ASPECT-ORIENTED SOFTWARE RELIABILITY ENGINEERING

Igor Safonov
●
International Unity Science Institute
1011 Arlington Blvd., Suite 403
Arlington, VA 22209, USA
isafonov@aol.com

ABSTRACT
Aspect-Oriented Approach to Software Development allows us effectively to effectively extract, evaluate and solve the main problem of contemporary tendency in Information Technology (particularly, in an Application Software) – a unification is alternated by a personalization. Increasing customer concerns about Performance, Quality, Reliability and Security (PQRS concept) can be satisfied only by symbiosis synergy of adequate models, techniques and tools on all stages of the Software lifecycle. We propose original methodology, formal models and simple methods of Software Reliability Engineering based on our many years experience of concern separation and aspect orientation in Software Development for Specialized Computers, Business Application and Government Institutions.

Keywords: Aspect-Oriented Software Development, Software Reliability Engineering, Application, Optimization, Model Driven Architecture, Formal Foundations, Structure-Algorithmic System

1. INTRODUCTION

By the late 20th century, two main tendencies of Software Engineering (SE) – extensification and unification were alternated by intensification and personalization. The changing reflected on formulation and decision of optimization problems. Multi-years experience in SE make us strong supporters of the Aspect-Oriented Software Development [12, 13] and allowed us to create and use the Customer-Oriented Approach (COA) to SE for Reliability (Performance, Quality, Security, Unification, etc.) [19]. The COA combines ideas of several directions in SE, requiring global (External-Internal, Behavior-Structure) and local (Property-Oriented) separation of concerns (Architecture Alternatives [1], Goal Patterns [2], Process Algebra [3, 11], Formal Languages [6, 11], Workflow [4, 14], Exception Handling [17, 21]), Application-Oriented Operating Systems [10], and Evolutionary Games [8, 9]). In this article, we limit our Software Reliability Engineering by the example of Checkpointing-Recovery System [5, 15, 16, 18, 20] widely accepted by developers.

The Ontology of Software Reliability Engineering specifies External (Functions) and Internal (Aspects) any Software Application at different stages of Development by the quadruple of correspondent Structure, Behavior, Goal and Resource models and their decomposition (top-down approach). The Ontology uses unified Extended Algorithm Algebra Language (XAAL) for separation of Internal Behavior and Structure concerns in process of Analysis, Optimization and Synthesis with required or possible degree of detailed elaboration for every stage of Software lifecycle (starting from Customer Requirements and Formal Specification, and finishing with Deployment and Maintenance). The simple and fast optimization techniques make the adaptation of Software Internal Behavior possible in order to compensate restrictions of universal operating systems and personalize application properties.

Aspect-Oriented Approach to Software Reliability Engineering was implemented for dozens of Aircraft, Spacecraft and Submarines Navigational Computers, for the Tax Modernization System, for Year 2K Problem, for Search Engines, and for a Government (including the Personal DSS for a Country President). In this time, we are designing Safety Aspect-Oriented Software of the Emergency Loss Prevention System (ELPS) for local governments with severe Requirements and Restrictions to Performance, Reliability, and Security. The ELPS is based on the Safety Wireless Network for Incident Management.

The proposed Software Model can specify any complex or simple software system being viewed as an object of design, at different stages of design, with the needed level of details $M_i(t)$, $i = 1, 2, \dots, K$, in which the structure and behavior of the system, the goals and resources of its design are represented, i. e.

$$M_i(t) = < MS_i(t), MB_i(t), MG_i(t), MR_i(t) >, \quad i = 1, 2, \dots, K, \qquad (1.1)$$

where $MS_i(t)$ – Structure Model, $MB_i(t)$ – Behavior Model, $MG_i(t)$ – Goal Model, $MR_i(t)$ – Resource

Model, and t – time.

In process of design by stages, we accomplish that level of detailing, which allow us to develop the documentation needed for design and subsequent production of designed system. If MBi (t) , i = 1, 2, … , K , is a program, all functional and logical operators of which can be structurally interpreted in terms of models MSj (t) , i <= j <= K , then Mi (t) , i = 1, 2, … , K, will be called the Structure-Algorithmic System (STALS). The process of STALS development is the multistage sequential (in most cases, labor-intensive and time-consuming) approximation to the required (needed and sufficient) for implementation level of details. For example, in the design process of computer-aided control systems for technological processes the stages of feasibility study, technical design and functional coding are executed, in the special computer design – the stages of system, algorithmic, logic-functional and technical development, in the applied software package design – the stages of requirements, specification, design, programming, and testing.

2. SOFTWARE RELIABILITY SYNTHESIS

By the Software Reliability Synthesis Model of STALS we shall name the formal specification of the software for realization of the designed system's external (functions) and internal (aspects) behavior containing information sufficient for the functionally equivalent additions to and transformations of the specification with the purpose to control reliability parameters of the system, i. e. sufficient for Formal Software Reliability Synthesis. As Reliability Synthesis Model of Structure-Algorithmic System, we shall use the submodel < MRSS, MRSB > of the model < MS, MB >, where MRSS – the Model for Reliability Synthesis of Structure, MRSS is a part of MS, and MRSB – the Model for Reliability Synthesis of Behavior, MRSB is a part of MB.

Because, eventually the goal of Reliability Synthesis of control components and systems consists of giving the detecting and correcting features to the object functioning in real-time scale, we therefore can execute Reliability Synthesis to fit the MRSB, making the corresponding additions to and transformations of MRSS. In doing so, the working hypothesis is the statement that the level of abstraction can be chosen in such way that to every operator (logical condition) we can set in accordance with the structure component realizing this operator (logical condition).

We shall add to and (or) transform of MRSB by governing relationships of the form:

$$A1 = A2, \tag{2.1}$$

where A1 – an arbitrary Reliability Design Component (any operator or logical condition out of MRSB, any syntactically correct, i. e. canonically represented, part of MRSB, or the complete MRSB), and A2 – an algorithm functionally equivalent to A1, except that A1 and A2 vary in their detecting and correcting properties, or in probabilities of correct execution, or in levels of other particular parameter of the quality or cost.

Let P (A1) ≠ P (A2), where P (A1) – a reliability parameter (in this case – the probability of correct execution) of A1, and P (A2) – a similar reliability parameter of A2. In the majority of practically interesting cases of Reliability Synthesis P (A1) ‹ P (A2), and the alternatives when P (A1) › P (A2), can result from the Design for Reliability of objects with natural redundancy (for example, in process of Design for Reliability of the STALS, realizing on the basis of heterogeneous or homogeneous computer networks [5, 14, 16]). Other quality parameters (timeliness, accuracy, etc.) and a cost of A1 and A2 can be different, but sometime can be the same. By the Formal Reliability Synthesis of programs, we shall name the sequence (may be one-component) functional equivalent additions to and (or) transformations of MRSB by the governing relationships of the form (2.1).

For initial illustration of XAAL, a simple example may be useful. The exception handling pseudo code is from [17]:

A = a3(a2(A1*a1(A2 V E) V A2) V A3*a4(a2(A1*a1(A2 V E) V A2) V E)),

where A1 ::= call a local handler; A2 ::= go to a higher (action) level handling;
A3 ::= local error detection; a1 = 1, if "not handled", else a1 = 0
a2 = 1, if "local handler exist", else a2 = 0
a3 = 1, if "exception is propagated from the component", else a3 = 0

a4 = 1, if "exception is raised (error has been found)", else a4 = 0
or

$$A = a3(A4 \vee A3*a4(A4 \vee E)),$$

where A3 ::= local error detection;
a3 = 1, if "exception is propagated from the component", else a3 = 0
a4 = 1, if "exception is raised (error has been found)", else a4 = 0

$$A4 ::= a2(A1*a1(A2 \vee E) \vee A2),$$

where A1 ::= call a local handler; A2 ::= go to a higher (action) level handling;
a1 = 1, if "not handled", else a1 = 0
a2 = 1, if "local handler exists", else a2 = 0

The methodology of Formal Reliability Synthesis of Algorithms can be illustrated by an application example of the very common governing relationship:

$$A = \nu\{ K \}\cdot A \cdot\omega\{ \pi(E \vee K \cdot\nu\{ K \})\cdot A \} , \qquad (2.2)$$

where A – an arbitrary Component of the Design for Reliability, K – a restoration operator,
ν – a logical condition "The realization of A is capable to work", defined thus:
ν = 1 , if the realization of A is capable to work, ν = 0 otherwise;
ω – a logical condition "A executed correctly",
π – a logical condition "Malfunction", defined in the same way as ν.

It is known (for example, [7]) that nearly 90 % of computer errors are the input/output errors and nearly 90% of computer failures require halt/reboot activity. It corresponds universally recognized but not generally accepted concept of pre-operational testing and diagnosis and post-operational detecting and correcting. Let us notice that an operator K is, in general, complex operator representing, for example, the next sequence of operators K = K1 · K2 · K3 , where K1 – a procedure for search of an inoperative components in realization of A, K2 – a procedure for restoration of capability to work of the A realization, K3 – a procedure for testing of capability to work of the A realization.

In different computer systems, most often the procedure K3 is realized by a hardware or hardware-software composition, the procedure K1 – by a software, and the procedure K2 – by a peopleware. In essence, implementation of governing relationship (2.2) implies the next. Before an execution of A, the testing of capability of it is performed. If the A realization is capable, then operator A is executed, otherwise operator K has been executed until the capability of the A realization is completely restored, and then the A is executed.

After execution of the A, the checking of correctness of its execution is performed. If the A was executed correctly, then in accordance with MRSB, an execution of next operator or checking of next logical condition is initiated. If the A is executed improperly, then a cause of its improper execution is clarified – "malfunction" or "fault". In case of malfunction, the execution of operator A is repeated, and in case of fault, the operator K is executed until the capability of the A realization is completely restored, and then operator A is executed. After the repeated execution of A, again the checking of its correct execution is performed. If A is correctly executed, then in accordance with MRSB, an execution of next operator or checking of next logical condition is initiated, and so forth.

Because different realization variants of ν, ω, π and K are feasible, the selection problem of the best their combination (optimization problem) for certain set of components of the Design for Reliability is complex and labor consuming (especially, taking the fixing a set of detecting and correcting means for one component restricts the selection of corresponding means for other components into account). However, in design of a data processing and management information systems on the base of unified computers and peripherals, and also in design of software packages for universal computers, we are forced to use the finished means of realization of ν, ω and π under restricted and regulated options of realization of K more frequently. Because of this, in the majority of cases we don't need to use all options presented with realization of the governing relationship (2.2), and in process of Formal Reliability Synthesis of Algorithms we can use more simple governing relations, which are specific cases of relationship (2.2). One such formal technique to obtain these relationships is fixation of corresponding Boolean variables ν, ω and π identically equal to 1 (i. e., a fixation of the identical truth of corresponding logical conditions).

Under the condition that ν = 1, we obtain

$$A = A \cdot\omega\{ \pi(E \vee K \cdot)\cdot A \} . \qquad (2.3)$$

Under the condition that ω = 1, we obtain

$$A = \nu\{ K \}\cdot A . \qquad (2.4)$$

Under the condition that $\pi = 1$, we obtain

$$A = \nu\{ K \} \cdot A \cdot \omega\{ A \} . \quad (2.5)$$

Under the condition that $(\nu = 1)$ & $(\omega = 1)$, we obtain

$$A = A, \quad (2.6)$$

which is degenerate, i. e. its application means that no detecting and correcting features are conferred to the operator A .

Under the condition that $(\nu = 1)$ & $(\pi = 1)$, we obtain

$$A = A \cdot \omega\{ A \} , \quad (2.7)$$

Under the condition that $(\omega = 1)$ & $(\pi = 1)$, we obtain the governing relationship (2.4), and under the condition that $(\nu = 1)$ & $(\omega = 1)$ & $(\pi = 1)$ – the relationship (2.6).

It is obvious that the governing relationship (2.3) is applied in the case when kA >> P (A) (here, kA – availability coefficient of the A realization), the governing relationship (2.4) is applied in the case when kA << P (A), etc.

3. SOFTWARE RELIABILITY ANALYSIS

In XAAL, the sequential execution of operators (any units of action) A1, A2, … , AN corresponds to the multiplying of these operators:

$$A = A1 * A2 * \dots * AN. \quad (3.1)$$

Let us assume that PA (TA), PA1 (TA1), PA2 (TA2), … , PAN (TAN) – the correct execution probabilities of the operators A, A1, A2, … , AN in the execution times T, T1, T2, … , TN respectively. It is obvious that

$$PA (TA) = [i = 1, N] \Pi PAi (Tai), \quad (3.2)$$

and

$$TA = [i = 1, N] \Sigma Tai. \quad (3.3)$$

For simplicity of the program execution reliability analysis, we use the generalized (canonical) a-disjunction operation instead of a-disjunction operation. The result A of its application to ordered operator sequence A1, A2, … , AN takes the form

$$A = a1(A1 \vee a2(A2 \vee \dots \vee a[N-1](A[N-1] \vee A[N])) \dots) , \quad (3.4)$$

where a1, a2, … , a[N-1] – logical conditions.

Let us assume that pai = Probability (ai = true). Then, supposing that logic conditions are checked momentarily and absolutely reliably, the correct execution probability of the disjunctive canonical program specified by the expression (3.4) is defined by formula

$$PA (TA) = [i = 1, N] \Sigma pai * PAi (TAi) + (1 – pa[N-1]) * PAN (TAN), \quad (3.5)$$

where

$$TA = [I = 1, N] \Sigma pai * TAi + (1 – pa[N-1]) *TAN. \quad (3.6)$$

The expression

$$B = a\{ A \} \quad (3.7)$$

means that the operator B is the result of the a-iteration operation to the operator A.

The correct execution probability of the operator B is defined by the formula

$$PB (TB) = pa / (1 – (1 –pa) * PA (TA), \quad (3.8)$$

where

$$TB = ((1 – pa) / pa) * TA. \quad (3.9)$$

If operators A1, A2, … , AN are executed in parallel the case may be noted

$$A = [A1, A2, \dots , AN] , \quad (3.10)$$

and we can use the formulas

$$PA (TA) = [i = 1, N] \Pi PAi (TAi), \quad (3.11)$$

$$TA = \max (TA1, TA2, \dots , TAN). \quad (3.12)$$

Because for modeling any program execution, it is enough to use operators (3.1), (3.4), (3.7) and (3.10), the probabilities of program correct execution and execution times can be evaluated by formulas (3.2), (3.3), (3.5), (3.6), (3.8), (3.9), (3.11) and (3.12).

We are reminded that the Aspect-Oriented Approach allows us separately develop External and Internal Behavior of Software. In context of Design for Reliability, the External Behavior executes

functional data processing without taking into account the natural (unpremeditated) errors, malfunctions, faults, conflicts, etc., and the Internal Behavior realizes aspect data processing tolerated the program execution to conditions of above-mentioned interferences. In context of a Design for Security, the Internal Behavior realizes aspect data processing tolerated an access to and execution of programs to conditions of artificial (premeditated) interferences, without intervention in External Behavior.

The common property for Design for Reliability and Design for Security is redundancy, which can be shared between them only in exceptional situations. It is why resources for Reliability and for Security must be separated in majority cases.

The next technique of automated correction of Software execution uses the functional redundancy approach. Suppose some functional equivalent programs for the same A problem processing are ordered with accordance to their priority: A1, A2, … , AN. Suppose further that ν1, ν2, … , νN – the conditions of potential possibility for correct execution (availability) of A1, A2, … , AN , correspondently. Formally, we can write

$$A = \nu 1(A1 \vee \nu 2(A2 \vee \dots \vee \nu N(AN \vee W)) \dots), \quad (3.13)$$

where W – the signal about the potential impossibility for correct execution of A.

A probability of correct execution of A is

$$PA(TA) = [I = 1, N] \Sigma\, kAi * Pai(TAi) * [l = 1, N; l \neq i] \Pi (1 - kAl),$$

where

$$T = [I = 1, N] \Sigma\, kAi ([l = 1, N; l \neq i] \Pi (1 - kAl))\, TAi ,$$

The availability levels kAi , i = 1, 2, … , N , are calculated by traditional structured reliability techniques.

Example. The program SK with structure (hardware) error detection and behavior (software) execution correction (aspect K) looks like

$$SK = S * \varphi(K * \varepsilon(S \vee W) \vee E), \quad (3.14)$$

where S – the functional equivalent of SK, but without detection and correction; φ – the logical condition of error detection in process of S execution; ε -- the logical condition of data renewal for S repetition; K – the correction code (aspect): $K = A * \alpha(\beta\{BC\}\, \chi(\delta(F \vee D) \vee D) \vee F)$,

where A – addition of the number of malfunctions to the malfunction counter, B – address forming for the type malfunction counter, C – increment of the type malfunction counter, D – restoration of data for functional code (in our case, for S program), F – transfer of control to emergency diagnostics, α -- logical condition “common number of malfunctions less than N”, β -- “the number of given type malfunctions has been counted”, χ -- “the current and foregoing malfunctions occurred in the same point of functional code”, and δ -- “the current and foregoing malfunctions have the same type”.

Let us calculate change of the mean time and correct execution probability of S, realized by technique (3.14), if we know that TS = 100 sec., and PS (100) = 0.9. Also, PA (TA = 2 sec.) = 0.999, PB (2) = 0.999, PC (1) = 0.9995, PD (5) = 0.99, PF (2) = 0.999, pα = 0.8, pβ = 0.5, pχ = 0.1, and pδ = 0.1. First, we calculated, that TK = 8.776 sec., and PK (8.776) = 0.9896. Then, with help of (3.14): TSK = 110.8 sec., and P SK = 0.989. Thus, the mean time of functional program S execution, after addition the error detection and correction of aspect code, increased less than 11%, but the error probability decreased approximately 9 times.

Consideration is being given to the case, when the governing relationships being used in the Reliability Synthesis of the STALS are special cases of the relationship

$$A = \varphi\{ K \}\, A\, \omega\{ \pi(L \vee K)\, A \} , \quad (3.15)$$

where A – a component of the Design for Reliability [1], with τA (the execution time of A) – a random value with the distribution function FA(t), and the A execution process is accompanied by the Poisson stream of malfunctions with the intensity Λm and Poisson stream of faults with the intensity Λf; L – the operator of restoration of conditions sufficient for repeating of A after a supposed malfunction; TL (the execution time of L) – the random value with the distribution function:

FL (t) = FLm\f (t) , if a malfunction happened, but a fault did not,

FL (t) = FLf (t), if a fault happened, and FL (t) = FLmf (t) in all other cases.

K – the operator of restoration of conditions sufficient for repeating of A after a supposed fault;

TK (the execution time of K) – the random value with the distribution function:

FK (t) = FKm\f (t) , if a fault happened,

FK (t) = FKf (t), if a malfunction happened, but a fault did not and

FK (t) = FKmf (t) in all other cases;
φ – the logical condition "realization of A is capable of working";
τφ (φ testing time) – a random value with the distribution function Fφ (t),
Pφ1 – the probability of a first type mistake (i. e., the probability of mistaken verification of a capacity for work of A realization),
Pφ2 – the probability of a second type mistake (i. e., the probability of mistaken verification of a unable to work of A realization),
ω – the logical condition "A execution is correct",
τω -- the random value with the distribution function Fω(t),
Pω1 – the probability of a first type mistake (i. e., the probability of mistaken verification of a correct execution of A realization),
Pω2 – the probability of a second type mistake (i. e., the probability of mistaken verification of a incorrect execution of A realization),
π – the logical condition "a malfunction is the cause of error",
Pπ1 and Pπ2 – the probabilities of corresponding mistakes of the first and second type (i. e., the probability of mistaken verification of a malfunction or fault).

The state of SAS at the instant t is a pair (X(t), Y(t)), where

$$X(t) \in \{-1, 0, 1\}, \quad Y(t) \in \{ A, L, K, \varphi, \omega, \pi, S, F \}.$$

X(t) – the parameter, characterized the Structure State of SAS and specified as follows:
X(t) = -1 , if in the instant t A realization is unable to work,
X(t) = 0 , if at the last A execution, preceded the instant t, a malfunction happened, but a fault did not, X(t) = 1 in all other cases;

Y (t) – the parameter, characterizing the Algorithmic State of SAS and specifying by a operator (logical condition) execution (testing) in the instant t, and S (F) – an arbitrary component of the Design for Reliability, an ending (beginning) of its execution is interpreted as the beginning (ending) of execution of the right side of the relationship (1).

The computational formulas are obtained for determination of the distribution function, mean time and dispersion of the execution time of the right side of relationship (1) and the probability of its correct execution in a given time. The application package is proposed for a statistical modeling of the STALS with purpose of a time-probability parameter estimation and optimization model stability testing.

4. SOFTWARE RELIABILITY OPTIMIZATION

Analysis of the accumulated experience in formulation and solution of optimization problems in design of algorithms and programs for computerized control and data processing systems can be useful for elaboration of profitable approach to formulation and solution of the appropriate optimization problems in conditions of a computer-aided design. Developers and customers of a developing object pursue the next prime (to a large degree, contradictory) goals.

1. A customer takes an interest to obtain the optimal in some a sense control system. This is most commonly done by specifying one or more quality parameters (efficiency, performance, etc.) of the design object for setting up an optimization problem. As this takes place, a certain amount of design resources identified with the design object can be expressed by the generalized cost criterion of manufacturing, operation and evolution of the object. In each specific case, this criterion can have one or another interpretation, for example it can characterize the realization complexity, volume of r/w memory, overall dimensions, energy consumed by the control system, number and skill of maintenance personnel. The identical with those problems, aimed at optimization of a design object, will be named the Design Object Optimization (Optimization Design).

2. Developers and customers of a control system take an interest to the Design Process Optimization. The interests of developers and customers coincide in an attempt to increase the productivity of a Computer-Aided Design System: a developer endeavors to timely meet contractual obligations to create a control system, and a customer – to timely obtain a capable to work system complete with operational documentation. However, a customer is less interested in the generalized cost of the design system (although the design cost may reflect on the project price) than a developer. At the same time, the quality parameters of a project documentation (especially, its validity) are the governing factors for a customer, because poor

project quality manifests itself in the time of operation in the most unexpected and undesirable form. Notice that the pace of a moral aging of methods and means for control of technological and business processes are ahead of the pace of control system development.

3. Neither the Design Object Optimization, nor the Design Process Optimization results the potentially possible effect of control automation without rational richness of control systems by models, algorithms, software and hardware for the Control Optimization. The realization volume of Control Optimization methods immediately not only causes on cost parameters of Control Systems, but it also influences on cost parameters of their Design Process. The interrelation between the realization volume of Control Optimization and many of quality parameters of Design and Control Systems is also obvious.

To partly illustrate the foregoing, let us cite one example of decision of Design Problem Optimization. The object of Design Problem Optimization is the algorithm of single execution of the control program

$$B = [A0, A4]\ \alpha5\{\ \alpha1(\ A1\ V\ \alpha2(\ A2\ V\ \alpha3(\ A3\ V\ E\ )))\}\ A5\ .$$

(Interpretation of functional operators and logical conditions of the algorithm are contained in [18]. There is also more information about the XAAL.)

It needs to minimize the average time of the algorithm B execution under condition that a probability of its correct execution is no less than 0.995. The required accuracy of the problem solution is five decimal places. The initial data (the durations and correct execution probabilities of functional operators A0, A1, A2, A3, A4 and A5, true probabilities of logical conditions α1, α2, α3 and α5):

t0 = 0.01 min., t1 = 1 min., t2 = 1 min.,
t3 = 5.00 min., t4 = 0.001 min., t5 = 0.001 min.,
P0 = 0.999, P1 = 0.900, P2 = 0.950,
P3 = 0.950, P4 = 0.999, P5 = 0.999,
p1 = 0.800, p2 = 0.150, p3 = 0.999, p5 = 0.100.

For organization of error detection and correction of the system's internal behavior, in efforts to enhance its reliability, we apply the next governing relationship

$$Ai = Ai\ \omega i(E\ V\ Ai\ \omega i(E\ V\ \ldots\ Ai\ \omega i(E\ V\ Ai))\ \ldots\ )\ ,$$

$$\text{-------}$$

$$Xi$$

where ωi – logical condition of correct execution of the functional operator Ai, E – the identical operator.

Let us presume

$$Pi\ (Xis) >= 0.99999\ ,\quad i = 0, 1, 2, 3, 4, 5\ ,\quad (s - \text{sufficient})$$

and calculate the vector of sufficient conditions for accomplishment of required reliability level:

$$Xs = (1, 4, 3, 3, 1, 1).$$

Algorithm B is a linear regular algorithm of the form

$$B = L1\ L2\ L3\ ,$$

where

$$L1 = [\ A0, A4\ ]\ ;$$
$$L2 = \alpha5\{\ \alpha1(\ A1\ V\ \alpha2(\ A2\ V\ \alpha3(\ A3\ V\ E\ )))\ \}\ ;$$
$$L3 = L5\ .$$

The sufficient conditions for algorithms L1, L2 and L3:

$$PL1\ (XL1) >= 0.9983\ ,\quad PL2\ (XL2) >= 0.9983\ ,\quad PL3\ (XL3) >= 0.9983\ .$$

Let us test the conditions:

$$PL1\ (0) = P0 \cdot P4 = 0.999 \cdot 0.999 = 0.998\ ;$$
$$PL2\ (0) = p5\ /\ (1 - (1 - p5)\ (\lambda1 \cdot P1 + \lambda2 \cdot P2 + \lambda3 \cdot P3 + \lambda4 \cdot 1)) =$$
$$= 0.1\ /\ (1 - 0.9(0.8 \cdot 0.9 + 0.03 \cdot 0.95 + 0.1698 \cdot 0.95 + 0.00017)) =$$
$$= 0.5525\ ;$$
$$PL3\ (0) = P5 = 0.999\ .$$

Thus, only the algorithm L3 = A5 meets the sufficient conditions, and because of this we can set X5 = 0 . Based on the known value of Xs, we select the initial values of the vector XL1, X0 = 1 and X4 = 1, i. e. XL1(0) = (1, 1). We solve the optimization problem for corresponding parallel algorithm by the coordinate hauling down technique starting from the sufficient value location

$$X0 = 0\quad \text{and}\quad X4 = 1\ ,\ \text{i. e.}\ \ X\ L1 = (\ 0, 1\ ).$$

Now, let us define the accomplished probability of correct execution of algorithm L1:

$$PL1\ (XL1) = P0\ (\ 1 - (\ 1 - P4)^{**}2 =$$
$$= 0.999\ (\ 1 - (\ 1 - 0.999)^{**}2\ ) = 0.99899$$

and refine the sufficient conditions for algorithm L2:

$$PL2\ (XL2) >= P0\ /\ (\ PL3\ (XL3) \cdot PL1\ (XL1) =$$
$$= 0.995\ /\ (\ 0.999 \cdot 0.99899\ ) = 0.997.$$

Algorithm L2 is iterative regular algorithm

$$L2 = \alpha 5\{\ C\ \},$$

where

$$C = \alpha 1(\ A1\ V\ \alpha 2(\ A2\ V\ \alpha 3(\ A3\ V\ E\ )))\ ,$$

And based on this, we can define the restriction for algorithm C:

$$PC\ (XC) >= (\ PL2\ (XL2) + q5 - 1\ )\ /\ (\ PL2\ (XL2) \cdot q5\ ) =$$
$$= (\ 0.997 + 0.9 - 1\ )\ /\ (\ 0.997 + 0.9\ ) = 0.99967\ .$$

After solution of optimization problem for the corresponding disjunctive regular algorithm, we obtain:

$$x1 = 3,\ \ x2 = 1,\ \ x3 = 2,\quad \text{i. e.}\quad XL2 = (\ 3, 1, 2\ ).$$

Thus, the necessary and sufficient conditions for optimal ensuring of the required reliability level of the algorithm B are:

$$x0 = 0,\ \ x1 = 3,\ \ x2 = 1,\ \ x3 = 2,\ \ x4 = 1\ \ \text{and}\ \ x5 = 0,$$
$$\text{i. e .}\ \ X = (\ 0, 3, 1, 2, 1, 0\ ).$$

After Reliability Synthesis, the control program takes the form:

$$D = (\alpha 0\ V\ \alpha 4)\{\ [\ A0,\ A4 \cdot \omega 4(\ E\ V\ A4\ )\ ] \cdot \alpha 5\{\ \alpha 1(\ A1 \cdot \omega 1(\ E\ V$$
$$A1 \cdot \omega 1(\ E\ V\ A1 \cdot \omega 1(\ E\ V\ A1\ ))))\ V\ \alpha 2(A2 \cdot \omega 2(\ E\ V\ A2\ )\ V$$
$$\alpha 3(A3 \cdot \omega 3(\ E\ V\ A3 \cdot \alpha 3(\ E\ V\ A3\ )))\ V\ E\ )))\ \}\ A5\ \}\ .$$

Appraising the average time of single execution of the control program before and after an optimization, we obtain correspondently:

$$tB\ (0) = 15.123\ \text{min.}\quad \text{and}\quad tB\ (X) = 16.002\ \text{min.}$$

Thus, we met required reliability level by an increase the average time of the algorithm execution less than 6 %.

The problems of optimization for business automation and data processing specified by the XAAL are being solved for a variety of criteria. The most frequently used parameters are a time of algorithm execution, probability of correct execution, complexity, and price. Any from these parameters (and also the number of functional operator types) may be selected as the goal function F, and other parameters must fit the restrictions. We developed different techniques for the Software Aspect Optimization problem based on the Branch and Bound, Convex-Programming and Dynamic Programming Methods.

5. CONCLUTIONS

On base of the Aspect-Oriented Approach to Software Engineering, we developed and implemented the common methodology and set of techniques for Optimal Enhancement of Software Reliability. The techniques use unified for all stages of Software Development Extended Algorithm Algebra Language, simple probability models of Reliability Analysis, formal methods of Reliability Synthesis and effective methods of Reliability Optimization. The next step of this R&D is Aspect-Oriented Software Development for the Adaptive External (Market Intelligence) and Internal (Conflict Resolution) Corporate Governance, for the Decision Support Systems of City Governments in emergency situation, for Reliability and Security Optimization.

6. REFERENCES

1. AATT TO24. Communications System Architecture Development. – AATT TO24 SAIC. – http://www.grc.nasa.gov/WWW/avsp/wxap2000/SAIC/sld001.htm
2. I. Alexander. Goal Patterns Generate Scenarios. Paper at RESG Scenarios Day, 1999. http://easyweb.easynet.co.uk/~iany/consultancy/goalpatt/goalpatt.htm

3. J. H. Andrews. Process-Algebraic Foundations of Aspect-Oriented Programming. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns (REFLECTION 2001), Kyoto, Japan (September 25-28, 2001). A. Yonezawa and S. Matsuoka (Eds.). LNCS 2192, Springer-Verlag, pp. 187—209, 2001.
4. B. Bachmendo and R. Unland. Aspect-Based Workflow Evolution. In Aspect-Oriented Programming and Separation of Concepts. Proceedings of the International Workshop, A. Rashid and L. Blair (Eds.), pp. 13—19, Lancaster University, UK, 24 August, 2001.
5. B. Bieker and E. Maehle. User-Transparent Checkpointing and Restart for Parallel Computers. In Foult-Tolerant Parallel and Distributed Systems, pp. 385—399, Boston: Kluwer Academic Publishers, 1998.
6. S. Clarke and R. J. Walker. Towards a Standard Design Language for AOSD. Aspect-Oriented Software Development. Proceedings of the 1st International Conference on Aspect-Oriented Software Development, pp. 113-119, Enschede, The Netherlands, 2002.
7. E. Daniel, R. Lal, and G. Choi. Warnings and Errors: A Measurement Study of a UNIX Server. The 29th International Symposium on Fault-Tolerant Computing. Madison, Wisconsin, USA, June 15-18, 1999.
8. V. Degtiar and I. Safonov. Evolutionary Mechanism of Conflict Resolution. New York, NY: Plenum Publishing Corporation. No. 2401-0114, 1988.
9. V. Degtiar and I. Safonov. An Evolution-Stable Conflict-Reducing Mechanism with Side Payments. New York, NY: Plenum Publishing Corporation. No. 2602-0297, 1990.
10. A. Frohlich and W. Schroder-Preikschat. High Performance Application-Oriented Operating Systems – the EPOS Approach. In Proceedings of the 11th Symposium on Computer Architecture and High Performance Computing, pp. 3—9, Natal, Brazil, September 1999. http://citeseer.nj.nec.com/augusto99high.html
11. V. Glushkov, A. Barabanov, L. Kalinichenko, S. Michnovskiy, and Z. Rabinovich. Computers with Developed Interpretation Systems. Kiev: Naukova Dumka, 1970.
12. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In ECOOP'97 – Object-Oriented Programming, 11th European Conference, LNCS 1241, pp. 220—242, 1997. http://citeseer.nj.nec.com/kiczales97aspectoriented.html
13. I. Kiselev. Aspect-Oriented Programming with AspectJ. Indianapolis, IN: SAMS, 2002.
14. S. W. Loke and A. Zaslavsky. Towards Distributed Workflow Enactment with Itineraries and Mobile Agent Management. E-Commerce Agents. J. Liu and Y. Ye (Eds.). LNAI 2033, pp. 283—294, Springer-Verlag, Berlin, Heidelberg, 2001.
15. J. S. Plank, Y. Chen, K. Li, M. Beck, and G. Kingsley. Memory Exclusion: Optimizing the Performance of Checkpointing Systems. Technical Report UT-CS-96-335, University of Tennessee, August 1996.
16. F. Quaglia, B. Ciciani, and R. Baldoni. A Checkpointing-Recovery Scheme for Domino-Free Distributed Systems. In Foult-Tolerant Parallel and Distributed Systems, pp. 93—107, Boston: Kluwer Academic Publishers, 1998.
17. A. Romanovsky. Exception Handling in Component-Based System Development. In the 25th International Computer Software and Application Conference (COMPSAC 2001), Illinois, USA, October, 2001.
18. I. Safonov. Design for Reliability of Control Algorithms. Vladivostok, USSR: VINITI, 1982.
19. I. Safonov. Trust Engineering and Risk Management of Complex Systems. Proceedings of the International Scientific School "Modelling and Analysis of Safety, Risk and Quality in Complex Systems", pp. 62—65. June 18-22, 2001, Saint-Petersburg, Russia, 2001.
20. J. Whaley. System Checkpointing Using Reflection and Program Analysis. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns (REFLECTION 2001), Kyoto, Japan (September 25-28, 2001). A. Yonezawa and S. Matsuoka (Eds.). LNCS 2192, Springer-Verlag, pp. 44—51, 2001.
21. I. S. Welch, R. J. Stroud, and A. Romanovsky. Aspects of Exceptions at the Meta-Level. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns (REFLECTION 2001), Kyoto, Japan (September 25-28, 2001). A. Yonezawa and S. Matsuoka (Eds.). LNCS 2192. Springer-Verlag, pp. 280-281, 2001.