Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 12, Number 1 (2016), pp. 167-190

http://www.ripublication.com

Applicability of Software Testing Methods to Software and Hardware Data Security Tools

Tatiana Mikhailovna Kanner
National Research Nuclear University MEPhI,
Kashirskoye sh., 31, Moscow, Russia.

Abstract

The article is devoted to the issues of testing software and hardware data security tools (DST) used in modern information systems (IS) to ensure the required level of security of the data being processed therein. The author substantiates the need of improving the existing approaches to testing of hardware and software DST due to the growing popularity of IS and reduction of time spent for their creation. The article considers the applicability of software testing methods to software and hardware DST, identifies features of software and hardware DST depending on the principles of their operation, which prevent from using the existing testing methods in their original form. A mathematical model for software and hardware DST has been developed, and the criteria of applicability of software testing methods to such DST have been formed. On the basis of the above criteria, the requirements for automation of the testing process have been established. As illustrated by one of the types of software and hardware DST, the author proves the fact that testing automation requires additional hardware equipment. The article describes the purpose, the structure and the principle of operation of a developed testing tool – a USB switcher used for automated testing of portable software and hardware DST operating in an OS of a computer and having a USB connection interface. The criteria proposed in the article allow to determine applicability of one or another testing method to a certain software and hardware DST, and the developed USB switcher allows to automate testing of most of the software and hardware DST having a USB connection interface.

Keywords: Testing, Software Testing Methods, Features Of Software And Hardware DST, Criteria Of Applicability Of Existing Software Testing Methods To Software And Hardware DST, A USB Switcher.

Introduction

To ensure data security in the process of their storage and processing in information systems (IS), software or software-hardware data security tools (DST) should be used. Software and hardware DST are more secure, because when you use a software DST, you cannot guarantee that it has not been changed and operates adequately. Therefore, data protection should be arranged by transferring some of security functions to the hardware level, i.e. with the help of software and hardware DST.

On the one hand, using of a DST improves system security, but on the other hand it can lead to a deterioration of user and functional characteristics of the IS. To minimize the adverse effect of DST, they must be tested before they are included in the IS.

Taking into account the fact that IS are currently created and developed very fast (due to the use of efficient tools), we need to maximally reduce the time spent to develop a DST picking out the stages that can be fully or partially automated. Such stages include testing, verification and elimination of the detected errors, which are repeated while developing a DST many times. At the same time, we should take into account the fact that existing testing methods (for example, software testing methods), including the automated testing method, can turn out to be inapplicable to software and hardware DST or applicable to them only subject to certain conditions or to availability of additional hardware equipment for testing.

There arises a contradiction between the need for reliable data protection and the need for reduction of the time consumed by inclusion of software and hardware DST into the IS. This contradiction can be eliminated by creating methods and developing hardware testing tools for software and hardware data security complexes. First of all, we need to create conditions of applicability of software testing methods to software and hardware DST, and to form applicability criteria on the basis of such conditions, which, in turn, will demonstrate what hardware equipment should be developed, and how testing methods should be adapted.

Widely-known scientific works pay much attention to the problem of quality and security of software, as well as to methods and tools used for its testing. The basics and the methods of software testing are considered in works by V.P. Kotlyarov [13], I.V. Stepanchenko [21], S.V. Sinitsyn and N.Yu. Nalyutin [20], R. Black [3-4], L. Tamres [22] and C. Kaner [10]. The issues of testing using the black box method, as well as functional software testing are examined to the fullest extent in works by B. Beizer [2]. The method of automated software testing is described in detail by E. Dustin [7]. Works by R. Culbertson [6] and K. Beck [1] are devoted to studying the concepts of extreme programming and rapid testing.

The issues of quality and reliability of software are considered in depth by V.V. Lipaev [15] and G. Myers [17-18] in their works. V.I. Grekul [8] examines software testing in terms of its implementation in information systems.

The above works consider the issues of testing that are applicable to software, but pay little attention to software and hardware DST. Some differences between testing of software and software-hardware complexes are studied in R. Black's work [4]. Therefore, the following scientific and technical task arises: to develop a methodological framework for improving testing methods used for software and hardware DST, and to develop hardware solutions that will allow to automate the process of testing such tools.

At the same time, the issue of testing software and hardware DST interests many vendors, but their practically used testing methods are usually not published in publicly available resources and have no scientific rationale.

Method

A. Forming conditions determining applicability of software testing methods to software and hardware DST

Let us consider the differences between software testing methods and the methods used to test software and hardware DST (the methods of positive and negative test cases; black box, white box and gray box methods; manual, automated or semi-automated testing methods [11]).

From the viewpoint of system knowledge (black box, white box and gray box methods) and positivity of test cases (positive or negative test cases) as a whole, if we do not consider the issue of testing automation with the help of these methods, it does not matter what the testing object is: software, software DST or software-hardware DST. Irrespective of the testing object, the corresponding tests are included in the Testing Program and Procedure (TPP).

At the same time, the methods of positive and negative test cases, as well as black, white and gray box methods can be included in the methods of manual, automated or semi-automated testing (separately or together). Irrespective of the testing method used, first of all, one should develop a TPP with the required test structure, procedure and testing methods. On the basis of the TPP, one can perform testing manually or develop automated tests using special automation tools. Therefore, particular attention should be paid to the following testing methods that can include the methods of positive or negative test cases, as well as black, white and gray box methods:

1. The method of manual testing in accordance with the TPP;
2. The method of automated testing using automation scripts;
3. The method of semi-automated testing if there is no possibility to ensure full testing automation.

Let us point out the major features arising while applying the above software testing methods to various software and hardware DST:

- A difficulty in fixing operation errors/bugs;
- Inability to use debuggers;
- Features connected with the hardware component:
    - Presence of various form factors of the hardware component;
    - A need for cross-testing of the hardware and the software components of the DST;
    - Dependence on the hardware platform;
- Features connected with the type and the operation principle of the hardware component:

    - Portability (for portable hardware component);
    - Inability to use automation tools;
    - A need to test components having no security functions (flash memory, etc.), and their interaction with components performing security functions;
- Inability to test using virtualization tools;
- A need to test limit/boundary values in the conditions of a limited memory of the hardware component;
- Features arising while using program interfaces to test the hardware component, a possibility to test the software and the hardware components separately.

Let us consider the above features in relation to one of the DST – a personal cryptographic data security tool (PCDST) SHIPKA [12], [19] representing a portable software and hardware DST operating in an OS of a computer. The following arguments will be generally applicable to all other DST of this type.
The PCDST SHIPKA operates in the environment of an OS, therefore unlike a DST operating independently from the computer, there is a possibility to fix errors of both the hardware and the software components of the PCDST SHIPKA in the OS, as well as a possibility to use debuggers and other test tools (automation tools, fuzzers, etc.).

The PCDST SHIPKA can be implemented on the basis of various form factors of USB devices, each of which can differently interact with the hardware component. Therefore, while testing it, one must take into account various ways of implementing the hardware component.
Since the hardware component of the PCDST SHIPKA is implemented as a USB device, and the DST itself operates in an OS, there is no need to test it on various hardware platforms (there are no limitations on computer hardware characteristics; compliance with the specification of the corresponding USB interface version will be enough).
As to the PCDST SHIPKA, there is a need for functional cross-testing of its components and their interaction (reliability of the management interface, reliability of any hardware components – the internal memory, etc.). Some versions of the PCDST SHIPKA include a flash memory. Let's call such DST "DST with a flash memory".
DST with a flash memory (like many other software and hardware DST operating in the OS) represent a complex made of software and a composite hardware device (an alienable one, if we speak about DST with a flash memory), which includes:
- Components directly performing data security functions (a microprocessor, internal software/firmware, software in the OS, etc.);
- Other components having no direct security functions, and if we speak about a DST with a flash memory – the flash memory.

While testing DST with a flash memory (both, manually or by automated means), the following tests should be also carried out:

1. Testing of the flash memory (determining speed characteristics, load tests, etc.);
2. Testing of interaction of the components performing security functions with the flash memory.

Thus, while testing DST operating in an OS, which includes a flash memory, the TPP should be supplemented with the above tests. At the same time one should remember that, for example, testing of a flash memory in terms of its reliability can be done manually, but it is unreasonable taking into account time consumption, because it requires a large number of cycled atomic read/write operations of variously-sized data blocks. To test the flash memory, it would be reasonable to use some existing and publicly available specialized tools, rather than to develop them [12].

While automating testing of software and hardware DST with a flash memory (and many other alienable software and hardware DST), there arises a nuance connected with the need to reconnect the hardware component in the process of testing their functions used by the end user. It is impossible to adequately emulate work of a real user of such DST without using additional testing tools.

To test the PCDST SHIPKA, virtualization tools may be used, because USB devices can be passed through to a virtual OS. However, in case of such passthrough of the hardware component to a virtual OS, new errors can arise that do not occur in a real computer and are associated with inadequate operation of virtualization tools.

In case of load and stress testing of the components performing security functions of the PCDST SHIPKA, one should also test limit values of the DST characteristics [12]. For example, correctness of operation with the maximum possible number of cryptographic keys in the device (which can exceed 200). To do this, one should generate the required number of keys, check correctness of operation of the PCDST SHIPKA, and in case of any errors – gradually reduce the number of keys and repeat testing until a certain critical value, which is quite difficult to do manually.

One of the ways to automate testing of the PCDST SHIPKA is to use the Software Development Kit (SDK) for standard interfaces, which is implemented for the PCDST SHIPKA for the purpose of possible use of cryptographic functions of the device by third-party developers. Three standard interfaces have been implemented for the PCDST SHIPKA so far: Crypto Application Programming Interface (Crypto API), Public-Key Cryptography Standard #11 (PKCS#11), Application Protocol Data Unit (APDU), and the corresponding SDK includes a set of examples how they may be used. Such examples can be used to automate functional testing of the PCDST SHIPKA, since their successful running guarantees that the utilities correctly running on the basis of such interfaces will run correctly on the PCDST SHIPKA. However, the software component of the PCDST SHIPKA should be tested separately (because its operation is not checked while using the SDK), for example, with the help of testing automation tools.

On the basis of the foregoing, testing of DST with a flash memory (operating in an OS) can be generally automated, subject to the following conditions:

Condition 1. Use of additional testing tools allowing to emulate work of a real user, including reconnection of the DST in the process of its configuring and use, if necessary.

Condition 2. Support for the OS environment where automation scripts can operate.

Condition 3. Presence of automation scripts and other tools of testing the components not performing security functions (the flash memory, etc), and interaction of the components of software and hardware DST.

Manual testing of such DST is rather difficult and requires at least using third-party programs to test the components not performing security functions (the flash memory, etc.) – i.e. at least partial automation is desirable.

It is necessary to meet conditions 1-3 to arrange automated testing of the DST operating in an OS and having a portable hardware component. As to the DST operating in an OS and having a stationary hardware component, it is enough to meet conditions 2-3.

Thus, we have formed conditions of applicability of software (software DST) testing methods to software and hardware DST operating in an OS. On the basis of the above conditions, criteria of applicability of software testing methods to software and hardware DST can now be established.

B. Formalizing criteria determining applicability of software testing methods to software and hardware DST

Let us formalize criteria determining applicability of software testing methods to software and hardware DST using key provisions of the system analysis, the formal system theory, as well as the algorithm and the set theories.

Let $M = \{m_1, m_2, m_3, m_4, ...\}$ denote a set [5] of all the software and hardware DST where $m_1, m_2, m_3, m_4, ...$ are some software and hardware DST (for example, the PCDST SHIPKA).

Let M_{na} denote a set of software and hardware DST, which can be tested manually (i.e. not automatically).

In general, $M = M_{na}$ for any consistent DST ($M_{na} \subseteq M$ by convention, and any consistent software and hardware DST can be tested manually). However, consistency of a particular DST, i.e. existence of a possibility to implement its functions in principle, should be reasoned.

The set M can be expressed in the following way:

$M_a \cup M_{sa} \subseteq M$

where:

M_a is a set of software and hardware DST, which can be tested automatically.

M_{sa} is a set of software and hardware DST, which can be tested automatically only in part (i.e. semi-automated).

Here:

1. $M_a \subseteq M_{na}$, $M_{sa} \subseteq M_{na}$ are subsets of the set M_{na}.
2. The subsets M_a and M_{sa} are disjoint – $M_a \cap M_{sa} = \varnothing$.

The set of software and hardware DST (M) is shown in Figure 1.

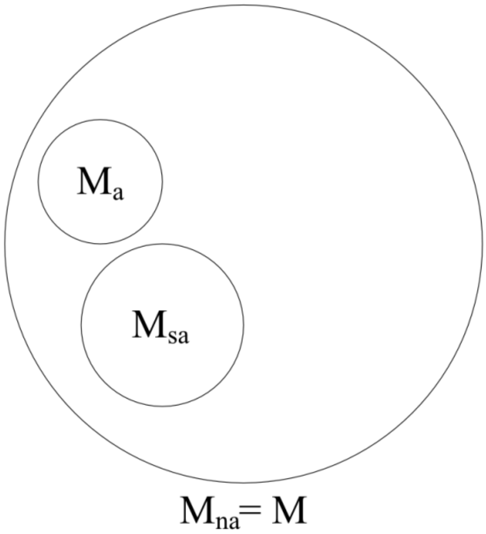

Figure 1: The set of software and hardware DST

In addition, the set M can be expressed in this way:
$M = P_{os} \cup P_{nos}$,
where:

P_{os} is a set of software and hardware DST operating in an OS.

P_{nos} is a set of software and hardware DST operating independently from an OS.

The set P_{os} consists of two subsets: $P_{os} = P_{portable} \cup P_{stationary}$,
where:

$P_{portable}$ is a set of portable software and hardware DST operating in an OS.

$P_{stationary}$ is a set of stationary software and hardware DST operating in an OS.

Here $P_{portable} \cap P_{stationary} = \varnothing$.

Definition 1. Let us construct a mathematical model [14, 23] of some abstract software and hardware DST in the following way: $\forall m \in M$ can be expressed as $m = <F, V, OP>$, which is a selection of the following sets:

- F is a set of objective functions performed by software and hardware DST (for example, encryption, use of a digital signature, access control);
- V is a set of possible states of the hardware component (here $v_0 \in V$ is an initial state, for example, when the hardware component has not been initialized);
- OP is a set of state transition functions, where the software and hardware DST transits from one state into another (for example, initialization of the hardware component and its transition to the initialized state).

Definition 2. The DST can be in a particular state at any specific time, let us denote it $(F^{'},v)$, where $v \in V$ is the current state of m, and $F^{'}$ is a subset of objective functions of m, which can be performed in this state v (i.e. a set of objective functions of the DST, which can be performed in the current state v, $F^{'} \subseteq F$). Here combining all the $\bigcup F^{'} = F$, where $F^{'} \subseteq F$, $\forall v \in V$.

Definition 3. The state transition function for the software and hardware DST can be expressed in the following way:

$$op \in OP : (F^{'}, v_1) \to (F^{''}, v_2)$$

This means that m transits from one state $(F^{'},v_1)$ into another $(F^{''},v_2)$.
Here, some objective functions of the DST can be performed only in particular states of the DST's hardware component. For example, as a rule, no function of a DST can be performed before first initialization of its hardware component; encryption functions of a cryptographic DST cannot be used unless encryption keys are generated, etc.

Thus, Definition 3 implies that in accordance with the algorithm theory [16], some functions of the set F are computable only in particular states of the set V , and these functions can turn to be incomputable in other states.
In addition, functions of the set OP can be functions performed directly in the software and hardware DST (nonobjective functions, such as the initialization function), or external functions that should be performed additionally (for example, entering a PIN code, reconnection of the hardware component, etc.).

Let us define the elements of the objective functions set performed by the software and hardware DST.

Definition 4. Let $m \in M$ and $m = < F,V,OP >$, where F,V,OP are the sets of objective functions, states and transition functions in accordance with Definitions 1-3. Let $F = \{F_1,...,F_n\}$, where $n \in \mathrm{N}$ (N is a set of natural numbers from now on) is a finite set of objective functions performed by m .

All the functions in F can be expressed as an ordered tuple [9] of subfunctions f performed one after another (steps required to perform the function):

$$\forall F_i \subseteq F, i = 1,\ldots,n, F_i = \begin{cases} f_1(input_1) \\ \ldots \\ f_k(input_k), \mathrm{k} \in \mathrm{N} \end{cases}$$

where:

input is a tuple of formal parameters transferred to F_i depending on implementation of the objective function (for example, in case of file encryption, this will be the file to be encrypted, the path for saving the encrypted result and the encryption key). $input_1,...,input_k$ are formal parameters of subfunctions of the objective function F_i.

Here, for $j = 1,...,k$:

$$\forall f_j \in F_i,\ f_j(input_j) = \begin{cases} True \text{ (if } f_j \text{ is performed successfully)} \\ False \text{ (if an error occurs)} \end{cases}$$

If $f_j = False$, the objective function F_i will return error.

If $f_j = True$, F_i will return success if $j = k$.

Let us define manual test functions for software and hardware DST on the basis of Definition 4 in the following way:

Definition 5. Let $m \in M$ and $m = <F,V,OP>$, where F,V,OP are the sets of objective functions, states and transition functions in accordance with Definitions 1-3. Let $F = \{F_1,...,F_n\}$, where $n \in \mathrm{N}$ is a finite set of objective functions performed by m. Since $M = M_{na}$, there will be $R = \{R_1,...,R_n\}$ for m – a finite set of manual test functions corresponding to functions $\{F_1,...,F_n\}$.

All the functions in R can be expressed in the form of an ordered tuple of subfunctions r performed one after another (steps required to perform the function):

$$\forall R_i \subseteq R, i = 1,...,n, R_i = \begin{cases} r_1(input_1, output_1) \\ ... \\ r_k(input_k, output_k), \mathrm{k} \in \mathrm{N} \end{cases}$$

where:

$input$ is a tuple of formal parameters passed to R_i, $input_1,...,input_k$ are formal parameters of subfunctions of the function R_i.

$output$ is a tuple of output data (an operation log, an error code, an additional result, etc.), $output_1,...,output_k$ are the output data of subfunctions of the function R_i.

Here, for $j = 1,...,k$:

$$\forall r_j \in R_i, r_j(input_j, output_j) = \begin{cases} True \text{ (if } f_j \text{ is performed successfully)} \\ False \text{ (if an error occurs)} \end{cases}$$

If $r_j = False$, the objective function R_i will return error, and $output = output_j$.

If $r_j = True$, then if $j = k$, R_i will return success, and $output = output_j$.

If $r_j = True$ and $j \neq k$, R_i will continue execution, and $output = output_{j+1}$.

Definition 6. $\forall r_j \in R_i$ the subfunction of the manual test function is computable on the basis of Definition 5, if $f_j \in F$ corresponding to it is computable for $\forall j = 1,...,k$, $k \in N$.

Definition 7. Let $m = <F,V,OP> \in M$. If $\forall R_i \subseteq R$ (the manual test function in accordance with Definition 5) contains only computable subfunctions in accordance with Definition 6, R_i will be a computable manual test function, $i = 1,...,n$, $n \in N$, and R will be a set of computable manual test functions (and DST $m \in M_{na}$).

Let us formalize a general criterion of the possibility to test software and hardware DST manually on the basis of Definitions 1-7.

Proposition 1. ***(General criterion of the possibility to test software and hardware DST manually)****:*
Let $m \in M$ *and* $m = <F,V,OP>$ *is in the state* $v_0 \in V$*, where* F,V,OP *are the sets of objective functions, states and transition functions in accordance with Definitions1-4.*

Let $R = \{R_1,...,R_n\}$, $n \in N$ *is a finite set of manual test functions corresponding to objective functions of* m *in accordance with Definition 5.*
Then $m \in M_{na} \Leftrightarrow$ *the following conditions are met:*
1. Either $\forall r_j \in R_i$, $i = 1,...,n$, $j = 1,...,k$, $k \in N$ *is computable in accordance with Definition 6 in the state* $v_0 \in V$ *for* $\forall R_i \subseteq R$ *(i.e.* R *is a set of computable manual test functions in the state* v_0 *in accordance with Definition 7);*
2. Or $\exists op_1,...,op_L \in OP$ *(functions of transition to the states* $v_1,...,v_L$*):* $\forall r_j \in R_i$*, which is incomputable in the state* v_0*, is computable in one of the states* $v_1,...,v_L$ *for* $\forall R_i \subseteq R$.
For example, this means that:

$$R_i = \begin{cases} r_1(input_1, output_1) \\ \ldots \\ op_z \\ r_j(input_j, output_j) \\ \ldots \\ r_k(input_k, output_k), k \in N \end{cases}$$

where:

r_j *is incomputable in the initial state.*

op_z *is a function of transition to a state where* r_j *is computable (*$z = 1,...,L$*).*

$input_1,...,input_k$ *are formal parameters of subfunctions of the function* R_i.

$output_1,...,output_k$ are output data of subfunctions of the function R_i.

Let us define a set of automation functions (scripts) for software and hardware DST on the basis of Definition 4.

Definition 8. Let $m \in M$ and $m = <F, V, OP>$, where F, V, OP are the sets of objective functions, states and transition functions in accordance with Definitions 1-3. Let $F = \{F_1,...,F_n\}$, where $n \in N$ is a finite set of objective functions performed by m.

Let us define a set of automation functions (scripts) for m, which will be applicable to software and hardware DST, as $S = \{S_1,...,S_n\}$, where $n \in N$ is a finite set of automated test functions corresponding to objective functions $\{F_1,...,F_n\}$.
If we express all the functions in F as a tuple of subfunctions f performed one after another (in accordance with Definition 4), we can define the automated test functions in the following way:

$$S_i = \begin{cases} for(l = 1,...,count)\{ \\ \qquad s_1(input_1, output_1) \\ \qquad ... \\ \qquad s_k(input_k, output_k), k \in N \\ \} \end{cases}$$

where:

count is a number of repeated executions of the automated test function.

input is a tuple of formal parameters transmitted to S_i, $input_1,...,input_k$ are formal parameters of subfunctions of the function S_i.

output is a tuple of output data (an operation log, an error code, an additional result, etc.) required for analysis and reporting of the automation tool.

$output_1,...,output_k$ are output data of subfunctions of the function S_i.

Here, for $j = 1,...,k$:

$$\forall s_j \in S_i, s_j(input_j, output_j) = \begin{cases} True \text{ (if } f_j \text{ is performed by automation tools successfully)} \\ False \text{ (if an error occurs)} \end{cases}$$

and:

$$output = \begin{cases} \{ \\ output'_1 \\ \dots \\ output'_{count} \\ \} \end{cases}$$

where:

$output'_l = output_j, l = 1,\dots,count$, if $s_j = False$ on iteration l. S_i continues the next iteration if $l \neq count$, and otherwise it is completed.

$output'_l = output_{j+1}, l = 1,\dots,count$ if $s_j = True$ on iteration l ($j \neq k$). S_i continues the current iteration.

$output'_l = output_j, l = 1,\dots,count$ if $s_j = True$ on iteration l ($j = k$). S_i continues the next iteration if $l \neq count$, and otherwise it is completed.

When S_i finishes, success or fail of a certain iteration should be determined on the basis of *output* data.

Definition 9. $\forall s_j \in S_i$ automation function indicated in Definition 8 is computable if $f_j \in F_i$ corresponding to it is also computable for $\forall j = 1,\dots,k$, $k \in N$, and there are sufficient conditions to perform s_j and a possibility to fix the result of s_j.

Definition 10. Let $m = <F,V,OP> \in M$. If $\forall S_i \subseteq S$ (the automation function in accordance with Definition 8) contains only computable automation subfunctions in accordance with Definition 9, S_i will be a computable automation function, $i = 1,\dots,n$, $n \in N$, and S will be a set of computable automation functions (and DST $m \in M_a$).

Let us formalize general criteria of possible full or partial testing automation for software and hardware DST on the basis of Definitions 1-4, 8-10.

Proposition 2. ***(General criterion of possible testing automation for software and hardware DST):***
Let $m \in M$ and $m = <F,V,OP>$ is in the state $v_0 \in V$, where F,V,OP are the sets of objective functions, states and transition functions in accordance with Definitions1-4.

Let $S = \{S_1,\dots,S_n\}$, $n \in N$ is a finite set of automation functions corresponding to objective functions of m in accordance with Definition 8.
Then $m \in M_a \Leftrightarrow$ the following conditions are met:
1. Either $\forall s_j \in S_i$, $i = 1,\dots,n$, $j = 1,\dots,k$, $k \in N$ is computable in accordance with Definition 9 in the state $v_0 \in V$ for $\forall S_i \subseteq S$ (i.e. S is a set of computable automation functions in the state v_0 in accordance with Definition 10);

2. Or $\exists op_1,...,op_L \in OP$ *(automated functions of transition to the states* $v_1,...,v_L$*):* $\forall s_j \in S_i$*, which is incomputable automation subfunctions in the state* v_0*, is computable automation subfunctions in one of the states* $v_1,...,v_L$ *for* $\forall S_i \subseteq S$.

For example, this means that:

$$S_i = \begin{cases} for(l=1,...,count)\{ \\ \quad s_1(input_1, output_1) \\ \quad ... \\ \quad op_z \\ \quad s_j(input_j, output_j) \\ \quad ... \\ \quad s_k(input_k, output_k), k \in \mathrm{N} \\ \} \end{cases}$$

where:

s_j *is incomputable in the initial state.*

op_z *is a function of automated transition to a state where* s_j *is computable* ($z = 1,...,L$).

$input_1,...,input_k$ *are formal parameters of subfunctions of the function* S_i.

$output_1,...,output_k$ *are output data of subfunctions of the function* S_i.

Corollary 1. (General criterion of possible partial automation of testing software and hardware DST):

Let $m \in M$ and $m = < F,V,OP >$ is in the state $v_0 \in V$, where F,V,OP are the sets of objective functions, states and transition functions in accordance with Definitions1-4.

Let $S = \{S_1,...,S_n\}$, $n \in \mathrm{N}$ is a finite set of automation functions corresponding to the objective functions of m in accordance with Definition 8.

Then $m \in M_{sa} \Leftrightarrow \exists$ at least one $S_i \subseteq S$, S_i is a computable automation function in accordance with Definition 10, $i = 1,...,n \Leftrightarrow$ the following conditions are met for such S_i:

1. Either $\forall s_j \in S_i$, $j = 1,...,k$, $k \in \mathrm{N}$ is a computable automation subfunction in accordance with Definition 9 in the state $v_0 \in V$ for $S_i \subseteq S$;

2. Or $\exists op_1,...,op_L \in OP$ (automated functions of transition to the states $v_1,...,v_L$): $\forall s_j \in S_i$ which is incomputable automation subfunctions in the state v_0, is computable automation subfunctions in one of the states $v_1,...,v_L$ for $S_i \subseteq S$.

C. Requirements for the testing tools applicable to software and hardware DST operating in an OS

Let us apply the general criterion determining the possibility of manual testing from Proposition 1 to the PCDST SHIPKA.

Let us denote the PCDST SHIPKA as m_{shipka}, $m_{shipka} \in M$.

Axiom 1. $m_{shipka} \in P_{portable} \subseteq P_{os}$.

Proposition 3. *Let* $m_{shipka} = < F_{shipka}, V_{shipka}, OP_{shipka} > \in M$ *be the PCDST SHIPKA in accordance with Definition 1. Then* $m_{shipka} = < F_{shipka}, V_{shipka}, OP_{shipka} > \in M_{na}$.

Proof: Let us prove Proposition 3. m_{shipka} can be in the following states $v_0, v_1, v_2, v_3, v_4 \in V_{shipka}$ [19]:

- v_0 is an initial state, when the device is not initialized (not formatted, and no authorization parameters, administrator password, and user PIN and PUK codes have been set).
- v_1 is a state, when the device is initialized and prepared for operation (it has been formatted, and authorization parameters, an administrator password, and user PIN and PUK codes have been set).
- v_2 is a state, when the device is ready for operation, and the user PIN code have not been entered (cryptographic keys and certificates needed for encryption and signing procedures have been generated or imported, or a user has been registered for protected login in the system, or pass cards have been created, or any two or more of the above conditions have been met).
- v_3 is a state, when the device is ready for operation like in the state v_2, but an adequate user PIN code have been entered.
- v_4 is a state, when the device is in the technological mode.

At the initial time, the PCDST SHIPKA can be in the states v_0, v_1, v_2, v_4 and cannot be in the state v_3.

At the same time, since the PCDST SHIPKA is a portable DST ($m_{shipka} \in P_{portable}$) in accordance with Axiom 1, for m_{shipka} there exist states $v_0^{'}, v_1^{'}, v_2^{'}, v_4^{'} \in V_{shipka}$ that are similar to the states $v_0, v_1, v_2, v_4 \in V_{shipka}$, but in which the device is not connected to a USB port of the computer. At the initial time the PCDST SHIPKA can also be in the states $v_0^{'}, v_1^{'}, v_2^{'}, v_4^{'}$.

The operation principle of the PCDST SHIPKA implies that all of its objective functions are computable only in the state v_3.

Therefore, in accordance with Proposition 1, to make the task of manual testing for the PCDST SHIPKA computable, there should exist at least transition functions $op_1,...,op_8 \in OP_{shipka}$ between the following states of m_{shipka}:

$- op_1 : v_2 \rightarrow v_3$;

$- op_2 : v_2^{'} \rightarrow v_3$;

$- op_3 : v_1 \rightarrow v_3$;

$- op_4 : v_1^{'} \rightarrow v_3$;

– $op_5 : v_0 \to v_3$;

– $op_6 : v_0^{'} \to v_3$;

– $op_7 : v_4 \to v_3$;

– $op_8 : v_4^{'} \to v_3$.

Possible states and transition functions of m_{shipka} are presented in Figure2.

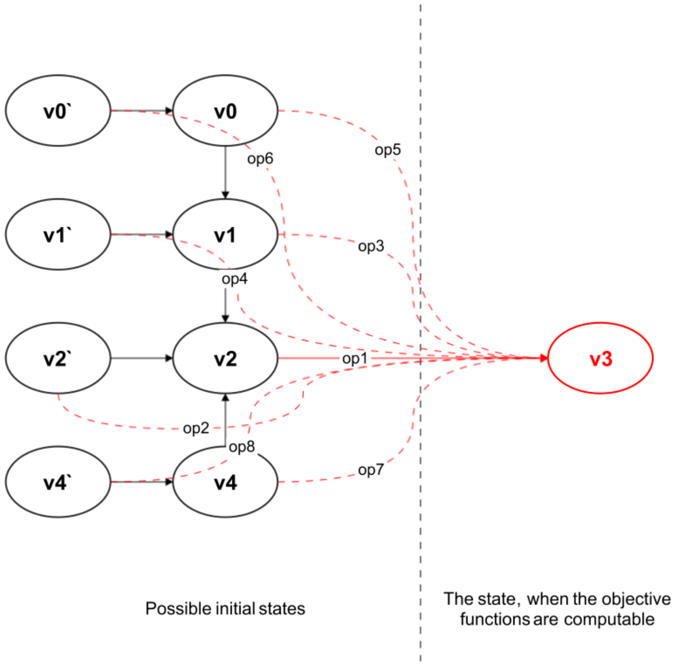

Figure 2: Possible states and transition functions of the PCDST SHIPKA

For m_{shipka}, such transition functions are as follows [19]:

op_1 – entering of the PIN code;

op_2 – a composite transition function comprising connection of the device to a USB port of the computer and execution of op_1;

op_3 – a composite transition function comprising generation procedures and import of cryptographic keys and certificates, or registration of a user for protected login in the operating system, or creation of a pass card, or a combination of any two or more of the above conditions, as well as execution of op_1;

op_4 – a composite transition function comprising connection of the device to a USB port of the computer and execution of op_3;

op_5 – a composite transition function comprising device initialization and execution of op_3;

op_6 – a composite transition function comprising connection of the device to a USB port of the computer and execution of op_5;

op_7 – a composite transition function comprising transition of the device from the technological mode to the state v_0, v_1, or v_2 and execution of op_5, op_3 or op_1, correspondingly;
op_8 – a composite transition function comprising connection of the device to a USB port of the computer and execution of op_7.

All the above transition functions can be performed manually, therefore all the objective functions can be computable after corresponding transitions in case of manual testing of the PCDST SHIPKA, which was to be proved.

Let us formalize particular criteria of possible automation of testing the PCDST SHIPKA using Conditions 1–3 and the general criterion provided in Proposition 2.

Proposition 4. (Particular criterion of possible automation of testing the PCDST SHIPKA):
Let $m_{shipka} = < F_{shipka}, V_{shipka}, OP_{shipka} > \in M$ *is the PCDST SHIPKA in accordance with Definition 1.*
Then $m_{shipka} = < F_{shipka}, V_{shipka}, OP_{shipka} > \in M_a \Leftrightarrow \forall S_i \subseteq S$ *(in accordance with Definition 8)* $\exists op_2, op_4, op_6, op_8 \in OP_{shipka}$ *(in accordance with the proof of Proposition 3), that allow to connect/reconnect the device to a USB port of the computer using automated tools, consequently incomputable automation functions* $s_j \in S_i$ *become computable.*

Proof: It follows from Definitions 8, 9, 10 and Proposition 2 that $\forall s_j \in S_i$ should be computable automation functions in any state $v \in V_{shipka}$.

It follows from the proof of Proposition 3 that all the transition functions from OP_{shipka}, except for $op_2, op_4, op_6, op_8 \in OP_{shipka}$, can be automated, and therefore, if we anyhow automate such transition functions, to make the task of automated testing of the PCDST SHIPKA computable in accordance with Proposition 2, it will be enough to initiate the corresponding function of automated transition from one state (in which some s_j is incomputable) into another one (in which this function s_j is computable).
The $op_2, op_4, op_6, op_8 \in OP_{shipka}$ indicated in the conditions should be a technical testing tool that can anyhow connect and disconnect the PCDST SHIPKA from a USB port of the computer.

Let us formulate a Corollary about possible testing of portable software and hardware DST operating in an OS on the basis of Conditions 1-3 and Proposition 4.

Corollary 2. (Requirement for the hardware equipment in case of automating testing of portable software and hardware DST operating in an OS):
Let $m = < F, V, OP > \in M$ in accordance with Definition 1, and $m \in P_{portable}$.
Then $m = < F, V, OP > \in M_a \Leftrightarrow$ the conditions of Proposition 2 are met, and one of the existing transition functions $op \in OP$ allows to connect/disconnect the DST to the computer using automation tools (i.e. there is hardware equipment allowing to automatically connect/disconnect the DST to the computer).

Results and Discussion

Proposition 3 demonstrates that software and hardware DST such as the PCDST SHIPKA can be tested manually without any additional hardware equipment. Proposition 4 and Corollary 2 set requirements for the hardware equipment needed for automated testing of the PCDST SHIPKA and portable DST operating in an OS. Taking into account these requirements, as well as Conditions 1-3, it becomes clear what properties a hardware testing tool should have and how automated and semi-automated testing methods should be adapted to make them applicable to portable DST operating in an OS.
As to stationary software and hardware DST operating in an OS, one should apply the general criterion stipulated in Proposition 2, since such DST possess no similar properties as those indicated in Corollary 2 and typical of all the DST of such a type.

Thus, we can conclude that software (software DST) testing methods are applicable to software and hardware DST operating in an OS, subject to the special characteristics of such DST (and sometimes, of each particular DST).If we speak about the PCDST SHIPKA operating in an OS, it will be reasonable to use only semi-automated testing (using tools for testing the flash memory), or automated testing and additional hardware equipment allowing to automatically connect/reconnect the PCDST SHIPKA to the computer, switching this equipment on or off while carrying out automated tests.
Connection/reconnection of the PCDST SHIPKA to the computer can be emulated with the help of software tools (using BIOS or settings of a particular operating system). However, such connection/reconnection cannot ensure full emulation of a physical disconnection (for example, power supply cannot be switched off), which makes it difficult to use such software for testing of portable software and hardware DST operating in an OS (including the PCDST SHIPKA). Moreover, such software cannot be universal for various versions of the OS, BIOS, etc. Therefore, to connect/reconnect the PCDST SHIPKA (and other portable software and hardware DST) to the computer, it will be reasonable to use hardware testing tools.
While developing hardware testing tools, one should take into account the type of the DST's connection interface to the computer (a USB interface in case of the PCDST SHIPKA), and the operation principle of such tools will be similar for various interfaces. The hardware testing tool implemented for one of the interface types can be used for various DST using this interface for connection to the computer.

Accordingly, the so-called USB switcher has been developed (its appearance is shown in Figure 3). It is a software and hardware complex, which represents:

- Hardware equipment installed between the USB interface of the computer and the DST on the same USB channel and switching it in accordance with the commands received through the control channel;
- Software allowing to send an appropriate command from the OS through the control channel (to switch on or off data transmission and power supply of the switch-controlled USB device).

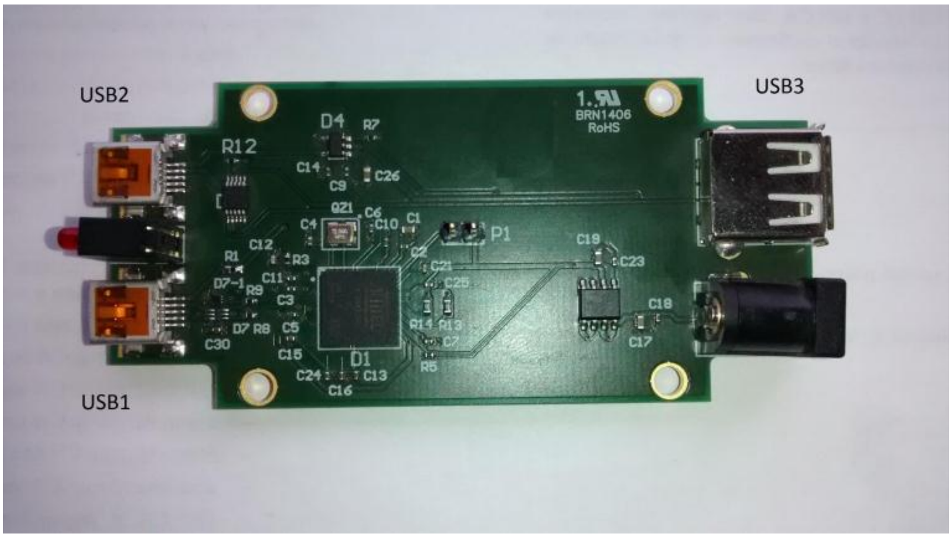

Figure 3: Appearance of the USB switcher

Such hardware equipment can physically interrupt power supply of various USB devices connected to it (for example, the PCDST SHIPKA) without physically disconnecting/reconnecting them. The USB switcher with the switch-controlled PCDST SHIPKA is shown in Figure 4.

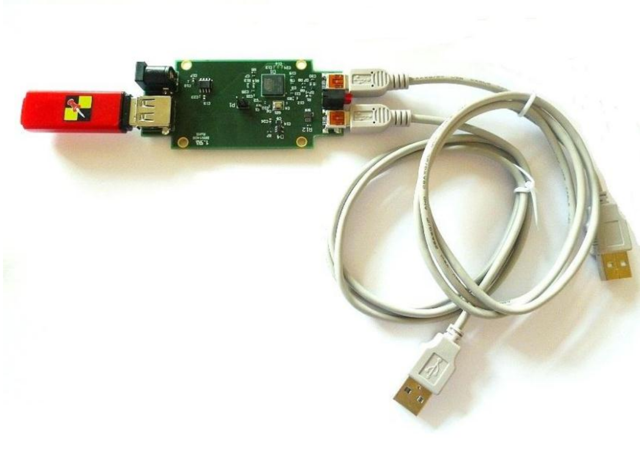

Figure 4: The USB switcher with the switch-controlled PCDST SHIPKA

The USB switcher is designed to perform the following functions:

1. Controlled reconnection of USB devices in the process of their automated testing (this means that the USB switcher is used as a hardware testing tool);
2. Controlled access lock to the USB channel (the USB switcher is used as a data security tool locking prohibited USB devices and allowing access to permitted USB devices).

The diagram of connecting a DST with a USB interface to a computer through the USB switcher is shown in Figure 5.

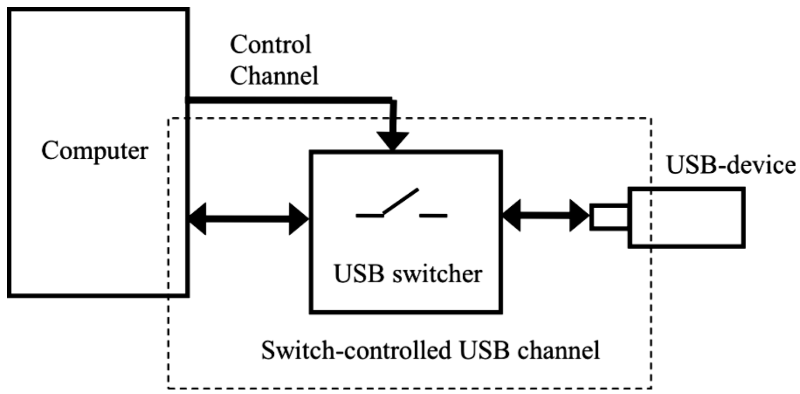

Figure 5. The diagram of connecting a DST with a USB interface to a computer through the USB switcher

Functionally, the USB switcher consists of the following components:

- A hardware component, comprising:
  - A microcontroller;
  - Two connectors USB – USB1 and USB2 (see Figure 3), where USB1 is used to pass control commands from the computer to the switcher, and USB2 is used to transmit data to and from the USB device connected to the switcher (USB1 and USB2 are connected to the computer with the help of connecting cables);
  - One connector USB – USB3 used to connect USB devices to the switcher;
  - A USB data line multiplexer USB2-USB3;
  - A power electric key switch USB2-USB3;
  - Light-emitting diodes indicating the switcher's status (red and green);
  - A reset switch (to reboot the internal software);
  - An external power input for the microcontroller, keys and light-emitting diodes (power can be supplied through USB1 or from an external source using this input, if necessary).
- Internal software (firmware) representing software launched from the hardware component upon receipt of power and performing the main functions of the software and hardware complex;
- External software, comprising:
  - Libraries to embed and use the complex in third-party software;

– Command-line utilities for switching USB devices (using the libraries).

Let us consider the operation principle of the USB switcher in detail.

Upon switching on power supply of the hardware component (i.e. upon connection of USB1 to the computer or provision of power to the external power input), the firmware is launched, which consecutively performs the following functions:

- Tests operability of the hardware component;
- Switches on the red LED (notifies that the hardware component is ready for work);
- Switches off data transmission and power supply of USB2-USB3;
- Switches off the green LED (notifies that USB2-USB3 are switched off);
- Waits for messages through the control interface USB1.

Using a library needed to embed the complex in third-party software or a command-line utility operating on its basis, the following commands can be sent to the hardware component:

- To switch on data transmission and power supply of USB2-USB3 (which is identified by a green LED on);
- To switch off data transmission and power supply (which is identified by a green LED off);
- Request the switching status.

Switching of the USB channel (USB2-USB3) is ensured by synchronous switching on and off of the data line multiplexer and power supply. Commands between the external software and the firmware are passed by exchanging messages in a special format through the control channel USB1.

Upon receipt of a message (containing a command for the hardware component) through the control channel USB1, the firmware performs the following actions:

- Extracts the command for the hardware component from the message;
- Executes the command (depending on the command – switches on or off data transmission and power supply of USB2-USB3 and correspondingly changes the state of the green LED);
- Returns the result of command execution through USB1.

For the purpose of using the USB switcher, there is a program interface (implemented in the library needed to embed and use the complex in third-party software) comprising a set of the following functions:

- *std::list<std::string>US_Enumerate()*;
- *std::string US_ON(std::string usbSwitcher)*;
- *std::string US_OFF(std::string usbSwitcher)*.

Where:

- usbSwitcher is a line (from the list returned by US_Enumerate()) identifying the USB switcher, to which the command is passed (like “USB Switcher X”, where X is the number of the switcher connected to the computer, beginning from “0”);

- US_Enumerate is a function searching for and outputting a list of found and currently available USB switchers (or an empty list, if no switchers are connected to the computer);
- US_ON is a function passing the command to switch on data transmission and power supply of the devices connected to the USB switcher (in case of successful operation, it returns “SUCCESS”, otherwise it generates an exception like std:string with a detailed description of the error);
- US_OFF is a function passing the command to switch off data transmission and power supply of the devices connected to the USB switcher (in case of successful operation, it returns “SUCCESS”, otherwise it generates an exception like std:string with a detailed description of the error).

While using the USB switcher, the following parameters should be passed as input to the command-line utility (USBSwitcherConsole.exe):

- list – to output a list of available USB switchers (i.e. the function US_Enumerate from the library of embedding and using the complex in third-party software is performed);
- on<switcherName> – to switch on data transmission and power supply of the devices connected to the corresponding switcher (i.e. the function US_ON from the library of embedding and using the complex in third-party software is performed). If the <switcherName> is not specified the command is passed to the first available switcher;
- off<switcherName> – to switch off data transmission and power supply of the devices connected to the corresponding switcher (i.e. the function US_OFF from the library of embedding and using the complex in third-party software is performed). If the <switcherName> is not specified, the command is passed to the first available switcher.

While developing the USB switcher, the existing solutions resembling it in terms of their functions have been analyzed. Thus, in spheres not connected with testing data security tools (for example, in systems like “smart house”), some tools are used containing a USB relay, which is able to remotely switch/turn off USB devices. However, if we use such a USB relay as a basis for the USB switcher, there can arise problems with signal alignment or with appearance of the so-called “parasite” currents (when the switcher is being turned off), and therefore it can be difficult to support some interfaces such as USB-3.0. To use the USB relay as a basis for the USB switcher, one should design and assemble a similar board (eliminating the above defects), and develop software and firmware. No publicly available ready-to-use analogs of the developed USB switcher for testing software and hardware DST, including those based on a USB relay, have been found.

Conclusion

As a rule, any software and hardware DST can be tested manually, without using any additional testing tools or adapting manual testing methods. However, testing of some DST, for example, DST with a flash memory, requires at least using automated testing tools for the flash memory, otherwise testing of such DST can turn out to be quite long. That is why, to test DST with a flash memory, it will be reasonable to use automated or semi-automated testing methods.

To automate testing of various software and hardware DST, the following general conditions should be met:

- Sufficient conditions for operation of automation scripts (in an OS of a computer or a DST);
- A possibility to fix the results of automated testing;
- Availability of automation scripts for functional cross-testing of the components of a software and hardware DST (the components performing and not performing security functions of the DST);
- A possibility of automated transition from such a state of a software and hardware DST, in which some of its functions are incomputable, to a state, in which these functions become computable.

In case the above conditions are not met for one or more functions of the DST, only the semi-automated testing method can be applied.

To automate testing of portable software and hardware DST operating in an OS, additional hardware equipment should be used allowing to automatically connect/disconnect the DST to a computer. Stationary software and hardware DST operating in an OS have no such properties (which are typical of all the DST of this type).

The developed software and hardware complex (the UBS switcher) can be used to automate testing of portable software and hardware DST operating in an OS and having a USB connection interface (including the PCDST SHIPKA), when it is necessary to automatically connect/reconnect the DST to a computer, for example to emulate a user's actions performed while working with a DST.

The USB switcher fully emulates physical disconnection and connection of the DST to a computer both, at the level of power supply and data transmission. Therefore, this software and hardware complex meets all the requirements for hardware equipment needed to automate testing of portable software and hardware DST operating in an OS from Corollary 2. Alongside with this target use of the USB switcher as a hardware testing tool, it can be used as a data security tool locking prohibited USB devices and allowing access to permitted devices.

Thus, the criteria proposed in this article allow to determine the possibility to apply one or another testing method to a particular software and hardware DST operating in an OS. To automate testing of one of the types of software and hardware DST (portable software and hardware DST operating in an OS and having a USB connection interface), hardware equipment has been developed to be used together with automated tests by embedding therein commands to controllably connect/disconnect the DST. Such hardware equipment allows to reduce the time needed to test a DST before it is included in an IS.

The topic discussed in this article can be further developed by improving/adapting the USB switcher in the following ways: to test portable software and hardware DST with another connection interface and to test DST supporting the interface USB-3.0. The USB switcher can be also improved to allow combining several connection interfaces of DST in one form factor (for example, USB-2.0 and USB-3.0) or to allow sequentially connect the DST being tested to various computers (this can be required to test some of the DST). Another promising field of development can be formation of similar criteria determining applicability of software testing methods to other types of software and hardware DST (operating independently from an OS or in an OS, with a portable or stationary/embedded hardware component), as well as use of the developed USB switcher in the process of testing such DST and development of new testing tools on its basis.

References

[1] Beck, K. (2004). *Extreme Programming Explained: Embrace Change* (2nd ed.). Boston: Addison-Wesley.

[2] Beizer, B. (1995). *Black-Box Testing Techniques for Functional Testing of Software and Systems*. New York: Wiley.

[3] Black, R. (2004). *Critical Testing Processes: Plan, Prepare, Perform, Perfect*. Boston: Addison-Wesley.

[4] Black, R. (2009). *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing* (3rd ed.). Indianapolis: Wiley.

[5] Bourbaki, N. (2004). *Theory of sets*. Berlin: Springer-Verlag. DOI:10.1007/978-3-642-59309-3.

[6] Culbertson, R., & Brown, C. (2002). *Rapid Testing*. Upper Saddle River, NJ: Prentice Hall PTR.

[7] Dustin, E., & Rashka, J. (1999). *Automated Software Testing: Introduction, Management, and Performance*. Reading, Mass.: Addison-Wesley.

[8] Grekul, V.I., Denishenko, G.N., & Korovkina, N.L. (2005). *Proektirovanie informatsionnykh sistem* [*Design of Information Systems*]. Moscow: NOU INTUIT. (in Russian).

[9] Hilbert, D., & Bernays, P. (1982). *Osnovaniya matematiki. Logicheskie ischisleniya i formalizatsiya arifmetiki* [*Foundations of Mathematics. Logical Calculus and Formalization of Arithmetics*]. Moscow: Nauka. (in Russian).

[10] Kaner, C., & Falk, J. (1999). *Testing Computer Software* (2nd ed.). New York: Wiley.

[11] Kanner (Borisova), T.M., & Gadasin, V.A. (2012). Zadacha testirovaniya apparatnykh sredstv zashity informatsii [The Task of Testing Hardware Data Protection Tools]. *Information Security Questions*, (3), 10–16. (in Russian).

[12] Kanner (Borisova), T.M., & Oblomova, A.I. (2013). Sposoby avtomatizatsii testirovaniya SZI, funktsioniruyushikh v OS, na primere PSKZI SHIPKA [Testing Automation Methods of Data Security Tools, Operating in OS: PCDST SHIPKA as an Example]. *Elektronika-info*, (6), 117–118. (in Russian).
[13] Kotlyarov, V.P., & Kolikova, T.V. (2006). *Osnovy testirovaniya programmnogo obespecheniya* [*Fundamentals of Software Testing*]. Moscow: Binom. (in Russian).
[14] Krupsky, V.N., & Plisko, V.E. (2009). *Teoriya algoritmov* [*Theory of Algorithms*]. Moscow: Academia. (in Russian).
[15] Lipaev, V.V. (1986). *Testirovanie programm* [*Testing Programs*]. Moscow: Radio i svyaz. (in Russian).
[16] Malcev, A. I. (1970). *Algorithms and Recursive Functions*, Groningen: Wolters-Noordhoff Pub.
[17] Myers, G. (1976). *Software reliability: Principles and practices*. New York: Wiley.
[18] Myers, G., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing* (3rd ed.). New Jersey: Wiley.
[19] OKB SAPR. (2011). *Personal Cryptographic Data Security Tool SHIPKA-K. Administrator Manual*. Moscow: JSC "OKB SAPR".
[20] Sinitsyn, S.V., & Nalyutin, N.Yu. (2008). *Verifikatsiya programmnogo obespecheniya* [*Software Verification*]. Moscow: Binom. (in Russian).
[21] Stepanchenko, I.V. (2006). *Metody testirovaniya programmnogo obespecheniya* [*Methods of Software Testing*]. Volgograd: VSTU. (in Russian).
[22] Tamres, L. (2002). *Introducing software testing*. Boston: Addison-Wesley.
[23] Zhuravlyov, Y.I., Flerov, Y.A., & Vyaly, M.N. (2010). *Diskretny analiz. Formalnyie sistemy i algoritmy* [*Discrete Analysis. Formal Systems and Algorithms*]. Moscow: Kontakt Plus. (in Russian).