# Distributed Application Extensions and Obsolescence as Risk Generating Factors

Cătălin Alexandru TĂNASIE
Bucharest University of Economic Studies, Doctoral School
catalin.tanasie@hotmail.com, catalin.tanasie@doesec.ase.ro

*Extending distributed IT&C applications raises risks associated to integrating new modules in already existing functional areas, influencing performance and data relevancy. The current paper details on the actors and operational, as well as analytical areas of impact. The software application used to test and refine assessment models, MERICS, is presented alongside obsolescence-related factors. A model on evaluating moral decay is detailed and conclusions are drawn relating to the effects of usage-stage distributed application-related activities.*
***Keywords:*** *Distributed Applications, Extensions, Obsolescence, MERICS, Risks*

## 1 Risks Identified in DIA Extensions

Distributed applications are built and used according to functional and technical specifications that reflect the expectations and requirements of an organization at the moment of the development decision. Company activity changes or extends to cover new, related areas of interest, and supporting applications need to be improved in order to address these changes. The intended lifespan of the application influences the number and extent of added features and components. In core banking applications, where the extent of usage reaches time spans in excess of 10 or 15 years due to the cost of replacing the components, along with dependencies in higher architectural level components, over 90% of the system's life spans after the initial release, with continuous updates on the structure and logic. New DIA modules, the alteration or addition of methods and parameters, changes in user management and security requirements developed after the initial live release of the applications are factors contributing to DIA extensions development, as the users, administrators, executives and various other organization actors discover issues with the existing form. Identifying DIA extension areas depends on:

- the *development and usage plan* relating to the format of the initial release; large applications are often split into stages, with areas of usage prioritized and implemented in a specific order; starting with the second stage, the developers and users include factors relating to the effects on the existing version, as testing is done on the whole and development potentially affects performance and business processes in already implemented components; MERICS, the model testing and refining application associated to the current research, is released in 3 stages, with successive completion and usage for operational, web-based interfaces and communication, as well as analytical modules;
- the *improvement or extension cause* affects the actors and format of the new release – functional or logical error removal, the adding of new functionalities, updating software technologies; the timing and budget allocated for the improvement differs accordingly – bug fixing is time-sensitive and costs incurred on the system budgeted as risks in the management's view, while new methods and components are predicted and provided for in a controlled environment.

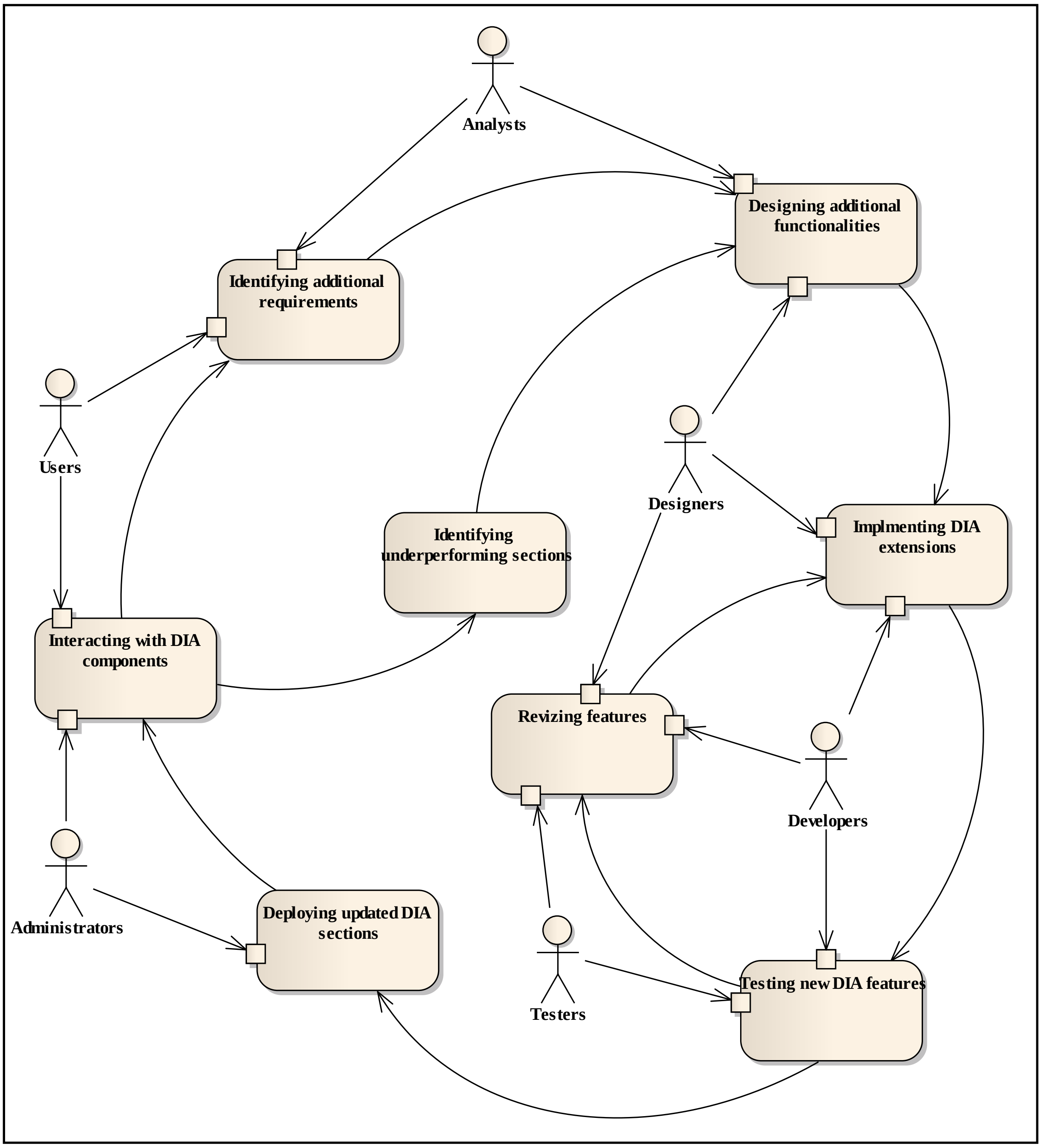


**Fig. 1.** Extending DIA components cycle

Figure 1 details on the actors and associated actions in extending DIAs. The graph's arches, linking action nodes, identify the order of the events. In addition, party interactions identify areas where *users*, *administrators*, *analysts*, *designers*, *developers* and *testers* provide their *input* in designing, implementing, validating and deploying new application features.

The triggering stage in the cycle of DIA extension consists of the *interaction* between operational users, system administrators and components or interfaces. Malfunctioning sections, *underperforming* graphical or abstract structures lacking in usability or performance are identified through repetitive accessing. Alternatively, new activities or changes in component requirements or external interactions – the expected unavailability of an information source, the changes made in Web services – trigger the updating and extension of DIA modules. *Analysts* and *designers* construct the *requirements and structure* of the added parts, leading to their *implementation* and incipient testing by internal or external teams. As *new components* affect the entirety of the system by interacting and sharing the same resources, testing is done by *revising* all previously built modules. Error detection and improperly constructed structures trigger the repetition of development and testing until a form considered appropriate for *deployment* is available. The users interacting with the application provide feedback over time and the cycle repeats itself.

Estimating costs is relevant to prioritizing extensions and determining the amount of resources – personnel, technological costs and licenses, development and testing hardware and application downtime due to reconfigurations and deployment – and requires the considering of factors brought to a common denominator – *sales hours* or *added value hours* for MERICS. In assessing costs to the enhancement of MERICS components, time was chosen as the unifying measurement criteria for the analyzed feature.

Let $i$ be the component, method or application usage parameter whose enhancement or fixing is considered, $F_i(h)$ the associated positive functionality impact and $L_i(h)$ the costs induced by the application dysfunction in its current form. The enhancement $E_i(h)$ is effective if the former are higher over the measured $h$ interval, or

$$E_i(h) = F_i(h) - L_i(h) > 0.$$

In order to properly address time valuation, the costs of the actions and incidents are represented based on the hourly income as follows:

$$c_i(h) = l_i,$$
$$l_i = k * s_i,$$
$$s_i = \frac{QP(i)}{H},$$

where:

- $s_i$ – *hourly* sales or added value as related to enhancement $i$;
- $l_i$ – costs associated to incident or extension $i$, as payments projected to be made by the organization to internal and external actors;
- $k$ – number of hourly sales, as loss representation;
- $QP(i)$ – Production function over the measured interval, either in monetary units or added value, as relevant to enhancement $i$;
- $H$ – total number of hours in the measured interval.

The individual gains, $F_i(h)$, losses $L_i(h)$ and overall effect $E_i(h)$ of enhancement $i$, decomposed in $m$ constituting factors, are determined by the equations

$$L_i(h) = \sum_{a=1}^{t} c_{aj}(h) = \sum_{a=1}^{t} l_{ij} = \sum_{a=1}^{t} k_j * s_{ia},$$

$$F_i(h) = h_i * \sum_{j=1}^{m} s_{ij} = \sum_{j=1}^{m} s_{ij} * h_{ij},$$

$$E_i(h) = F_i(h) - L_i(h) = \sum_{j=1}^{m} s_{ij} * h_j - \sum_{a=1}^{t} k_{ia} * s_{ia},$$

where:

- $m$ – number of factors contributing to the measurement of enhancement $i$ ***gain*** effects;
- $t$ – number of factors contributing to the measurement of enhancement $i$ ***loss*** effects;
- $h_{ij}$ – number of hours gained by the implementation of extension $i$ determined by factor $j$;

$k_{ia}$ – number of hourly sales, as loss representation for extension $i$ due to factor $a$;
$QP$ – production over the measured interval, either in monetary units or added value;
$H$ – total number of hours in the measured interval.

Consequently, the total net gain $E(h)$, total added value of the extensions $F(h)$, as well as subsequent total costs $L(h)$, are measured by summing up the individual enhancements. For $n$ updated items, $m$ positive and $t$ negative effects, the specified indicators result from applying to the formulas

$$F(h) = \sum_{i=1}^{n} F_i(h) = \sum_{i=1}^{n}\sum_{j=1}^{m} s_{ij} * h_{ij},$$

$$L(h) = \sum_{i=1}^{n} L_i(h) = \sum_{i=1}^{n}\sum_{a=1}^{t} k_j * s_{ia},$$

and

$$E(h) = F(h) - L(h) = \sum_{i=1}^{n}\sum_{j=1}^{m} s_{ij} * h_{ij} - \sum_{i=1}^{n}\sum_{a=1}^{t} k_j * s_{ia}.$$

The following considerations constitute factors in the valuation of the $F$ and $L$ indicators:

- *upgrade effect on application productivity*, $\boldsymbol{w(h)}$, part of $F$, measured in the user and process time gains derived from the increase in speed and optimization of interactions, as well as the removal of unwanted glitches or dysfunctional algorithms;
- *reduction of risk incidence*, $\boldsymbol{r(h)}$, part of measured over the analyzed period as a product of losses and frequency;
- *development and testing time*, $\boldsymbol{d(h)}$ part of $L$, measured in hours added for all members of the involved teams, or derived from the costs in externalizing the service, measured in hours using the $\boldsymbol{c_i}$ function;
- *negative effects on security*, $\boldsymbol{es(h)}$**,** part of $L$, including damage repair time or costs, as well as the losses caused by the application downtime;
- *hardware and software costs* ***hs(h)***, part of $L$, measured either by replacing damaged units or affected source code, either by the upgrading and extending of existing ones – buying licenses for new software frameworks, adding processing and storage capacity.

The factors contributing to the building of the model are extendable in number with no negative impact on the relevancy of the estimator, as long as the cost correlation is maintained.

The model presented in the previous section is applied to 4 characteristics of the algorithms and components in MERICS image processing tasks – *adding a histogram-based method for static frames comparison* (1), *implementing PKI-based security* in Web services effects on image processing (2), and *completing the MERICS.COMMON module* (3). As MERICS is developed by the author, no commercial value in sales is measured. $s_i$ is valued at **6,25** units as a measure of an average loading of 2000 images and derived video frames per month over a period of 60 days, considering an 8-hour weekday interval: $QP = 2000, H = 40 * 8 = 320$. *Table 1* details on the results obtained in measuring the indicators, with the affected MERICS component shown.

**Table 1.** MERICS impact assessment for the three considerations

| No | Affected component | $w_i(h)$ | $r_i(h)$ | $d(h)$ | $es(h)$ | hs(h) | $F_i(h)$ | $L_i(h)$ | $E_i(h)$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | *MERICS.OPERATIONAL* | *112,5* | *31,25* | *24* | *0* | *50* | *143,75* | *74* | *69,75* |
| 2 | *MERICS.WCF*<br>*MERICS.SERVICE*<br>*MERICS.OPERATIONAL*<br>*MERICS.AUTH.* | *87,5* | *268,75* | *187,5* | *0* | *106,25* | *356,25* | *293,75* | *62,5* |
| 3 | *MERICS.WCF*<br>*MERICS.SERVICE*<br>*MERICS.OPERATIONAL*<br>*MERICS.AUTH.*<br>*MERICS.WEBAPP*<br>*MERICS.COMMON* | *356,25* | *62,5* | *500* | *18,75* | *137,5* | *418,75* | *626,25* | *-237,5* |
| | ***TOTAL*** | ***556,25*** | ***362,5*** | ***711,5*** | ***18,75*** | ***293,75*** | ***918,75*** | ***994*** | ***-75,25*** |

Consequently, for the $m$ = 2 positive and $t = 3$ negative factors mentioned as factors, as well as $n = 3$ enhancements detailed, the generic formulas for $F$ and $L$ become

$$L_i(h) = d_i(h) + es_i(h) + hs_i(h),$$
$$F(h) = \sum_{i=1}^{3} F_i(h),$$
$$F_i(h) = w_i(h) + r_i(h),$$
$$L(h) = \sum_{i=1}^{3} F_i(h),$$
$$E_i(h) = F_i(h) - L_i(h) = w_i(h) + r_i(h) - d_i(h) - es_i(h) - hs_i(h),$$
$$E(h) = F(h) - L(h) =$$
$$= \sum_{i=1}^{3} F_i(h) - \sum_{i=1}^{3} F_i(h) = \sum_{i=1}^{3} \big(w_i(h) + r_i(h) - d_i(h) - es_i(h) - hs_i(h)\big).$$

In addition, factors are quantifiable as per total of affected enhancements, with $W(h)$, $R(h)$, $D(h)$, $ES(h)$, $HS(h)$ calculated as follows – generic and three item MERICS form (shown in *table 1*):

$$W(h) = \sum_{i=1}^{n} w_i(h)$$
$$R(h) = \sum_{i=1}^{n} r_i(h),$$
$$D(h) = \sum_{i=1}^{n} d_i(h),$$
$$ES(h) = \sum_{i=1}^{n} es_i(h),$$
$$HS(h) = \sum_{i=1}^{n} hs_i(h).$$

For $n$ = 3, the values for the indicators above as determined as

$$W(h) = w_1(h) + w_2(h) + w_3(h)$$
$$R(h) = r_1(h) + r_2(h) + r_3(h),$$
$$D(h) = d_1(h) + d_2(h) + d_3(h),$$
$$ES(h) = es_1(h) + es_2(h) + es_3(h),$$
$$HS(h) = hs_1(h) + hs_2(h) + hs_3(h).$$

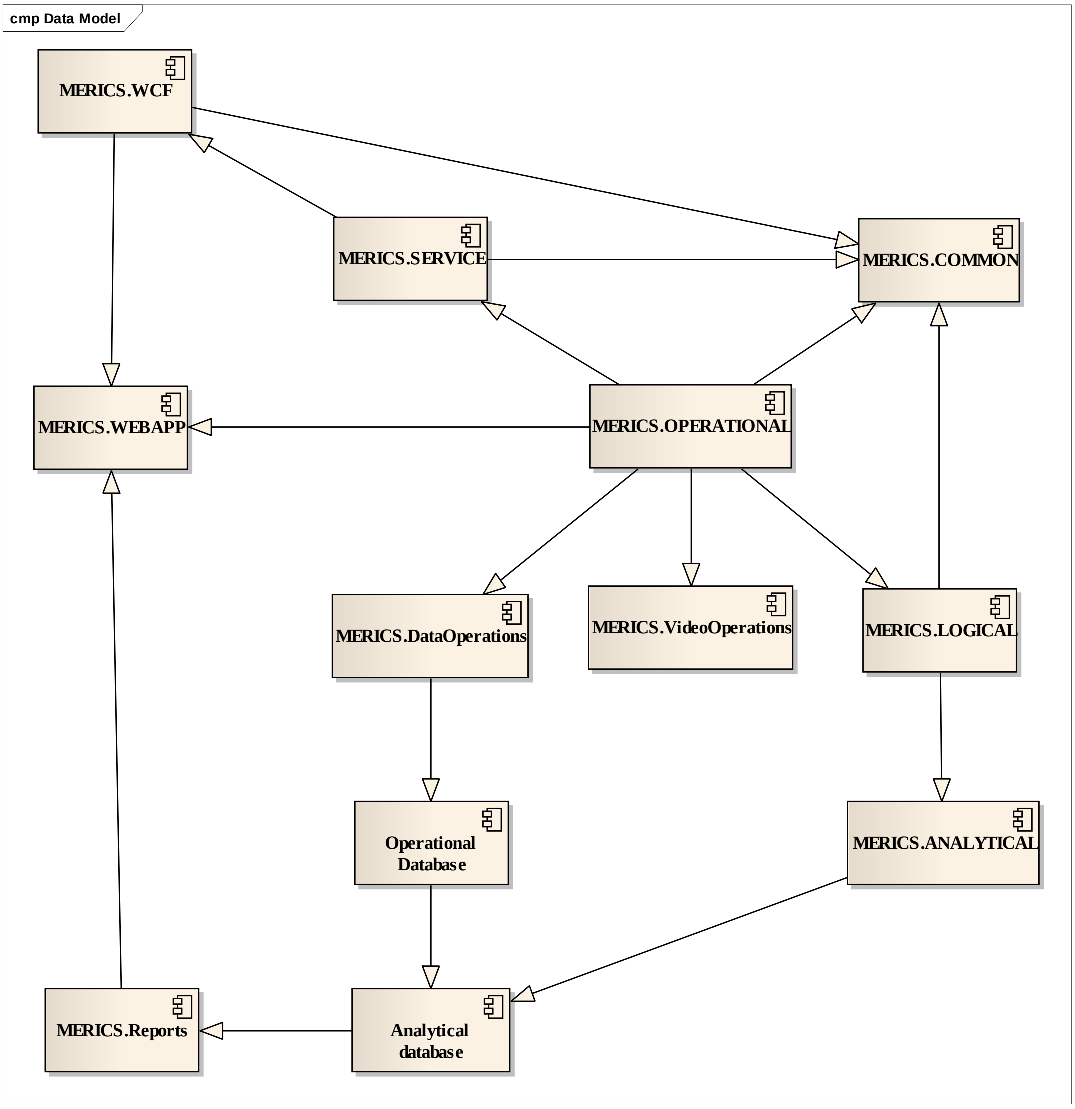


**Fig. 2.** MERICS components testing dependency, incident-originating component representation

Figure 2 identifies testing steps in the evaluation of changes made as part of the MERICS.OPERATIONAL module, with the affected neighbors in the architectural tree of the application shown along with their own influence on others forming second degree dependencies to the subject of the measuring. The impact of additional workload is calculated using the $c$ function if necessary and included in the $L$ indicator.

As observed, the implementation of the *MERICS.COMMON* module has not, when strictly considering image processing speed and quality increase, contributed to the improvement of DIA performance. The uniting of common features under the same architectural construction is however relevant and profitable when considering security, flexibility and usability over longer time periods. The total, ***-75,25***, is negative, indicating that with no other considerations, the update of the application within the specified criteria is not recommended.

Evaluating the effects of extending or modifying one of the application's components is done through testing sessions, within the extent of the affected processes. Within the first 6 months of MERICS usage, testing for deficiencies and extended features, as measured in work-hours, accounted for 60% of the time, as per the deployment and usage plan. Industry applications sometimes associate more than 90% of post-development and roll-out

resources to testing, as even small changes in core components have the capacity to affect the whole of the application.
Deficiencies in extended DIA components are searched for by specialized testing teams and users based on the behavior of the measured item and impact evaluation on co-dependent modules, in decreasing order of their relationship with the current one.

## 2 Obsolescence-Related Risks

The software market is characterized by permanent evolution in software technologies, triggered by the improvement of hardware in both memory and computation speed. The lifespan of distributed applications is linked to their capacity of solving tasks in acceptable parameters with respect to alternative solutions. Alongside technical considerations, development budgets and usage history determine resolutions on the continuation of usage. *Moral decay*, or *obsolescence* in functions, data structures and technologies, is the main factor in determining the moment of discontinuing usage, and is defined as a continuous depreciation in value for the software components, triggered by new alternatives and changes in both the user organization activity and development practices. Constructing an assessment indicator for the degree of obsolescence in DIA components considers the following relative factors and implications in using the MERICS application:

- *relative component age*, as the generic indicator of moral decay, considering the dynamicity of the industry; the basis of the estimation is the predicted life duration of the entire system as envisioned by the owners based on operational specifications; as the application transitions through successive versions, the updates include changes in framework specifications or the rewriting of source code in order to comply with performance standards and provide flexibility and extensibility in view of future changes;
- *relative software technology usage time* – based on development and deployment platforms age or software development technology usage duration; in older programming frameworks, the increase in stability due to repeated testing and elimination of bugs, viewed as a supporting factor in technological choice, is diminished by the lack of support as developers change their specialization to match trends in software evolution, and software development platform producers discontinue maintenance and upgrade operations on their products, leading to increased costs for the DIA operators;
- *error frequency dynamics*, measured considering the changes in user and security specifications, operational computing load on components, communication strain due to increases in message size and encryption algorithms processing resource needs, as larger ciphers are used in preventing brute force attacks, with the increase in large-scale availability hardware leading to continuous improvements in synchronous and asynchronous encryption;
- the *inverted relative numerical evolution of user accounts*, indicating the percentage of the current user load that the component was originally designed to serve; the indicator is calculated by dividing initial to current numbers.

Considering the enumerated factors, let $RO_w$ identify the DIA obsolescence risk in component $w$, as an average value measured in the [0,1] interval relative to the moral decay of DIA components as follows – general model:

$$RO_w = \frac{\sum_{i=1}^{n} md_i}{n},$$

where:

| | | |
|---|---|---|
| $md_i$ | – | moral decay-triggering relative factor $i$; |
| $w$ | – | measured component indicator; |
| $n$ | – | number of identified factors, |

or, for the identified elements mentioned in the current section

$$RO_w = \frac{comp_w + st_w + (1 - err_w) + (1 - u_w)}{4},$$

where:

$comp_w$ – relative age of component $w$;
$st_w$ – component $w$ relative software technology usage time;
$err_w$ – relative $w$ error incidence,
$u_w$ – inverted relative numerical evolution, $w$ component.

Interactions in distributed application components depend on the compatibility in protocols and message formatting, encryption and encoding technologies. Deprecation in software technologies in one of the endpoints leads to risks in the parsing and de-serialization of the message content. Although mediated by procedures that exclude runtime errors due to differences in standards or communication protocols, communication is required to comply with standards in software development as an insurance policy when considering development costs – as collaboration in distributed systems is critical to obtaining the output, obsolescence is prevented by constant improving and evaluation of the technological state of the components.

Considering the $RO_w$ indicator assessment, *table 2* identifies year-based relative values in MERICS modules that include communication functions, considering a lifetime of 10 years for the application, 8 years for deployment platforms and 4 years for software frameworks – major versions, chosen based on design specifications and research project objectives, as well as usage expectancy and backward compatibility for Microsoft technologies.

Research done on MERICS modules and communication context took place over a period of 6 months and information collecting is ongoing as of May 2012. Data sources for included operational, derived analytical, logging, system information from the application and associated deployment environment. Errors were filtered and written in specialized segments of the file system, and operational methods had their activity monitored and associated to operational information through unique IDs and session information.

**Table 2.** MERICS – communication obsolescence measurement

| **Component** | **Main software framework component** | **Deployment platform** | $comp_w$ | $st_w$ | $err_w$ | $u_w$ | $RO_w$ |
|---|---|---|---|---|---|---|---|
| *MERICS. DataOperations* | ADO.NET Entity Framework 4.1 | Microsoft Windows Server 2008 R2 | ***1/10 =*** 0,1 | ***2/8=*** 0,25 | 0,88 | 0,7 | **0,19** |
| *MERICS.WCF* | WCF (API, .NET 4.0 version) | Microsoft Windows Server 2008 R2 | 0,05 | 0,5 | 0,75 | 0,7 | **0,27** |
| *MERICS.WEB APP* | ASP.NET 4.0 | Microsoft Windows Server 2008 R2 | 0,05 | 0,5 | 0,6 | 0,7 | **0,31** |
| *MERICS.TEST. Desktop* | Windows Forms (API) Microsoft .NET 4.0 | Microsoft Windows Server 2008 R2 | 0,1 | 0,5 | 0,6 | 0,7 | **0,32** |
| **Average values, $RO$** | - | - | 0,07 | 0,43 | 0,7 | 0,7 | **0,27** |

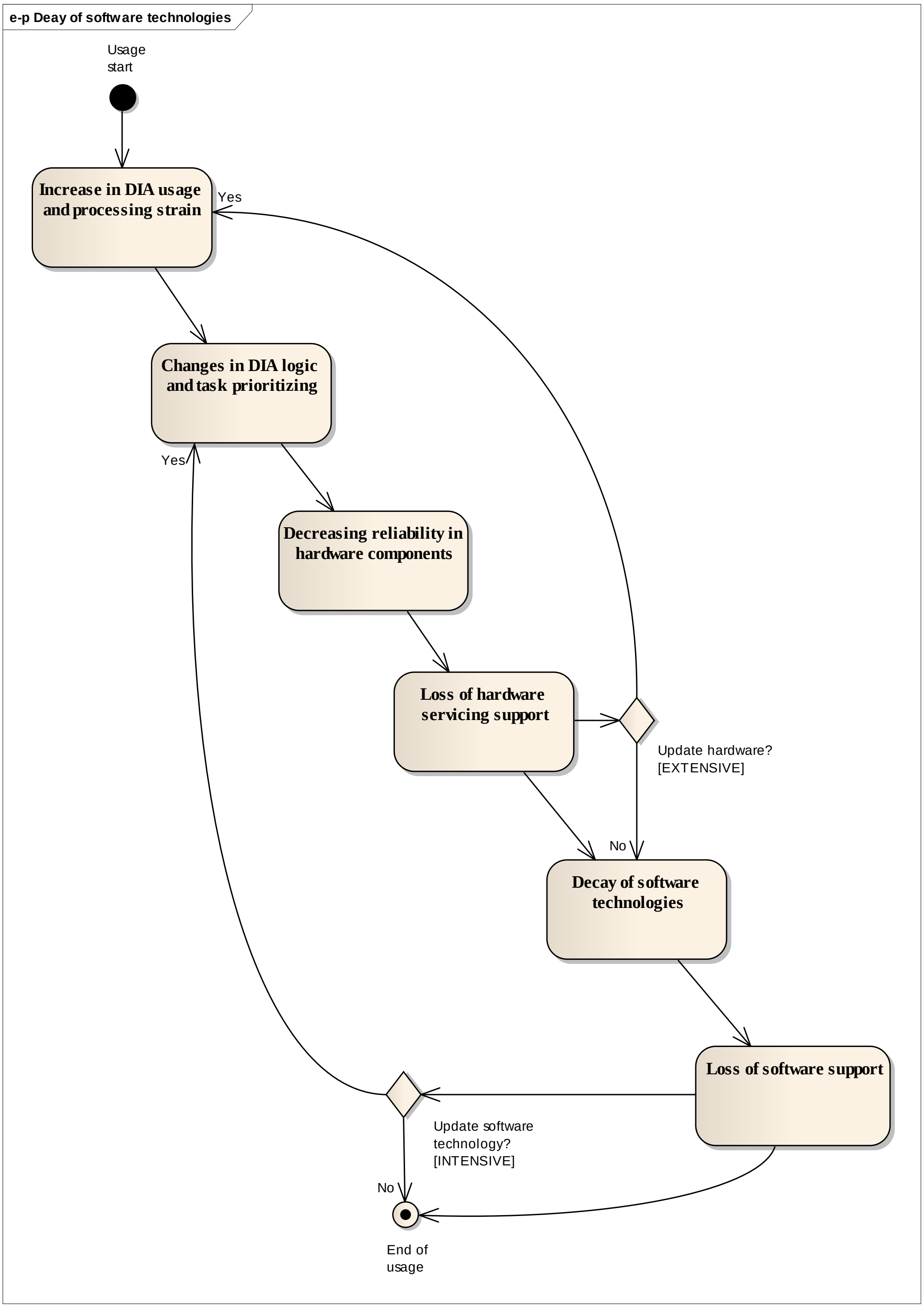


**Fig. 3.** Obsolescence-related performance drop and countermeasures

Considering the obsolescence assessment model and project objectives, MERICS communication components have a global relative decay value of **0,27**, or **27%**, with *MERICS.DataOperations* using the newest technologies, having a **19%** obsolescence factor value, and *MERICS.TEST.Desktop* a higher **32%**, due to a higher error incidence in testing environments and slightly longer deployment age.

The evaluation of component and cross-application obsolescence allows for the early identification of vulnerable areas and planning of update and replacement in both software and hardware supporting technologies. *Figure 3* shows the update-extended lifetime of a DIA component, differentiating between extensive, hardware augmenting updates, and intensive, software actualization and extension solutions.

DIA usage and processing increase, as the application is deployed and popularized through segments of the organization, leads to both communication and computing augmentation.

Let the $umd()$ function define the moral decay in distributed application components and technologies. As shown in *figure 3*, correlations exist between it and functions describing the evolution in number of users $u$, processing load $pu$ and hardware technology degradation $ht$. As all three of the enumerated factors are time-dependent or translatable to time-related functions measured in number of hours, the following are considered for set $X$ consisting of the entirety of DIA items

$$X = \{x_1, x_2, \dots x_i, \dots x_n\}, i = \overline{1,n}$$
$$umd(x_i) = f_i\big(u(h), pl(h), ht(h)\big),$$
$$u(h) = u * h_u, \qquad u: N^+ \rightarrow N$$
$$pl(h) = pu * h_{pu}, \qquad pl: N^+ \rightarrow N$$
$$ht(h) = ht * h_{th}, \qquad ht: N^+ \rightarrow N$$

where

| | | |
|---|---|---|
| $x_i$ | – | measured moral decay-affected DIA component or process $i$; |
| $i$ | – | specific measured item; |
| $f_i()$ | – | cross-system moral decay function for element $i$; |
| $h, h_u, h_{up}, h_{ht}$ | – | number of hours, as relevant to the specific function; |
| $u(h)$ | – | moral decay as a function of usage time; |
| $pl(h)$ | – | moral decay as a function of processing resources, time-based representation, |
| $ht(h)$ | – | moral decay as a function of hardware decay or age, time-based representation. |

## 3 Conclusions

Changes in DIA activity prioritization due to transitions in user activity and optimization of runtime parameters leads to a decrease in original logic relevancy. Hardware and software support diminishes with time as acquisition of newer versions of the external products is encouraged. Updating the moral decay factors, either by choosing software or hardware solutions, increases the life expectancy for the targeted component, yet the efficiency of the measure decreases with successive uses due to changes in external supporting factors – protocols, industry standards, runtime platform support.

Augmenting DIA components benefits specific tasks within the system's activity domain. As such, this operation is susceptible to difficulties in evaluating the global impact of changes:

- determining the *effects of updating logical and operational components* on the quality of information, as tested and refined scenarios become irrelevant to the new specifics of the data flows; users rely on the validity of output, leading to chained vulnerability patterns in operational and analytical activities; solutions rely on documenting and disseminating factor change information, allowing interested parties to account eventual incidents;
- *treating malfunctions* triggered by underperformance in *external* DIA dependencies, outside the control of the user organization's decision factors; extending the controls for better prevention of newly encountered incidents is not known a priori and therefore constitutes a risk element itself; developing flexible and easily configurable application components helps reducing the incident effects removal time, for both developers and administrators;
- understanding existing *implementation specifics*, as development teams change through-out the lifetime of the application, either at individual or organization level; documenting the development specifics

reduces this risk, yet time pressures often prevent the proper description of features, as well as testing time, allowing for technical bugs to remain undiscovered, which leads to increased pressure on the incident fixing team.

Due to changes in actors as compared to the development stage, as well as qualitative effects deriving from the increased time span of DIA usage, procedural and technical information is not readily available and must be accounted for in management decisions. Ensuring the proper conditions for DIA usage reduces costs and helps interacting parties rely on information quality in addressing operational activities.

## References

[1] M.P. Lima, J.M.N. David, 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, ISBN 978-1-4244-8445-4.

[2] B. Boehm, J. Bhuta, *Balancing Opportunities and Risks in Component-Based Software Development*, IEEE Software, Volume 25, Issue 6, December 2008, pp. 56 - 63, ISSN 0740-7459.

[3] W. Yu, X. Tuo, 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, ISBN 978-1-4244-5537-9.

[4] J.A. Ibrahim, M. Majid, A.H. Hashim, R.M. Tahar, *Risk Quantification in Coal Procurement for Power Generation: The Development of Supply Shortage Impact Matrix,* Second International Conference on Computational Intelligence, Modelling and Simulation (CIMSiM), 28-30 September 2010, pp. 401 - 406, ISBN 978-1-4244-8652-6.

[5] C. Huo, Y. Teng, *Risks and risk management in the development of information system,* Second International Conference on Communication Systems, Networks and Applications (ICCSNA), Volume 2, June 29 - July 1 2010, pp. 345 - 349, ISBN 978-1-4244-7475-2

**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 In-formatics 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.