuration of the virtual infrastructure ω^k is aimed at the selection of the technology stack s^* satisfying the following condition:

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

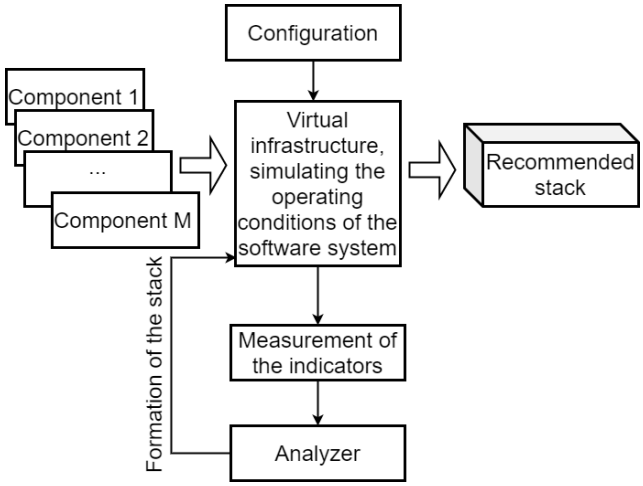

Fig. 1. The process of the effective selection of software components based on the experimental evaluation of the quality of operation

III. The Use of the Model in Software Engineering

With the model and process introduced above, let us consider the case of selecting Node.js components for the development of the Digital Psychological Tools for Conducting the Large-Scale Psychological Research in Russia.

The aim of the platform design is to facilitate the countrywide simultaneous online psychological surveys in schools. Due to the unstable Internet connectivity in the villages and remote territories, it is crucial to provide the guaranteed data delivery even if the communication channel suddenly breaks down. The questionnaire includes the description structure and may include additional resources such as images. Figure 2 shows the standard scheme of questionnaire transmission without the resource preloading. All the images are downloaded from the server in the survey process.

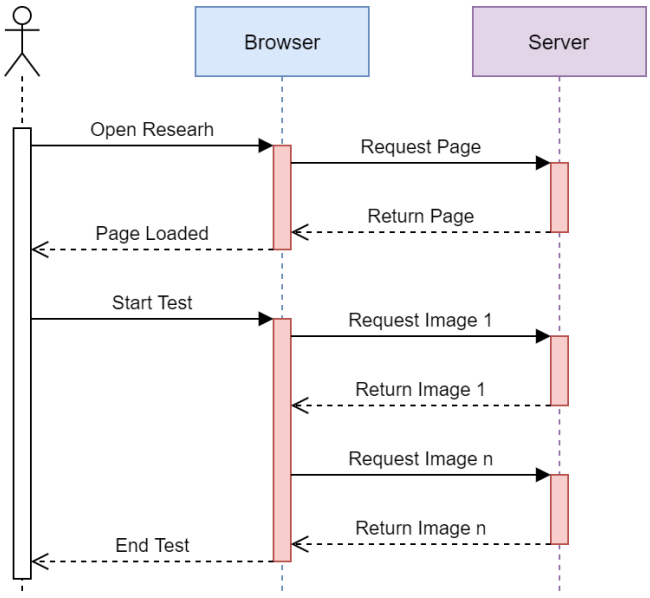

Fig. 2. Scheme of questionnaire transmission without the resource preloading

In the case of an unstable Internet connection, the participant waits for an image downloading to be able to answer the question. This may significantly affect the reliability of the research. Therefore, the batch approach was chosen to preload the questionnaire resources from the platform server with a single file. An archive with the data acts as the batch. Figure 3 shows the transmission of the questionnaire with the preloading of all the resources.

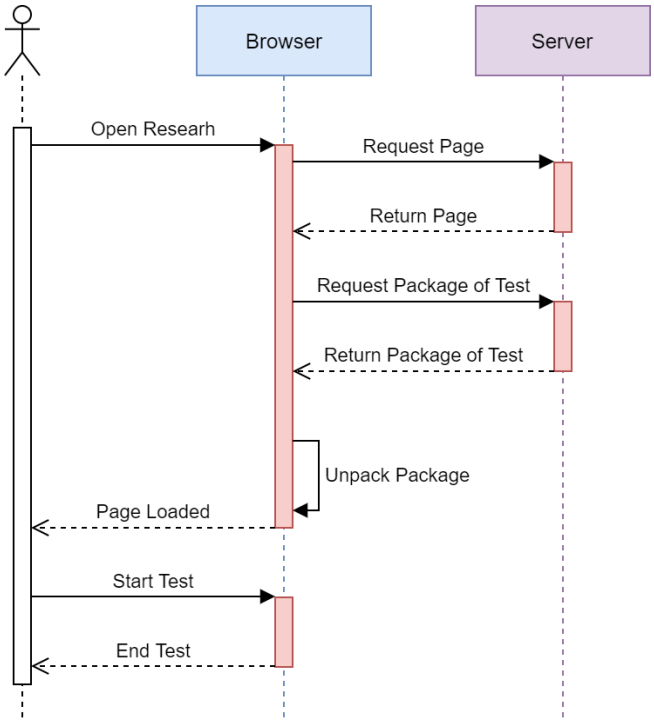

Fig. 3. The questionnaire transmission with the batch resource preloading

The batch approach provides the following advantages: it becomes necessary to check the checksum of a single file only, the number of HTTP requests to the server is reduced, an Internet connection is only required to download the questionnaire and send the results back to the platform server, a poor Internet connection will not affect the research process.

The chosen architecture is a good option for similar software systems that need to ensure stable operation without constant access to the Internet. However, the software components that are used to implement the archiving may show different performance in a stack of software components implementing the rest of the system while operating in the desirable conditions. Thus, there is a need to select the archiving components and the rest of the components effectively, experimentally evaluating various options together.

To meet the goal of the surveys facilitating, the following set of functional requirements and alternative Node.js components was considered: q_1 – "sequentially check all the elements of the array for compliance with the condition and return an array consisting of elements, for which the check gave the value "True", alternative components: Lodash, Underscore; q_2 – "apply the specified function to all the elements of an array, thereby return a new array consisting of the transformed elements", alternative components: "Lodash", "Underscore", JavaScript language

tools; q_3 – "return the first element of an array", alternative components: Lodash, Underscore; q_4 – "generate the full path to the file or directory based on the specified array of path elements", alternative components: "Path"; q_5 – "find and replace a substring in the string passed", alternative components: JavaScript language tools; q_6 – "perform archiving of the transferred file array and return the generated Zip-archive", alternative components: "Adm-zip", "Jszip", "Zipit"; q_7 – "calculate the MD5 hash for the specified data set", alternative components: "Hasha", "md5", "Ts-md5"; q_8 – "read the data from a file", alternative components: "Fs-Extra", "Fs"; q_9 – "read the contents of a directory, returning an array of file and subdirectory names in the directory", alternative components: "Fs-extra"; q_{10} – "recursively read the contents of a directory and return an array of file and subdirectory names in the directory", alternative components: "Recursive-readdir".

Thus, $n=10, p=216$.

The considered configuration $\omega^k, k=t=1$ of the virtual infrastructure included host CPU: Intel® Core ™ i7–7700; number of cores: 4; number of logical processors: 8; clock frequency: 3.60 GHz; host RAM: 12 GB; host operating system: Ubuntu 16.04 LTS; Vagrant version: 2.2.4; Node.js version: 10.15.3; 2 virtual CPU cores; 2.0 GB of virtual RAM; virtual machine operating system: Ubuntu 16.04 LTS; provisioning software: Ansible; file exchange tools for the virtual machine: NFS-server + BindFS inside the virtual machine; additional system software: git, make, htop, iotop, rsync, node-gyp.

The evaluation of the quality of operation is performed with respect to the $f=14$ partial quality indicators: $r_1^{k,j}$ 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.

The reason for choosing these indicators was that the client devices at schools across the country were represented by a wide range of mobile and desktop devices running various operating systems (see Figure 4) so the selection of the software stack should be aimed at minimizing the hardware resource consumption and facilitating the survey operation at the devices with the lowest hardware features.

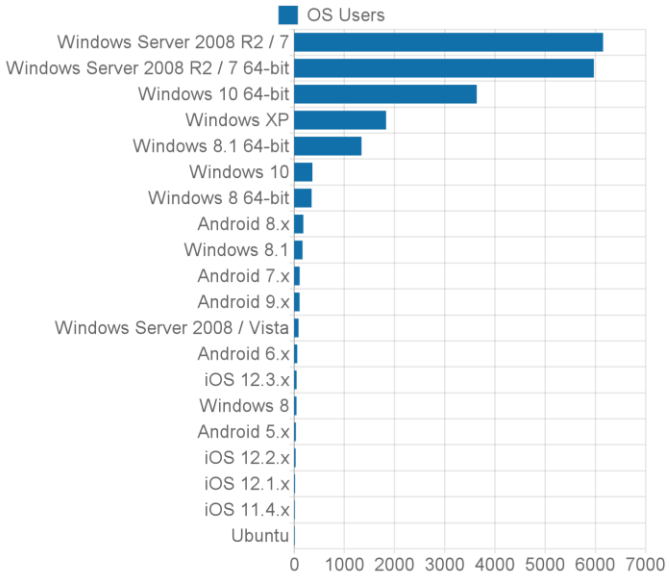

Fig. 4. The amounts of clients running different operating systems

The set of indicators coincides with all the available indicators provided by the Node.js global object "process".

The platform development team set up the weighting factors for the indicators as $w_2 = w_{11} = 0.08$; $w_\xi = 0.07 (\xi = 1, \xi = \overline{3,10}, \xi = 12)$ to reach a compromise between them and prevent too much influence of a particular indicator on the decision.

When conducting the experiment the indicators $r_\xi^{k,j}$ $(\xi = \overline{1,14}, k = \overline{1,t}, j = \overline{1,p})$ 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, which satisfies the criterion (2).

IV. Experimental Methodology

The selection of a software stack can be carried out with a complete search algorithm for the set of all the possible software stacks. However, due to the unavoidable measurement noise caused by non-linear delays associated with access to information on the hard drives and network devices during the execution of the experiment algorithm, it is not possible to use the results of a single measurement of

quality indicators.

It was found in the experiments that reproducible results of selecting a software stack can be obtained after at least 5 consecutive experimental evaluations of each stack with averaging the resulting integral indicator (1).

Thus, 1080 runs of the experiment algorithm should be carried out to evaluate all the 216 possible stacks. Another option is using an evolutionary algorithm, for example, the genetic algorithm to reduce the number of evaluations through the rejection of "unpromising" stacks at an early stage and reevaluation the fitting stacks in the subsequent generations. The integral indicator (1) can be used as a fitness function for the genetic algorithm.

The experimental methodology with the use of a genetic algorithm in MATLAB environment is presented in Figure 5.

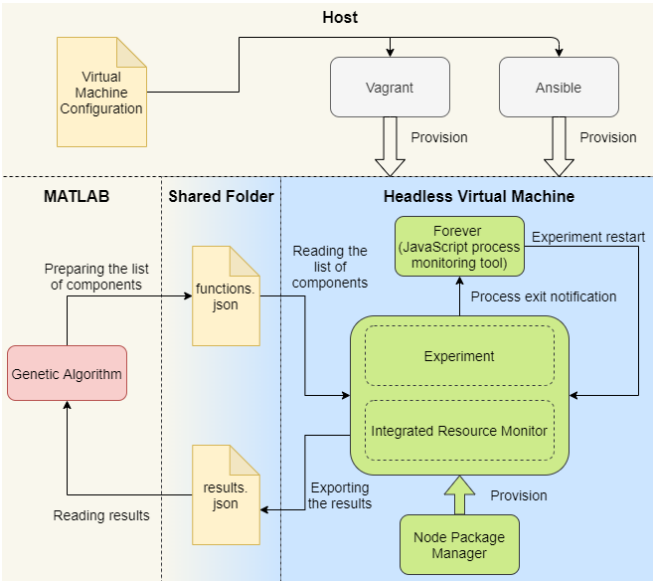

Fig. 5. Experimental methodology

At the beginning of the experiment the genetic algorithm generates the *functions.json* file, which is a json representation of the stack under evaluation which defines a set of alternative Node.js components to implement the functional requirements in the experiment.

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

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

where each $\alpha_{i_g}^h$ takes its values in the range from 1 to $\left|m_i\right|$, 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 g^{th} generation, $H = \left|\Theta_g\right|, H << p$. 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:

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

here 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. *Creating the initial population*: assign $g = 1$; generate N random genotypes constituting the initial population $\Theta_1 = \{\zeta_1^1, \zeta_1^2, \ldots, \zeta_1^H\}$, for each ζ_1^h get the corresponding choice of the software components $s_1^j = C^{-1}(\zeta_1^j)$, 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 $\mu_{min} = \min_{j=1,H} \mu_j$.

2. *Start creating the next generation*: assign $\kappa = 1$.

3. *Select of 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 $\Theta_{g+1} = \{\zeta_g^{'1}, \zeta_g^{'2}, \ldots, \zeta_g^{'H}\}$ of individuals selected earlier; for each $\zeta_g^{'\kappa}$ get the corresponding choice of the software components $s_g^{'j} = C^{-1}(\zeta_g^{'j})$, perform the computational experiment and compute the value vector of integral criterion (1) $\mu' = (\mu'_1, \mu'_2, \ldots, \mu'_H)$, $\mu'_i = \Psi(\omega^k, s_g^{'j})$; set $\mu_{min} = \min_{j=1,H} \mu_j$.

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.

The algorithm may stop when 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.

The genetic algorithm is executed with the following parameters: MATLAB version: R2018a; integer constrains: all the genes are integer-valued; selection operator: tournament selection; mutation operator: extended power mutation; crossover: Laplace crossover; probability of mutation, P_M : 0.01; probability of crossover, P_K : 0.8; elite count: 1; population size: 20; max generations: 100; max stall generations: 10; function tolerance: 0.01.

The evaluation of a stack consists of two main stages: the initialization of the components and the execution of the experimental algorithm with those components.

The integration of the components of the stack is implemented using a functional approach, which is the most convenient way of combining 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.

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 code names of the functional requirements. 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. 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 following for each path:
   - 5.1 Read all the list of files recursively.
   - 5.2 Read and load into the RAM all the files.
   - 5.3 Create a Zip-archive in the RAM.
   - 5.4 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 indicator (1). This data is obtained through the interface of the "process" object of Node.js.

V. Results

The genetic algorithm converged to the solution of the problem (3) after 11 generations, performing 220 experimental evaluations of the various software stacks. The genotype of the solution is [2 3 2 1 1 1 2 2 1 1], which corresponds to the following selection of the components to meet the functional requirements: q_1, q_2 and q_3 are implemented with the component "Underscore"; q_4, q_5 and q_8 are implemented with the JavaScript language tools; q_6 is implemented with the component "Adm-zip"; q_7 is implemented with the component "Md5"; q_9 is implemented with the component "Fs-extra"; q_{10} is implemented with the component "Recursive-readdir". The integral indicator (1) for the solution was equal to 0.242436.

Experimental measurements for the terminal generation of the genetic algorithm are presented in the Table I for the partial quality indicators $r_1^{k,j}...r_6^{k,j}(k=1)$ and in the Table II for the indicators $r_7...r_{10}^{k,j}, r_{12}...r_{14}^{k,j}, \Psi(k=1)$. The $r_{11}^{k,j}$ indicator was equal to zero for all the individuals in the terminal generation.

TABLE I
EXPERIMENTAL MEASUREMENTS FOR THE TERMINAL GENERATION OF THE GENETIC SEARCH FOR THE INDICATORS $r_1^{k,j}...r_6^{k,j}(k=1)$

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 indicators in the terminal generation of the genetic algorithm. The measurements corresponding to the effective solution are **highlighted**

The graph of genetic search, reflecting the solution process for the problem of minimizing the integral quality indicator Ψ, is shown in the Figure 6.

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

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 indicators in the terminal generation of the genetic algorithm. The measurements corresponding to the effective solution are **highlighted**

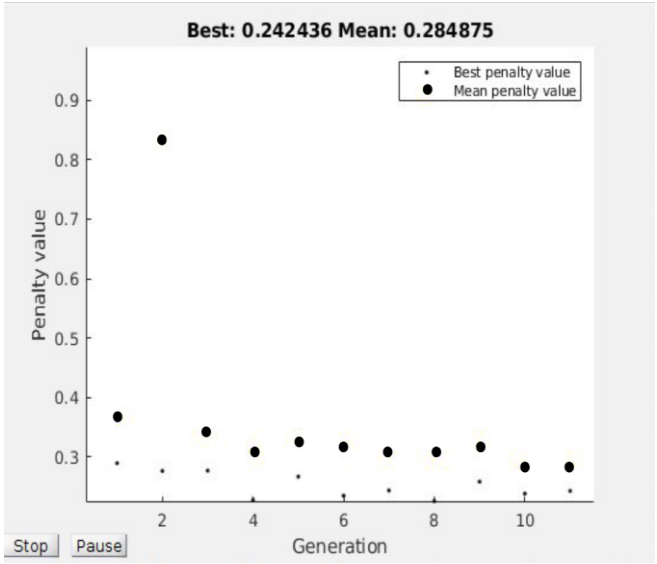

Fig. 6. Graph of genetic search. Penalty value is equal to the Ψ for the specific stack being evaluated in the experiment. Best penalty value is the minimal Ψ in the generation. Mean penalty value is the average Ψ in the generation

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 execution time by the machine.

However, as the genetic search proceeds, the genotype of the best selection begins to predominate from generation to generation which is confirmed by decreasing mean value of Ψ and the best selection is identified.

VI. Discussion

As the digital economy becomes the major application of distributed software systems providing government, municipal and banking services, traffic control, household automation etc. the problem of maintaining their quality of service in normal and peak modes turns to be the crucial one. As those distributed systems grow rapidly the wrong decisions made at the development stage can become an obstacle for their continuous operation. Modern component-oriented development environments and frameworks provide plenty of development tools, significant in their number and approximately the same in their functionality. The specific selection of such tools brings different values of performance indicators depending on the desired operating conditions of the software system being developed.

The framework architectural approach allows the rapid construction of software systems based on template solutions combining the permanent piece of software (the framework) and variable components compatible with the permanent part so the software system is made up by components selection for the appropriate framework, while it is possible to use alternative sets of components that have a similar interface to implement similar functionality.

The task of quality evaluation is particularly important for the development and operation of software systems in the desired conditions. The results of numerical evaluation of sets of software components can be the basis for the formalization and finding the solution of the problem of selecting 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

The aim of the platform development is to facilitate the countrywide simultaneous online psychological surveys in schools in the conditions of unstable Internet connection and the large variety of desktop and mobile client devices, running different operating systems and browsers. The module, which development is considered in the paper, should provide the functionality for archiving and checksum verification of the survey forms and graphical data. The 216 possible sets of software components were available to build the module. With the experimental approach proposed in the paper, the effective set of components was identified based on evaluations of 14 quality of operation indicators. To simulate the desired operating conditions and to guarantee the reproducibility of the experiments, the virtual infrastructure was configured, and the genetic algorithm was applied to reduce the number of experiments with the unpromising sets of software components. The application of the genetic algorithm led to the reproducible results of components selection after 220 experiments instead of 1080 experiments needed by the exhaustive search algorithm.

A software engineering model is proposed. The model is aimed at providing an effective choice of technologies and components used in integrated design, and based on experimental evaluations that consider the specific operating conditions and the environment of the designed distributed system.

The methodology for effective selection of software components based on experimental evaluations of their quality of operation was discussed and applied to the problem of development psychological surveys facilitating module for the countrywide Digital Psychological Tools for Conducting the Large-Scale Psychological Research. It was shown how the software developers can set the desired quality indicators and perform a search for the appropriate set of software components using the virtual infrastructure, simulating the planned operating conditions of the software system. To reduce the number of experiments the developers can use the evolutionary approach presented in the article.

References

[1] J. C. Pereira and R. de F. S. M. Russo, "Design thinking integrated in agile software development: A systematic literature review," *Procedia computer science*, vol. 138, pp. 775-782, 2018.

[2] N. Belhaj, D. Belaïd, and H. Mukhtar, "Framework for building self-adaptive component applications based on reinforcement learning,". In *2018 IEEE International Conference on Services Computing (SCC)*, IEEE, pp. 17-24, 2018.

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

[4] 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.

[5] S. Gerasimou, R. Calinescu and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering," *Automated Software Engineering*, vol. 25, no. 4, pp. 785-831, 2018.

[6] R. C. Shock and T. C. Hartrum, "A classification scheme for software modules," *Journal of Systems and Software*, vol. 42, no. 1, pp. 29-44, 1998.

[7] S. Sarkar, G. M. Rama and A. C. Kak, "API-Based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 14-32, 2007.

[8] B. Mitchell, M. Traverso and S. Mancoridis, "An architecture for distributing the computation of software clustering algorithms," In *Proceedings Working IEEE/IFIP Conference on Software Architecture*, Amsterdam, Netherlands, 2001, pp. 181-190.

[9] C. K. Kwong, L. F. Mu, J. F. Tang and X. G. Luo, "Optimization of software components selection for component-based software system development," *Computers & Industrial Engineering*, vol. 58, no. 4, pp. 618-624, 2010.

[10] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 193-208, 2006.

[11] S. J. Huang, N. H. Chiu and L. W. Chen, "Integration of the grey relational analysis with genetic algorithm for software effort estimation," *European Journal of Operational Research*, vol. 188, no. 3, pp. 898-909, 2008.

[12] Y. S. Dai and X. L. Wang, "Optimal resource allocation on grid systems for maximizing service reliability using a genetic algorithm," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 1071-1082, 2006.

[13] C. C. Michael, G. McGraw and M. A. Schatz, "Generating software test data by evolution," *IEEE Transactions on Software Engineering*, vol. 27, no. 12, pp. 1085-1110, 2001.

[14] S. H. Aljahdali and M. E. El-Telbany, "Software reliability prediction using multi-objective genetic algorithm," In *2009 IEEE/ACS International Conference on Computer Systems and Applications*, Rabat, 2009, pp. 293-300.

[15] R. Feldt, "Generating diverse software versions with genetic programming: an experimental study," *IEE Proceedings – Software*, vol. 145, no. 6, p. 228, 1998.

[16] 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.

[17] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach," *Information and Software Technology*, vol. 46, no. 4, p. 243-253, 2004.

[18] D. Saha, R. S. Mitra and A. Basu, "Hardware software partitioning using genetic algorithm," In *Proceedings Tenth International Conference on VLSI Design*, Hyderabad, India, 1997, pp. 155-160.

[19] 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 *GECCO '13: Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pp. 1461-1468, 2013.

[20] K. Shuang, S. Yu and S. Su, "TTS-Coded Genetic Algorithm for QoS-driven web service selection," *In 2009 IEEE International Conference on Communications Technology and Applications*, Beijing, 2009, pp. 885-890.

[21] E. Nikulchev, D. Ilin, P. Kolyasnikov, V. Ismatullina, I. Zakharov and S. Malykh, "Development of the Open Digital Platform for Conducting the Large-Scale Psychological Research," *Russian Foundation for Basic Research Journal*, no. 4 (104), pp. 114-119, 2019.

[22] 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.