ISSN 0005-1179 (print), ISSN 1608-3032 (online), Automation and Remote Control, 2023, Vol. 84, No. 10, pp. 1177–1185.

TOPICAL ISSUE

A Comprehensive Software Verification Technology for Onboard Control Systems of Spacecraft

V. V. Kul'ba[*,a], **E. A. Mikrin**[*†], **B. V. Pavlov**[*,b], **and S. K. Somov**[*,c]

Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia
e-mail: [a]*kulba@ipu.ru,* [b]*pavlov@ipu.ru,* [c]*ssomov2016@ipu.ru*

Received June 19, 2023
Revised July 18, 2023
Accepted August 2, 2023

Abstract—This paper conceptualizes the main principles of comprehensive software verification for an onboard spacecraft control system. An optimal comprehensive verification strategy for onboard software is selected by rigorously stating and solving the corresponding optimization problem. Software verification methods with functional correctness indicators are proposed.

Keywords: spacecraft, software, onboard control system, comprehensive software verification

DOI: 10.25728/arcRAS.2023.99.27.001

1. INTRODUCTION

A wide variety of tasks performed by spacecraft, extreme conditions of spacecraft operation, and different technologies and protocols of information interaction between onboard hardware elements and software, including various sensors and indicators, determine the need to create new approaches, methods, and technologies to support R&D works in the field of advanced space technology. The papers [1, 5], the preprint [2], and the books [3, 4] were devoted to the development and implementation of modeling methods in the aerospace industry. An important place therein was occupied by the issues of digital modeling, a relevant method for studying different aspects in the operation of onboard spacecraft control systems (OSCSs), including the design, development, and verification of their software [6, 7]. According to the experience of application of modern organizational, methodological, and technical solutions used to verify OSCS software, it is necessary to develop basic principles of a comprehensive verification methodology for OSCS software [6]. The solution of this problem is urgent for the effective development and verification of OSCS software using software prototypes, early functional integration, and the iterative checking of software requirements. This methodology is used to optimize the comprehensive software verification process in terms of time and cost criteria considering various technological constraints.

2. SELECTING AN OPTIMAL COMPREHENSIVE VERIFICATION STRATEGY

Formal problem statements on selecting an optimal comprehensive verification strategy for OSCS software often involve two optimality criteria: the minimum time of verification and the minimum cost of verification. The general problem of selecting an optimal comprehensive verification strategy is to determine the following elements: 1) an optimal partition of the software complex into separate parts, 2) the set of necessary subprograms (mocks and drivers), and 3) a scenario to verify the separate parts of the software complex. When stating the problem, constraints are used to determine the admissible partitions and unions of a special graph $\boldsymbol{\Gamma}$, whose vertices correspond to

[†] Deceased.

the program modules and whose arcs are control links between them. When developing tests and localizing errors, graph models are used to detail the graph $\boldsymbol{\Gamma}$. These models formalize the detailed flowcharts of the software complex and, in addition, the detailed flowcharts of separate program modules (PMs).

The set of different comprehensive verification strategies for OSCS software is defined as follows. At the initial stage, it is necessary to determine the set of all admissible partitions of the graph $\boldsymbol{\Gamma}$ into subgraphs. Autonomous testing is conducted for each resulting subgraph. Next, the set of all admissible unions of the resulting subgraphs is determined. These unions are used for joint software testing. Each comprehensive verification strategy is defined as follows. First, it is the set of subgraphs $p^m = \{p_1, \ldots, p_l, \ldots, p_M\}$ obtained by partitioning the graph $\boldsymbol{\Gamma}$. Second, it depends on the order in which these subgraphs are united. Uniting the subgraphs in a given order yields the original graph structure $\tilde{p}^{mn} = \{\tilde{p}_1^{mn}, \ldots, \tilde{p}_k^{mn}, \ldots, \tilde{p}_N^{mn}\}$, where $\tilde{p}_N^{mn}$ coincides with the graph $\boldsymbol{\Gamma}$.

The objective of selecting an optimal comprehensive verification strategy is to find a partition p^{m^*} of the graph $\boldsymbol{\Gamma}$ and a sequence of uniting the subgraphs $\tilde{p}^{\,mn^*}$ that, when used together, yield a comprehensive verification scenario with the optimal values of time and cost characteristics of the verification process.

If the verification process involves the mn-strategy, the time and cost of comprehensive verification consist of two components, $\overline{T}_{mn}^{p}\left(\overline{C}_{mn}^{p}\right)$ and $\overline{T}_{mn}^{o}\left(\overline{C}_{mn}^{o}\right)$. The first component is the time $\overline{T}_{mn}^{p}$ (and cost $\overline{C}_{mn}^{p}$) of the autonomous verification of the subgraphs obtained by partitioning the graph $\boldsymbol{\Gamma}$ for the mn-strategy. The second component is the time $\overline{T}_{mn}^{o}$ (and cost $\overline{C}_{mn}^{o}$) of implementing the uniting stages for these subgraphs and performing the subsequent joint verification of the subgraphs for the mn-strategy. The time and cost of verifying autonomously the subgraphs obtained by partitioning the graph $\boldsymbol{\Gamma}$ are given by

$$\bar{T}_{mn}^{p} = \sum_{\nu} t_{\nu mn}, \quad \bar{C}_{mn}^{p} = \sum_{\nu} C_{\nu mn},$$

where t_{vmn} and C_{vmn} denote the time and cost of the autonomous verification of the νth subgraph of the graph $\boldsymbol{\Gamma}$.

When uniting the subgraphs, the time and cost characteristics of joint verification are given by

$$\overline{T}_{mn}^{o} = \sum_{k} b_{kmn}, \quad \overline{C}_{mn}^{o} = \sum_{k} S_{kmn},$$

where b_{kmn} and S_{kmn} denote the time and cost of joint verification at the kth subgraph uniting stage.

The problem of determining an optimal verification strategy with the time criterion has the following general statement: it is required to minimize the expression

$$\sum_{mn} \left(\overline{T}_{mn}^{p} + \overline{T}_{mn}^{o}\right) x_{mn}$$

subject to the verification cost constraint

$$\sum_{mn} \left(\overline{C}_{mn}^{p} + \overline{C}_{mn}^{o}\right) x_{mn} \leqslant C.$$

In this constraint,

$$x_{mn} = \begin{cases} 1 & \text{if the } mn\text{-strategy is chosen for comprehensive verification} \\ 0 & \text{otherwise.} \end{cases}$$

The constraint also includes a constant $\mathbf{C}$, which specifies the maximum allowable cost of comprehensive verification.

In the process of solving this problem, possible partitions of the graph $\boldsymbol{\Gamma}$ into subgraphs are found by selecting the composition of V groups of the program modules of the software complex, where V is the number of program modules in the software complex of the onboard control system. When solving the problem, it is required to observe the constraints on the admissible combinations of program modules for each of $\mathbf{V}$ groups.

In the course of selecting a union of subgraphs from the subgraph set $p^m = \{p_1, \ldots, p_m, \ldots, p_M\}$ into the original graph $\boldsymbol{\Gamma}$, it is required to determine the list of stages to unite $\mathbf{V}^*$ non-empty subgraphs into $\boldsymbol{\Gamma}$. The maximum number of such stages must be equal to the number of non-empty subgraphs $\mathbf{V}^*$.

However, if the number of program modules in the software complex (the number of software components) is high, then the set of possible system verification strategies becomes very large as well. Due to this fact, estimating the time and cost characteristics of system verification strategies becomes an extremely resource-intensive and time-consuming task. To eliminate this difficulty, we propose to find particular optimal software verification strategies: such problems are most commonly encountered in practice.

We define the set $\overline{P}^p$ of admissible partitions of the graph $\boldsymbol{\Gamma}$ into subgraphs as follows:

$$\overline{P}^p = \{P^m\}, \quad m = \overline{1, M}.$$

Here, $P^m = \{p_1^m, \ldots, p_\nu^m, \ldots, p_{D_m}^m\}$ and p_ν^m denote the mth partition and the νth subgraph, respectively, and D_m is the number of subgraphs in the mth partition.

The set $\overline{P}^o = \{\tilde{P}^{mn}\}$, $(n = \overline{1, N_m},\ m = \overline{1, M})$ defines the admissible unions of the graph $\boldsymbol{\Gamma}$. The element $\tilde{P}^{mn} = \left\{\tilde{p}_1^{mn}, \ldots, \tilde{p}_k^{mn}, \ldots, \tilde{p}_{F_{mn}}^{mn}\right\}$ of the set $\overline{P}^o$ is the nth union under the mth partition of the graph $\boldsymbol{\Gamma}$. The values N_m and F_{mn} are the number of the resulting unions of subgraphs and the number of uniting stages under the mth partition of the graph $\boldsymbol{\Gamma}$.

For the nth union, the element $\tilde{p}_k^{mn}$ is defined as follows:

$$\tilde{p}_k^{mn} = \bigcup_{v \in R1_k^{mn}} p_v^m \bigcup_{i \in R2_k^{mn}} p_i^m.$$

Here, $R1_k^{mn}$ is the index set of the subgraphs from P^m and $R2_k^{mn}$ is the index set of the subgraphs from $\tilde{P}^{mn}$ included in the kth joint verification stage under the mth partition and the nth union of the graph $\boldsymbol{\Gamma}$.

On the one hand, the comprehensive verification strategy is determined by the partition of the graph $\boldsymbol{\Gamma}$ into subgraphs $P^m \in \overline{P}^p$; on the other, by the union of the resulting subgraphs $\tilde{P}^{mn} \in \overline{P}^p$ into the original graph.

The time t_ν and cost C_ν of the autonomous verification of each vth subgraph in a partition consist of three components as follows: the time and cost (t_ν^n, C_ν^n) of preparing test data, the time and cost (t_ν^p, C_ν^p) of executing the testing process, and the time and cost $\left(t_\nu^{\text{loc}}, C_\nu^{\text{loc}}\right)$ of localizing the errors detected during subgraph testing, i.e.,

$$t_\nu = t_\nu^n + t_\nu^p + t_\nu^{\text{loc}}, \quad C_\nu = C_\nu^n + C_\nu^p + C_\nu^{\text{loc}},$$

where

$$t_\nu^n = t_\nu^{\text{gen}} + t_\nu^{\text{mock}} + t_\nu^{\text{dri}}, \quad C_\nu^n = C_\nu^{\text{gen}} + C_\nu^{\text{mock}} + C_\nu^{\text{dri}}.$$

These formulas have the following notations: t_ν^{gen} and C_ν^{gen} are the time and cost of generating test data for the vth subgraph, respectively; t_ν^{mock} and C_ν^{mock} are the time and cost of developing

mock subprograms to verify the vth subgraph, respectively; t_ν^{dri} and C_ν^{dri} are the time and cost of developing driver subprograms to verify the vth subgraph, respectively; t_ν^{p} and C_ν^{p} is the time and cost of carrying out tests for the vth subgraph, respectively; finally, t_ν^{loc} and C_ν^{loc} are the time and cost of localizing errors detected when testing the vth subgraph, respectively.

An optimal system verification strategy can be found in two steps as follows. The first step is to select an admissible partition of the graph $\boldsymbol{\Gamma}$ into subgraphs $P^m \in \overline{P}^p$ for their autonomous verification. The second step is to select an admissible union of these subgraphs from the set $\overline{P}^o$ for joint verification. The two steps ensure the comprehensive verification process with minimum time and cost under the existing time and cost constraints.

For the problem statement under consideration, we define the variable

$$y_{mn} = \begin{cases} 1 & \text{if for the } m\text{th partition of the graph } \Gamma \text{ the } n\text{th union is selected} \\ 0 & \text{otherwise.} \end{cases}$$

This problem is solved using the following initial data:

1) the sets $\overline{P}^p = \{P^m\},\ m = \overline{1, M}$ and $\overline{P}^o = \{\tilde{P}^{mn}\},\ n = \overline{1, N_m},\ m = \overline{1, M}$,

2) the time and cost characteristics of the autonomous and joint verification processes.

The time and cost of comprehensive verification are given by

$$\overline{T}^k = \overline{T}_m^p + \overline{T}_{mn}^o, \quad \overline{C}^k = \overline{C}_m^p + \overline{C}_{mn}^o,$$

where $\overline{T}_m^p$ and $\overline{C}_m^p$ denote the time and cost of autonomous verification under the mth partition of the graph $\boldsymbol{\Gamma}$, respectively; $\overline{T}_{mn}^o$ and $\overline{C}_{mn}^o$ denote the time and cost of joint verification under the mth partition of the graph $\boldsymbol{\Gamma}$ and the nth union of the graph Γ.

The time $\overline{T}_m^p$ and cost $\overline{C}_m^p$ of autonomous verification are given by

$$\bar{T}_m^p = \sum_\nu \left(t_{\nu m}^{\text{gen}} + t_{\nu m}^{\text{mock}} + t_{\nu m}^{\text{dri}} + t_{\nu m}^{\text{loc}} \right),$$
$$\bar{C}_m^p = \sum_\nu \left(c_{\nu m}^{\text{gen}} + c_{\nu m}^{\text{mock}} + c_{\nu m}^{\text{dri}} + c_{\nu m}^{\text{loc}} \right).$$

Let the test sets to debug the subgraphs $p_\nu^m \in P^m$ be determined, and let the corresponding time and cost characteristics be known for them. Then the time and cost characteristics of autonomous debugging are calculated as

$$t_{\nu m}^{\text{gen}} = \sum_{j=1}^{J_\nu m} t_{j\nu m}^{\text{gen}}, \qquad c_{\nu m}^{\text{gen}} = \sum_{j=1}^{J_{\nu m}} \hat{c}_{j\nu m}^{\text{gen}},$$
$$t_m^{\text{progr}} = \sum_{j=1}^{J_\nu m} t_{j\nu m}^{\text{progr}}, \quad c_{\nu m}^{\text{progr}} = \sum_{j=1}^{J_{\nu m}} \hat{c}_{j\nu m}^{\text{progr}},$$
$$t_m^{\text{loc}} = \sum_{j=1}^{J_\nu m} t_{j\nu m}^{\text{loc}} \rho, \qquad c_{\nu m}^{\text{loc}} = \sum_{j=1}^{J_{\nu m}} \hat{c}_{j\nu m}^{\text{loc}} \rho.$$

In these formulas, $J_{\nu m}$ is the set of tests to verify the subgraph p_ν^m.

The variables $t_{\nu m}^{\text{mock}}$ and $c_{\nu m}^{\text{mock}}$ specify the time and cost of developing all mock subprograms to verify the subgraph p_ν^m, i.e.,

$$t_{\nu m}^{\text{mock}} = \sum_{i=1}^{I_{\nu m}} \hat{t}_{i\nu}^{\text{mock}}, \quad c_{\nu m}^{\text{mock}} = \sum_{i=1}^{I_{\nu m}} \hat{c}_{i\nu}^{\text{mock}}.$$

Here, I_{vm} is the number of mock subprograms to verify the subgraph p_v^m.

The time $\overline{T}^{\,o}$ and cost $\overline{C}^{\,o}$ of executing joint verification stages under the mth partition and the nth union of the graph $\boldsymbol{\Gamma}$ are given by

$$\overline{T}^{o} = \sum_{k=1}^{F_{mn}} \left(b_{kmn}^{n} + b_{kmn}^{\text{progr}} + b_{kmn}^{\text{loc}} \right),$$

$$\overline{C}^{o} = \sum_{k=1}^{F_{mn}} \left(S_{kmn}^{n} + S_{kmn}^{\text{progr}} + S_{kmn}^{\text{loc}} \right).$$

Suppose that the subgraphs $\tilde{p}_k^{mn} \in \tilde{P}^{mn}$ are verified using test sets with known time and cost characteristics. Then the time and cost of executing the kth joint verification stage under the mth partition and the nth union of the graph $\boldsymbol{\Gamma}$ are given by

$$b_{kmn}^{\text{n}} = \sum_{j=1}^{J_{kmn}} \hat{b}_{jkmn}^{\text{gen}}; \quad b_{kmn}^{\text{progr}} = \sum_{j=1}^{J_{kmn}} \hat{b}_{jkmn}^{\text{progr}}; \quad b_{kmn}^{\text{loc}} = \sum_{j=1}^{J_{kmn}} b_{jkmn}^{\text{loc}} \rho,$$

$$S_{kmn}^{\text{n}} = \sum_{j=1}^{J_{kmn}} \hat{S}_{jkmn}^{\text{gen}}; \quad S_{kmn}^{\text{progr}} = \sum_{j=1}^{J_{kmn}} \hat{S}_{jkmn}^{\text{progr}}; \quad S_{kmn}^{\text{loc}} = \sum_{j=1}^{J_{kmn}} S_{jkmn}^{\text{loc}} \rho.$$

With all these expressions for the time and cost characteristics of the software verification process, we formally state an optimization problem to find an optimal strategy for implementing a comprehensive verification scenario in terms of the minimum total time:

$$\sum_{m} \left(\sum_{m} \overline{T}_m^{p} \sum_{n=1}^{N_m} y_{mn} + \sum_{n=1}^{N_m} \overline{T}_m^{o} y_{mn} \right) \to \min.$$

This problem is solved subject to the following constraints:

—the maximum allowable cost of implementing the verification process,

$$\sum_{m} \left(\overline{C}_m^{p} \sum_{m} y_{mn} + \sum_{n=1}^{N_m} \overline{C}_m^{o} y_{mn} \right) \leqslant C;$$

—the set of M constraints on the variables y_{mn},

$$\sum_{n=1}^{N_m} y_{mn} = 1, \quad m = \overline{1, \text{M}}.$$

The problem of finding an optimal comprehensive verification strategy with the minimum cost criterion is formulated by analogy:

$$\sum_{m} \left(\overline{C}_m^{p} \sum_{m} y_{mn} + \sum_{n=1}^{N_m} \overline{C}_m^{o} y_{mn} \right) \to \min$$

subject to the time constraint imposed on the verification process,

$$\sum_{m} \left(\sum_{m} \overline{T}_m^{p} \sum_{n=1}^{N_m} y_{mn} + \sum_{n=1}^{N_m} \overline{T}_m^{o} y_{mn} \right) \leqslant T,$$

and the set of M constraints imposed on the variables y_{mn}.

These problems belong to the class of linear mathematical programming problems widely used in practice.

3. VERIFICATION METHODS FOR OSCS SOFTWARE WITH FUNCTIONAL CORRECTNESS INDICATORS

At the early functional integration stage of OSCS components, functional correctness indicators are used to assess the proper implementation of the functions of OSCS software. Each functionality of the software complex is implemented on some set of data processing routes. Along these routes, the input parameters of a function are transformed into one output result of this function or into a set of its output results. In order to check the correct operation of a function fully, it is necessary to check the entire set of data processing routes used by this function for a given set of its input parameters. Correct operation is validated if the output results of functions completely coincide with the reference results provided in the specifications of the program complex. Checking correct operation on the entire set of input data and on all data processing routes is a task of very high complexity. Therefore, one should select a bounded subset of data processing routes for their checking. This subset must be sufficient to check the implementation of the main functions of the software complex.

Nowadays, there are two approaches to check the correct operation of software: functional and structural. The functional approach involves the "black box" representation of software. The structural approach is based on checking the correct implementation of data processing routes; when preparing tests, it considers the structural peculiarities of separate modules of the software complex as well as the peculiarities of inter-module interaction within the complex. Both functional and structural approaches have significant disadvantages from the standpoint of efficient software verification implementation [2].

Due to this fact, we propose a method with the positive properties of both approaches. The method implies selecting a set of tests with the functional correctness indicators of the program complex that are necessary to check its correct operation. The quality of software operation is assessed based on the results of carrying out a set of selected tests. Consider this method in detail.

Let $\mathbf{F}$ be the set of all functions of the software complex implementing all primary and auxiliary functions. It is required to select a subset $\overline{F} < F$ of functions to be checked so that their correct operation will yield the desired values of the functional correctness indicators of the software complex.

For the software complex, an input data domain $\overline{E}$ is defined. For each function $F_j \in \overline{F}$, the corresponding subset $E_j \in \overline{E}$ of this domain is defined as well. Each such function transforms data of the input domain $E_j \in \overline{E}$ into the corresponding data of the output domain $y_j \in \overline{Y}$. Here, the set y_j contains all possible values of the output data for the function F_j $\left(j = \overline{1, J}\right)$.

The output results $y_{kj} \in Y$ of the software complex are obtained when implementing the sets of routes M_{jk} $\left(j = \overline{1, J}, k = \overline{1, K}\right)$ to process the data. Hence, to check the set of functions $\overline{F}$ of the software complex, it is necessary to check the correct operation of the set of data processing routes. Implementing these routes gives the necessary output results y_{kj} for each function F_j from the set $\overline{F}$ using the input data subsets $E_j \in \overline{E}$.

A function F_j of the software complex will be considered checked if, for all output results $y_{kj} \in Y_j$ of this function, the correctness of passing the set $M_{jk}\left(j = \overline{1, J}, k = \overline{1, K}\right)$ is successfully checked for all data processing routes yielding the output results for the function F_j. The sets $M_{jk} \in \overline{M_j}$, $k = \overline{1, K}$, of such routes will be considered the sets $\overline{M_j}$ of backbone routes for the function F_j. The correctness of obtaining the result of the jth function will be assessed using the indicator

$$N_{kj} = \frac{n_{kj}^{\text{chec}}}{n_{kj}^{\text{tot}}}.$$

In this formula, n_{kj}^{chec} is the number of checked backbone routes and n_{kj}^{gen} is the total number of backbone routes forming the results $y_{kj} \in Y_j$. The total number of backbone routes equals the cardinality of the set M_{kj}.

We will use the backbone route as the main element to be checked when assessing the functional correctness indicator of software and the graph model $\boldsymbol{\Gamma}(V, C)$ of the enlarged flowchart of the software complex when executing the verification scenario and determining the backbone paths for the functions of the set $\overline{F}$.

In the graph model, V is the vertex set of the graph $\boldsymbol{\Gamma}$, which corresponds to the set of blocks in the enlarged flowchart of the software complex, and C is the arc set of the graph. The arcs C represent the transfer of control between the flowchart blocks. These blocks are separate procedures and their aggregates or the program modules of the software complex. An arc between blocks i and j means the transfer of control from the former to the latter. In the model under consideration, vertex $\nu_i \in V$ of the graph $\boldsymbol{\Gamma}(V, C)$ is associated with the sets of its arguments $A_i = \{a_{\text{in}}\}$ and the sets of its results $R_i = \{r_{ij}\}$.

An information processing route m in the graph $\boldsymbol{\Gamma}(V, C)$ is a sequence $(v_0, c_0, v_1, c_1, \ldots, c_{I-1}, v_I)$ containing vertices and arcs. In this sequence, v_i $(0 \leqslant i \leqslant I)$ is a vertex of the graph $\boldsymbol{\Gamma}(V, C)$ and c_i $(1 \leqslant i \leqslant I-1)$ is an arc connecting vertices v_i and v_{i+1}. In turn, a sequence $(v_0, \ldots, v_I)$ of vertices corresponds to the transformations implemented on a data processing route m. Such a sequence is called a transformer of route m, and a sequence $(c_0, \ldots, c_{I-1})$ of arcs corresponds to the conditions to be satisfied on route m and is called the condition of route m.

For the result $y_{jk} \in Y_j$ of a function $F_j \in \overline{F}$, the backbone route m_{jk} is a route whose transformer $(v_0, \ldots, v_i)$ includes at least one of the possible sequences of external and internal information links. These external and internal links must start at vertex v_0 and end at vertex v_i to obtain the result y_{jk}.

4. SOFTWARE VERIFICATION FOR THE ONBOARD CONTROL SYSTEM OF THE RUSSIAN SEGMENT OF THE ISS

In this section, as one example, the concept described above is used to verify software configuration elements (SCEs) of the Russian Segment of the International Space Station (ISS) [3].

The following operations are carried out stage-by-stage to verify the SCEs:

(1) the autonomous testing of the software complex;
(2) the comprehensive verification of the SCEs on a ground verification bench;
(3) software verification jointly with C&C MDM (Command and Control Multiplexor DeMultiplexor, the onboard central computer of the US Segment and the entire ISS);
(4) the formal qualification tests of the SCEs.

The listed verification stages of the SCEs allow detecting, localizing, and eliminating the errors arising in the software verification process as well as confirming software operability and assessing software compliance with the technical specifications.

The **autonomous testing** of software is conducted based on an autonomous PC workstation and on the SDDF complex (software project development tools). Testing is conducted using a methodology that includes the following elements: the description of the testing procedure, initial testing conditions, and test cases. After the software testing process is finished, it is handed over to the configuration control group, which integrates the tested software into the SCEs of the onboard central computer.

The **comprehensive verification of the SCEs** is performed according to a special scenario to solve the following tasks:

(1) quality checking for the operating system;

(2) onboard control system software assembly and comprehensive verification in accordance with the flight plan and the operating modes of the Russian Segment and the service module simultaneously with flight safety control (i.e., checking the correct implementation of all subgraphs and the entire graph $\boldsymbol{\Gamma}$);
(3) spot checks of the backbone routes corresponding to the most probable abnormal situations, the localization of abnormal situations, and their elimination;
(4) checking the compliance of onboard control system software with the documents (ICD SSP 50 097);
(5) resource allocation control (memory, CPU time, and I/O channels).

Joint tests with C&C MDM were conducted on SITE-C, EGSE, and SVF, dedicated benches with special test implementation scenarios. During the tests, the onboard software of both onboard control systems (the US Segment and the Russian Segment) as well as the model software of both onboard systems (the US Segment and the Russian Segment) were used.

Formal qualification tests or acceptance tests and docking tests is a process that verifies the compliance of the SCEs of the onboard central computer with the requirement specification and ICD.

A certain subset is selected from the set of tests conducted using the NKO ground verification complex. This subset serves to verify the correctness of implementing a given set of backbone routes. Upon completion of the formal qualification testing, the Customer signs the report that the SCEs of the onboard central computer are ready for docking tests.

Docking tests were conducted in accordance with a dedicated methodology. The hardware and software means of the onboard central computer undergo docking tests with real hardware or its analogs using the NKO-2 ground verification complex. Docking tests of the hardware and software means of the onboard central computers (the Russian Segment with the US Segment) were conducted using the NKO-1 ground verification complex. They were carried out in accordance with the NASA–RSA Phase 2-3 Bilateral Integration and Verification Plan (SSP50101). The hardware and software means of the onboard central computer as part of the Zvezda service module (product index 17KSM) were tested on complex bench No. 24008 and on the control and test station in a required volume.

5. CONCLUSIONS

This paper has presented the existing experience as well as organizational, methodological, and technical solutions concerning software verification for onboard spacecraft control systems. The main features of a comprehensive software verification technology for onboard spacecraft control systems have been described. This technology ensures effective software development and debugging based on software prototypes, the iterative refinement of requirements, and early functional integration. The proposed technology has been implemented within the computer-aided software development and verification system for onboard spacecraft control systems. As a result, the total number of errors in the process of software development and verification has been significantly reduced for the Russian Segment of the ISS.

REFERENCES

1. Mikrin, E.A., Kul'ba, V.V., and Pavlov, B.V., Developing Models and Design Methods for Information Management Systems in Space Vehicles, *Autom. Remote Control*, 2013, vol. 74, no. 3, pp. 348–357.
2. Mikrin, E.A., Kul'ba, V.V., Kosyachenko, S.A., Somov, D.S., and Gladkov, Yu.M., *Kompleksnaya otrabotka programmnogo obespecheniya bortovogo kompleksa upravleniya kosmicheskimi apparatami i imitatsionnye modeli funktsionirovaniya bortovykh sistem i vneshnei sredy* (Comprehensive Software Verification for an Onboard Spacecraft Control System and Simulation Models of Onboard Systems and

Environment), Preprint of Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, 2011.

3. Kul'ba, V.V., Mikrin, E.A., Pavlov, B.V., and Platonov, V.N., *Teoreticheskie osnovy proektirovaniya informatsionno-upravlyayushchikh sistem kosmicheskikh apparatov* (Theoretical Foundations of Designing Spacecraft Information and Control Systems), Moscow: Nauka, 2006.
4. Kurenkov, V.I. and Kucherov, A.S., *Metody issledovaniya effektivnosti raketno-kosmicheskikh sistem. Problemno-orientirovannye sistemy avtomatizirovannogo proektirovaniya* (Methods for Studying the Efficiency of Rocket and Space Systems. Problem-Oriented Computer-Aided Design Systems), Samara: Samara State Aerospace University, 2012.
5. Zelentsov, V., Kovalev, A., Okhtilev, M., Sokolov, B., and Yusupov, R., Creation and Application Methodology of the Intelligent Information Technology of Complexity Objects Space and Ground Based Monitoring, *SPIIRAS Proceedings*, 2013, vol. 5, no. 28, pp. 7–81.
6. Mikrin, E.A., *Bortovye kompleksy upravleniya kosmicheskimi apparatami i proektirovanie ikh programmnogo obespecheniya* (Onboard Spacecraft Control Systems and Their Software Development), Moscow: Bauman Moscow State Technical University, 2003.
7. Mikrin, E.A., Sukhanov, N.A., Platonov, V.N., et al., Design Concepts of Onboard Control Complexes for Automatic Spacecrafts, *Control Sciences*, 2004, no. 3, pp. 62–66.

This paper was recommended for publication by V.M. Glumov, a member of the Editorial Board