Integrating Usage Stage Risk Measurement Tools in Distributed Applications

Cătălin Alexandru TĂNASIE
Academy of Economic Studies, Bucharest, Romania
catalin.tanasie@hotmail.com, catalin.tanasie@doesec.ase.ro

Distributed application features are presented along with associated risks occurring during the development and production stages. Issues raised by incidents catalogued as risks are described. The MERICS software application is presented, being built for user-distributed system interaction analysis and component reliability measurement. The steps taken to accurately describe risks from an end-user scope, based on a comparative system, are shown. The model is implemented using previously-described elements as parameters. The operational impact of analysis is detailed within successive versions of modules composing distributed applications. Risk management decisions are to be based on the analytic database built by applying the above.
Keywords**:** *Distributed IT Applications, Risks, Models, Release, Management*

1 Introduction

Distributed IT applications – DIA – define a collection of software modules separated based on function or location and who interact in order to provide an optimized solution to a request.

DIA components are characterized by:

- *diversity*, in the sense that module development technologies vary widely;
- *functional orientation*, each component having a pre-determined, well defined role in the system;
- *technical autonomy*, the property of a component to function even if other components it interacts with are not accessible;
- *logical autonomy*, derived as a notion from the technical one, but encompassing a module's ability to perform its individual tasks at maximum efficiency and without waiting for an external process-based input; when the latter is required, the interaction occurs asynchronously;
- *redundancy*, the presence of common functions in more than one component, used to improve system-wide performance or as backup in case an incident occurs.

The distributed software application *post-release* or *live* stage describes the interval starting with the first usage of the application in solving the tasks it was designed to perform and ending with the moment it is replaced. Between these two the application is constantly improved by applying patches designed to fix eventual errors in design or implementation and by extending its original functionality.

The dynamic manner of software development techniques, hardware and by extension products introduces the issue of obsolescence, which shortens an application's lifespan. As such, having inter-communicating components comes with the advantage of replacing old, poorly performing modules with new ones in a user-transparent manner.

IT post-release stage risks describe the probability of occurrence for an event which causes the partial or total loss of function for the application, as well as the probability of losing data integrity and implicitly lowering result quality.

Based on the DIA risk definition as concerns operational and analytical data, these incidents produce:

- risks concerning data quality or completeness in storage;
- user-performed analysis risks.

2 User-distributed system interaction

Through the duration of the *live* stage, the jobs that DIA perform are triggered directly or indirectly by user actions. *Users* are either

individuals, with varied degrees of technical knowledge, or identities associated to processes or components, that interact in performing a task whose output is given by previous operations or is based on them. Identifying target end-user groups is essential to DIA development, as their members participate in stages throughout the development of the solution.

Chronologically ordered based on a DIA's lifecycle, the activities involving user input are:

- *functional specification definition*, based on the consulting of the end-users of the application by analysts, concerning the identification of the processes that need to be implemented or optimized;
- *functional testing*, stage which precedes the application's release and in which user feedback is necessary in order to expose technical errors or functional ones appearing in the applied design and which are easier to discover using test scenarios; an important part of this stage is *acceptance testing*, testing the candidate version for release on a global scope;
- *the roll-out and live stages*, involving the gradual to full usage of the released application, raising risks linked to the application's activity domain, environment, user and inter-component authentication as part of obtaining the result set;
- *maintenance*, the continuous set of processes through which DIAs are kept in a optimally running state and incident effects are removed; it is considered *technical* or *functional* based on the role the user plays and his input;
- *extending DIA functional coverage*, through associating new modules, process that includes design, development and testing based partially on user input.

The users represent the main source for the input for the application's modules, in a direct or indirect manner, and their activities influence the application's operational state and the quality of stored information and offered results. *IT risks*, defined as probabilities of occurrence for events that cause losses through affecting the application's operating status and data quality, arise through this interaction.

In classifying DIA risks factors such as frequency, damage assessment through qualitative or quantitative costs and context of occurrence are taken into account [1].

In a distributed informatics system, the communication channels, component complexity, chosen architectural model, maintainability, module interdependency and user actions constitute factors of risk.

The actors involved in post-release DIA interactions generate *security risks* described by scenarios that include compromising authentication credentials through losing or accidentally or consciously divulging them towards unauthorized users. The potential damage is increased in situations where the communication channel is not under the control of the desired parties, as is the case with most geographically and architecturally separated applications. Privacy through usage of secure message protocols and techniques is not always an option, as it leads to increased complexity.

DIA post-release usage develops data quality risks through bad practices such as:

- deficient assignment of user roles associated to interacting with the application, resulting in allowing for initiating specialized tasks by users that do not poses required functional or technical knowledge, such as composing and generating irrelevant reports;
- removal or altering data which in turn acts as a basis for operations performed through automated processes in a transparent, asynchronous manner, not part of a business flow, as in the case of altering of operational information by the database administrator, event that will affect analysis;
- user impersonation, resulting in storing erroneous data or altering previously saved information; this scenario differs from security risks by the active role assumed by the attacker as concerning modifying information;

- process impersonation, affecting inter-component message content;
- inconsistent data acquisition due to insufficient validation or collecting irrelevant information;
- performing unwanted operations on logically associated information, deriving from design issues and insufficient testing, as is the case with deleting foreign key information or intersection table records.

The third risk category is due to external factors, as viewed from the DIA scope and consists of:

- vulnerabilities in application availability due to natural disasters, power surges or the temporary unavailability of key functional or technical personnel;
- loss of inter-module or user-to-application communication channels;
- economical context factors such as diminishing the maintenance and operational budget.

The *risk management* concept defines the resource allocation and package of measures destined to preventing incident occurrence and disaster recovery through minimizing downtime and damage control. It consists of:

- application startup procedures;
- building a backup framework including a minimum of functionality, usually started in a transparent manner related to the users, as well as designing and implementing a level of native redundancy such as doubling key functionality across multiple components;
- development of specialized components destined for transient information storage, such as active session data;
- complex logging of scheduled and triggered tasks aiming at rebuilding failed processes and providing error information;
- assuming incident-related costs as part of the business plan and assigning a risk-related budget either independent of or part of the maintenance budget;
- documenting incidents for the purpose of future avoidance or minimizing occurrence.

Risk management is an integral part of the overall project management and involves decisions based on target user groups input and client feedback in the context of a dynamic environment.

3 MERICS. A software application for data collection, quality assessment and risk analysis

In order to properly illustrate the post-release management and usage of a distributed application and to define the practical background for analysis processes the following section presents the MERICS (*Modele de Estimare a Riscurilor Informatice în Contextul Securizării* / *IT Risk Estimation Models in a Secured Environment*) software application.

MERICS components are destined to help acquire, transform and store information obtained through user input as well as operational processes in vision of future analysis, as well as to determine the efficiency of used algorithms and to minimize risks through the development of an evolutional model of their form based on evaluating their performance as compared to a set of results considered as an optimal form.

Conceptually, the MERICS application is composed of the following two parts:

- the operational package, reuniting components destined to help with image and video content loading, processing and comparison, marking the differences and storing the output information; the application's use cases are shown in Figure 1;
- the analytical package, grouping processes and technologies for operational algorithm analysis as well as evaluating the application's global performance under criteria such as reliability, scalability, processing speed and security.

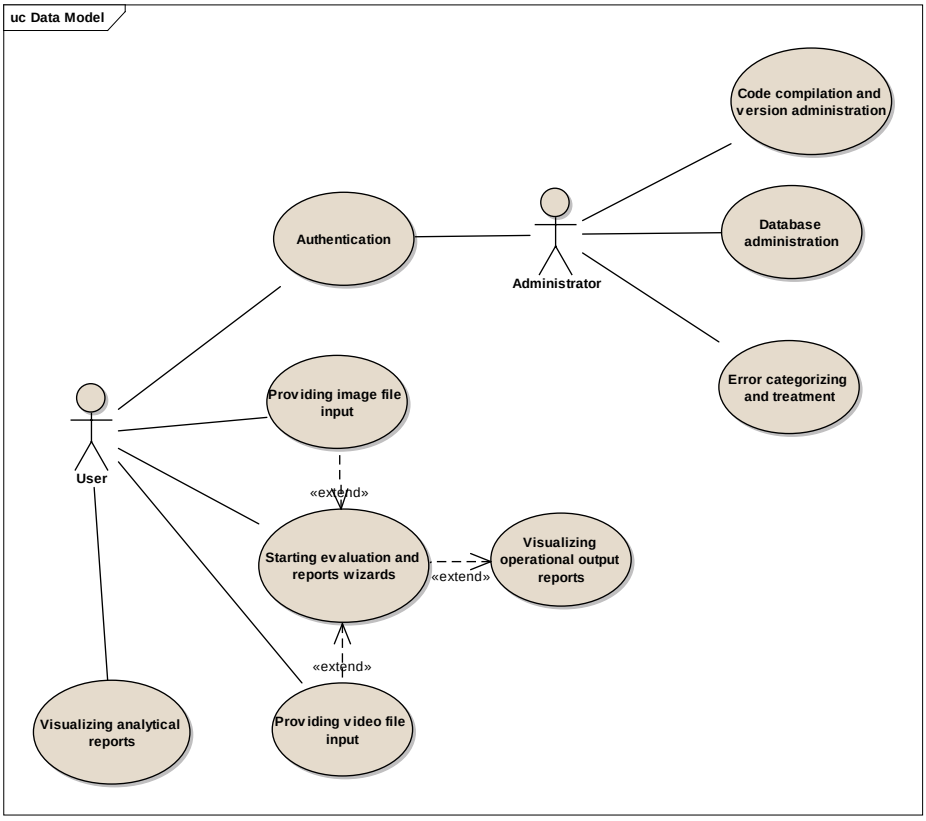

Fig. 1. *MERICS Use Cases*

Architecturally, the MERICS application is built around components as presented in Figure 2 and the following paragraphs, in a decreasing order as concerning the logical layer they belong to related to the interface, starting with the databases.

The databases are structured based on input provided by analyzing target group requirements, containing additional elements for storage of contextual information.

The operational database is designed to store primary information like pictures or video files, as well as meta-information concerning their disk storage location, loading date, the subject of the study and operational methods used, as well as acquired results. Attributes relating to the context of their acquirement are added, among them the session ID. This enables multiple instances of the same base content in order to serve as comparison or to provide backup.

The analytical database is aimed at helping with the extraction and formatting of information obtained through processing the data in its operational counterpart on analytical processing dimensions – method, stage, user, subject and file – and to generate data for building reports. This information constitutes input for the analytical methods, but the latter are stored in the operational database, directly or through referring the code block they belong to, as their usage is an operation allowed for distinct user roles.

MERICS.DataOperationsLayer represents the interface between the logical layer and the databases, serving as intermediary for insert, update and query operations performed on data, as permitted by the application's design. Programmatically, it includes two object classes designed to enable:

- mirroring the operational and analytical database structures by building around class-table, object-record and attribute-column associations;
- implementing the database interaction procedures and ensuring operation order, as in the case of inter-table, foreign key based interdependencies.

MERICS.Reports is composed of tools used for preparing reports by applying templates and filters on information in order to prepare

the content and format necessary for the presentation layer, either through interface-built controls or through generating report files.

MERICS.COMMON encompasses the collection of objects, properties and methods common to more than one component, such as reusable objects, enumerations, structures reunited in order to avoid circular dependencies and unwanted redundancy. An example of such an object is the *image* or *video content* itself, which is transferred throughout the application logical and persistence layers. These objects have no relevancy outside the application's components or at the database level and they are not standard development platform items as they contain additional information.

MERICS.OperationalLayer is the application's architectural nucleus, in the sense that it integrates functional methods, storage and data presentation operators, targeting both operational and analytical content and reports. Once a new method is developed, its logic is implemented at this level and a unique signature is associated in order to recognize the resulting output at a later stage. Additionally, this module contains error processing components. The reports and user-required analysis translates to supplementary methods, but no distinction is made concerning the analytical or operational scope at this level.

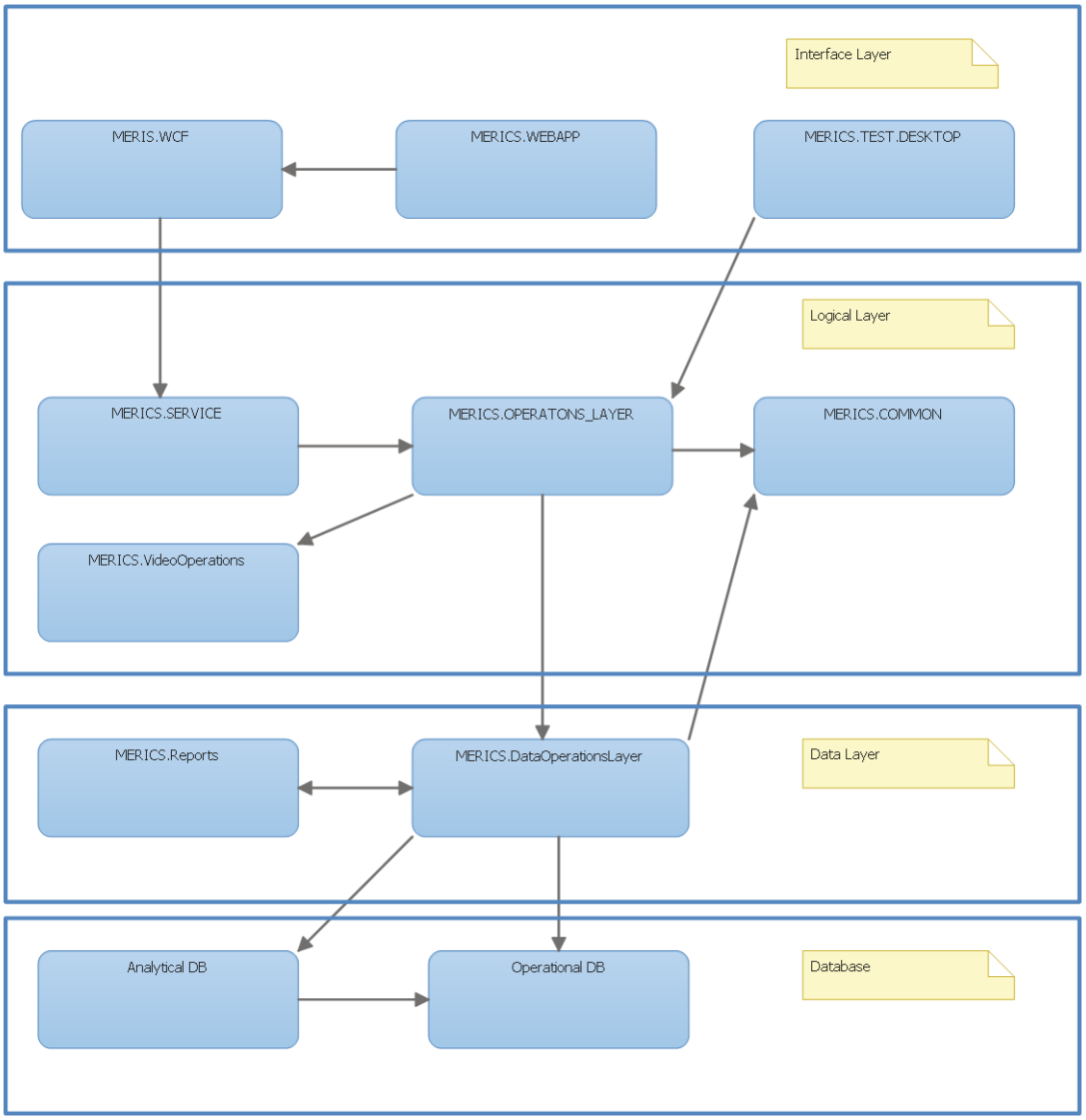

Fig. 2. *MERICS Components and Layers*

MERICS.VideoOperations targets video file processing. It is not a part of *MERICS.OperationalLayer* due to incompatible software technologies, an intermediary layer of encapsulation being required. The operations are partially based on functionality stored in libraries belonging to the *Windows Application Programming Interface – Windows API*.

MERICS.WEBAPP is a Web application performing the role of a client for the services that compose MERICS.WCF and presenting the user with authentication means and, based on role-granted access levels, with a means to perform operations on input data, either by measuring using varied operations, or by generating operational or analytical reports.

MERICS.TEST.DESKTOP is a module used as part of the development process, as a tool that allows for accessing the logical layer without consuming additional resources necessary for Web service and application enabling and without compromising data through its unsecured exposure to external environments. It is a form-based application used for testing components added as part of the logical and persistence layers, as well as the alterations, reports or database contents. The interface copies the Web version, except for authentications, which is integrated with the Operating System.

Currently the beta version of MERICS is undergoing testing as part of pre-deployment, as components are successively activated and functionalities extended.

4 Data quality risks

The term *operational method* is defined as the sum of procedures applied on a dataset as described by an application use case and resulting in a series of values interpreted in the given context. MERICS has operating methods reuniting a series of steps in image processing.

An analytical method represents an evaluation function associated to a process that occurs as part of the application's domain or to the latter's performance as opposed to a set of criteria that describe the optimum behavior. It can involve comparing different forms of the same algorithm, as available at different moments or as part of distinct application versions.

Let *O* be the set of operational methods associated to corresponding use cases, defined as:

$$O = \{O_1, O_2 \dots, O_k, \dots, O_n\},$$

where:

$n -$ number of operations applied on collected data, corresponding to functional use cases;

$O_i -$ operational method, part of set *O* at position *i*, $i=\overline{1,n}$.

Let T be the set defining the second dimension in operational analysis, grouping the different stages in the evolution of set *O* members or their components. Due to the time arrow associated to application version progress, the *m* components correspond to different dates, but a chronological order of these is no required:

$$T = \{T_1, T_2, \dots, T_l, \dots, T_m\},$$

where:

$m -$ the number of stages through which different operational methods pass through during their lifetime; component or application based;

$T_j -$ stage at position *j* in set *T*.

Applying operational method O_i at the timestamp or application version corresponding to stage T_j leads to result r_{ij}.

Consider the *R* matrix as being composed of items corresponding to results obtained through the execution of operational methods belonging to set *O*, measured according to stages as defined in set *T*,

$$R = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1j} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2j} & \dots & r_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{i1} & r_{i2} & \dots & r_{ij} & \dots & r_{im} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{n1} & r_{n2} & \dots & r_{nj} & \dots & r_{nm} \end{pmatrix} (1),$$

where:

$n -$ operations number,

$m -$ number of sets,

r_{ij} – numerical coefficient representing the output obtained by applying operation *i* at time or stage *j*, part of the *R* matrix; in case qualitative levels are associated to numerical values, the means of obtaining it remains the same.

Le matrix $\bar{R}$ be composed of results considered to represent the optimum,

$$\bar{R} = \begin{pmatrix} \overline{r_{11}} & \overline{r_{12}} & \dots & \overline{r_{1j}} & \dots & \overline{r_{1m}} \\ \overline{r_{21}} & \overline{r_{22}} & \dots & \overline{r_{2j}} & \dots & \overline{r_{2m}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \overline{r_{i1}} & \overline{r_{i2}} & \dots & \overline{r_{ij}} & \dots & \overline{r_{im}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \overline{r_{n1}} & \overline{r_{n2}} & \dots & \overline{r_{nj}} & \dots & \overline{r_{nm}} \end{pmatrix} (3),$$

$$i = \overline{1,n}, j = \overline{1,m},$$

where:

n – operations number,

m – number of sets,

$\bar{r}_{ij}$ – numerical coefficient representing the *optimum* output obtained by applying operation *i* at time or stage *j*, part of the $\bar{R}$ matrix; in case qualitative levels are associated to numerical values, the means of obtaining it remains the same. It is calculated based on predetermined known information, such as specifications, industry standards or context-enabled features – in MERICS' case, identical images are compared.

If the result of applying a given method is not quantifiable or impossible to evaluate due to insufficient data, a value equal to the average based on the method or stage is associated to the corresponding item in the matrix. For a set composed of 3 methods, O_a, O_b, O_c, that in forms T_u, T_v, T_w – $a, b, c = \overline{1,n}$, $u, v, w = \overline{1,m}$ – does not have a measurable output, for *n* operations and *m* stages we obtain:

$$r_{ij} = \frac{\frac{\sum^{n}_{q=1,\ q\neq a,b,c} r_{qj}}{n - nl_j} + \frac{\sum^{m}_{z=1,\ z\neq u,v,w} r_{iz}}{m - nc_i}}{2} (3),$$

where:

a, b, c – identifiers matched to unquantifiable operations on the chosen value scale;

u, v, w – identifiers associated to stages in which no r for operations O_a, O_b, O_c can be determined;

r_{qj} – numerical coefficient representing the output obtained by applying operation *q* at time or stage *j*, part of the *R* matrix;

r_{iz} – numerical coefficient representing the output obtained by applying operation *i* at time or stage *z*, part of the *R* matrix;

nl_j – number of elements with unquantifiable values on the current j column in matrix *R*;

nc_i – number of elements with unquantifiable values on the current i line in matrix *R*.

Analytical methods, grouped is set A, are essentially a series of operations applied to set *R* in order to minimize differences between obtained results and optimum corresponding values, through selecting the methods considered efficient by comparing their output to set $\bar{R}$ items and revising the ones that underperform based on the same criteria.

$A_o(i)$, the aggregated index of operational evolution for method *i*, is calculated as:

$$A_o(i) = f(r_{i.}, T) = \sum_{j=1}^{m} f(r_{ij}, T_j), \forall\, i = \overline{1,n},$$

$$r_{i.} = \{r_{ij} |\, j = \overline{1,m}\}, R = \{r_i |\, i = \overline{1,n}\},$$

where:

$f()$ – operational method analysis function;

r_{ij} – numerical coefficient representing the output obtained by applying operation *i* at time or stage *j*, part of the *R* matrix;

$r_{i.}$ – the collection of results obtained by applying operation *i* through all applicable stages;

T_j – stage at position *j* in set *T*;

m – number of stages.

$A_s(j)$, the index of application evolution for stage *j*, is calculated as:

$$A_s(j) = g(r_i, O) = \sum_{i=\overline{1,m}}^{n} g(r_{ij}, O_i), \forall\, j = \overline{1,m},$$
$$r_{.j} = \{r_{ij} \mid i = \overline{1,n}\}, R = \{r_j \mid j = \overline{1,m}\},$$

where:

$g()$ – stage analysis function;

r_{ij} – numerical coefficient representing the output obtained by applying operation *i* at time or stage *j*, part of the *R* matrix;

$r_{.j}$ – the collection of results obtained by applying all operations in the version corresponding to stage *j*;

O_i – operational method at position *i* in set *O;*

n – number of operations.

Table 1 reflects the relationship between operations and application stages, as well as the form of associated analytical forms. These constitute input to global evolution indicators, marked as for A_o operations and A_s for stages.

Table 1. *Operation-stage comparison*

E	T_1	T_2	…	T_l	…	T_m	***Total ops***
O_1	$f(r_{11}, T_1)$ $/g(r_{11}, O_1)$	$f(r_{12}, T_2)$ $/g(r_{12}, O_1)$	…	$f(r_{1l}, T_l)$ $/g(r_{1l}, O_1)$	…	$f(r_{1m}, T_m)$ $/g(r_{1m}, O_1)$	$A_o(1)$
O_2	$f(r_{21}, T_1)$ $/g(r_{21}, O_2)$	$f(r_{22}, T_2)$ $/g(r_{22}, O_2)$	…	$f(r_{2l}, T_l)$ $/g(r_{2l}, O_2)$	…	$f(r_{2m}, T_m)$ $/g(r_{2m}, O_2)$	$A_o(2)$
⋮	⋮	⋮		⋮		⋮	⋮
O_k	$f(r_{k1}, T_1)$ $/g(r_{k1}, O_k)$	$f(r_{k2}, T_2)$ $/g(r_{k2}, O_k)$	…	$f(r_{kl}, T_l)$ $/g(r_{kl}, O_k)$	…	$f(r_{km}, T_m)$ $/g(r_{km}, O_k)$	$A_o(k)$
⋮	⋮	⋮		⋮		⋮	⋮
O_n	$f(r_{n1}, T_1)$ $/g(r_{n1}, O_n)$	$f(r_{n2}, T_2)$ $/g(r_{n2}, O_n)$	…	$f(r_{nl}, T_l)$ $/g(r_{nl}, O_n)$	…	$f(r_{nm}, T_m)$ $/g(r_{nm}, O_n)$	$A_o(n)$
Total stages	$A_s(1)$	$A_s(2)$	…	$A_s(l)$	…	$A_s(m)$	A_o/A_s

The MERICS modules implement the before mentioned metric as part of the logical layer. The analytical methods and their output are stored in the analytical database and processed for reports.

5 An evolutional model for method optimization

During a DIA's post-release lifespan, the user encounters the situation of picking the right method in the analysis of its input. Most of the time, he is unaware of implementation details or is lacking the technical knowledge needed to understand the algorithms, as they are not a part of his interest area. In case the form of the methods uses is modified, usually in a user-transparent, no interface altering mode, the user will lack the ability or awareness to distinguish between the results obtained. The variation in results will be due to changes in input values, evaluation algorithms or a combination of both. Yet

integrating a set of bounded functional or analytical activities and reports in an IT system leads to logical dependencies in the input information format and data relevancy.
MERICS deals with this issue by establishing an application-wide system of measurement that relies on performance estimators for algorithms that are designed to process images and video content across different formats and with varying contextual accuracy. A histogram is performing well in evaluating similar format and resolution content such as video frames, but less relevant or in need of improvement when dealing with images that undergo rotation operations or color reversal.
MERICS analytical algorithms performing tasks like error count, operational algorithm efficiency measurement, processing speed and cross-stage evaluation describe in an exact manner a fundamentally inexact chain of events.
Lacking the ability to provide, in a classical sense, proofs for the predictions obtained as result of applying risk assessment algorithms leads to the development of an evolutional algorithm for optimizing the DIA's methods by using techniques specific to genetic algorithms applied to operations as defined in chapter 4.
Algorithms, defined [1] as *heuristic methods for selection and optimization that imitate processes associated to natural selection*, introduce the concepts of:

- *representation*, usually as an array of binary values. MERICS uses this element at operation or module level and in various stages. The correspondence is not the same as in the genetic representation, which usually groups randomly generated collections of tens or hundreds of thousands of elements;
- the *fitness function* used for decision-making;
- the *fitness landscape*, concept that groups efficiency evaluators in the process of analyzing the global application performance in treating vulnerabilities that affect the quality of data or the system's availability;
- the *crossover* and *mutation* functions, represented in MERICS through algorithms highlight the performance of operational methods in processing compatible input and identifying new forms for methods as stages advance.

Consider set *I* of consisting of *z* input items serving as parameters for the *O* set of operations presented in chapter 4:

$$I = \{i_1, i_2 \dots, i_t, \dots, i_z\},$$
$$op: I \to R_{m.},$$

where:
$op() -$ operational input processing function part of set *O*;
$i_1 .. i_z -$ operational input;
$R_{m.} -$ the array formed by assembling the results of the operational method *op()* along the m stages in which it is used; part of matrix *R*.

The binary arrays composing the representation are determined through evaluating information *i* through method *op()*. The fidelity of the model relates to describing the operational steps as detailed as possible. The design stage involves describing DIA dynamics through UML diagrams or similar tools.
Figure 3 describes the steps performed in calculating the similarity degree between two Bitmap images, by way of splitting them in pre-determined size cells and calculating the total values of color components on the RGBA (Red, Green, Blue, Alpha) scale, in which the last component indicates the transparency degree. There occurs an observable dependency on previously-computed values, as is the case with calculating averages for each component, needing the total cell average, the latter in turn depending on identifying cell number.
The *CalculateFramePixelComponentSimilarityUsingHistogramsV2()* method is itself an optimized form of the same algorithm applied on the RGB color representation scale with the purpose of comparing images.
The algorithm's steps are identified as:

- the setting or calculating of cell matrix properties;

- computing the total value of numerical indices associated to each cell's composing pixels;
- calculating the total cell-to-cell differences between the video frames or pictures;
- determining the global similarity and the angle at which rotating one of the input items obtains a maximum of resemblance to the other, 0º if the feature is disabled.

Consider the (0,0,0,0) vector representing the algorithm's 4 component steps as described in the paragraph above and in their initial form.

The RGB to RGBA scale transition consists in altering the way in which the totals are calculated. Cell totals and pixel differences are not affected, as the manner in which these color component-based values are determined remains unchanged.

Consequently, the second generation array for describing the algorithm's steps is (0,0,0,1), the value 1 indicating that a change has occurred in the last step.

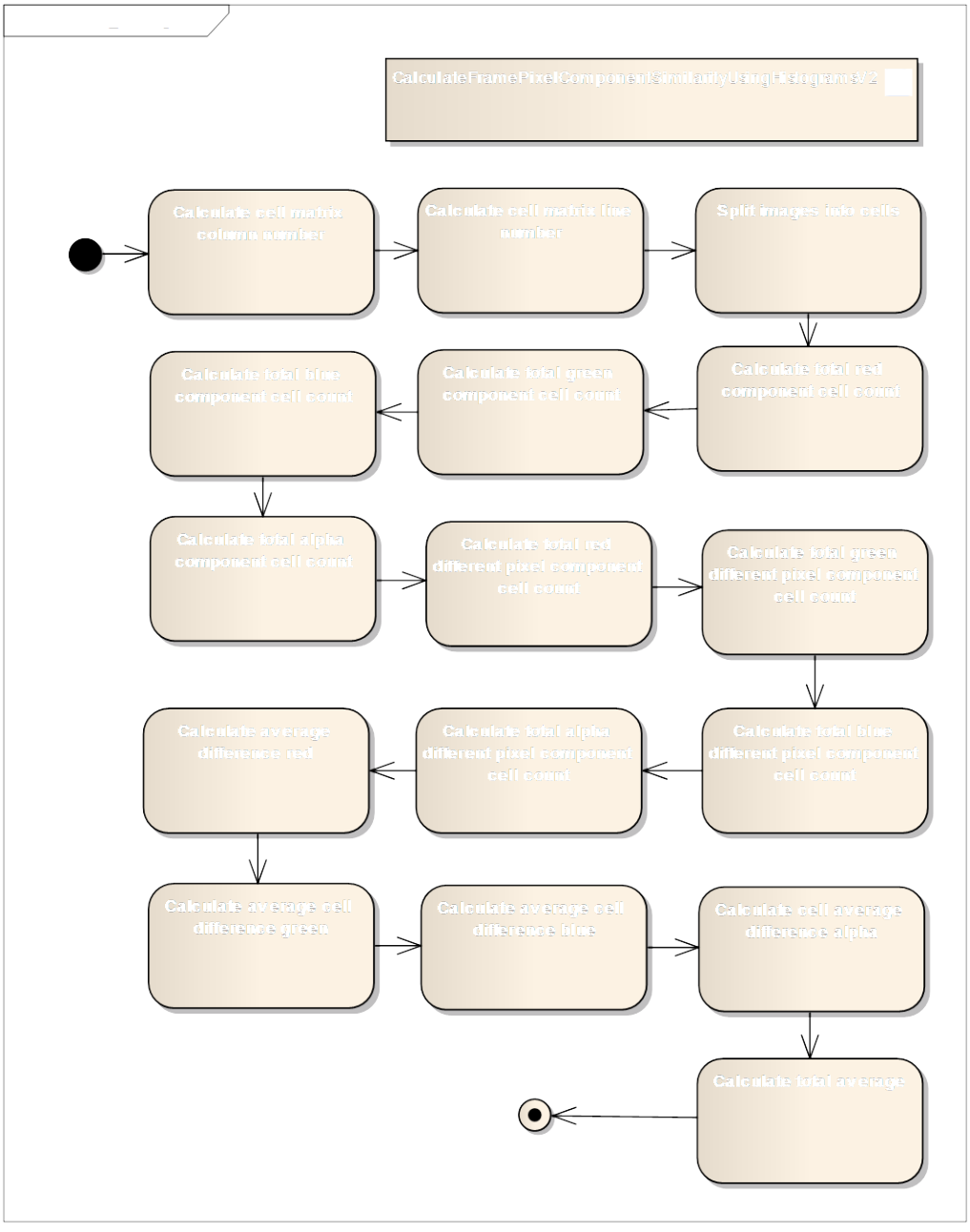

Fig. 3. CalculateFramePixelComponentSimilarityUsingHistogramsV2() steps

The fitness function role for these operations is played by use of *f()* and *g()* functions as described chapter 4's model, based on the analysis of the result's fidelity in providing relevant data – determined through comparing successive stages for improvements or the global impact of the alteration. The better performance of the histogram algorithm by switching to the RGBA scale is determined through this criterion.

The next stage for the analytical MERICS module consists of determining the impact that incidents and security breaches have in the variation of these values. Based on image sets the vulnerabilities in designing the algorithms are identified, as well as the zones that must be refined through successive modifications while keeping performing features.

Steps in method evolution appear as follows:

Step 1. Form 1. (0,0,0,0)

Step 2. Form 1. (1,0,0,0) and determined to be inferior to 1.

Step 3. Form 1. (0,1,0,0) is superior to 1 ➔ this form is now described as the reference (0,0,0,0)

Step 4. Form 2. (1,0,0,0) superior to 1 ➔ this form is now described as the reference (0,0,0,0)

Step 5. Form 3.

If global application performance is impacted at a random step *k*, the global estimators are computed considering the current method at its k form and the rest at their previous, *k-1*, forms.

6 Conclusions

Risk evaluation as applied to data quality and security creates a framework enabling decisions related to risk management, associated to DIAs post-release stages. Based on analytical algorithms, incidents are identified and measures taken ensuring the continual improvement of the operational methods.

Losses caused by the occurrence of events negatively impact application performance or informational content, as well as influence the data collection and analysis. The cost is estimated by the user or owner of the application based on associating value to functionalities [2].The valuation models include this subjective factor in their algorithms, the budget or activity domain leading to different weights for similar incidents and subsequently to the refusal or acceptance of a change in the operations.

An online vendor specializing in digital content may find that requesting a client's physical address in the checkout forms lowers overall sales due to customer resilience in providing such information, as opposed to goods that require shipping, even if the purpose is clearly stated – an online survey, for instance. This leads to users associating different effects for the same procedure.

MERICS associates operational and analytical methods aiming at constantly improving their behavior. The latter's dependency on data quality and availability, as well as including meta-information tools such as logs and error treatment in the application's source code or environment leads to establishing a mechanism for refining the algorithms and optimizing the architectural layout.

DIAs features such as component reliance in task solving allow for processing optimization and increase reliability through enabling flexible security levels, with enhancements in work speed and inter-component communication, as well as proper user identification and message contents encoding.

Method stages help in describing and comparing operational functions through different application versions.

The model allows for abstracting method components, process used in developing automatic analytical instruments.

Acknowledgements

The current research effort is augmented by the help of the *DoEsEc* project, financed through the European Social Fund's Human Resource Development program. Parts of the current article were presented in Bucharest University of Economic Studies' *11th*

International Conference on Informatics in Economy in May 2012 [11].

References

[1] W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via http://lulu.com, http://www.gp-field-guide.org.uk, 2008. (With contributions by J. R. Koza), 2008, 250 pp.

[2] C. A. Tănasie, S. Vînturis, A. Grigorovici, *Reliability in Distributed Software Applications*, Informatica Economica, Volume 15, No. 4.

[3] S. Hoermann, M. Aust, M. Schermann, H. Krcmar, *Comparing Risks in Individual Software Development and Standard Software Implementation Projects: A Delphi Study*, 45th Hawaii International Conference on System Science (HICSS), 4-7 January 2012.

[4] M.P. Lima, J.M.N. David and B.T. Dantas, *Risk Management and Context in a Collaborative Project Management Environment for Software Development*, Brazilian Symposium of Collaborative Systems - Simposio Brasileiro de Sistemas Colaborativos (SBSC), 5-8 October 2010, pp. 95 - 102

[5] B. Boehm and J. Bhuta, *Balancing Opportunities and Risks in Component-Based Software Development*, IEEE Software, Volume 25, Issue 6, December 2008, pp. 56 – 63.

[6] Y. Wang, X. Tuo and T. Zhao, *A concrete model of software risk development*, 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), 9-11 July 2010, pp. 472 – 474.

[7] J.A. Ibrahim, M. Majid, A.H. Hashim and R.M. Tahar, *Risk Quantification in Coal Procurement for Power Generation: The Development of Supply Shortage Impact Matrix*, Second International Conference on Computational Intelligence, Modeling and Simulation (CIMSiM), 28-30 September 2010, pp. 401 – 406.

[8] M.Q. Saleem, J. Jaafar and M.F. Hassan, *Model driven security frameworks for addressing security problems of Service Oriented Architecture*, 2010 International Symposium in Information Technology (ITSim), 15-17 June 2010, pp. 1341 – 1346.

[9] J. Pearl, *Causality: Models, Reasoning and Inference*, 2nd Edition, Cambridge University Press, 2009, 484 pp.

[10] M. Sadiq, M.K.I. Rahmani, M.W. Ahmad, S. Jung, *Software risk assessment and evaluation process (SRAEP) using model based approach, Networking and Information Technology (ICNIT)*, 2010 International Conference, 11-12 June 2010, pp. 171 – 177.

[11] *Post-release distributed software applications risk management*, Proceedings of the Eleventh International Conference on Informatics in Economy IE 2012, 10-11 May 2012, pg. 36 – 40.

Catalin Alexandru TĂNASIE, born at 18.08.1984 in Pitesti, Arges, is a graduate of the I. C. Bratianu National College and of the Faculty of Cybernetics, Statistics and Economic Informatics within the Bucharest University of Economic Studies, the Economic Informatics specialization, 2007 promotion. Starting 2007 he attended the Informatics Security Master in the same institution, and is currently a PhD student at the Doctoral School within the Bucharest University of Economic Studies. He has concerns in the field of distributed applications programming, evolutionary algorithms development, part of the field of artificial intelligence - neural and genetic programming. Currently he works as an application designer in a financial institution. He is involved in creating commercial applications using development platforms belonging to leaders in the field, companies including Microsoft, Oracle and IBM.