Metric of Vulnerability at the Base of the Life Cycle of Software Representations

Mikhail Buinevich*, Konstantin Izrailov*, Andrei Vladyko*

*The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Saint Petersburg, Russia

bmv1958@yandex.ru, konstantin.izrailov@mail.ru, vladyko@sut.ru

***Abstract*— This article investigates the problem of the origin of software vulnerabilities in terms of their life cycle. For this purpose, the process of creating a software by a person is examined in detail and partially formalized with in philosophical categories – «Form vs Essence». There is proposed main representations of life cycle and intermediate stages of the transformation between it. As an example there's hypothetical comparison of two software transformations with a graphical interpretation of the security level of each of final representations. This article is a author's works continuation on the fundamental research of the security problems of the software domain.**

***Keyword*— information safety, machine code, software representation, vulnerability, methodology, metric, life cycle**

I. Introduction

Current problem of information security is existence of vulnerabilities in the software (further – SW). It is necessary for effective counteraction to vulnerabilities all-round studies of properties of this object, thus not only static, but also dynamic. Last define lifetime of vulnerabilities according to a scale of process of creation of a SW. So, the analysis of points of appearance of vulnerabilities will allow to evaluate the current security level of a product and to make its forecast for different variations of development process. The analysis of points on a scale where vulnerabilities can be guaranteed found, will allow to construct the systems of their neutralization more effectively.

As process of creation SW is interactive and step-by-step, and points of emergence of vulnerabilities have to correspond to steps of interaction of the person with him – at the same time vulnerabilities can possess various degree of premeditation from the person. It will allow to create typification of vulnerabilities on points of their emergence and prerequisites to it.

Manuscript received data is October 13, 2017. This work is a follow-up of the invited journal to the accepted conference paper of the 18th International Conference on Advanced Communication Technology.

M. V. Buinevich is with The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Russian Federation, Saint-Petersburg, 22-1 Prospekt Bolshevikov (e-mail: bmv1958@yandex.ru).

K. E. Izrailov is with The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Russian Federation, Saint-Petersburg, 22-1 Prospekt Bolshevikov (corresponding author to provide phone: +7(921)555-2389; fax: none; e-mail: konstantin.izrailov@mail.ru).

A. G. Vladyko is with The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Russian Federation, Saint-Petersburg, 22-1 Prospekt Bolshevikov (e-mail: vladyko@sut.ru).

Processing of a program code automatic means can be considered an analog of interaction of the person with a code as the rules put in them are also created by the person – for disposal of monotonous work. The set of steps of interaction in this case will be similar.

II. Areas of Life of Vulnerabilities in Submissions of the Software

The research of dynamic properties of vulnerabilities begun in author's article [1] is under construction on states SW in certain points of life cycle. Such states can be described by means of the philosophical categories reflecting interrelation of two parties of everyone representations of a program code (further – Representation): his form (external manifestation – the flowchart [2], a programming language, etc.) and essence (inner meaning – logic of use of elements of schemes, operators of language, etc.). In fact, the form is only one of possible reflections of essence within the current environment (in particular it is various for the person – the text and images, and machine – a binary code). Justification of the chosen division into states it is possible to find possible use at construction SW appropriate design means: experts of a certain profession and the standard automating programs (further – Utilities). Each of means according to own mission carries out transformation of Representation from previous to the subsequent (and with application a reverse engineering and in the opposite direction [3, 4, 5, 6], for example, for the benefit of search of vulnerabilities). Offered in above to the mentioned author's article of Representation, describing the creations SW typical process for telecommunication devices, and involved for this means, are given in Table 1.

TABLE 1.
Representations Software and Design Means for Their Transformations

Order №	Initial representations	Project tool for transformation Name	Type
	---	Creator	Specialist
1	Main idea	Concept developer	Specialist
2	Concept model	Design architect	Specialist
3	Architecture	Algorithmist	Specialist
4	Code algorithms	Coder	Specialist
5	Source code	Compiler	Utility
6	Assembler code	Assembler	Utility
7	Machine code	Linker	Utility
8	Image file	---	---

Note. The first Representation arises only owing to activity of the creator, and the last finishes process of creation SW.

It should be noted that a profession the algorithmist and the coder extremely seldom meets in modern practice, though it

was demanded at initial stages of development of the region of programming. In current state of IT area they are united by a profession the programmer, leading to the fact that experts of the last are forced to be engaged in transformation of two Representations with various appointment at once (see No. 3 and No. 4 in Table 1). Authors consider that such situation is caused by purely economic reasons and hypothetically is the indirect source of vulnerabilities influencing quality (in sense of safety) the final product.

Considered in [1] states point to significant jumps between forms and essence SW, lowering details of transformations between the next Representations. And as states in itself are completely static – any changes without participation of design means (from Table 1) are impossible – that cognitive stages of transformations between Representations are of special interest as in them new vulnerabilities ([7, 8]) can appear.

III. Representation transformation stages

According to the interpretations of form and essence entered above, reflection of any Representation in the objective (physical) world can be created by means of only his form which is obviously consisting of the system of others, smaller – elementary forms. In the subjective (mental) world, Representation has an appearance of the complete essence consisting also of the system of elementary essence. And as the structure and properties of forms of various Representations are, as a rule, essentially various and don't give in to external regularity, it is possible to make transformation only consciously and only through their essence storing all communications lacking for this purpose. A rare exception are transformations between extremely similar Representations by means of trivial rules (for example, the translation of a binary code from hexadecimal Intel HEX in binary record). We will repeatedly note that in case of use of Utilities the situation essentially doesn't change as though the last and make a transfer automatically, however generally these means will transform an entrance form to own internal representation – i.e. work with essence, and then generate the last according to an output form.

We will consider abstractly separate taken transformation of Representation of *i* to the subsequent Representation of *i+1* (further – Representation of *j*) which is carried out by a certain project tool. The essence of work of any such tools comes down to three fundamental phases: analysis of Representation of *i*, its processing and synthesis of Representation of *j*.

As an example we will use conversion of algorithms of a program code (Representation of *i*) to the source code (Representation of *j*) the coder (the expert – project means for creation of the code implementing the given algorithm). So, the coder studies the given algorithm (for example, it has flowchart appearance), realizes its sense, selects a suitable programming language (if it is not set in advance) and implements on it a necessary functionality. Obviously, the coder shall the be sign, both with the form of the flowchart of an algorithm, and with a paradigm of a programming language and its syntax; also his abilities and experience shall allow to work with the corresponding essence of Representations.

A. Phase 1. Analysis

The phase of the analysis of Representation by the person consisting in his understanding is rather difficult and debatable; nevertheless, the existing researches (in particular, in the field of Text Comprehension [9]), allow to allocate the following stages of this process.

Stage 1. Perception

At this stage of people makes initial recognition of a form of Representation of *i* by means of sense organs (as a rule, sight). As it isn't possible to person to capture all form of Representation entirely (as though the flowchart consisted of the only graphic element), his breakdown on elementary forms is made. At the same time the elementary sense (or in terms of the entered categorial couple – essence) compared to each such elementary form remains not certain so far. Representation of *i* from area of the physical world passes into own world of the subject studying him. So, the coder, studying the flowchart, obtains initial visual information on her elements and their communications.

Stage 2. Understanding

At this stage of person compares the received elementary forms to certain elementary essence according to own thesaurus (the system of comparisons of terms and their concepts). Representation of *i* acquires a certain semantic essence, though rather separate. So, the coder begins to perceive elements of flowcharts with a condition (a symbol – the Decision) as conditional transitions to elements with data processing (a symbol – Process).

Stage 3. Interpretation

At this stage person synthesizes the uniform essence of Representation of *i* which is the coordinated whole in a brain, using the elementary essence understood by him. At the same time, as a rule, the total sense of Representation is much more difficult, than summation of separate semantic elements from the previous stage – classical synergetic effect. Judgment of Representation by semantic interpretation of the understood forms is in this way made. This way a coder understands the general sense of work of an algorithm.

B. Phase 2. Processing

As Representations of *i* and *j* are based on own unique elements of essence (corresponding to the missions), between the last the corresponding converting is necessary. The phase of processing of representation in working memory of the person [10] consisting of the only stage is for this purpose intended.

Stage 4. Re-comprehension

The stage can be considered the most difficult and the least studied – at it the sense SW constructed on elements of essence of Representation of *i* in identical sense on elements of essence of Representation of *j* will be transformed. The stage is similar to modeling process where the essence of Representation of *i* has an appearance of a certain internal model in a brain of the person which then is investigated for the purpose of receiving (or transformations in her) the new model corresponding to the essence of Representation of *j*. So, the coder, using own cognitive skills, not just finds

unambiguous compliance of steps of an algorithm to programming language designs, namely will transform them in more difficult way. At the same time the rule «one in one» doesn't work as the flowchart of an algorithm already is 2D (the unidirectional count with cycles), and a program code – 1D (a set of lines with own structure). The situation many times becomes complicated if the algorithm corresponds to one paradigm of programming (for example, imperative), and the required programming language – another (for example, functional).

C. Phase 3. Synthesis

The synthesis phase following a reconsideration phase has to construct his form on the essence of Representation of *j* – to transfer Representation from the subjective world of the person to objective surrounding. Thus, the phase reasonably can consist of the stages belonging to an analysis phase, but which are carried out upside-down. There are even utilities with the close purposes and the principles of work (for example, SWIM [11]).

Stage 5. Shaping

At this stage person breaks the uniform essence of Representation on elementary, ready for reflection in the physical world. There is a certain structure of maintenance of Representation of *j* though without any specification concerning her form. So, the coder from conscious sense of work of an algorithm (step-by-step) receives the scheme of performance according to a programming paradigm (in this case, imperative).

Stage 6. Exposition

At this stage person compares to each elementary essence the same elementary form. The return thesaurus (the system of comparisons of concepts and their terms) is for this purpose used. Programming language forms are compared to semantic elements of Representation of *j*. So, the coder presents the scheme of performance in the form of set of designs of language.

Stage 7. Decoration

At this stage person makes concrete entry of Representation of *j* strictly in the required look – graphic, text, binary and so forth. Representation has a uniform consistent form. So, the coder makes the end result of the activity – writes down a program code of an algorithm.

Stages can be considered intuitive and logical on a set of examples. The structure of phases and stages and also graphic interpretation of their work in categories of a form/ essence is given in the Fig. 1. Zone 1, 2 and 3 correspond to conditional degree of sensibleness of essence by the person for each of Phase and Stage of transformation of Representations.

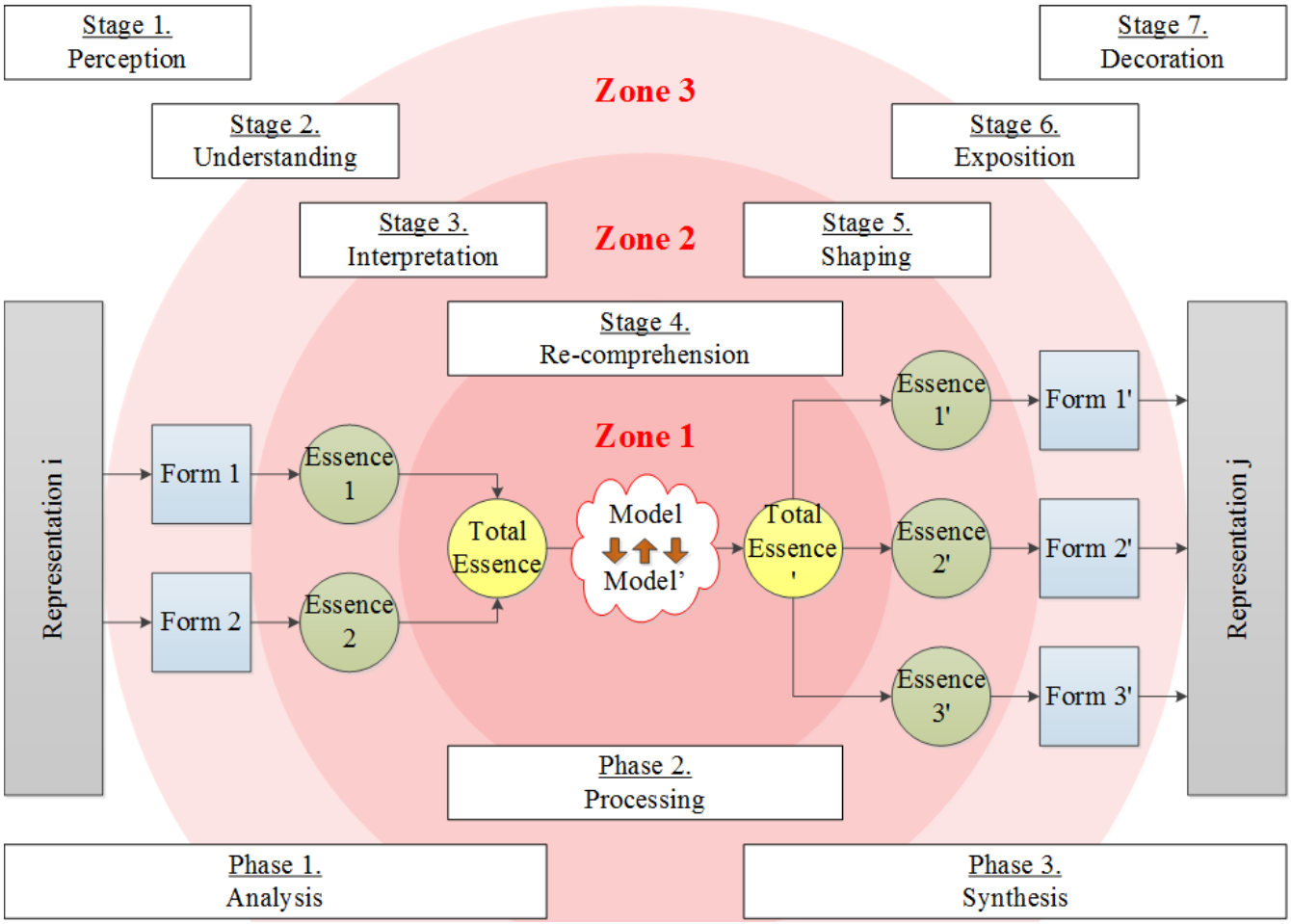
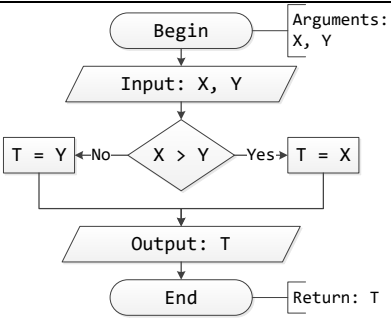


Fig. 1. Scheme of transformation of Representation and its graphic interpretation

Explanations to the diagram in the Fig. 1 for the example consisting in creation of the source code of function of a choice maximum of two numbers according to the given flowchart of an algorithm are provided in Table 2.

TABLE 2.
EXPLANATIONS TO THE SCHEME OF TRANSFORMATION OF REPRESENTATION AND EXAMPLE OF CODING

№	Result of transformation	Elements of the scheme of transformation (for example)
	Submission of the flowchart of an algorithm (Representation *i*)	
1	Visual elements of the flowchart of an algorithm (Form 1, 2)	«Begin» with receiving two variables: X and Y; «Decision» on transition (conditional branch) with a condition of comparing of two variables «X > Y»; two «Processes», each of which appropriates to temporary variable T one of two X and U variables; «End» with return of a variable T; unidirectional connectors of all elements;
2	Sense of elements of the flowchart of an algorithm (Essence 1, 2)	Obtaining value of the X and Y variables; comparing of the X and Y variables; if X > Y, then assignment of a variable T of variable X value , else – value variable Y; resetting of variable T value;
3	Sense of all algorithm (Total Essence)	Comparing of values of the X and Y variables and resetting maximum of them through variable T value;
4	Sense of the source code implementing the algorithm (Total Essence') received by the intermediate models (Model, Model')	Determination of function with arguments X and Y returning the value, maximum from them; use for storage of result of temporary variable T;
5	Sense of elements of the source code (Essence 1', 2', 3')	Targeting a signature of function with arguments of X and Y; comparing values of the X and Y variables of construction IF-ELSE; depending on result of comparing locating in temporary variable T of one of values of the X and Y variables; resetting from function variable value T;
6	Constructions of the source code according to a language syntax (Form 1', 2', 3')	Function title with two arguments X and Y; beginning and end of a function body; declaration of temporary variable T; conditional GOTO statement for «X > Y»; branches of the conditional branch on the separate units of a code containing assignment of a variable T of the X or Y variables; return statement from function of a variable T;

7	Representation listing of the source code (Representation *j*)	`int funct(int x, int y) {`
		`  int t;`
		`  if (x>y) {`
		`  t = x;`
		`  } else {`
		`  t = y;`
		`  }`
		`  return t;`
		`}`

Note. First column «№» means order number of Stage.

IV. TYPIFICATION OF VULNERABILITIES WHEN TRANSFORMING REPRESENTATION

As it was specified, vulnerabilities can appear in the course of change of Representations (incremental and step by step), it is expedient to enter their typification in the place in which they appeared – i.e. to the stage initiating vulnerability. So, the erratic understanding the coder on Stage will lead 2 senses of an element of the conditional branching (for example, the incorrect sign of comparing) to incorrect interpretation of a sense of all algorithm and, as a result, to writing of the incorrect source code (for example, returning the minimum number instead of maximum) even if all remained stages were executed absolutely truly.

The premeditation level corresponding to a conscious participle of the person to vulnerability appearance can be additional typification reasonably. The following can be such types of premeditation for the given stages:

- unwitting – shown in the mistakes made by the person not consciously i.e. owing to its physical or psychological state, or because of external factors;
- malicious – based on conscious entering of mistakes into Representations, for example, owing to personal ambitions, the order or the compelled reactions.

It is obvious that if the casual nature of vulnerabilities is inherent with different degree in all stages of transformation of Representations, then malicious can belong only to a stage of reconsideration and the next to him. It follows from the fact that around a stage there is sufficient degree of sensibleness of essence by the malefactor (corresponding to zones in the Fig. 1); at all other stages person just tries to understand sense of Representation through perception, and then to describe it.

V. ASSESSMENT OF SAFETY OF TRANSFORMATION OF REPRESENTATION

We will consider process of receiving each new Representation of safety change, previous from a position. We will enter the level of safety of Representation of *i* as density of his essence without vulnerabilities (or the relation of «volume» of essence without vulnerabilities to general «volume») and we will designate him the S_i parameter. It is obvious that for Representation without vulnerabilities of S_i=1. Safety of the following Representation *j* is defined by S_j; at the same time $S_j < S_i$ condition will be always met. Equality of $S_j = S_i$ is possible only at automatic transformation by the entrusted Utilities and won't be considered here. The situation when $S_j > S_i$ meaning that as a result of transformation, a part of vulnerabilities has disappeared and in general is extremely improbable (though some Utilities of assembly can signal about use of unsafe operations and offers options of their correction).

We will consider features of each of the specified SW development process stages and also types of vulnerabilities from the point of view of premeditation. For this purpose we will compare coefficient of increase in level of insecurity of Representation with each stage – $K_N^P(t)$, corresponding to

probability of emergence of vulnerability like P (U – Unwitting, M – Malicious) at the Stage *N* (from 1 to 7) in timepoint of t (a point on a scale of performance of stages). The value of coefficient can lie in the range *[0..1]*. Then coefficients influence the level of safety of Representation as:

$$S_j = \prod_{N=1..7} \left(1 - K_N^P(t)\right) \times S_i.$$

Thus, each stage due to «nonzero» probability of emergence of vulnerabilities inevitably reduces safety of new Representation.

The coefficient $K_N^P(t)$ has dependence on timepoint because of stages aren't carried out instantly, and have a certain duration in process of which (from the beginning of a stage before its end) vulnerabilities can appear with various speed (the corresponding derivative from $K_N^P(t)$ in a certain timepoint of t_0). Such assumption is competent as at some stages of vulnerability practically don't appear (because of simple subconscious work of a human brain), on some have the increased intensity (because of need to the person to work with the large volume of memory), and on some have growth rate, time-dependent (because of complex processing by a brain both the initial, and constantly obtained new information). We will consider coefficient $K_N^P(t)$ for each of a stage and type of premeditation of vulnerability.

For definition of a temporary type of coefficient of increase in level of insecurity we will use the simplified reasoning – but, nevertheless, reasonable logic and experience as definition of an exact formula for $K_N^P(t)$ demands carrying out a full-fledged research on a set of statistics (and its processing) concerning places and types of emergence of vulnerabilities on each of stages of transformation of Representations. Also we will divide a type of conversion within each stage from the point of view of topology into 4 groups:

– «one to one» (further, «*1:1*») or the transformation of one element to other element;

– «one to several» (further, «*1:N*») or the transformation of a uniform element to a set of other elements;

– «several to one» (further, «*N:1*») or the transformation a set of one elements in a uniform element;

– «several to several» (further, «*N:N*») or the transformation a set of one elements to a set of other elements.

VI. Influence of Subjective Components of the Person

The Stage 1 is defined by consecutive perception of elementary forms of Representation of *i* from a uniform form and is carried out by the person with good degree of automaticity, though in the form of «*1:N*». As essence by the person is still not understood, emergence only of casual vulnerabilities is possible. Thus, the stage can be characterized by the average value of coefficient having a linear appearance:

$$\begin{cases} K_1^U(t) = k_1^U \times t \\ K_1^M(t) \equiv 0 \end{cases}.$$

The Stage 2 is defined by consecutive understanding of elementary forms of Representation of *i* by their «replacement» by the corresponding elements of essence, i.e. in the form of «*1:1*». As the understanding of elementary essence doesn't give a full picture of sense an Idea, emergence only of casual vulnerabilities is possible. Thus, the stage can be characterized by low value of coefficient and have a linear appearance:

$$\begin{cases} K_2^U(t) = k_2^U \times t \\ K_2^M(t) \equiv 0 \end{cases}.$$

The Stage 3 is defined by consecutive interpretation of elementary essence of Representation of *i* in uniform essence which is carried out by the person with good degree of automaticity, though in the form of «*N:1*». As the person at the end of a stage already has an idea of full essence, besides casual vulnerabilities emergence of rare malicious vulnerabilities is possible. Thus, the stage can be characterized by average value of coefficient for casual and very low for malicious vulnerabilities and have a linear appearance:

$$\begin{cases} K_3^U(t) = k_3^U \times t \\ K_3^M(t) = k_3^M \times t \end{cases}.$$

The Stage 4 is defined by transformation of sense of Representation of *i* to sense of Representation of *j* (by their essence) which is made by the person most consciously. Transformation is similar to process of «the solution of a task» – poorly studied creative action. Nevertheless, the approximate type of coefficient of K_4^P can be received as a result of the following reasonings. Firstly, the stage isn't consecutive (on extremely measure, in the same degree as others) since there is no uniform algorithm of the decision any tasks – it is creative with emergence of Insight. Secondly, though transformation is also similar to a look «*1:1*», nevertheless, because of their dimensions of keeping of people is capable to work only with their parts that leads to a type of «*N:N*». And, thirdly, standard feature of work with non-standard tasks is existence of a step of assessment i.e. as far as the decision has come or has achieved the objectives. Thus, it is possible to assume that generally the stage will gradually transform one parts of essence of Representation *i* to other parts of essence of Representation *j*, checking at the same time each new received part for coordination both with initial, and with already received from the point of view of proximity to a goal.

According to the made assumption, with each transformation of a part of essence the number of the made actions increases by one – because of need of coordination of each following part for a limit with all previous. The last corresponds to the arithmetic sequence (which sum of members is equal: $1 + 2 + \cdots + n = \frac{(n+1)\times n}{2}$) and directly influences emergence of casual vulnerabilities.

As the person has complete idea of the essence of Representations, this stage is most preferable to the vulnerabilities introduced by the malefactor; at the same time, their addition can be carried out in process of transformation. Thus, the stage can be characterized by coefficient with the high value for casual vulnerabilities having a square appearance and an average for malicious vulnerabilities and

to have a linear appearance:

$$\begin{cases} K_4^U = k_4^U \times t^2 \\ K_4^M = k_4^M \times t \end{cases}.$$

The Stage 5 is defined consecutive decomposition of uniform essence of Representation of *j* in elementary essence which is carried out by the person with good degree of automaticity, though in the form of «*1:N*». As the person at the beginning of a stage still operates with full essence, besides casual vulnerabilities emergence rare malicious is possible. Thus, the stage can be characterized by average value of coefficient for casual and very low for malicious vulnerabilities and have a linear appearance:

$$\begin{cases} K_5^U(t) = k_5^U \times t \\ K_5^M(t) = k_5^M \times t \end{cases}.$$

The Stage 6 is defined by consecutive formation of elementary forms of Representation of *j* by «replacement» of the corresponding elements of essence with them, i.e. in the form of «*1:1*». As work with elementary essence doesn't allow to operate with full sense of Representation, emergence only of casual vulnerabilities is possible. Thus, the stage can be characterized by low value of coefficient and have a linear appearance:

$$\begin{cases} K_6^U(t) = k_6^U \times t \\ K_6^M(t) \equiv 0 \end{cases}.$$

The Stage 7 is defined by the consecutive description of a uniform form of Representation of *j* from elementary forms which is carried out by the person with good degree of automaticity, though in the form of «*N:1*». As the person have completely departed from the essence of Representation, emergence only of casual vulnerabilities is possible. Thus, the stage can be characterized by average value of coefficient and have a linear appearance:

$$\begin{cases} K_7^U(t) = k_7^U \times t \\ K_7^M(t) \equiv 0 \end{cases}.$$

It is expedient to write down the level of insecurity of Representation a two-component vector which elements set density of each of types of vulnerabilities (casual and malicious): $S_i \equiv [S_i^U, S_i^M]$. The general deterioration in safety in this case and also according to the entered coefficients, will be defined by the equation $[S_j^U, S_j^M] = M \times [S_i^U, S_i^M]$, where M – a metrics (in the form of a single matrix) increases in insecurity when transforming Representations. The metrics is defined how:

$$\mathrm{M} \equiv \prod_{N=1..7} \left(\begin{bmatrix} 1 - K_{0_N}^{\ U} & 0 \\ 0 & 1 - K_{0_N}^{\ M} \end{bmatrix} \times [S_i^U, S_i^M] \right),$$

where $K_{0_N}^{\ P}$ – coefficient of increase in level of insecurity on the termination of a stage *N* for type of premeditation of vulnerability of U or M. Thus, the metrics doesn't depend on time and her components are calculated as number. Unlike existing (diverse and specialized for own tasks, for example [12, 13, 14, 15]), the entered metrics at rather high level of abstraction sets safety of the gained Impression on everyone a stage of interaction of people code. At the same time, the metrics allows to predict safety level in the future as it depends only on the chosen life cycle of creation SW.

In case of more real reflection of safety the matrix of *M* won't be single since unwitting and malicious vulnerabilities exert impact at each other on each of stages of transformation of Representations. Formalization of this situation is more difficult and demands carrying out additional researches.

Graphic interpretation of coefficient $K_N^P(t)$ for various stages and types of vulnerabilities, illustrating their distinctions and features, has the following stylized appearance (Fig. 2):

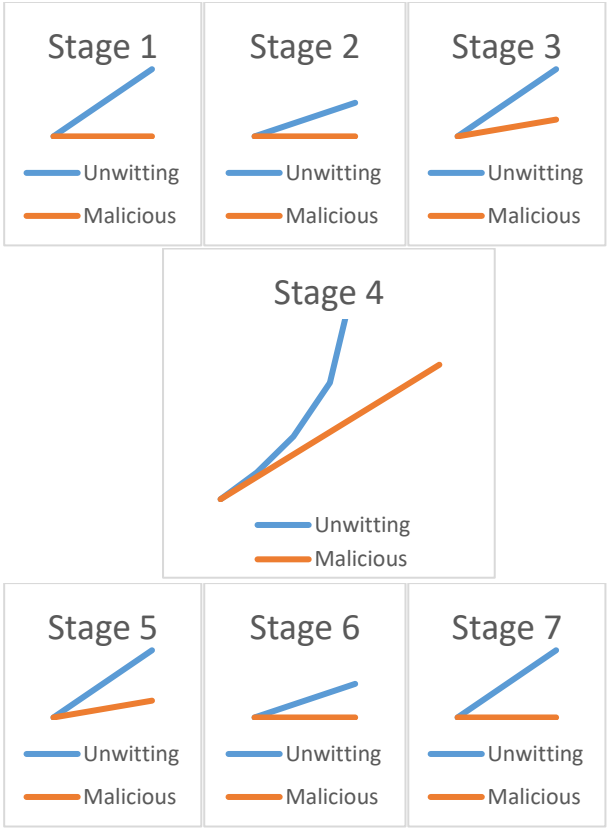


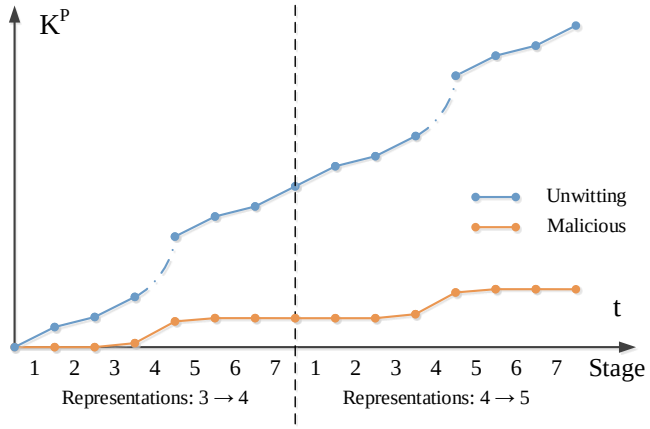
Fig. 2. Graphic interpretation of coefficient of increase in level of insecurity of Representations for each phase

VII. Comparison of Safety of Transformation of Representations for Standard Scenarios of Software Development

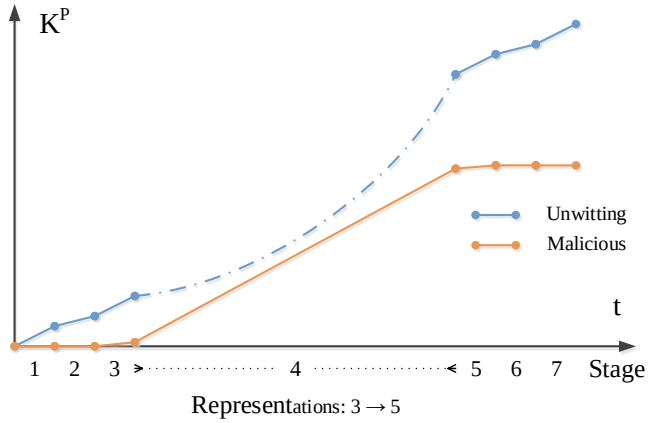
As an example of potential applicability of the explained approach to assessment of safety we will make hypothetical comparing of two methods of creation of a program code (Representation 5) on its architecture (Representation 3) – with use of the intermediate algorithmized representation (Representation 4) and without it. In the second case the programmer receives the essence of Representation of algorithms of a code implicitly in the course of reconsideration of essence of Representation of architecture in Representation of the source code, passing thereby phases of synthesis of algorithmized Representation, its explicit processing and the subsequent analysis.

With sufficient degree of convention, but without loss of

sense, change of the general coefficient of increase in level of insecurity of Representations $K^P(t)$ (as compound of $K_N^P(t)$ on each phase N) for both types of vulnerabilities ($P = \{I, M\}$) in a case from use of algorithmized Representation (further – Case 1) and without him (further – Case 2) is shown on the following schedules (Fig. 3).



a) transformation through Representation 4 (3→4→5)



b) transformation without Representation 4 (3→5)

Fig. 3. Graphic interpretation of the general coefficient of increase in insecurity when transforming Representations 3 in 5

At the Fig. 3 the dashed line has shown coefficient $K^P(t)$ with square dependence on t time. Units of measurements of schedules and the scale of Stage 4 on graphics 3b are conditional, designed to reflect the general behavior of coefficient and difference between him for two options of transformation.

The analysis of graphics 3) and 3) in the Fig. 3 allows to draw the following conclusions. First, in Case 1 using of intermediate Representation 4 means emergence of additional Stages 5, 6, 7 (for transformation of Representations 3 → 4) and Stages 1, 2, 3 (for transformation of Representations 4 → 5) that inevitably leads to accumulative increase in insecurity of Representation 5 because of casual vulnerabilities. Secondly, in Case 2 the refusal of use of intermediate Representation 4 means increase in Stage 4 twice that leads to a bigger increase in insecurity of Representation 5 because of casual vulnerabilities (because of a square type of coefficient $K_4^U(2t)$). Thirdly, existence in Case 2 twice of smaller quantity of Stages 3, 4, 5 having not zero $K_N^M(t)$ leads to smaller increase in insecurity of Representation 5 because of malicious vulnerabilities, than in Case 1.

According to the above-mentioned analysis, it is possible to claim that the choice of each of the described ways of creation SW results in various probability of emergence of vulnerabilities in Representation, at the same time the quantity of their one type can increase, and another to go down.

VIII. Conclusion

The described approach to safety assessment SW for various stages of his creation allows to create the formalized mathematical apparatus, as for assessment of current state of safety SW, and his forecasting (for example, as in [16]). The last one helps to make the reasonable choice among different variations of SW development, calculating at the same time potentially dangerous places. Collecting expert opinions and statistical data which will allow to set necessary invariant parameters of the device is necessary for transformation of a theoretical research into practical tools. The future of development of a research can develop into full processing of all process of creation SW – removal of one of his representations, addition of others, modification of the third – that will allow to receive final SW set safety level. Also, adaptation of process of creation safe SW under the field of its application, such for example as telecommunication devices is possible.

Reference

[1] M. Buinevich, K. Izrailov, A. Vladyko, "The life cycle of vulnerabilities in the representations of software for telecommunication devices," in Proc. 8th IEEE International Conference on Advanced Communication Technology (ICACT), South Korea, PyeongChang, 2016, pp. 430-435.

[2] K. Xinogalos, "Using flowchart-based programming environments for simplifying programming and software engineering processes," IEEE Global Engineering Education Conference (EDUCON), Germany, Berlin, 2013, pp. 1313-1322

[3] M. Buinevich, K. Izrailov, A. Vladyko, "Method for Partial Recovering Source Code of Telecommunication Devices for Vulnerability Search," in Proc. IEEE 17th International Conference on Advanced Communication Technology (ICACT), South Korea, PyeongChang, 2015, pp. 76-80..

[4] M. Buinevich, K. Izrailov, A. Vladyko, "Method and Prototype of Utility for Partial Recovering Source Code for Low-Level and Medium-Level Vulnerability Search," in Proc. IEEE 18th International Conference on Advanced Communication Technology (ICACT), South Korea, PyeongChang 2016, pp. 700-707.

[5] M. V. Buinevich, K. E. Izrailov, "Method and utility for recovering code algorithms of telecommunication devices for vulnerability search," in Proc. IEEE 16th International Conference on Advanced Communication Technology (ICACT) , South Korea, PyeongChang, 2014, pp. 172-176.

[6] P. Tripathy, K. Naik. "Reengineering," Software Evolution and Maintenance: A Practitioner's Approach, 2014, pp. 133-186.

[7] R. Aliyev, L. Peñalver, "Analyzing Vulnerability Databases," in Proc. 10th IEEE International Conference on Application of Information and Communication Technologies (AICT), Baku, Azerbaijan, 2016.

[8] A. Fedorchenko, I. Kotenko, A. Chechulin, "Design of Integrated Vulnerabilities Database for Computer Networks Security Analysis," in Proc. 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Turku, Finland, 2015, pp. 559-566.

[9] M. Gernsbacher, M. Kaschak. "Text Comprehension," The Oxford Handbook of Cognitive Psychology, 2013, pp. 462-474.

[10] M. Yeari, "The role of working memory in inference generation during reading comprehension: Retention, (re)activation, or suppression of verbal information?," Learning and Individual Differences, 2017, Vol. 56, pp. 1-12.

[11] M. Raghothaman, Y. Wei, Y. Hamadi. “SWIM: Synthesizing What I Mean – Code Search and Idiomatic Snippet Synthesis”, in Proc. IEEE 38th International Conference on Software Engineering (ICSE), USA, New York, 2016, pp. 357-367.
[12] J. Knight, B. Rodes, K. Wasson, “A security metric based on security arguments,” In Proc. 5th International Workshop on Emerging Trends in Software Metrics, USA, New York, 2014, pp. 66–72.
[13] X. Cheng, N. He, M. Hsiao, “A New Security Sensitivity Measurement for Software Variables,” In Proc. IEEE Conference on Technologies for Homeland Security, USA, Waltham, 2008, pp. 593-598.
[14] Metrics and measurement of trustworthy systems // Military Communications Conference (MILCOM). IEEE. 2016. pp. 1237-1242.
[15] J. Cho, P. Hurley, S. Xu “Metrics and measurement of trustworthy systems,” in Proc. IEEE 35th Military Communications Conference (MILCOM 2016), Baltimore, United States, 2016, pp. 1237-1242.
[16] M. Buinevich, P. Fabrikantov, E. Stolyarova, K. Izrailov, A. Vladyko "Software Defined Internet of Things: Cyber Antifragility and Vulnerability Forecast," in Proc. IEEE 11th International Conference on Application of Information and Communication Technologies (AICT), Moscow, Russia, 2017, pp. 293-297.

Mikhail Buinevich received the D.Sc degree in Engineering from the Naval Institute of Radio Electronics, St. Petersburg, Russia, in 2010.
He has 20 years of experience in research and development in IT security. Now he is a Professor at The Bonch-Bruevich Saint-Petersburg State University of Telecommunications and supervises post graduates.
Prof. Buinevich is the Editor-in-chief of “Proceedings of Telecommunication Universities” scientific journal.

Konstantin Izrailov defense his degree of PhD at The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Russia in 2017. Now he is an Associate Professor at that University.
Mr. Izrailov interests include: Information & Network Security; Software-Defined Networking; Internet of Things.

Andrei Vladyko (IEEE member (M'14)) acquired his degree of PhD at Komsomolsk-on-Amur State Technical University, Russia in 2001.
At present he is a head of R&D department of The Bonch-Bruevich Saint-Petersburg State University of Telecommunications.
Mr. Vladyko major interests include: Information & Network Security; Software-Defined Networking; Internet of Things; Wireless Sensor Network.