**Journal of Software Engineering and Applications, 2014, 7, 371-386**
Published Online May 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.75034

# An Application of Paraconsistent Annotated Logic for Design Software Testing Strategies

**Marcos Ribeiro do Nascimento[1], Luiz Alberto Vieira Dias[1], João Inácio Da Silva Filho[2]**

[1]Brazillian Aeronautics Institute of Technology—ITA, São José dos Campos, Brazil
[2]Santa Cecília University—UNISANTA, Santos, Brazil
Email: mribeiro@ita.br, vdias@ita.br, inacio@unisanta.br

Received 26 February 2014; revised 20 March 2014; accepted 28 March 2014

## Abstract

**Nowadays, application model systems for decision-making based on non-classical logic such as Paraconsistent Logic are used successfully in the treatment of uncertainties. The method presented in this paper is based on the fundamental concepts of Paraconsistent Annotated Logic with annotation of 2 values (PAL2v). In this study, two algorithms based on PAL2v are presented gradually, to extract the effects of the contradiction in signals of information from a database of uncertain knowledge. The Paraconsistent Extractors Algorithms of Contradiction Effect-Para Extr$_{ctr}$ is applied to filters of networks of analyses (PANets) of signal information, where uncertain and contradictory signals may be found. Software test case scenarios are subordinated to an application model of Paraconsistent decision-making, which provides an analysis using Paraconsistent Logic in the treatment of uncertainties for design software testing strategies. This quality-quantity criterion to evaluate the software product quality is based on the characteristics of software testability analysis. The Para consistent reasoning application model system presented in this case study, reveals itself to be more efficient than the traditional methods because it has the potential to offer an appropriate treatment to different originally contradicting source information.**

## Keywords

**Paraconsistent Logic, Design Testing Strategies, Software Testability, Paraconsistent Decision Making Model**

## 1. Introduction

This paper is the result of studies carried out for the Brazilian National Water Agency, within the scope of the Amazonian Integration and Cooperation Project for the Modernization of Hydrological Monitoring (in Portuguese, ICAMMH). The project was developed at the Brazilian Aeronautics Institute of Technology (ITA) [1].

The purpose of this paper is to present an application of Artificial Intelligence techniques using Paraconsistent Logic in the treatment of uncertainties, for design software testing strategies, in terms that allow an analysis of a quality-quantity *criterion* to evaluate the software product quality based on the characteristics of software testability: ISO/IEC 25010:2011 [2] and heuristics of software testability by Bach [3].

During the software product development activity on the ICAMMH project, software product engineers used the quality management core assets, according to the software product development plan and a description of the particular software product to be built in Software Product Lines (SPL) [4] [5]. These planned activities carry out actions to program measurement of software quality and evaluate the application software testing standards. There is a large gap between the requirement that software quality measurement should be carried out and the standards and practices on how to carry out the measurements of software quality. In this context of the ICAMMH project, the uncertainty in decision making on the design software testing strategies, as regarding defect and faults origins of the software product, is rarely captured explicitly, in order to provide continuous quality improvement of the software product engineering, during the development process of the software.

The problem to be addressed is how to create a quality-quantity *criterion* that can measure continuous improvement actions during the process of software product development by SPL, in accordance to software quality standards. The specific approach focuses on software product quality by software testability: sub characteristics of the maintainability SQuaRE 25010:2011 [2] and heuristics of software testability of Bach [3].

Two research questions (RQ) were formulated:

RQ1. Is it possible to construct a Paraconsistent Analysis System Model in a way that facilitates the establishment of test *criteria* and quality performance of tests based on the quality-quantity *criteria* to evaluate the software product?

RQ2. Is it possible to develop an application with artificial intelligence tools to evaluate uncertainty in software testability *criteria*: 1) the measurements are feasible enough to allow testers diagnose techniques and methods; and 2) the practices and reachable test criteria of software product quality?

While RQ1 is addressed in the domain of Software Product Quality Engineering based on Paraconsistent Analysis Model for Uncertainty Treatment in Design Testing Strategies on SPL, RQ2 is focused on the interpretation results of Paraconsistent analysis from the quality-quantity characteristics of testability *criteria* corresponding to the software product quality standards application mapped.

This research provides important tools, in order to develop an application of Paraconsistent Annotated Logic (PAL) in designing software testing to act as powerful computational tools, dedicated to software testability analysis, and based on quality-quantity criteria without contamination by contradictory information signs.

This paper is organized in five sessions: Section 2 presents the fundamental theories; Section 3, the methodology for designing testing software model analysis, with Paraconsistent Logic (PL); Section 4, an application of the Paraconsistent Annotated Logic (PAL); Section 5, the conclusions reached.

## 2. Fundamentals Theories

### 2.1. Environment Software Testing for the ICAMMH Project

Eight software product lines were originally planned for the ICAMMH, but only one of them was actually available at the time of this study. Thus, only the Software Product Line 2 (SPL2) was utilized in experimental research. Requirements data and specifications design were collected from two different experiments and placed within the following software testing artifacts: test plan, test procedures, test cases, and test scenarios, as in IEEE Std. 829 [6]. This was done using functional and structural techniques, described in Perry [7], and defect prevention analysis, as in Sharma *et al.* [8].

The Software Test (ST) team used the following black-box techniques in SPL2 in the functional test: 1) Equivalence Class Partitioning; 2) Boundary Value Analysis, and 3) Cause Effect Graphing. This ST team prepared 256 test cases for the 8 usecases developed for SPL2. To obtain the software testing results, Test Cases (TC), as in [7], based on use case requirements, as in [9], they were manually performed for SPL2.

Different metrics were used to assess the SPL2, as seen in **Table 1**. In the first column, the indicators and metrics, in the second column, the calculation formulas. The third column displays the values calculated for the first Integration Test (IT) of SPL2. The Regression Testing (RT) results (1˚ RT, and 2˚ RT 3˚ RT) are shown in subsequent columns. **Table 1** shows the primary metrics: 1) testing coverage percent use case by the PUC indi-

Table 1. Software Production Line 2—Derived Metrics and Indicators [Test Cases Have Found Defects—TCFD (*)].

| Indicators and Metrics | Formula | 1˚ IT | 1˚ RT | 2˚ RT | 3˚ RT |
|---|---|---|---|---|---|
| Testing Coverage Percent Use Case (PUC)—Indicator | (qty.) UC/total of UC | 6/8 = 0.75 [75%] | 8/8 = 1 [100%] | 8/8 = 1 [100%] | 8/8 = 1 [100%] |
| Defects Found by Software Test (DFST)—measure of efficacy | TCFD/total of TC | 91/170 = 0.53 [53%] | 66/235 = 0.28 [28%] | 57/238 = 0.24 [24%] | 60/256 = 0.23 [23%] |
| Defects Found by week (DEW)—indicator | TCFD/total of weeks | 91/6 = 15.2 | 66/4 = 16.5 | 57/4 = 14.25 | 60/3 = 20 |
| Effectiveness Defect Detection (EDD)—indicator | TCFD (n − 1) RT /TCFD (n) RT | N/A | 91/66 = 1.38 | 66/57 = 1.16 | 57/60 = 0.95 |
| Effectiveness of Removal of Defects (ERD)—indicator | Remove TC defects No. RT/TCFD | N/A | 4/66 = 0.06 [6%] | 24/57 = 0.42 [42%] | 29/60 = 0.48 [48%] |
| Percentage of Test Coverage (PTC)—metric efficiency | (qty.) TC exercised/ total of TC | 95/170 = 0.56 [56%] | 194/235 = 0.83 [83%] | 199/238 = 0.84 [84%] | 215/256 = 0.84 [84%] |
| Density Efficiency Software Test (DEST)—indicator efficacy | TCFD/(qty.) TC exercised | 91/95 = 0.95 [95%] | 66/194 = 0.34 [34%] | 57/199 = 0.29 [29%] | 60/215 = 0.28 [28%] |

cator; 2) the efficiency in the detection of defects per week, calculated by the DEW indicator, which increased with time; and 3) the percentage of effectiveness in detecting defects, calculated by the DEST indicator, which decreased with time.

**Table 1** also shows the secondary metrics: 1) the percentage of effectiveness in detection of defects, calculated by the DFTS measure of efficacy, decreased with time; and 2) the percentage of efficiency in the execution of ST, calculated by PTC metric efficiency, which increased with time.

Also in **Table 1**, a comparison with RT (previous to the current one), it is observed that: 1) an efficient defect detection, calculated by the EDD indicator, decreased with time, and that the percentage of effectiveness of removal of defects, calculated by the ERD indicator, increased with time. The derived or secondary metrics present in **Table 1**, should provide the attributes of software product quality, as well as mechanisms to control the software process. The relationship between object-oriented design metrics and testability of classes uses on **Table 1** as reference study by [9] [10].

## 2.2. The Use of Reference Standards for Testability Software

The theoretical approach used in this experimental research in terms of building components for a Paraconsistent quality-quantity analysis were based on the standards below.

The software product quality model SQuaRE 25010:2011 is composed of eight properties (Functional Suitability, Reliability, Performance Efficiency, Operability, Security, Compatibility, Maintainability, Transferability), which are further subdivided into sub-characteristics that can be measured internally or externally. Maintainability quality property refers to the product's ability to be modifiable and changeable. Testability is one such sub-characteristic; the capability of the software is determined by a set of internal attributes that can be measured [2].

The ICAMMH Project also made use of the James Bach's testability heuristics, to design internal facilitators for the testability of the software in the form of a set of quality attributes to describe the testability sub-characteristic, within the maintainability property of SQuaRE 25000:2011 standards. Khan and Mustafa [11] "extended James Bach internal facilitators to 2 (two) new approaches: external and environmental facilitators".

## 2.3. Paraconsistent Logic and the Treatment of Uncertainties

Paraconsistent Logic (PL) belongs to a non-classical logic category and its main feature is the revocation of the principle of non-contradiction (see [12]-[15]). The pioneers of PL are J. Lukasiewicz and the Russian philosopher N. A. Vasilév [11], who around 1910, simultaneously, but independently, suggested the possibility of the existence of a logic that did not use the principle of non-contradiction.

The initial systems of PL contained all logical levels such as propositional and predicate *calculi* as well as the Logic of Superior Order are due to N. C. A. da Costa (see [16]-[18]). PAL belongs to a family of PL and can be represented through a lattice of four vertices. These four vertices represent extreme logical states of propositions.

According to Da Costa *et al*. [19] [20], in PAL, the proposition *P* is accompanied with annotations; each annotation belongs to a finite lattice and attributes a logic value to the correspondent logical proposition. It is considered that each evidence degree attributes a logic value to the proposition that belongs to a group of values composed of the constants of annotation of the lattice $\{\top, t, F, \perp\}$ for which the following order relationship is defined: $\perp < t$, $\perp < F$, $t < \top$ and $F < \top$. PAL may be represented through a Hasse diagram in which the constants in the vertices of a lattice will show extreme logical states to the propositions sentence, where: $\top$ = Inconsistent, t = True, F = False and $\perp$ = Paracompleteness. The PAL language, the semantics of a complete set of connectives and axioms is found with details in Da Silva Filho *et al*. [21] and [22].

## 2.4. The Main Algorithms of the Paraconsistent Annotated Logic with Annotation of 2 Values—PAL2v

According to Da Silva Filho *et al.* [21] and [22] a representation of how the annotations, or evidences, express the knowledge about a certain proposition *P* can be obtained through PAL. This is done through a lattice on the real plane with pairs (μ, λ) which are the annotations seen in **Figure 1**. In this representation an operator ~ is fixed, defined as follows: $\sim:|\tau| \rightarrow |\tau|$ where $\tau = \{(\mu, \lambda)| \mu, \lambda \in [0, 1] \subset \Re\}$.

If *P* is a basic formula then: $\sim [(\mu, \lambda)] = (\lambda, \mu)$, where $\mu, \lambda \in [0, 1] \subset \Re$. The operator ~ stands for the "meaning" of the logical symbol of negation of the system to be considered. The *P*(μ, λ), can be intuitively read: "It is assumed that *P*'s belief degree (or favorable evidence) is μ and disbelief degree (or contrary evidence) is λ. Thus, (1,0) intuitively indicates total belief, (0,1) indicates total disbelief, (1,1) indicates total inconsistency, and (0,0) indicates total paracompleteness (indetermination)" [21] [22].

This Paraconsistent system for the treatment of uncertainties using the Paraconsistent algorithm can calculate values that are representative of information signals. In a Paraconsistent analysis system for decision-making, the inputs μ and λ are values contained in the closed interval between 0 and 1, which belong to the set of real numbers. These two values come from two or more information sources, which search for favorable or contrary evidences in respect to the same proposition *P*. Since they originate from different sources, these values may be similar, thus representing consistency, or be different, representing a contradiction. The resulting Paraconsistent system is called basic Paraconsistent Analysis Node (PAN), according to Da Silva Filho *et al.* [21] [22].

The projects that utilize PANs may carry out information treatment through Paraconsistent analysis networking, without using signal adjustments, neither weight that may leave the processing dependent on external factors. With these results, a PAN may be connected to other PAN, forming a Paraconsistent Analysis Decision Network (PANNet) to estimate values and perform analysis of information in different conditions.

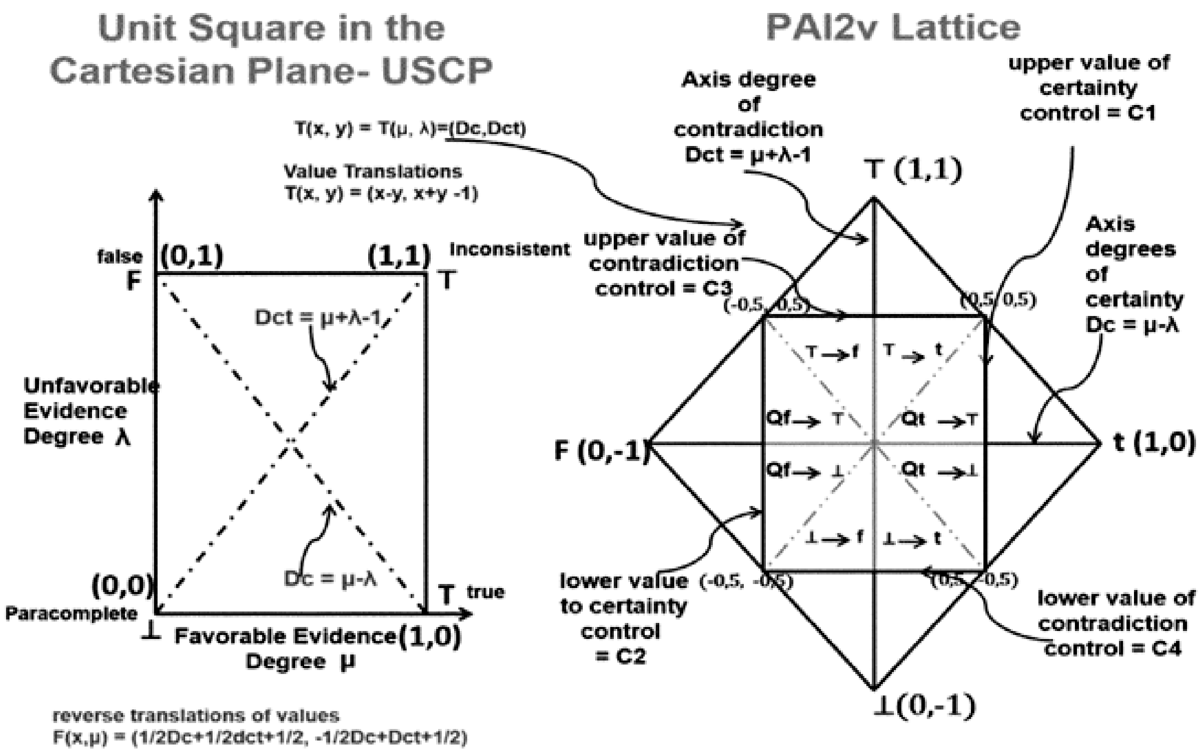
The special case where it is known initially where exist many contradictions attributed to certain evidence of Proposition A (*P*a) should be considered. This weakens the evidence going on to another Proposition B (*P*b*)*. In this case, a special configuration is used, where output represented by an interval of evidence $\varphi_{ext}$ of the Paraconsistent Analysis Node (PAN) controls the analysis carried out by another PAN. This configuration, where three-dimensional analysis occurs, is called the Paraconsistent Cube Analyzer (PCA), according to Da Silva Filho *et al.* [20] [21].

## 3. Methodology for Designing Testing Software Model Analysis

The topology of PANNet is arranged in order to treat uncertainties; it can be built with several interlinked PAN-algorithms. Thus, in a basic configuration, the preliminary analysis in a PAN produces a value of real Degree Certainty $D_{CR}$ and a signaled interval of certainty $\varphi_e(\pm)$, referring to a unique proposition. According to Da Silva Filho *et al*. [23] [24] the Paraconsistent Algorithm Extractor of Contradiction effects ($ParaExtr_{ctr}$) is applied to connections of the PANNet. This configuration forms a Paraconsistent Analysis Network, which is capable of extracting the effects of the contradiction, gradually from the signals of information that come from the uncertain knowledge database source. **Figure 2** shows the representation of the algorithm extractor of contradiction effects that uses a network of three PANs.

The Paraconsistent Analysis for Extraction of Contradiction effects ($ParaExtr_{ctr}$) algorithm is based on a topology that enables PANNet to represent a range of situations, where the existing ($\varphi_{ext}$) group of clusters of similar semantic nature elements coming from several sources of information representing a modeling PAN Kernels, according to Da Silva Filho *et al.* [23] [24].

***ParaExtr**$_{ctr}$*—Extractor of Contradiction Effects Algorithm—Part (a), similar semantic nature


Unit Square in the Cartesian Plane- USCP
T(x, y) = T(μ, λ)=(Dc,Dct)
Value Translations
T(x, y) = (x-y, x+y -1)
false
F
(0,1)
(1,1)
Inconsistent
T
Dct = μ+λ-1
Unfavorable Evidence Degree λ
Dc = μ-λ
(0,0)
Paracomplete
⊥
T true
Favorable Evidence Degree μ
(1,0)
reverse translations of values
F(x,μ) = (1/2Dc+1/2dct+1/2, -1/2Dc+Dct+1/2)
PAl2v Lattice
Axis degree of contradiction Dct = μ+λ-1
upper value of certainty control = C1
T (1,1)
upper value of contradiction control = C3
(-0,5, 0,5)
(0,5, 0,5)
Axis degrees of certainty Dc = μ-λ
T→f
T→t
Qf→T
Qt→T
F (0,-1)
t (1,0)
Qf→⊥
Qt→⊥
⊥→f
⊥→t
lower value to certainty control = C2
(-0,5, -0,5)
(0,5, -0,5)
lower value of contradiction control = C4
⊥(0,-1)

| Extreme States | Symbol | Non-extreme states | Symbol |
|---|---|---|---|
| True | t | Quasi-true tending to Inconsistent | Q t →T |
| False | F | Quasi-true tending to Paracomplete | Q t →⊥ |
| Inconsistent | T | Quasi-false tending to Inconsistent | QF→T |
| Paracomplete | ⊥ | Quasi-false tending to Paracomplete | QF→⊥ |
| | | Quasi-inconsistent tending to True | QT→t |
| | | Quasi-inconsistent tending to False | QT→F |
| | | Quasi-paracomplete tending to True | Q⊥→t |
| | | Quasi-paracomplete tending to False | Q⊥→F |

**Figure 1.** Organization associated lattice of the paraconsistent annotated logic [22].

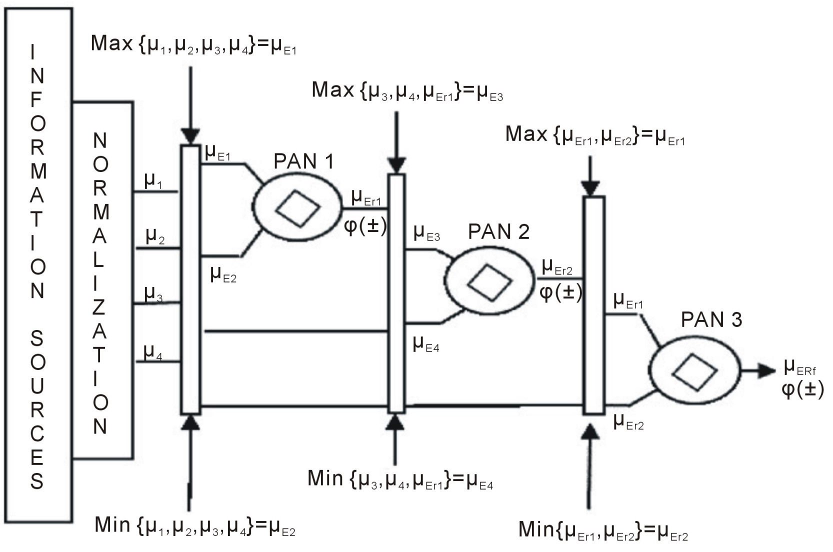


**Figure 2.** The Paraconsistent algorithm extractor of contradiction effects ($ParaExtr_{ctr}$).

1) Present $n$ values degrees of evidence that compose the group under study

$$G_\mu = (\mu_A, \mu_B, \mu_C, \cdots, \mu_n) */\text{degrees of evidence} \quad 0 \le \mu \le 1*/ \ .$$

2) Select the largest value among the degrees of evidence the group under study

$$\mu_{\max A} = \max(\mu_A, \mu_B, \mu_C, \cdots, \mu_n).$$

3) Consider the largest value among the degrees of evidence the group under study in the favorable degree of evidence $\mu_{\max A} = \mu_{\text{sel}}$.

4) Consider the smallest value among the degrees of evidence the group under study in the favorable degree of evidence: $\mu_{\min A} = \min(\mu_A, \mu_B, \mu_C, \cdots, \mu_n)$.

5) Transform the smallest value among the degrees of evidence the group under study in the unfavorable degree of evidence $\lambda_{\text{sel}} = 1 - \mu_{\min A}$.

6) Do the Paraconsistent analysis among the selected values.

$\mu_{R1} = \mu_{\text{sel}} \lozenge \lambda_{\text{sel}}$ */ where $\lozenge$ is a Paraconsistent action of the **PAN kernel** */.

7) Increase the obtained value μR1 in the group under study, excluding the two values μmax e μmim from this, selected previously.

$$G_\mu = (\mu_{R1}, \mu_A, \mu_B, \mu_C, \cdots, \mu_n) - (\mu_{\text{maxi}}, \mu_{\text{mini}}); \ \text{where } i = A, \cdots, N$$

8) Return to item 2 until the Group under study has only 1 element resulting from the analyses,

Go to item 2 until $G_\mu = (\mu_{Er})$.

```
Function PAN Kernel (μ, λ){
1) Enter with the input values
μ */favorable evidence Degree $0 \le \mu \le 1$
λ */unfavorable evidence Degree $0 \le \lambda \le 1$
2) Calculate the normalized Contradiction Degree
$\mu_{ctr} = (\mu + \lambda)/2$
3) Calculate the interval of evidence resulting
$\varphi_{eint} = 1 - |2\mu_{ctr} - 1|$
4) Calculate the Certainty Degree
$Dc = \mu - \lambda$
5) Calcule the Contradiction Degree
$Dct = \mu - \lambda + 1$
6) Calculate the distance d into Lattice
$d = \text{root}((1 - |Dc|)^2 + Dct^2))$
7) Compute the output signal
If $\varphi_{eint} < 0, 25$ OR $d > 1$, then
do: S1 =0,5 e S2 = φe( ±):
uncertainty go to item 11
Else: go to next step
8) Calculate the real Certainty Degree
If $Dc > 0$ then $G_{cr} = (1 - d)$
If $Dc < 0$ entãoG$_{cr} = (d - 1)$
9) Calculate the real Evidence Degree
$\mu_{er} = (G_{cr} + 1)/2$
10) Present the outputs
$S1 = \mu_{er}$ and $S2 = \varphi e(\pm)$
11) End}
```

The new algorithm *(ParaExtr$_{ctrPlus}$)* is based on a topology that enables PANNet to represent a range of situations, where the existing ($\varphi_{\text{ext}}$) group with possibility to represent also clusters, with different semantic nature elements coming from several sources of information representing a modeling PCA Kernels, As exposed below.

***ParaExtr$_{ctrPlus_s}$***—Extractor of Contradiction Effects plus Algorithm—Part (b), different semantic nature:

1) Present *n* values degrees of evidence that compose the group under study

$$G_\mu = (\mu_A, \mu_B, \mu_C, \cdots, \mu_n) * / \text{degrees of evidence} \quad 0 \le \mu \le 1 * / \ .$$

2) Select the largest value among the degrees of evidence the group under study

$$\mu_{\max A} = \max(\mu_A, \mu_B, \mu_C, \cdots, \mu_n).$$

3) Consider the smallest value among the degrees of evidence the group under study in the favorable degree of evidence: $\mu_{\text{sel}} = \mu_{\max A}$.

4) Consider the smallest value among the degrees of evidence the group under study in the favorable degree of evidence: $\mu_{\min A} = \min(\mu_A, \mu_B, \mu_C, \cdots, \mu_n)$.

5) Transform the smallest value among the degrees of evidence the group under study in the unfavorable degree of evidence.

If $\mu_{\text{minA}} = \mu_{\text{source}}$ then do $\lambda_{\text{sel}} = \lambda\mu_{\text{minA}}$ else $\lambda_{\text{sel}} = 1 - \mu_{\text{minA}}$;

/* if there is more than one $\mu_{\text{minA}} = \mu_{\text{source}}$, choose the largest λ corresponding to $\mu_{\text{minA}}$ */

6) Do the Paraconsistent analysis among the selected values

$\mu_{\text{R1}} = \mu_{\text{sel}} \Diamond \lambda_{\text{sel}}$; */ where ◊ is a Paraconsistent action of the **PCA Kernel** */

7) Increase the obtained value μR1 in the group under study, excluding from this the two values μmax e μmim, selected previously $G_\mu = (\mu_{R1}, \mu_A, \mu_B, \mu_C, \cdots, \mu_n) - (\mu_{\text{maxi}}, \mu_{\text{mini}})$; where $i = A, \cdots, N$.

8) Return to item 2 until the Group under study has only 1 element resulting from the analyses,

Go to item 2 until $G_\mu = (\mu_{Er})$.

**Function PCA Kernel** ($\varphi_{\text{ext}}$,μ, λ) {

1) Enter with the input values

μ */favorable evidence Degree $0 \le \mu \le 1$

λ */unfavorable evidence Degree $0 \le \lambda \le 1$

$\varphi_{\text{ext}}$ */external interval Evidence*/

2) Verify initial conditions

If I$\varphi_{\text{ext}} < 0, 25$, then

do: S1 = 0.5e S2 = $\varphi_{\text{ext}}$: uncertainty go to item 15

Else: go to next step

3) Calculate the max value of Evidence degreetending to max (true, false)

$\mu_{\text{Emaxt}} = (1 + \varphi_{\text{ext}})/2$

$\mu_{\text{Emaxf}} = (1 - \varphi_{\text{ext}})/2$

4) Calculate the Evidence degree internal results $\mu_{\text{eint}} = (\mu - \lambda + 1)/2$

5) Verify the conditions

If $\mu_{\text{eint}} >= \mu_{\text{Emaxt}}$, then do: Dc = $\varphi_{\text{ext}}$, Dct = 1 − $\varphi_{\text{ext}}$, and $\varphi_{\text{ext}} = \varphi_{\text{e}}$ (+) go to item 12;

If $\mu_{\text{eint}} >= \mu_{\text{Emaxf}}$ then do: Dc = −$\varphi_{\text{ext}}$, Dct = 1 − $\varphi_{\text{ext}}$, and $\varphi_{\text{ext}} = \varphi_{\text{e}}$(+) go to item 12; else next item.

6) Calculate the normalized Contradiction Degree $\mu_{\text{ctr}} = (\mu + \lambda)/2$

7) Calculate the interval of Evidence resulting

$\varphi_{\text{eint}} = 1 - |2\mu_{\text{ctr}} - 1|$

8) Calculate the Certainty Degree

Dc = μ – λ

9) Calcule the Contradiction Degree

Dct = μ − λ + 1

10) Calculate the distance d into Lattice

d = root((1 − |Dc|)$^2$ + Dct$^2$))

11) Compute the output signal

if $\varphi_{\text{eint}} < 0, 25$ OR d > 1, then

do: S1 = 0,5 e S2 = $\varphi_{\text{eint}}$(±): uncertainty go to item 15

Else: go to next step

12) Calculate the real Certainty Degree

If Dc > 0 then Dcr = (1 − d)

```
If Dc < 0 then Dcr = (d − 1)
13) Calculate the real Evidence Degree
$\mu_{er}$ = (Dcr + 1)/2
14) Present the outputs
S1 = $\mu_{er}$ and S2 = $\varphi_e(\pm)$
15) End}
```

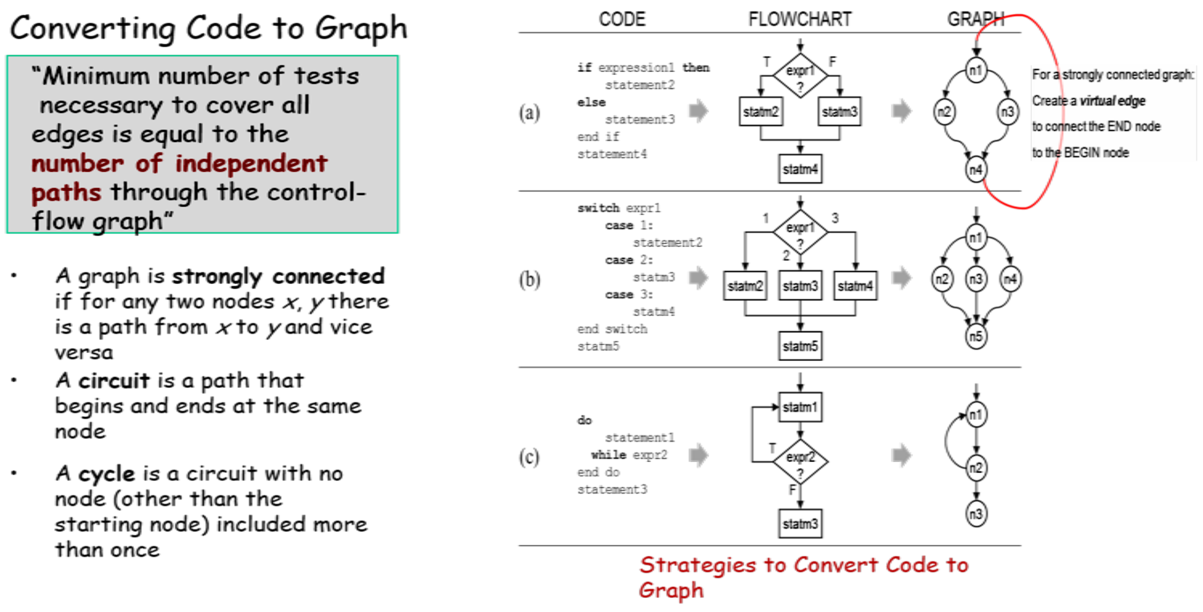
Structural metrics are based on the properties of flow graph models of programs. The attention is focused on control-flow and data-flow complexity. Linguistic complexity is ignored. Cyclomatic Complexity V(G) is ameasure of the complexity of a module's decision structure. It is the number of linearly independent paths and therefore, the minimum number of paths that should be tested. Invented by Thomas McCabe (1974) to measure the complexity of a program's conditional logic: counts the number of decisions in the program, under the assumption that decisions are difficult for people and makes assumptions about decision-counting rules and linear dependence of the total count to complexity. McCabe's Cyclomatic complexity is defined as $V(G) = e - n + 2p$, where:

e = number of links in the flow graph;
n = number of nodes in the flow graph; and
P = number of disconnected parts of the flow graph.

The $ParaExtr_{ctr}$ and $ParaExtr_{ctrPlus}$ algorithms are made using C programming language and both modules PAN and PCA Kernel. Control flow graphs describe the logic structure of software modules. A module corresponds to a single function or subroutine in typical languages, has a single entry and exit point, and is able to be used as a design component via a call/return mechanism. Each flow graph consists of nodes and edges. The nodes represent computational statements or expressions, and the edges represent transfer of control between nodes. As seen **Figure 3**.

The complexity of several graphs considered together is equal to the sum of the individual complexities of those graphs. The strategy of modularization $ParaExtr_{ctr}$ and $ParaExt_{rctrPlus}$ algorithms aims to implement C language source code programs for optimal number of cyclomatic complexity for each PAN and PCA Kernel function around values lower than 10. As seen **Figure 4** and **Figure 5**.

Practical Software Quality Issues where considered: 1) no program module should exceed a cyclomatic complexity of 10; and 2) software refactoring are aimed at reducing the complexity of a program's conditional logic. Essentially, with cyclomatic complexity, higher numbers are "bad" and lower numbers are "good". We use cyclomatic complexity to get a sense of how hard any given code may be to test, maintain, or troubleshoot as well as an indication of how likely the code will be to produce errors.



Converting Code to Graph

"Minimum number of tests necessary to cover all edges is equal to the **number of independent paths** through the control-flow graph"

- A graph is **strongly connected** if for any two nodes *x*, *y* there is a path from *x* to *y* and vice versa
- A **circuit** is a path that begins and ends at the same node
- A **cycle** is a circuit with no node (other than the starting node) included more than once

(a)
```
if expression1 then
   statement2
else
   statement3
end if
statement4
```

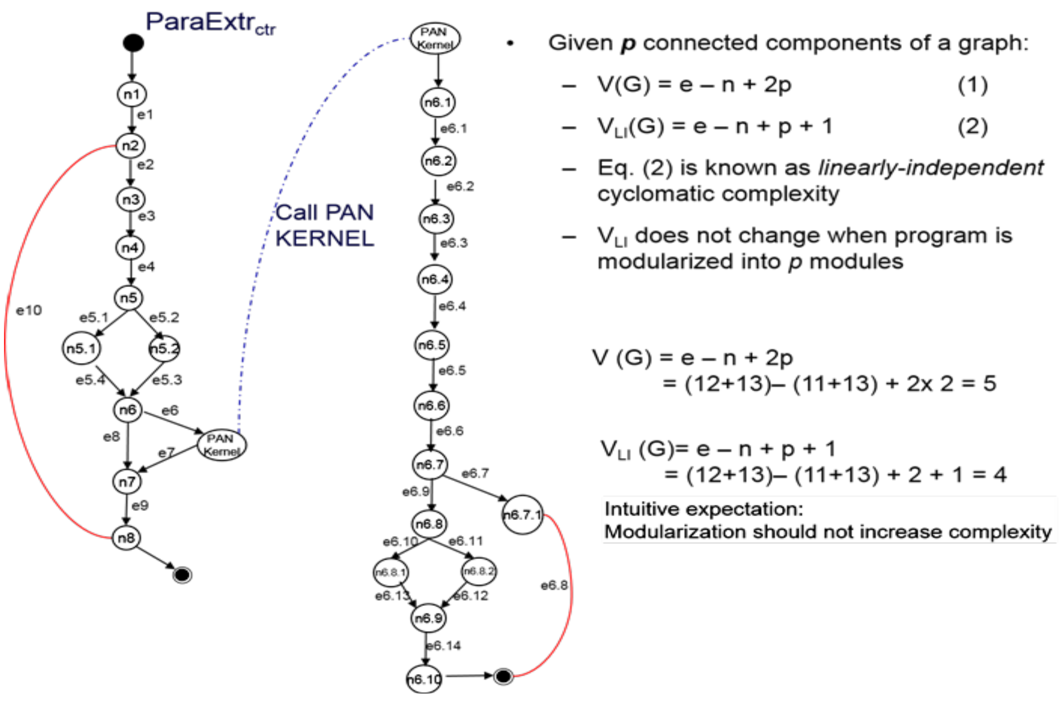
(b)
```
switch expr1
   case 1:
      statement2
   case 2:
      statm3
   case 3:
      statm4
end switch
statm5
```

(c)
```
do
   statement1
  while expr2
end do
statement3
```

For a strongly connected graph:
Create a ***virtual edge*** to connect the END node to the BEGIN node

Strategies to Convert Code to Graph

**Figure 3.** Strategies to Convert Code to Graph (*Maccbe V*(*g*)).



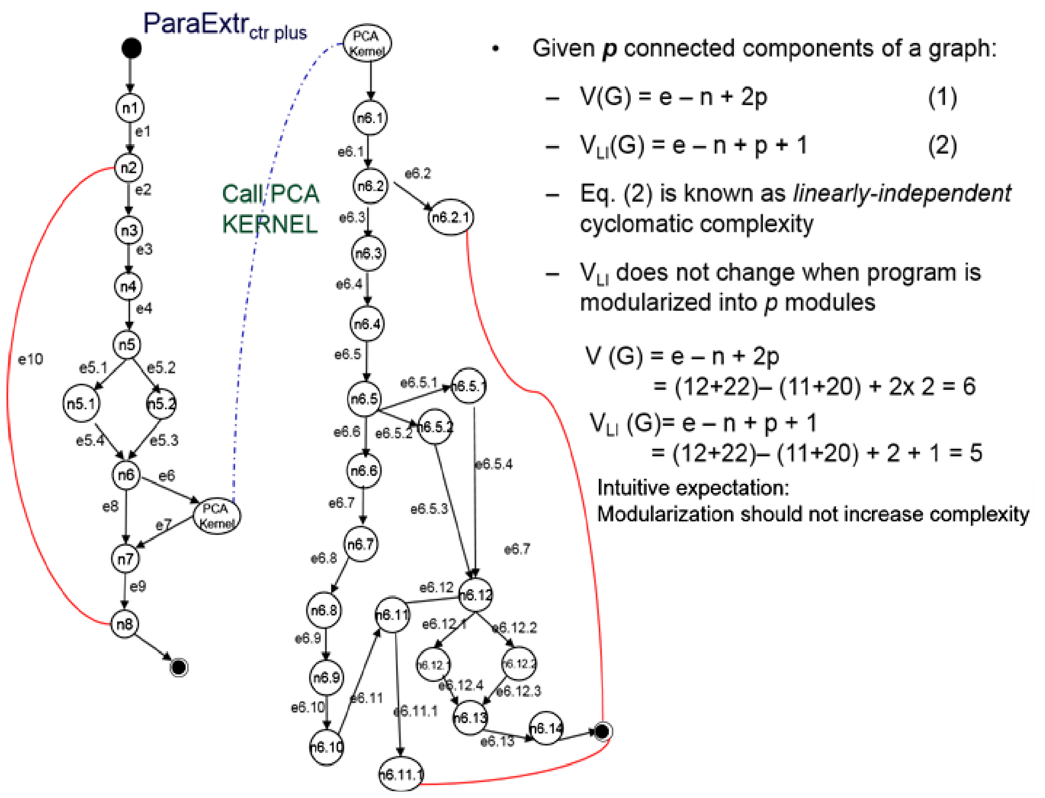
- Given ***p*** connected components of a graph:
  - V(G) = e – n + 2p (1)
  - $V_{LI}$(G) = e – n + p + 1 (2)
  - Eq. (2) is known as *linearly-independent* cyclomatic complexity
  - $V_{LI}$ does not change when program is modularized into *p* modules

V (G) = e – n + 2p
= (12+13)– (11+13) + 2x 2 = 5

$V_{LI}$ (G)= e – n + p + 1
= (12+13)– (11+13) + 2 + 1 = 4

Intuitive expectation:
Modularization should not increase complexity

**Figure 4.** McCabe's Cyclomatic Analysis *ParaExtr*$_{ctr}$ Algorithm.



- Given ***p*** connected components of a graph:
  - V(G) = e – n + 2p (1)
  - $V_{LI}$(G) = e – n + p + 1 (2)
  - Eq. (2) is known as *linearly-independent* cyclomatic complexity
  - $V_{LI}$ does not change when program is modularized into *p* modules

V (G) = e – n + 2p
= (12+22)– (11+20) + 2x 2 = 6

$V_{LI}$ (G)= e – n + p + 1
= (12+22)– (11+20) + 2 + 1 = 5

Intuitive expectation:
Modularization should not increase complexity

**Figure 5.** McCabe's Cyclomatic Analysis *ParaExtr*$_{ctrPlus}$ Algorithm.

## 4. Application of the PAL2v for Design Testing Software and Discussion

Developers at ITA Project were also responsible for ensuring that the testability issues were addressed in the requirements and design phases of development to support test planning and test design. Testers interacted with a team of designers to deal with the occurrence of testability problems, as well as the documentation of the design specifications used as the basis of test cases that would support integration testing and unit testing. The master test plan was revisited and enhanced with new information as needed upon every release of every SPL.

The ITA Software Testing (ST) team consisted of four (4) members with the following responsibilities for profile: senior test leader, usability test engineer, manual test engineer and automated test engineer. According to Elfriede [25] the capabilities of the ST team may affect the success or failure of the testing effort. It is not enough for a testing team to be technically proficient with the testing techniques and tools necessary to carry out the actual tests. Depending on the complexity of the domain, a test team should also include members who have a comprehensive understanding of the problem domain.

Using the above standards in SPL2, a treatment system of uncertainty for decision-making based on the PAL2v, was applied to investigate how to treat degrees of evidence related to the use for designing testing techniques based on software testability by quality-quantity indicators. Bach's quality facilitator heuristics for software testability were matched to the testability quantity indicators of SQuaRE 25010 to analyze the impact on best practices and techniques the test scenarios analysis.

Considering two information sources, [A] and [R], that send the evidence concerning a certain Proposition *Pn* to the analysis and decision-making system, as seen in **Figure 6** for the 1st step—(primary sources of information: accept or reject the acquisition data). The data: [A] —acceptance—degree of acceptance value of the proposition in the interval [0,10]; [R]—rejection—degree of rejection value of the proposition in the interval [0,10].

The degrees of evidence acquisition of the samples were obtained through mathematical equations, through normalization from modelling evidence signals, by acceptance and rejection acquisition data, with the exposed values in the degrees of evidence valued between 0 and 1 in the PAL2v lattice.

The degree of favorable evidence ($\mu$), expressing the level of acceptance of a statement $P_n$ and the degree of unfavorable evidence ($\lambda$), expressing the level of non-acceptance, were defined by responses to each object proposition [statements ($P_1 \ldots P_7$)]. The values were calculated respectively according to the following Relative Strengh Index (RSI) formula in **Figure 7**, for the 2nd step it should begin with the sentence "*RSI function PAL2v modeling Paraconsistent proposition* $P_n$", resulting in a degree of individual evidence from each tester (3rd step).

As seen **Figure 8**, in the 4th step, the Paraconsistent Extractor of Contradiction Effects ($ParaExtr_{ctr}$) Algorithm is composed by connections among PANs. This configuration forms a Paraconsistent network capable to extract the effects of the contradiction, in gradual way of the signals of information, from resulting degree of individual evidence of heuristic of software testability facilitator ($\mu_{er}$).

The degree of evidence $\mu_{erf}$ is a single value corresponding to a tester group (I = 1, 2, 3, 4) opinion for each Bach testability heuristics facilitator. Through successive Paraconsistent analyses, the effects of the contradictions are extracted and presented. As seen **Figure 8**, in the 4th step and Table-A.

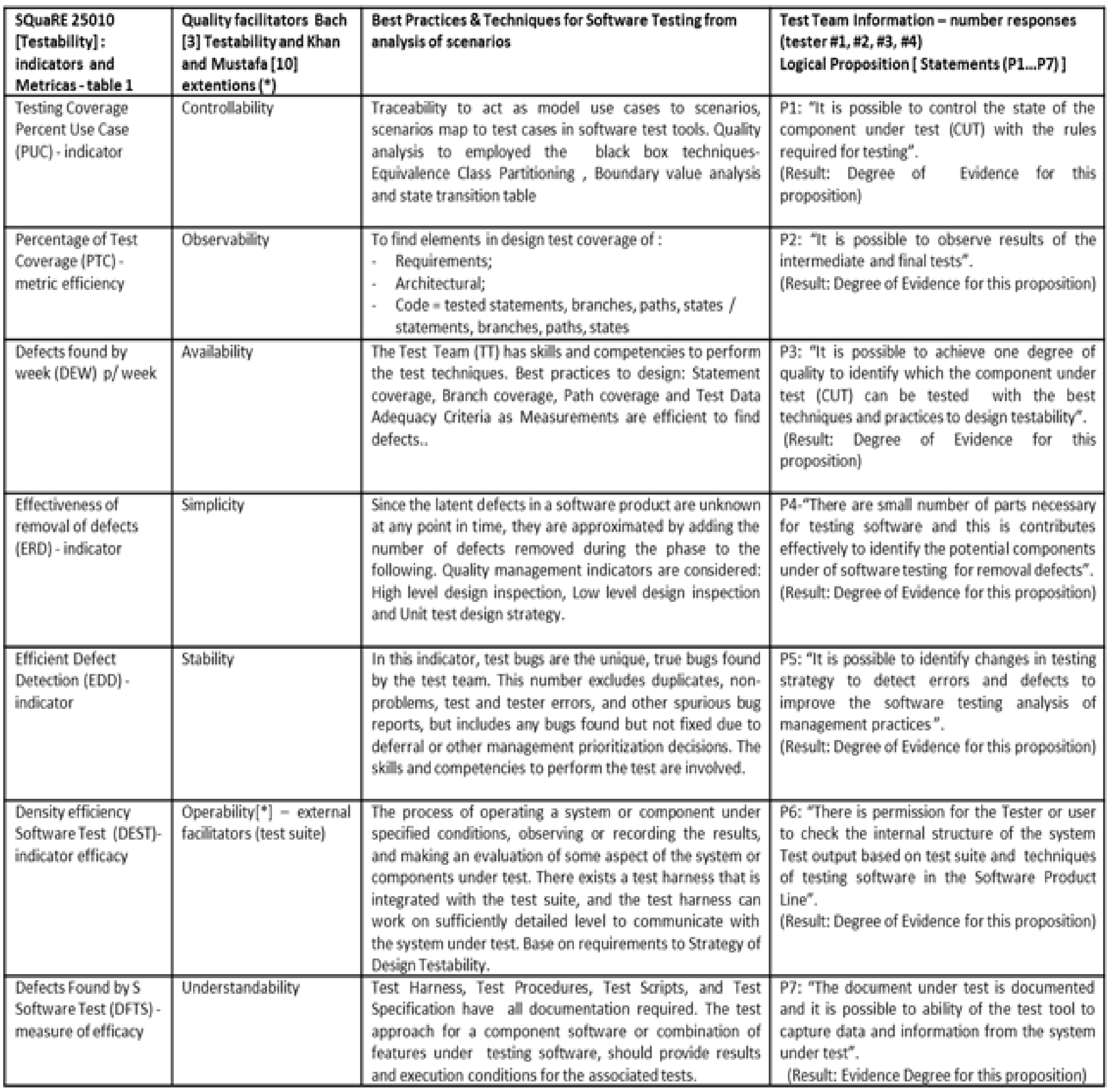
Thus, contradictions ($\mu_{ctr}$) in certain proposition analyses may affect the value of degree of certainty supplied by the analysis of an Object Proposition ($P_1, \ldots, P_7$). This PANNet has a process of normalization where each PAN output, through to $ParaExtr_{ctr}$ Algorithm, represents a Degree of Resultant Real Evidence value $\mu_{erf}$ accompanied by a Resultant Interval of Certainty value φeint , as seen **Figure 8**, in Table-A.

Three basic rules may be considered to make the combinations of the results in Paraconsistent Analysis network:

1) Propositon analyzed in the PANs maybe logically combined through the Degrees of resultant Real Certainty originated from the analysis, and thus make Paraconsistent and thus make different interconnections in the Paraconsistent Analysis Network;

2) The values of the Degrees of Resultant Real Certainty, as well as the intervals of Real certainty originated from the PANs, referring to the different propositions, may be logical treated by conjunction (AND) and disjunction (OR) or algebraically by addition or subtraction of their values, according to the characteristics and topology of the PANNet project;

3) The values of the Degrees of Resultant real certainty may be transformed, by normalization, into values between o and 1, in the real number interval, and thus considered as Degrees of Evidence of other propositions which are being analyzed by other different PANs. In this way, the interconnections among the PANs, will be donethrough the analysis of the evidences.



| SQuaRE 25010 [Testability]: indicators and Metricas - table 1 | Quality facilitators Bach [3] Testability and Khan and Mustafa [10] extentions (*) | Best Practices & Techniques for Software Testing from analysis of scenarios | Test Team Information – number responses (tester #1, #2, #3, #4) Logical Proposition [ Statements (P1...P7) ] |
|---|---|---|---|
| Testing Coverage Percent Use Case (PUC) - indicator | Controllability | Traceability to act as model use cases to scenarios, scenarios map to test cases in software test tools. Quality analysis to employed the black box techniques- Equivalence Class Partitioning , Boundary value analysis and state transition table | P1: "It is possible to control the state of the component under test (CUT) with the rules required for testing". (Result: Degree of Evidence for this proposition) |
| Percentage of Test Coverage (PTC) - metric efficiency | Observability | To find elements in design test coverage of : - Requirements; - Architectural; - Code = tested statements, branches, paths, states / statements, branches, paths, states | P2: "It is possible to observe results of the intermediate and final tests". (Result: Degree of Evidence for this proposition) |
| Defects found by week (DEW) p/ week | Availability | The Test Team (TT) has skills and competencies to perform the test techniques. Best practices to design: Statement coverage, Branch coverage, Path coverage and Test Data Adequacy Criteria as Measurements are efficient to find defects.. | P3: "It is possible to achieve one degree of quality to identify which the component under test (CUT) can be tested with the best techniques and practices to design testability". (Result: Degree of Evidence for this proposition) |
| Effectiveness of removal of defects (ERD) - indicator | Simplicity | Since the latent defects in a software product are unknown at any point in time, they are approximated by adding the number of defects removed during the phase to the following. Quality management indicators are considered: High level design inspection, Low level design inspection and Unit test design strategy. | P4-"There are small number of parts necessary for testing software and this is contributes effectively to identify the potential components under of software testing for removal defects". (Result: Degree of Evidence for this proposition) |
| Efficient Defect Detection (EDD) - indicator | Stability | In this indicator, test bugs are the unique, true bugs found by the test team. This number excludes duplicates, non-problems, test and tester errors, and other spurious bug reports, but includes any bugs found but not fixed due to deferral or other management prioritization decisions. The skills and competencies to perform the test are involved. | P5: "It is possible to identify changes in testing strategy to detect errors and defects to improve the software testing analysis of management practices". (Result: Degree of Evidence for this proposition) |
| Density efficiency Software Test (DEST)- indicator efficacy | Operability[*] = external facilitators (test suite) | The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or components under test. There exists a test harness that is integrated with the test suite, and the test harness can work on sufficiently detailed level to communicate with the system under test. Base on requirements to Strategy of Design Testability. | P6: "There is permission for the Tester or user to check the internal structure of the system Test output based on test suite and techniques of testing software in the Software Product Line". (Result: Degree of Evidence for this proposition) |
| Defects Found by S Software Test (DFTS) - measure of efficacy | Understandability | Test Harness, Test Procedures, Test Scripts, and Test Specification have all documentation required. The test approach for a component software or combination of features under testing software, should provide results and execution conditions for the associated tests. | P7: "The document under test is documented and it is possible to ability of the test tool to capture data and information from the system under test". (Result: Evidence Degree for this proposition) |

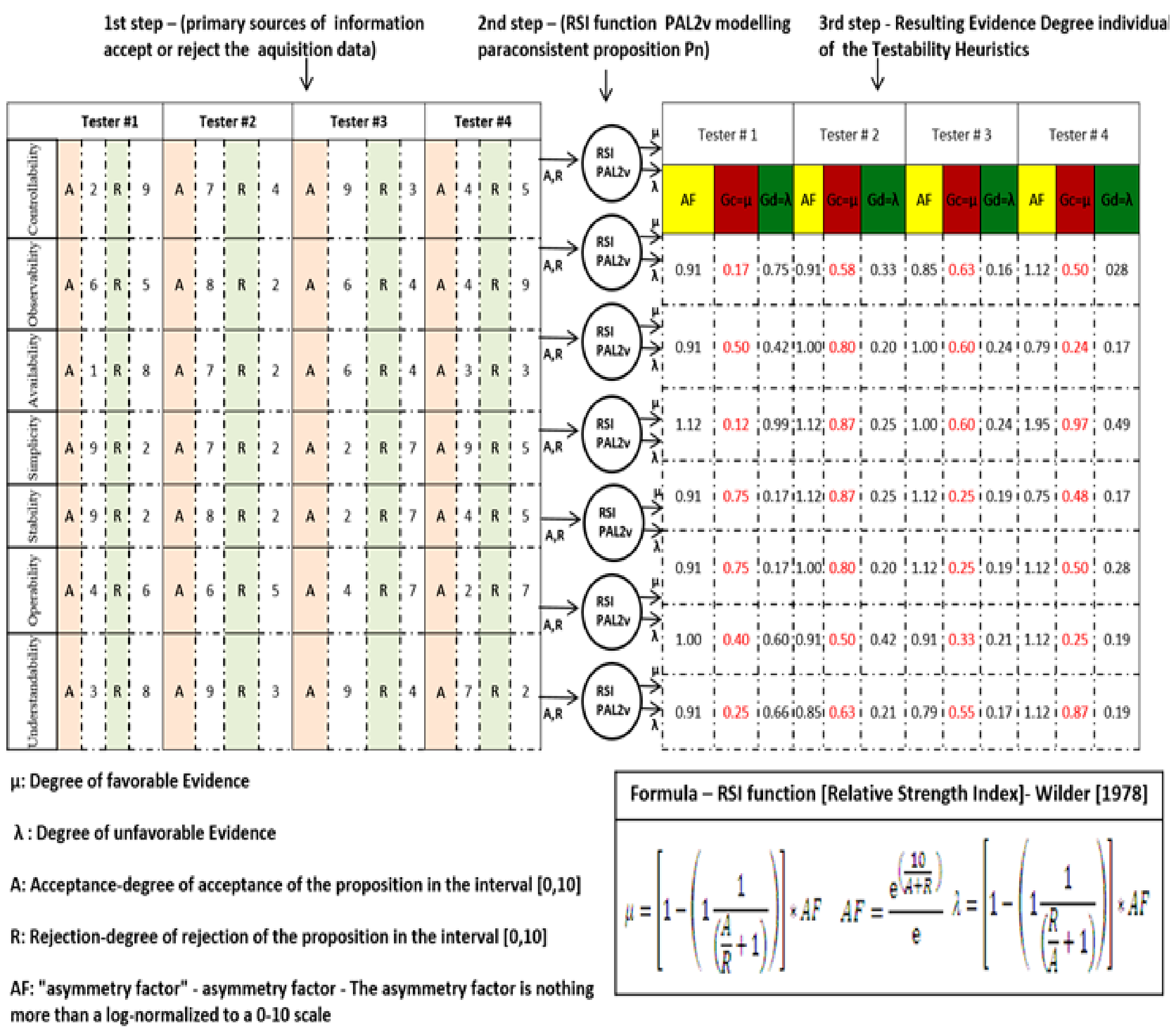
**Figure 6.** Bach's testability heuristics associated to a SQuaRE 25010 with the acquisition data for the results of the degree of acceptance or rejection for each proposition.

The relations between the resulting interval of certainty value $\varphi_e$ and normalized degree of contradiction $\mu_{ctr}$ value show how the level of contradiction in the analysis of PANNet. When taken together, they will promote evidence to make a decision that results in higher accuracy and reliability, as seen in **Table 2**.

It was verified from the characteristic line on the graph [$\varphi_{e \text{ versus}}$ $\mu_{ctr}$] from data in **Table 2**: that when the resultant interval of evidence is maximum value $\varphi_e = 1$ and the normalized degree of contradiction value $\mu_{ctr\,=}\,0.5$, represents a null contradiction.

Maximum contradictions, represented by a value of null resultant Interval of evidence, occur in two values of degree of contradiction $\mu_{ctr}$: 1) when the normalized degree of contradiction is equal to zero, indicating maximum contradiction by indetermination ($\perp$), and 2) when the normalized degree of contradiction is maximum, and indicating maximum contradiction by inconsistency ($\top$).

In this situation there will be two conditions: 1) if the unfavorable evidence $\lambda$ is greater than the favorable evidence $\mu$, the resultant interval of evidence $\varphi_e$ may present values lower than 0.5 and greater or equal to zero,


1st step – (primary sources of information accept or reject the aquisition data)
2nd step – (RSI function PAL2v modelling paraconsistent proposition Pn)
3rd step - Resulting Evidence Degree individual of the Testability Heuristics
A,R
RSI PAL2v
μ
λ

| | Tester #1 | | | | Tester #2 | | | | Tester #3 | | | | Tester #4 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Controllability | A | 2 | R | 9 | A | 7 | R | 4 | A | 9 | R | 3 | A | 4 | R | 5 |
| Observability | A | 6 | R | 5 | A | 8 | R | 2 | A | 6 | R | 4 | A | 4 | R | 9 |
| Availability | A | 1 | R | 8 | A | 7 | R | 2 | A | 6 | R | 4 | A | 3 | R | 3 |
| Simplicity | A | 9 | R | 2 | A | 7 | R | 2 | A | 2 | R | 7 | A | 9 | R | 5 |
| Stability | A | 9 | R | 2 | A | 8 | R | 2 | A | 2 | R | 7 | A | 4 | R | 5 |
| Operability | A | 4 | R | 6 | A | 6 | R | 5 | A | 4 | R | 7 | A | 2 | R | 7 |
| Understandability | A | 3 | R | 8 | A | 9 | R | 3 | A | 9 | R | 4 | A | 7 | R | 2 |

| Tester # 1 | | | Tester # 2 | | | Tester # 3 | | | Tester # 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| AF | Gc=μ | Gd=λ | AF | Gc=μ | Gd=λ | AF | Gc=μ | Gd=λ | AF | Gc=μ | Gd=λ |
| 0.91 | 0.17 | 0.75 | 0.91 | 0.58 | 0.33 | 0.85 | 0.63 | 0.16 | 1.12 | 0.50 | 028 |
| 0.91 | 0.50 | 0.42 | 1.00 | 0.80 | 0.20 | 1.00 | 0.60 | 0.24 | 0.79 | 0.24 | 0.17 |
| 1.12 | 0.12 | 0.99 | 1.12 | 0.87 | 0.25 | 1.00 | 0.60 | 0.24 | 1.95 | 0.97 | 0.49 |
| 0.91 | 0.75 | 0.17 | 1.12 | 0.87 | 0.25 | 1.12 | 0.25 | 0.19 | 0.75 | 0.48 | 0.17 |
| 0.91 | 0.75 | 0.17 | 1.00 | 0.80 | 0.20 | 1.12 | 0.25 | 0.19 | 1.12 | 0.50 | 0.28 |
| 1.00 | 0.40 | 0.60 | 0.91 | 0.50 | 0.42 | 0.91 | 0.33 | 0.21 | 1.12 | 0.25 | 0.19 |
| 0.91 | 0.25 | 0.66 | 0.85 | 0.63 | 0.21 | 0.79 | 0.55 | 0.17 | 1.12 | 0.87 | 0.19 |

μ: Degree of favorable Evidence

λ : Degree of unfavorable Evidence

A: Acceptance-degree of acceptance of the proposition in the interval [0,10]

R: Rejection-degree of rejection of the proposition in the interval [0,10]

AF: "asymmetry factor" - asymmetry factor - The asymmetry factor is nothing more than a log-normalized to a 0-10 scale

Formula – RSI function [Relative Strength Index]- Wilder [1978]

$$\mu=\left[1-\left(1\frac{1}{\left(\frac{A}{R}+1\right)}\right)\right]*AF \quad AF=\frac{e^{\left(\frac{10}{A+R}\right)}}{e} \quad \lambda=\left[1-\left(1\frac{1}{\left(\frac{R}{A}+1\right)}\right)\right]*AF$$

**Figure 7.** The RSI oscillator function modelling degree of evidence the PAL2v lattice[1].

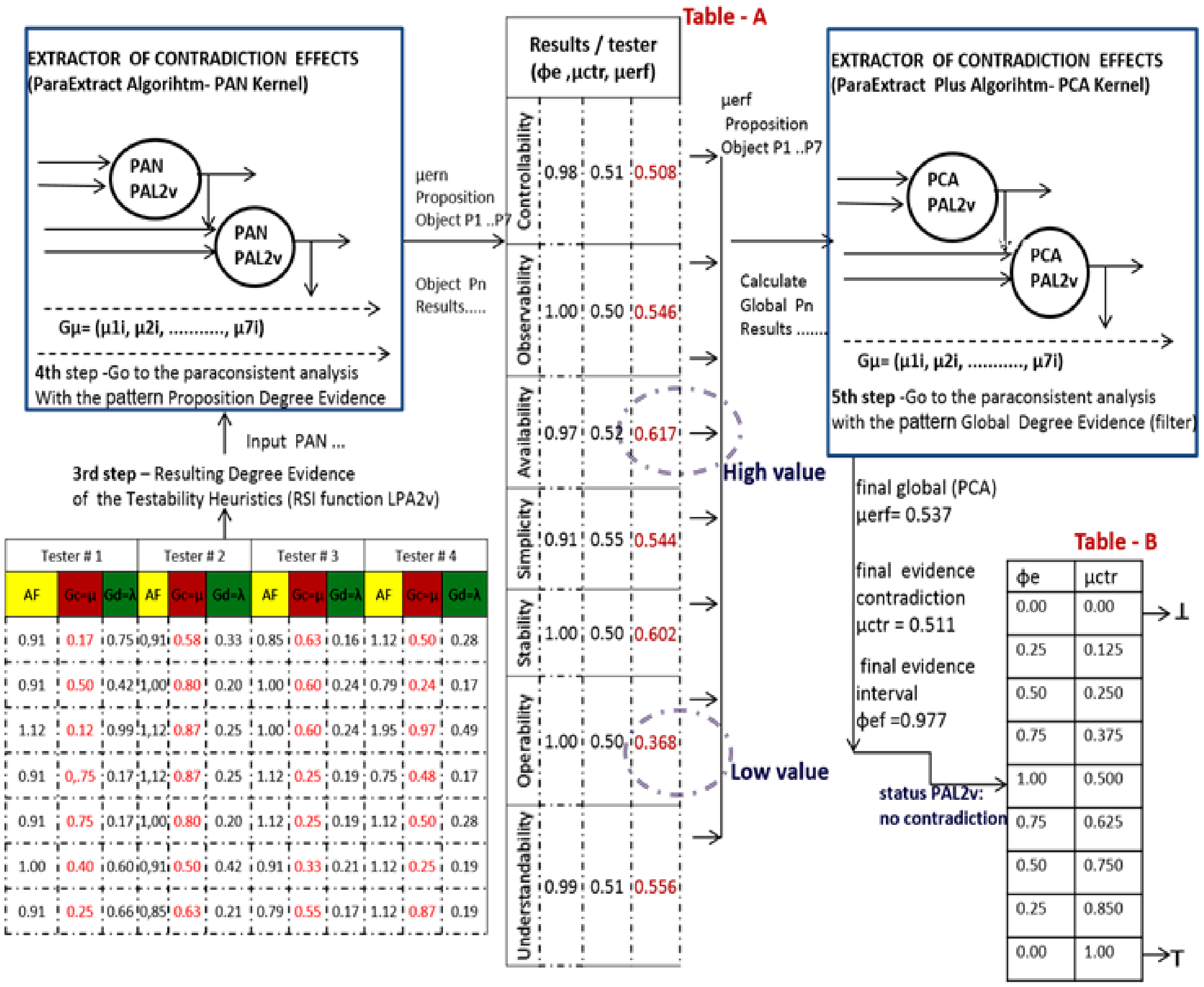
**Table 2.** Contradiction Analysis—Lattice PAL2v.

| Resultant Interval of Evidence ($\varphi_e$) | Normalized degree of Contradiction ($\mu_{ctr}$) | Analysis |
|---|---|---|
| 0.00 | 0.000 | Indetermination ⊥ |
| 0.25 | 0.125 | boundary limit ⊥ |
| 0.50 | 0.250 | Data valid |
| 0.75 | 0.375 | Data valid |
| 1.00 | 0.500 | Null contradiction |
| 0.75 | 0.625 | Data valid |
| 0.50 | 0.750 | Data valid |
| 0.25 | 0.875 | boundary limit T |
| 0.00 | 1.000 | Inconsistency T |

and 2) if the favorable evidence μ is greater than the unfavorable evidence λ the resultant interval of evidence $\varphi_e$ may present values greater than 0.5 and lower or equal to zero.

The **Table 3** shows the characteristics found of the resultant interval of evidence $\varphi_e$ related with the values of

[1]This formula was designed by Welles Wilder [1978], the RSI (Relative Strength Index) is an oscillator that allows to compare the average magnitude of gains from the average of the losses on a scale interval.


EXTRACTOR OF CONTRADICTION EFFECTS
(ParaExtract Algorihtm- PAN Kernel)
PAN PAL2v
PAN PAL2v
Gμ= (μ1i, μ2i, ..........., μ7i)
4th step -Go to the paraconsistent analysis
With the pattern Proposition Degree Evidence
Input PAN ...
3rd step – Resulting Degree Evidence
of the Testability Heuristics (RSI function LPA2v)
μern
Proposition
Object P1 ..P7
Object Pn
Results.....
μerf
Proposition
Object P1 ..P7
Calculate
Global Pn
Results .......
High value
Low value
EXTRACTOR OF CONTRADICTION EFFECTS
(ParaExtract Plus Algorihtm- PCA Kernel)
PCA PAL2v
PCA PAL2v
Gμ= (μ1i, μ2i, ..........., μ7i)
5th step -Go to the paraconsistent analysis
with the pattern Global Degree Evidence (filter)
final global (PCA)
μerf= 0.537
final evidence
contradiction
μctr = 0.511
final evidence
interval
ϕef =0.977
status PAL2v:
no contradiction
→⊥
→T

| Tester # 1 | | | Tester # 2 | | | Tester # 3 | | | Tester # 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| AF | Gc=μ | Gd=λ | AF | Gc=μ | Gd=λ | AF | Gc=μ | Gd=λ | AF | Gc=μ | Gd=λ |
| 0.91 | 0.17 | 0.75 | 0,91 | 0.58 | 0.33 | 0.85 | 0.63 | 0.16 | 1.12 | 0.50 | 0.28 |
| 0.91 | 0.50 | 0.42 | 1,00 | 0.80 | 0.20 | 1.00 | 0.60 | 0.24 | 0.79 | 0.24 | 0.17 |
| 1.12 | 0.12 | 0.99 | 1,12 | 0.87 | 0.25 | 1.00 | 0.60 | 0.24 | 1.95 | 0.97 | 0.49 |
| 0.91 | 0,.75 | 0.17 | 1,12 | 0.87 | 0.25 | 1.12 | 0.25 | 0.19 | 0.75 | 0.48 | 0.17 |
| 0.91 | 0.75 | 0.17 | 1,00 | 0.80 | 0.20 | 1.12 | 0.25 | 0.19 | 1.12 | 0.50 | 0.28 |
| 1.00 | 0.40 | 0.60 | 0,91 | 0.50 | 0.42 | 0.91 | 0.33 | 0.21 | 1.12 | 0.25 | 0.19 |
| 0.91 | 0.25 | 0.66 | 0,85 | 0.63 | 0.21 | 0.79 | 0.55 | 0.17 | 1.12 | 0.87 | 0.19 |

Table - A

| Results / tester (ϕe ,μctr, μerf) | | | |
|---|---|---|---|
| Controllability | 0.98 | 0.51 | 0.508 |
| Observability | 1.00 | 0.50 | 0.546 |
| Availability | 0.97 | 0.52 | 0.617 |
| Simplicity | 0.91 | 0.55 | 0.544 |
| Stability | 1.00 | 0.50 | 0.602 |
| Operability | 1.00 | 0.50 | 0.368 |
| Understandability | 0.99 | 0.51 | 0.556 |

Table - B

| ϕe | μctr |
|---|---|
| 0.00 | 0.00 |
| 0.25 | 0.125 |
| 0.50 | 0.250 |
| 0.75 | 0.375 |
| 1.00 | 0.500 |
| 0.75 | 0.625 |
| 0.50 | 0.750 |
| 0.25 | 0.850 |
| 0.00 | 1.00 |

**Figure 8.** Diagram of *ParaExtr*$_{ctr}$ and of *ParaExtr*$_{ctrPlus}$ algorithms from the acquisition data referring of heuristics of software testability.

resultant degree of real evidence $\mu_{erf}$.

In the characteristic segment line on the graph [$\varphi_e$ versus $\mu_{erf}$] from data in **Table 3** enables to verify that the resultant interval of evidence $\varphi_e$, represents the permitted value for variation of resultant degree of real evidence $\mu_{erf}$, in situation of the normalized degree of contradiction value $\mu_{ctr}$ .

The resultant interval of evidence $\varphi_e$ impacts on the maximum of the output degree of evidence tending to logical state True or False, as seen in **Figure 9**.

In the end of the analysis, the *ParaExtr*$_{ctr}$ algorithm presents as result a real evidence degree values $\mu_{erf}$ representative of the evidence degree group, was validated in agreement, the interval of contradiction analysis, output values from 4th step, according to **Table 2**: between boundary limit indetermination (⊥) and boundary limit inconsistency or paracompleteness (T).

When viewing Table-A of results for each proposition $P_n$ of heuristics of software testability in the 4th step, in **Figure 8**, the small value $\mu_{erf} = 0.38$, refers to the operability facilitator. This indicates a potential deficiency in the training of the technical staff of software testing, compromising the quality of the software product through poor implementation of best practices for testing and lack of operational experience.

Another of Bach's facilitators that showed medium results was controllability, with the value $\mu_{erf} = 0.508$. This means there were problems in the process of operating a system or software components under specified conditions of testing coverage. Some strategies that apply in test case scenarios with black box techniques: Equivalence Class Partitioning, Boundary Value Analysis, and State Transition Table, have shown to be inconsistent and partly failing.

Table 3. Characteristics resultant degree evidence—PAL2v Lattice.

| Resultant Interval of Evidence ($\varphi_e$) | Resultant degree evidence($\mu_{erf}$) | Analysis |
|---|---|---|
| 0.00 | 0.000 | False **F** |
| 0.25 | 0.125 | boundary limit **F** |
| 0.50 | 0.250 | Data valid |
| 0.75 | 0.375 | Data valid |
| 1.00 | 0.500 | Indetermination ⊥ |
| 0.75 | 0.625 | Data valid |
| 0.50 | 0.750 | Data valid |
| 0.25 | 0.875 | boundary limit **t** |
| 0.00 | 1.000 | True **t** |

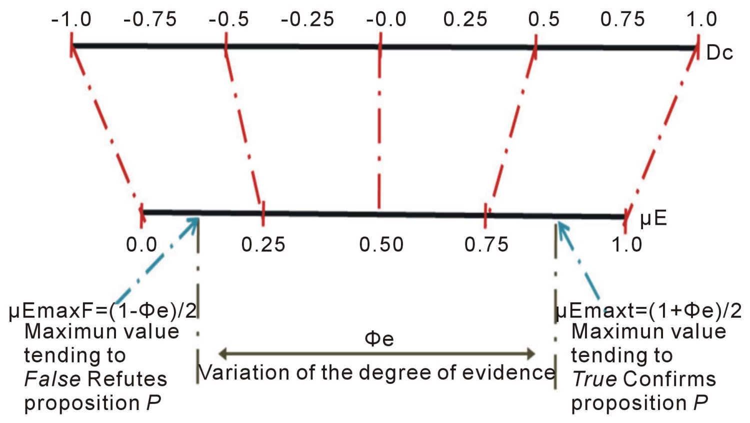

-1.0
-0.75
-0.5
-0.25
-0.0
0.25
0.5
0.75
1.0
Dc
μE
0.0
0.25
0.50
0.75
1.0
μEmaxF=(1-Φe)/2
Maximun value tending to *False* Refutes proposition *P*
Φe
Variation of the degree of evidence
μEmaxt=(1+Φe)/2
Maximun value tending to *True* Confirms proposition *P*

Figure 9. Location Logical states [F, t] base on variation of the degree of evidence.

The stability facilitator, in Figure 8, showed better numbers, it had a high value of $\mu_{erf} = 0.602$ and null contradiction. This facilitator showed a detection of defects with a satisfactory level in each release of the Software Product Line (SPL). It represents the changes in Test Cases (TC) and testing techniques employed by the software test team to find bugs which were assimilated incrementally with efficiency.

The 5th step is "go to the Paraconsistent analysis with the global evidence degree (filter) pattern", uses the $ParaExtr_{ctrPlus}$ algorithm in order to homogenize different facilitators, which is key to achieving an overall proposition concerning an object: $P_0$—"It is possible to matched quantity testability indicators of SQuaRE 25010 with the quality facilitators heuristics of software testability by James Bach, to analyze the impact of design testability using best practices and techniques on test scenarios".

The $ParaExtr_{ctrPlus}$ algorithm is a network for treatment of uncertainties, which can be built with several interlinked PCA-algorithms. The hypothesis of extraction of the effects of the contradiction from several non-homogeneous sources has the principle that: 1) if the first treated signals are the most contradictory, the impact each propagation of resultant Interval of Evidence must be analyzed, subsequently to another PCA kernel; and 2) gradually filters inconsistencies to treat the effects of contradiction in information from each resultant real degree evidence through the PANNet to obtain the influence on global final results.

In the 5th step, the $ParaExtr_{ctrPlus}$ algorithms to produce the global resultant real degree of evidence $\mu_{efr}$ is equal to 0.537, where $\mu_{ctr} = 0.511$ and $\varphi_e$ is positive near to value 1, representing a tendency of null contradiction. As seen in Table B, Figure 8.

The extraction process reduces the effects of the inconsistencies and it presents as answer a closer representative value of the reality, through the signals of contradiction by evidence of their ranges to reduce the partial contradiction $\mu_{erf}$ between the different Bach's testability facilitators.

In each analysis carried out the PAN interlinks of PANNet, PCA or PAN Kernel, the Interval of certainty is available, besides the Resultant Degree of Evidence according to **Table 3**. The two values will allow an identification of the condition for analysis of uncertainty of each proposition in the PAN Nettopology. With the value of Interval of Certainty, there exists an indication of where the system may react. Whether reduce of strengthen the evidences with the objective of diminishing the conflicts and increasing of Degree of Certainty.

The level of demand for the system makes the decision favorable; in other words, the final result $\mu_{efr}$ is "true" and satisfactory for the global proposition object $P_0$. The quality-quantity mapping [modelling] standards of Bach's testability model [3] and SQuaRE 25010 [2], provided an integrated view of complementary mechanisms to improve best practices, methods and techniques results on the diagnostic analysis for designing testability of software adopted by ICAMMH project Test Team.

## 5. Conclusions

This paper described the design characteristics of the PANNet topology carrying out different connections, which provided result as a real evidence degree representative on PAL2v lattice. The results demonstrate the feasibility of the quality-quantity process model for measure software testability.

The $ParaExtr_{ctr}$ algorithms presented are capable of extracting contradiction effects from groups of evidence signals regarding quality-quantity propositions of testability through basic concepts of Paraconsistent Annotated Logic for two values (PAL2v).

The modelling through PANNet topology combining PAN and PCA kernels of $ParaExtr_{ctr}$ algorithms have provided safe information about propositions with a higher or lower degree of contradiction. With this information the system is able to make more reliable decisions, besides having the values to act on the input signal control, weakening or strengthening evidences, in order to reduce contradiction effects.

The $ParaExtr_{ctr}$ and the $ParaExtr_{ctrPlus}$ algorithms have proposed new structures and different configurations of Paraconsistent analyses networks that were designed for treatment of uncertainties for designing software testing strategies. In that context the new approach aims to improve quality assurance for the development of software products.

Testability is a key factor in the quality of the software product, to propose new approaches to verifying and validating the quality for design strategies for software, during the development cycle.

This paper demonstrated an effective improvement in testability analysis and software because with the use of PAL2v algorithms, it presents a new way to treat the effects of gradually contradiction in information, also provides monitoring conditions, and adjustments in steering of the process.

The results obtained in this paper bring contributions to build robust Paraconsistent Specialists Systems, to act as powerful computational tools dedicated to software testability analysis without contamination by contradictory information signals.

## Acknowledgements

The Authors would like to thank the Brazilian Aeronautics Institute of Technology (ITA), for its scientific development incentives by using Research & Development Projects at ITA; the Brazilian National Water Agency (ANA), for the opportunity of participating in its Project; the Finance Agency for Studies and Projects (FINEP); and also the Casimiro Montenegro Filho Foundation (FCMF) for its available infrastructure.

## References

[1] da Silva, G.B., Romano, B.L., Campos, H.C., Vieira R.G., da Cunha A.M. and Vieira Dias, L.A. (2009) Integrating Amazonic Heterogeneous Hydrometeorological Databases. *Information Technology*: *New Generations, Third International Conference*, 119-124.

[2] ISO/IEC 25010 (2011) Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733

[3] Bach, J. (2003) Heuristics of Software Testability. http://www.satisfice.com/tools/testable.pdf

[4] Clements, P.C. and Northrop, L. (2001) Software Product Lines: Practices and Patterns. SEI Series in Software Engineering, Addison-Wesley.

[5] Northrop, L.M., Clements, P.C., Bachmann, F., Bergey, J., Chastek, G., Cohen, S., Donohoe, P., Jones, L., Krut, R.,

Little, R., McGregor, J. and O'Brien, L. (2009) A Framework for Software Product Line Practice, Version 5.0.

[6] IEEE Std 829 (1998) IEEE Standard for Software and System Test Documentation. 2008. Institute of Electrical and Electronics Engineers. Revision of IEEE Std 829.

[7] Perry, W. (2006) Effective Methods for Software Testing. 3rd Edition, Wiley, New York.

[8] Sharma, A., Kumar, V. and Pachori, S. (2012) Defect Prevention Technique used in Test Case for Quality Improvement. *International Journal of Computer Applications*, **43**, 17-21.

[9] Badri, M. and Toure, F. (2012) Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes. *Journal of Software Engineering and Applications*, **5**, 513-526. http://dx.doi.org/10.4236/jsea.2012.57060

[10] Rosenberg, D. and Sthephens, M. (2007) Use Case Driven Object Modeling with UML: Theory and Practice. Springer-Verlag, New York, 50-51.

[11] Khan, R.A. and Mustafa, K. (2009) Software Testability. Department of Computer Science, New Delhi. http://developeriq.in/articles/2009/jan/02/software-testability/

[12] Arruda, A.I. (1980) A survey of Paraconsistent Logic, In: Arruda, A.I., Chuaqui, R. and Da Costa, N.C.A., Eds., *Studies in Logic and the Foundations of Mathematics* (Vol. 99), Elsevier, 1-41.

[13] Blair, H.A. and Subrahmanian, V.S. (1988) Paraconsistent Foundations for Logic Programming. *Journal of NonClassical Logic*, **5**, 45-73.

[14] Subrahmanian, V.S. (1987) On the Semantics of Quantitative Logic Programming. *Proceedings of* 4*th IEEE Symposium on Logic Programming*, September, San Francisco, 173-182.

[15] Da Costa, N.C.A. (1974) On the Theory of Inconsistent Formal Systems. *Notre Dame Journal of Formal Logic*, **15**, 497-510. http://dx.doi.org/10.1305/ndjfl/1093891487

[16] Da Costa N. C. A. and Marconi D. (1989) An Overview of Paraconsistent Logic in the 80's. *The Journal of Non-Classical Logic*, **6**, 5-31.

[17] Da Costa, N.C.A., Subrahmanian, V.S. and Vago, C. (1991) The Paraconsistent Logic PT. *Zeitschrift fur MathematischeLogik und Grundlagen der Mathematik*, **37**, 139-148. http://dx.doi.org/10.1002/malq.19910370903

[18] Subrahmanian, V.S. (1990) Mechanical Proof Procedures for Many Valued Lattice-Based Logic Programming. *Journal of Non-Classical Logic*, **7**, 7-41.

[19] Da Costa, N.C.A., Abe, J.M. and Subrahmanian, V.S. (1991) Remarks on Annotated Logic. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, **37**, 561-570.

[20] Da Costa, N.C.A., Da Silva Filho, J.I., Abe, J.M., Murolo, F.F. and Soares, C.F. (1999) Lógica Paraconsistente Aplicada. Editora, Atlas.

[21] Da Silva Filho, J.I., Abe, J.M. and Lambert-Torres, G. (2009) Inteligência Artificial com as Redes de Análises Paraconsistentes Teoria e Aplicações. LTC, Editora, 328 p.

[22] Da Silva Filho, J.I., Abe, J.M. and Lambert-Torres, G. (2010) Uncertainty Treatment Using Paraconsistent Logic— Introducing Paraconsistent Artificial Neural Networks (Vol. 21). IOS Press, Amsterdam, 328 p.

[23] Da Silva Filho, J.I., Lambert-Torres, G., Ferrara, L.F.P., Mário, M.C., Dos Santos, M.R., Onuki, A.S., Camargo, J.M. and Rocco, A. (2011) "Paraconsistent Algorithm Extractor of Contradiction Effects—ParaExtrctr. *Journal Software Engineering & Applications*, **4**, 579-584. http://dx.doi.org/10.4236/jsea.2011.410067

[24] Da Silva Filho, J.I., Onuki, A.S., Ferrara, L.F.P., Mário, M.C., Camargo, J.M., Garcia, D.V., Dos Santos, M.R. and Rocco, A. (2012) Electric Power System Operation Decision Support by Expert System Built with Paraconsistent Annotated Logic, Advances in Expert Systems. InTech. http://dx.doi.org/10.5772/51379

[25] Elfriede, D. (2002) Effective Software Testing: 50 Ways to Improve Your Software Testing. Addison-Wesley Longman Publishing Co., Inc., Boston.