Classification of Defect Types in Requirements Specifications: Literature Review, Proposal and Assessment

Isabel Lopes Margarido*, João Pascoal Faria, Raul Moreira Vidal
Department of Informatics Engineering, Faculty of Engineering of the University of Porto
Porto, Portugal
{isabel.margarido, jpfaria, rmvidal@fe.up.pt

Marco Vieira
Department of Informatics Engineering, Faculty of Sciences, University of Coimbra
Coimbra, Portugal
mvieira@dei.uc.p

Abstract— **Requirements defects have a major impact throughout the whole software lifecycle. Having a specific defects classification for requirements is important to analyse the root causes of problems, build checklists that support requirements reviews and to reduce risks associated with requirements problems. In our research we analyse several defects classifiers; select the ones applicable to requirements specifications, following rules to build defects taxonomies; and assess the classification validity in an experiment of requirements defects classification performed by graduate and undergraduate students. Not all subjects used the same type of defect to classify the same defect, which suggests that defects classification is not consensual. Considering our results we give recommendations to industry and other researchers on the design of classification schemes and treatment of classification results.**

Keywords- software; requirements; defects; classification; taxonomy

I. INTRODUCTION

In this paper we consider that a defect is a fault, as defined in [1], extended to include all the software development artefacts (code, documentation, requirements, etc.). A defect is a problem that occurs in an artefact and may lead to a failure. We consider the requirements review as an inspection method.

In 2009, Chen and Huang analysed the impact of software development defects on software maintainability, and concluded that several documentation and requirements problems are amongst the top 10 higher-severity problems (see table I) [2]. The authors demonstrated the impact of the software requirements defects in the maintenance phase of a software project, when the defects affect the client, in case of failure. In the same year Hamill and Goseva-Popstojanova showed that requirements defects are among the most common types of defects in software development and that the major sources of failures are defects in requirements (32.65%) and code (32.58%) [3]. Therefore it is crucial to impede the propagation of requirements defects to posterior phases.

Card stated in 1998 that "Classifying or grouping problems helps to identify clusters in which systematic errors are likely to be found. [4]" Hence, it is important to have an adequate taxonomy to classify requirements defects, that support the following goals: **(1)** identify types of defects that are more frequent or have a higher cost impact; **(2)** analyse the root cause of requirements defects; **(3)** prepare requirements reviews checklists; **(4)** reduce risks associated with common problems in the requirements management process, such as bad communication, incomplete requirements, and final acceptance difficulties.

The Orthogonal Defect Classification (ODC) is frequently used by practitioners, but it is more adequate to classify code defects than defects in the requirements specifications [5-6].

There are several classifications identified in the literature, but none of them is indicated as being the most adequate for the classification of requirements defects, and, to the best of our knowledge, their quality properties were not validated. In our research we do a literature review and propose values for the attribute **type of defect** in the case of requirements using the recommendations of Fermut *et al* [6]. We conducted an experiment to validate the quality properties of the proposal and test the following hypotheses, when reviewing requirements specifications: null hypothesis (H_0) **all subjects use the same value to classify the type of a defect**; the alternative hypothesis (H_1) **not all subjects use the same value to classify the type of a defect**. Our results demonstrate that there is no guarantee that all subjects use the same value to indicate the type of a defect. Considering such results we give recommendations to industry and other researchers on the design of classification schemes and treatment of classification results.

The following sections contain: Section II – a literature review about defects classification, particularly the ones applicable to requirements; Section III – the assembly of the requirements defects classification list; Section IV – the validation of the classification list and results analysis; and finally, conclusions and future research are given in section V.

II. LITERATURE REVIEW

In 2009, Chen and Huang performed an e-mail survey with several software projects, and presented the top 10 higher-severity problem factors affecting software maintainability, as summarised in table I [2].

The authors indicated the following causes of software defects [2]: **(1)** a significant percentage of defects is caused by

Work partially funded by Fundação para a Ciência e a Tecnologia (FCT): Programa Operacional Potencial Humano (POPH) of QREN, and Fundo Social Europeu (FSE)

incorrect specifications and translation of requirements, or incomplete ones [7-8]; **(2)** half of the problems rooted in requirements are due to ambiguous, poorly written, unclear and incorrect requirements, the other half result of omitted requirements [9]. In 2003, Lutz and Mikulski analysed the impact and causes of requirements defects discovered in the testing phase, resulting from non documented changes or defects in the requirements, and proposed guidelines to distinguish and respond to each situation [10]. Their work emphasises the importance of requirements management.

TABLE I. TOP 10 HIGHER-SEVERITY PROBLEM FACTORS [2]

#	Software Development Factors	Problem Dimension
1	Inadequacy of source code comments	Programming Quality
2	Documentation obscure/untrustworthy	Documentation Quality
3	Changes not adequately documented	Documentation Quality
4	Lack of traceability	Documentation Quality
5	Lack of adherence to standards	Programming Quality
6	Lack of integrity/consistency	Documentation Quality
7	Continually changing requirements	System Requirements
8	Frequent turnover within the project team	Personnel Resources
9	Improper usage of techniques	Programming Quality
10	Lack of consideration for software quality requirements	System Requirements

Considering the problems that occur in the requirements specifications we present in the following subsections work that is related with or includes a requirements defects classification.

A. *Code Defects Classifications, 1992*

ODC is applicable in all the development phases except the requirements phase. The defect types used are: **function**, **interface**, **checking**, **assignment**, **timing/serialisation**, **build/package/merge**, **documentation** and **algorithm**. For each defect it is necessary to indicate if the feature is incorrect or missing [11]. Such classifiers do not seem completely adequate to classify requirements defects, and **Documentation** is too generic to give further information on the defect. The Hewlett-Packard (HP) [12] categorises the defects by **mode**, **type** and **origin**, (see figure 1) [6]. From the types of defects with origin in the **requirements/specifications phase**, the **requirements/ specifications** seems to be vague and the interfaces ones are too detailed and more adequate to design specification defects.

B. *Quality Based Classifiers, 1976 – 2010*

In this section we present the work of several authors that applied quality based classifiers to requirements defects.

In 1976, Bell and Thayer did a research to verify the impact of software requirements defects. Not surprisingly, they concluded that software systems meeting defective requirements will not effectively solve basic needs [13]. They aggregated the defects in categories, as presented in table III (Annex A). In 1981, Basili and Weiss categorised defects found in requirements documents and gathered a set of questions to be asked while reviewing them (as a review checklist) [14]. Table III shows the distribution of the 79 errors by different categories. Later, in 1989, Ackerman *et al* analysed the effectiveness of software inspections as a verification process [15]. They presented a sample requirements checklist to use in inspections of requirements documents, containing questions organised by defect categories: **completeness**, **consistency** and **ambiguity**. And in 1991, Sakthivel performed a survey about requirement verification techniques and presented a requirements defects taxonomy based on a literature review. The classes that the author proposed are: **incomplete**, **inconsistent**, **infeasible**, **untestable**, **redundant** and **incorrect**. For each class, Sakthivel presented different defects and an example [16].

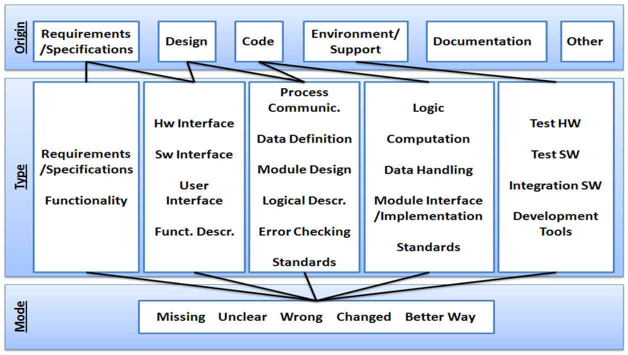

Figure 1. HP defects classification scheme [6].

In 2003, Hayes developed a requirements fault taxonomy for NASA's critical/catastrophic high-risk systems. Hayes stated that ODC refers to design and code while their approach emphasised requirements, so they adapted the Nuclear Regulatory Commission (NRC) requirement fault taxonomy from NUREG/CR-6316 (1995). [17] Afterwards, in 2006, Hayes *et al* analysed a software product related with the previous to build a common cause tree [18]. In both works **unachievable** was reserved for future. In 2006, the same was also done with **infeasible** and **non verifiable** (Table III shows their results).

Defects classification is important to support the analysis of the root causes of defects. In 2010, Kalinowski *et al* were aware that Defect Causal Analysis (DCA) could reduce defect rates by over 50%, reducing rework, and improving quality and performance [19]. To enhance DCA, they improved their framework named Defect Prevention Based Process Improvement (DPPI) used to conduct, measure and control DCA. The authors mentioned the necessity of collecting metrics for DCA and the importance of considering: **(1)** context when collecting metrics; **(2)** stability of the inspection; **(3)** technology/similarity of projects in inspections. When demonstrating their approach they reported the requirements defects distribution, classified by nature (see table III).

C. *Functional and Quality Based Classifiers, 1992 – 2009*

In this section we present defect classification taxonomies that are functional and quality based. In our research we consider that the functional classifiers represent the function of the requirement in the product (e.g. interface, performance, environment, functional).

In 1992, Schneider *et al* identified two classes of requirements defects to use when reviewing user requirements documents: **Missing Information** and **Wrong Information** (table III) [20]. In 1995, Porter *et al* compared requirements inspection methods. They performed an experiment where two Software Requirements Specification (SRS) documents were inspected with a combination of ad hoc, checklist and scenario inspection methods. The checklist was organised in categories, resembling a defect classification: **omission** (**missing functionality**, **performance**, **environment** or **interface**) and **commission (ambiguous** or **inconsistent information**, **incorrect** or **extra functionality**, **wrong section**). The scenarios also included categories: **data type consistency**, **incorrect functionality**, **ambiguity**, and **missing functionality**. The authors concluded from their results that the scenario inspection method was the most effective for requirements [21]. Later, in 2007, Walia *et al* repeated an experiment to show the importance of requirements defects taxonomy. They involved software engineering students in a SRS document review using a defect checklist. The students repeated the review, after being trained in the error abstraction process. The results of the experiment showed that error abstraction leads to more defects found without losses of efficiency and the abstraction is harder when people are not involved in the elaboration of the SRS and have no contact with developers. Requirements defects were classified as: **general**, **missing functionality**, **missing performance**, **missing interface**, **missing environment**, **ambiguous information**, **inconsistent information**, **incorrect or extra functionality**, **wrong section**, **other faults** [22]. This experiment was applied to error abstraction; we consider that a similar experiment is useful to validate defects classification.

Along the years researchers introduced classifiers to fulfil the specificities of requirements defects. Some reused existent classifications and conducted experiments to analyse the impact of different methodologies in SRS inspections. Table III summarises the relation between authors and classifiers.

III. Proposal of a Classification of Defect Types in Requirements Specifications

In 2005, Freimut *et al* [6] indicate the quality properties of a good classification scheme: **1. clearly** and **meaningfully define** the **attributes** of the classification scheme; **2. clearly define** the **values** of the attributes; **3.** ensure it is **complete** (every defect is classifiable by using the scheme); **4.** guarantee that it contains a **small number of attribute values -** the authors recommend 5 to 9 attributes, since this is the number of items that human short-memory can retain [11]; **5. aggregate attribute values**, to reduce ambiguity [13], whenever they are less significant, i.e. when they rarely occur, and detailed categories may be aggregated into a single one. For the attribute "type of defect" we consider that it is important that the values are **unambiguous**, i.e. only one value is applicable to one defect. Considering these recommendations we assembled a list of values for the attribute type of defect.

From the literature review, presented in section 2, we collected several different taxonomies and the frequency of the defects classifiers of the researchers' experiences (see table III). We analysed the frequency with which each classifier was used and its adequacy to classify a requirement defect.

The following classifiers were excluded for the indicated reasons:

- Considered important only for change management: **Not in current baseline**, **New** and **Changed Requirement** and **Not Traceable**;
- Too vague (given the intention of having a complete and clearly defined list of values): **General, Other** and **Inadequate**;
- Subsumed by another (**Inconsistent**): **Incompatible**;
- Too generic (given the existence of separate, more specific, classifiers): **Incorrect or Extra Functionality;**
- Over-detailed (given the existence of the more generic classifiers **Missing/Omission**, **Incorrect** and **Inconsistent,** and the intention of keeping a small number of attribute values): classifiers 19 to 33 and 35 in Table III detailing what is missing, incorrect or inconsistent (the details can be given in the defect description).

The following classifiers with overlapping meanings (and small frequencies in some cases) were aggregated into a single one, to avoid ambiguity:

- **Missing/Omission** and **Incomplete** $\rightarrow$ **Missing or Incomplete**;
- **Over-specification**, **Out of scope**, **Intentional Deviation** and **Extraneous Information** $\rightarrow$ **Not Relevant or Extraneous**;
- **Unclear** and **Ambiguity** $\rightarrow$ **Ambiguous or Unclear**;
- **Infeasible**, **Unachievable**, **Non Verifiable** and **Unstestable/Non Verifiable** $\rightarrow$ **Infeasible or Non-verifiable**.

Finally, some classifiers were slightly renamed.

The resulting 9 values for the type of defect attribute, with definitions and examples, are listed in Table II. We tried to give a clear and meaningful definition for each value.

IV. Practical Application and Results

In this section we present two experiments to verify the properties of our classifier against the recommendations of Fermut *et al* [6]. Formalising the hypothesis H, when reviewing requirements specifications: **H_0 - all subjects use the same value to classify the type of a defect**; **H_1 - not all subjects use the same value to classify the type of a defect**.

We conducted two experiments with different groups of people and similar classifiers. The final list (table II) used in the 2nd group had more detail in the values, definitions and examples. The 1st group was composed of master graduate students that had learnt how to develop a SRS document, and were familiar with inspections and defect classifications. The 2nd group was composed of third year undergraduate students that were familiar with SRS documents, inspections and defect classifications. We provided to each group the same SRS and the list of its defects. The subjects should register the type of defect in a form that included: the defects to classify, and distinct fields for the classifier, doubts between classifiers or to a new classifier and corresponding definition. The classification of the defects would indicate if the classifiers were ambiguous (one defect with different classifiers), meaningless (incorrectly classified) or incomplete (new classifier suggested).

TABLE II. REQUIREMENTS DEFECT CLASSIFICATION (FINAL VERSION)

Classifier	Definition	Example
Missing or Incomplete	The requirement is not present in the requirements document .Information relevant to the requirement is missing, therefore the requirement is incomplete. If a word is missing without affecting the meaning of the requirement the defect shall be classified as a typo.	"The system will allow authentication of authorised users." The way to access the system is not detailed. Is it by using a login and corresponding password? Using a card? And what happens when a non-authorised user tries to access it? If the requirement includes the expression To be Defined (TBD) it is incomplete.
Incorrect Information	The information contained in the requirement is incorrect or false, excluding typographical/grammatical errors or missing words. The requirement is in conflict with preceding documents.	Stating that "The Value Added Tax is 23%" when the correct value is 12%.
Inconsistent	The requirement or the information contained in the requirement is inconsistent with the overall document or in conflict with another requirement that is correctly specified.	One requirement may state that "all lights shall be green" while another may state that all "lights shall be blue"[23]. One requirement states "The house shall have 2 windows, 1 door and a chimney." and the second one states "The house shall have 2 windows and 1 door." one of the requirements is inconsistent with the other.
Ambiguous or Unclear	The requirement contains information or vocabulary that can have more than one interpretation. The information in the requirement is subjective. The requirement specification is difficult to read and understand. The meaning of a statement is not clear.	The requirement "An operator shall not have to wait for the transaction to complete." is ambiguous, depends on each person's interpretation. To be correctly specified it should be, e.g., "95% of the transactions shall be processed in less than 1 s." [23].
Misplaced	The requirement is misplaced either in the section of the requirements specification document or in the functionalities, packages or system it is referring to.	Include a requirement about the server application in the section that refers to the web-client application.
Infeasible or Non-verifiable	The requirement is not implementable, due to technology limitations, for instance. The requirement implementation can not be verified in a code inspection, by a test or by any other verification method. If the requirement is non-verifiable due to ambiguity, incorrectness or missing information, use the corresponding classifier instead.	"The service users will be admitted in the room by a teleportation system." The teleportation technology has not sufficiently evolved to allow the implementation of such requirement. "The message sent to the space for potential extraterrestrial beings should be readable for at least 1000 years."
Redundant or Duplicate	The requirement is a duplicate of another requirement or part of the information it contains is already present in the document becoming redundant.	The same requirement appears more than once in the requirements specification document, or the same information is repeated.
Typo or Formatting	Orthographic, semantic, grammatical error or missing word. Misspelled words due to hurry. Formatting problems can be classified in this category.	"The system reacts to the user sensibility, i.e. if the user is screaming the system stops." The word sensibility is different from sensitivity. When a picture is out of the print area.
Not relevant or Extraneous	The requirement or part of its specification is out of the scope of the project, does not concern the project or refers to information of the detailed design. The requirement has unnecessary information.	If the customer is expecting a truck then the requirement stating "The vehicle is cabriolet." is out of the scope of the project. A requirement that should have been removed is still in the document.

The results of the experiments are summarised in the pictograms on figure 2, that show the proximity of the subjects' answers (dark circles) to the classifier that we expected them to use (bright circles) in each defect. The size of the circle gives the number of the students that used a certain classifier. There were 29 defects to classify (x axis). The classifiers, **doubt** between classifiers or **new** classifier are represented in the y axis. We noticed that in the 1st experiment no defect was unanimously classified and in the 2nd several ones were. In both experiments certain defects were classified differently but with similar percentages. These observations induce us to conclude that certain defects will be differently classified, for their own characteristics. The full report of our work includes all experiments' results and analysis [24].

The two experiments we did are not totally comparable: the experience of the individuals on defects classification and the size of the group and the treatment (values of the type of defect attribute) were different. Despite that, the degree of agreement of the subjects, given by the Fleiss' kappa measure, was moderate in both experiments (0.46 in the 1st experiment and 0.44 in the 2nd) [25]. We also did a Cochran test to verify our hypotheses. Since the test is binomial, we considered that when the subjects chose the most used classifier they answered as the **majority (1)** and when they used any other classifier, they chose **other (0)**. The significance value indicates that the subjects answered the same way (0.60 in the 1st group and 0.63 in the 2nd, i.e. p-value > 0.05 which indicates that we cannot reject $\mathbf{H_0}$). Using the same transformation of data we did the McNemar test to verify if the results of the experiments were similar. The percentages of subjects classifying as the majority or using other classifier were similar on both experiments (see figure 3).

In our opinion, the following facts may have contributed to the subjects moderate degree of agreement: **(1)** the subjects were not the ones identifying the defects which may increase the error of misinterpretation (and consequent misclassification) of the defects; **(2)** the subjects were not involved in the development and did not have access to the developers of the SRS document. This is similar to the problem reported in an experiment of Walia and Craver [22]; **(3)** certain words in the description of defects induced the selection of the classifier named with a similar word; **(4)** the defects are expressed in natural language, which introduces ambiguity in the classification process.

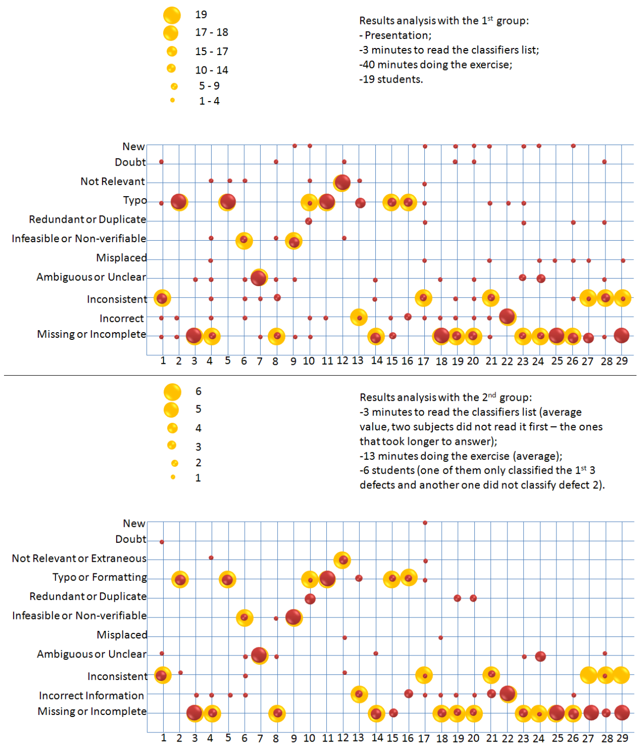

Figure 2. Results of the 1st experiment are represented the upper pictogram and of the 2nd are in the bottom.The collumn chart presents the McNemar test.

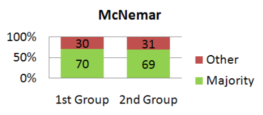

Figure 3. McNemar test.

Based on the experiments conducted, we suggest some recommendations for organizations that want to use requirements defects' classifications in an effective and consensual way: **(1)** people should be trained in the usage of the defects classification focusing in the distinction of the classifiers, the clarification of their definitions, practical examples and exercises; **(2)** to avoid that people apply a classifier based on its name only (often insufficient), without considering its definition, have the definition easily available, e.g., as a tool tip in a tool.

V. Conclusions and Future Work

We agree with Card when he states that a defect taxonomy should be created in such a way that it supports the specific analysis interests of the organisation that is going to use it, namely in the implementation of defect causal analysis [26]. In our work based on a literature review we assembled a classification for defect types in requirements specifications, following the recommendations in [6]. Such classification is important to support the analysis of root causes of defects and their resolution, to create checklists that improve requirements reviews and to prevent risks resulting from requirements defects. We evaluated our classification scheme through two experiments where students had to classify defects identified in a SRS document. We concluded that, even after refining the classification list, different people may classify the same defect in a different way. Hence, when choosing a classification for requirements' defects, organisations need to be aware of the problems of using them. People may interpret the classifiers differently and doing retrospective analysis of defects simply based on the type of defects might be misleading. Experiments similar to the ones presented in this paper may be conducted to determine the degree of consensus among their personnel.

As future research work we intend to improve the classifier and perform modified experiments "on the job", i.e.: **(1)** using individuals from industry; **(2)** using a SRS document from a project they are involved in; **(3)** having each individual conduct a complete SRS review to detect and subsequently classify defects. We expect that the classification difficulties will be attenuated in this setting, leading to more accurate and unanimous classifications. We will also use the defects classification to create a checklist to be used in the requirements inspections, and will conduct experiments (with a control group not using the checklist) to assess their impact on the review efficacy (percentage of defects detected), efficiency (time spent per defect) and convergence (of defect classification). We will verify if the classification of defects and application of the checklist reduce the number of defects in subsequent software development phases.

VI. References

[1] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," ed: IEEE Press, 1990.

[2] J.-C. Chen and S.-J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," Journal of Systems and Software, vol. 82, pp. 981-992 June 2009.

[3] M. Hamill and G.-P. Katerina, "Common Trends in Software Fault and Failure Data," IEEE Trans. Softw. Eng., vol. 35, pp. 484-496, 2009.

[4] D. N. Card, "Learning from Our Mistakes with Defect Causal Analysis," IEEE Softw., vol. 15, pp. 56-63, 1998.

[5] K. Henningsson and C. Wohlin, "Assuring Fault Classification Agreement - An Empirical Evaluation," presented at the International Symposium on Empirical Software Engineering, Redondo Beach, California, 2004.

[6] B. Freimut, et al., "An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management," presented at the Proceedings of the 11th IEEE International Software Metrics Symposium, 2005.

[7] L. Apfelbaum and J. Doyle, "Model based testing," presented at the 10th International Software Quality Week Conference, San Francisco, 1997.

[8] O. Monkevich, "SDL-based specification and testing strategy for communication network protocols," presented at the Proceedings of the 9th SDL Forum, Montreal, Canada, 1999.

[9] G. Mogyorodi, "Requirements-based testing: an overview," presented at the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), 2001.

[10] R. R. Lutz and C. Mikulski, "Requirements discovery during the testing of safety-critical software," presented at the Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, 2003.

[11] R. Chillarege, et al., "Orthogonal Defect Classification - A Concept for In-Process Measurements," IEEE Transactions on Software Engineering, vol. 18, pp. 943-956, November 1992.

[12] R. B. Grady, Practical software metrics for project management and process improvement: Prentice-Hall, Inc., 1992.

[13] T. E. Bell and T. A. Thayer, "Software requirements: Are they really a problem?," presented at the Proceedings of the 2nd international conference on Software engineering, San Francisco, California, United States, 1976.

[14] V. R. Basili and D. M. Weiss, "Evaluation of a software requirements document by analysis of change data," presented at the Proceedings of the 5th international conference on Software engineering, San Diego, California, United States, 1981.

[15] A. F. Ackerman, et al., "Software Inspections: An Effective Verification Process," IEEE Software, vol. 6, pp. 31-36, May 1989.

[16] G. S. Walia and J. C. Carver, "Development of Requirement Error Taxonomy as a Quality Improvement Approach: A Systematic Literature Review," Department of Computer Science and Engineering MSU-070404, 2007.

[17] J. H. Hayes, "Building a Requirement Fault Taxonomy: Experiences from a NASA Verification and Validation Research Project," presented at the Proceedings of the International Symposium on Software Reliability Engineering, Denver, CO, 2003.

[18] J. H. Hayes, et al., "Case History of International Space Station Requirement Faults," presented at the Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems, Standford, California, 2006.

[19] M. Kalinowski, et al., "Applying DPPI: A Defect Causal Analysis Approach Using Bayesian Networks," in Product-Focused Software Process Improvement. vol. 6156, M. Ali Babar, et al., Eds., ed: Springer Berlin / Heidelberg, 2010, pp. 92-106.

[20] G. M. Schneider, et al., "An experimental study of fault detection in user requirements documents," ACM Trans. Softw. Eng. Methodol., vol. 1, pp. 188-204, 1992.

[21] A. A. Porter, et al., "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment," IEEE Transactions on Software Engineering, vol. 21, pp. 563-575, June 1995 1995.

[22] G. S. Walia, et al., "Requirement Error Abstraction and Classification: A Control Group Replicated Study," presented at the Proceedings of the The 18th IEEE International Symposium on Software Reliability, 2007.

[23] IEEE, "IEEE Recommended Practice for Software Requirements Specifications," ed: IEEE, 1998.

[24] I. Lopes Margarido, "Requirements Defects Classification List," Faculty of Engineering, University of Porto, Technical Report 2010. http://paginas.fe.up.pt/~pro09003/publications.html.

[25] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," Biometrics, vol. 33, pp. 159-174, 1977.

[26] M. Kalinowski, et al., "Guidance for Efficiently Implementing Defect Causal Analysis," presented at the Brazilian Software Quality Symposium,VII SBSQ Florianópolis, Brazil, 2008.

VII. Annex A

TABLE III. Defect Classifiers Per author BY Chronological Order From Left to right.

		[13]	[14]	[15]	[16]	[11]	[12]	[20]	[21]	[17-18]	[22]	[19]	Sum
1	Not in current baseline	1.50%											1
2	Out of scope	7.20%											1
3	Missing/Omission	21.00%	24.00%							10.80%		23.50%	4
4	Incomplete	merged		Yes	Yes					23.30%			4
5	Inadequate	merged											1
6	Incorrect	34.80%	37.00%		Yes					30.11%		35.30%	5
7	Inconsistent	9.10%	10.00%	Yes	Yes			23	Yes	13.07%	Yes	5.90%	9
8	Incompatible	merged											1
9	New	7.20%											1
10	Changed Requirement	merged											1
11	Typos/Clerical	9.90%	23.00%										2
12	Unclear	9.30%											1
13	Ambiguity		4.00%	Yes				15	Yes	13.07%	Yes	11.80%	7
14	Wrong Section/Misplaced		1.00%						Yes	1.14%	Yes		4
15	Other		1.00%								Yes	5.90%	3
16	Infeasible				Yes					0.00%			2
17	Untestable/Non-verifiable				Yes					0.00%			2
18	Redundant/Duplicate				Yes					2.27%			3
19	Missing Functionality/Feature						/u/w/c/b	34	Yes		Yes		4
20	Missing Interface					/incorrect		11	Yes		Yes		4
21	Missing Performance							7	Yes		Yes		3
22	Missing Environment							9	Yes		Yes		3
23	Missing Software Interface						/u/w/c/b						1
24	Missing Hardware Interface						/u/w/c/b						1
25	Missing User Interface						/u/w/c/b						1
26	Missing Function/Description					/incorrect	/u/w/c/b						2
27	Missing Requirement/Specification						Inadequate						0
28	Missing/Incorrect Checking					Yes							1
29	Missing/Incorrect Assignment					Yes							1
30	Missing/Incorrect Timing/Serialization					Inadequate							0
31	Missing/Incorrect Build/Package/Merge					Inadequate							0
32	Missing/Incorrect Documentation					Inadequate							0
33	Missing/Incorrect Algorithm					Formal Spec							0
34	Incorrect or Extra Functionality								Yes		Yes		2
35	Data Type Consistency								Yes				1
36	Over-specification									1.14%			1
37	Not Traceable									2.27%			1
38	Unachievable									0.57%			1
39	Intentional Deviation									2.27%			1
40	General										Yes		1
41	Extraneous Information											17.60%	

For each defect classifier we indicate the authors who used it. The following information appears: **Yes** if we have no further information; the percentage of occurrence of a defect using the data of the experiment done with more data points; the quantity of defects; **merged** when the author used it merged with the classifier that is above that one; **Inadequate** when we consider that the classifier is not useful for requirements defects; **/incorrect**, indicating that the authors also used the 'incorrect' prefix; **/u/w/c/b** indicating the authors also used the prefixes 'Unclear', 'Wrong', 'Changed' and 'Better Way; **Formal Spec.** (Formal Specification) when we consider that such defect classifier would only be applicable if the requirements were specified with formal language.