Benefits of Defect Taxonomies and Validation of a new Defect Classification for Health Software

H. Ketheswarasarma Rajaram[1] , J. Loane[2], S. T. MacMahon[3], F. Mc Caffery[4]
Hamsini.Ketheswarasarma@dkit.ie[1], john.loane@dkit.ie[2], silvana.macmahon@dkit.ie[3], fergal.mccaffery@dkit.ie[4]
Dundalk Institute of Technology, Ireland

Abstract

Defect-based testing is a powerful tool for finding errors in software, including medical device software. Many software manufacturers avoid this method because it requires a detailed defect taxonomy that is expensive to construct and difficult to validate. SW91[1] is new defect taxonomy for health software being developed by the Association for the Advancement of Medical Instrumentation. This paper explains how defect taxonomies have been used and the benefits to industry. The initial steps of the validation of SW91 include mapping vulnerabilities from the Common Weakness Enumeration and a dataset from a medical device software development company in Ireland. Finally, the paper details future plans for validation, including taxonomy based testing which will be used to validate the efficiency, reliability, ability to perform useful analyses and defect coverage of SW91.

Keywords

Defect taxonomy, Defect Classification scheme for health software, Validation, Taxonomy based testing

1. *Introduction*

Medical devices increasingly rely on software to provide functionality [24]. Software complexity and the rapid growth of the software industry make it difficult to control and prevent defects [17]. Due to the introduction of advanced technologies, the medical device software industry is facing massive growth of complex software [24]. This massive growth of medical device software leads to quality risks.

The US Food and Drug Administration (FDA) reports that from 2005 to 2011, 19.4% of medical device recalls were related to software [28]. Another study focused

1 This document is still under study and subject to change

on recalls of medical devices related to computer-based failures such as software, hardware, inputs, outputs, or battery. This study reported that 2,303,441 recalls out of 12,024,836 were related to software. Software issues accounted for 33.3% of class I recalls, 65.6% of class II recalls and 75.3% of class III recalls [15,31]. The FDA recall process includes specifically identifying software-related recalls in order to improve medical device quality and to ensure patient safety [7].

Software quality assurance (SQA) practices have been integrated into the software development process to find defects and ensure software quality. SQA processes aim to minimize software defects and show that software meets requirements. There are many SQA activities, such as testing and inspections, which can be used to validate software [30]. Research studies suggest that a defect taxonomy is the best way to prevent and control defects [5,8,9]. People use customized or original defect taxonomies in different domains such as the safety critical domain, the business domain, and the telecommunications domain. Before they use defect taxonomies, they validate their defect taxonomies in terms of reliability, efficiency, and completeness.

This study focuses on validating a new defect taxonomy called SW91 [2]. This paper is structured as follows. Section 2 explains what a defect taxonomy is and how industries have used defect taxonomies during software development. Section 3 explains the benefits of using defect classification schemes. Section 4 explains the development of a new defect classification scheme for health software, SW91. Section 5 explains initial steps taken to validate SW91. Section 6 explains a new testing method called “taxonomy based testing” which details how defect categories from a taxonomy can be used in testing. Section 7 outlines plans for future work where taxonomy based testing will be used to validate SW91. Section 8 presents the summary and conclusions.

2. *State of the art – The use of defect taxonomies in industry*

A defect taxonomy is a system of hierarchical categories designed to be a useful aid for reproducibly classifying defects in the software development lifecycle [14]. There are many other terms for defect taxonomy including fault categorization, defect classification, fault classification scheme and bug taxonomy. In this paper, the terms defect taxonomy and defect classification scheme are used interchangeably. This section explains how different industries used various defect classification schemes for a variety of purposes in different phases of the software development lifecycle in order to improve software quality.

In 1998, at the Motorola Corporate Software Centre, the GSM Products Division’s Base Station Systems (GSMBSS) conducted a study on how the ODC scheme can be used to measure the progress of software development [4]. The ODC scheme was applied to an existing project with data collected by Fagan inspection [30]. After a successful feasibility study using the gathered data to verify the suitability of the ODC scheme, the team mapped defect data with minor modifications into the ODC scheme. This study proved that software development progress measurement and process improvement feedback can be produced from the data which was collected using an existing inspection method by adopting the ODC scheme. The authors of this study believed the ODC scheme can easily be applied to enhance software quality and to improve customer satisfaction while developing defect prevention and qualitative process management techniques [4].

In 2004, Lutz and Mikulski [20] published work on the analysis of 199 anomalies from seven spacecraft at the Jet Propulsion Laboratory. The purpose of this study was to improve the safety of future missions. The ODC scheme was selected to classify the post-launch safety critical software anomalies in order to extract the defect signature. The following outcomes were highlighted from this study:

- Training on documentation of anomalies can limit the reoccurrence of anomalies.
- The benefit of maintaining the documentation of system requirements for the operational process has been identified.
- Anomalies' analysis enhances the reusability of knowledge from one system to another.
- When comparing the outcomes from other methods related to operational risk, the anomaly patterns obtained by classifying the anomalies using the ODC scheme provided additional understanding of operational risks.

Finally, the authors of this study stated classifying anomalies lead to understanding the anomalies triggers and contributed to preventing operational anomalies.

Freimut et al. [10] conducted a study at Robert Bosch GmbH in the business unit for Gasoline Systems (GS) and published their work in 2005. Gasoline Systems developed electronic control units for gasoline engines with embedded software as a key component. To overcome the lack of information related to quality assurance and overall system quality at Bosch GS, it was decided to apply quantitative data management techniques in quality assurance strategies. They defined, introduced and validated a customized defect classification scheme to track defects which are involved in software development and process measurement [10]. The following outcomes were obtained after applying a defect classification scheme:

- Defect flow distribution and its outputs were observed.
- Identification of the defects introduced in early stages and identified in later stages.
- Defect data from the case study which identified categories with a high number of defects.
- Providing the measurement outputs to management.

In 2007, Robillard et al. [26] published work detailing the measurable test efficiency in a software product due to changing testing practices. This research was conducted with a team that developed audio software for video games. Two different testing phases, A and B, were used to measure the test efficiency. Phase A used implicit testing practices to record defects. In the second phase, B, an "Easy to follow" scheme was proposed to record the testing practices in order to make developers aware of the type of testing activities involved. A modified ODC scheme was used to record the defects in both testing phases and the following outcomes were observed [26]:

- The distribution of defects for the type of activity conducted in each phase, such as design review, code inspection, and unit test.
- The distribution of defects based on the discovery attributes. Discover attributes indicate who found the defects.
- The distribution of defects based on the activity qualifier attributes. The activity qualifier attribute indicates whether defects were found opportunistically or during planned testing.

- The ODC scheme was selected to get a statistical understanding of software process measurement using both types of testing.

In 2008, the ODC scheme was used in NASA flight projects. The ODC scheme was used as an extension of the COnstructive QUALity Model (COQUALMO) developed by Raymond Madachy and Barry Boehm from the University of Southern California, USA. The ODC COQUALMO model was used for critical NASA flight projects [21]. The ODC scheme was successfully adopted in defect reduction strategies. The ODC COQUALMO model helped in providing a highly detailed view of the defect profiles and their impact on specific risks. The ODC COQUALMO model with automated risk minimization helped to meet the quality goals of NASA’s flight projects in a shorter time with fewer resources [21].

In 2010, Li et al. [19] presented an extended and modified defect classification scheme named the Orthogonal Defect Classification scheme for Black-box Defect (ODC-BD) which was created based on the ODC scheme. This empirical study was aimed at helping black-box defect analysers and black-box testers to improve their testing efficiency and analysis. It was proved that the effort for defect analysis was reduced by 15% after applying the ODC-BD. The test efficiency, measured as the number of detected defects per unit time, in the first week, without using the ODC-BD scheme, was 0.075. In the second month, the test efficiency increased to 0.125 using the ODC-BD scheme [19]. This empirical study with 1660 black-box defects is a good example of measuring the black-box testing process with the ODC scheme.

In 2012, Mellegård et al. [23] published their work on developing an effective and systematic software defect classification scheme at Volvo car corporation in Sweden. They developed a software defect classification scheme “the Light-weight Defect Classification scheme (LiDeC)”, which complements the IEEE standard classification for software anomalies [12,13]. This study demonstrated the customization of a generic defect classification scheme to classify defects. Adopting a customized defect classification scheme minimized the time required to find defects while helping to characterize the defects. The HP scheme has been used within Hewlett-Packard departments for many different purposes such as root cause analysis and defect presentation [11,25].

In 2014, Nuno Silva and Marco Vieira [27] published their work which demonstrated the importance of domain specific defect classification schemes. They focused on four systems from the aerospace and space industries. They demonstrated the following problems in adopting a generic classification scheme into a safety critical domain:

- There are problems with adopting a generic defect classification scheme without considering the defect propagation effects and the interconnection of defects from different phases of the software development lifecycle.
- Inability to cover all the defects with existing listed defect types, defect triggers and defect impacts.
- Differentiating dimensions to keep the orthogonality of the defects was not easy to achieve.
- Not showing the connection to the quality models.
- There were difficulties getting the necessary level of information related to each defect to map with the defect type, defect trigger or defect impact.
- Difficulties mapping non-functional defects.
- Difficulties mapping the defects to a related standard.

The above points clearly demonstrate the problems in adopting a generic classification scheme into a safety critical domain. This study highlights the need for improved and domain specific defect taxonomies to classify defects.

Since medical device software is often safety critical, the necessity of a domain specific defect classification scheme has been identified and a defect classification scheme is being developed for healthcare software called SW91 [29]. The development of SW91 is explained in Section 4.

This section detailed how different defect classification schemes were used in different industries from 1998 to 2014 and the importance of domain specific defect classification schemes. The next section discusses the benefits of defect classification schemes.

3. *Benefits of defect classification schemes*

Bernd Freimut [11] has detailed various benefits of defect classification schemes including characterization of the defects found, defect prevention, control inspections, evaluate and improve technologies, control testing, plan testing and reduce field defects.

Vallespir et al. [32] stated a defect classification scheme makes it easy to find the injected defects while providing information on phases, activities, and disciplines throughout the software development lifecycle. Robert B. Grady from Hewlett Packard stated that the benefit of classifying defects is to help find the correct quality assurance activity. He stated: “As you categorize the defects, you will uncover a variety of symptoms. A typical first step will be for you to decide to do better or different inspections or tests” [25]. When combining both Vallespir et al. and Robert B. Grady’s statements, injected defects could be identified and classified by a defect classification scheme. Those classified defects will inform the choice of the correct quality assurance activities.

Kelly and Shepard stated a detailed defect classification scheme plays a significant role in understanding the software development process [18]. Defect classification schemes can be created for several different purposes in a software development organization. These include:

1. Making decisions during software development
2. Tracking defects for process improvement
3. Guiding the selection of test cases
4. Analysing research results [18]

Vogel has detailed a procedure for medical device software defect management. “Classification” is the second of eight steps. Vogel’s defect procedure supports the need for defect classification schemes in medical device software development. He stated the importance of classification in medical device software as follows: “the classification is important for later determination on the recommendations and means for verifying any changes made to deal with the defect”. Safety critical domains have utilized defect classification schemes to reduce defects and to improve analysis [21,24,27]. Safety critical domains have a requirement for a unique defect classification scheme tailored to their unique needs [33]. The medical device software industry is such a safety critical domain and hence should have its own domain specific defect classification scheme.

The Association for the Advancement of Medical Instrumentation (AAMI) is developing a defect classification scheme for health software which includes medical device software. Section 4 explains the development of the defect classification scheme for health software called SW91. Prior to the development of SW91, there has been no defect classification scheme specifically developed for use in the medical device software industry. It is hoped that applying a defect classification scheme into the medical device software industry will bring similar benefits observed in Sections 2 and 3.

4. *AAMI and Development of SW91*

AAMI is a non-profit organization founded in 1967 [1]. AAMI is developing a defect classification scheme named “Classification of Defects in Health Software-SW91” as a standard. This work started in 2014 and aims to provide a common language to classify defects and improve software quality in health software including medical device software [29]. SW91 was published in September 2016 for first public comment and again published in April, 2017 for the second round of public comment. It is expected that the final version of SW91 will be published later in 2017. SW91 includes defect categories from planning a system to maintenance and release of a system.

It contains multi-level defect categories such as parent level and bottom level. Each defect category has its own defect code with a unique number. The numbering system followed in SW91 is flexible to allow new categories to be added as necessary under any parent level of defect category. The next section explains how SW91 has been validated to date.

5. *Validation of SW91*

Before starting the validation of SW91, a brief comparison was done among other relevant defect classification schemes such as the ODC scheme, the IEEE Standard Classification for Software Anomalies and the HP scheme. Bernd Freimut analyzed different defect classification schemes based on their structure and their usability [11]. In the literature defect classification schemes were validated for their reliability, ability to perform useful analysis, and efficiency [10,19,20,22]. These terms will be considered in the validation of SW91. In addition to the above terms, the defect coverage of SW91 will also be validated by this research. Our plan has following three different tracks:

1. Mapping defects from databases.
2. Mapping defects from medical device software companies.
3. Taxonomy based testing.

Section 5.1 explains the first track of the validation and explains how SW91 was mapped with open source data. Section 5.2 presents the second track of validation. The third track of validation is a new approach using taxonomy based testing. Section 6 explains how taxonomy based testing has been used in other industries. Section 7 explains future plans to use taxonomy based testing to validate SW91.

5.1 Mapping SW91 with CWE

The CWE is an open source list containing common software weaknesses and their vulnerabilities. CWE Version 2.9 [6] was the latest version available when the mapping was started. CWE Version 2.9 contains 1004 vulnerabilities. This version of CWE has multiple views such as full dictionary view, development view, research view and fault pattern view.

Prior to the mapping, it was necessary to select the appropriate view of vulnerabilities from the CWE. After carefully analysing the multiple views, SW91 was mapped with the cross section view from the CWE. The approach to the mapping and the selection of the cross section view was discussed and finalized with the SW91 development team. The SW91 development team is composed of members from a number of relevant disciplines such as medical device product development, software engineering, software quality, and regulatory policy with members considered to be expert in their field.

The cross section view contains a selection of software weaknesses which represent the range of weaknesses captured in the CWE. These weaknesses include a total of 158 vulnerabilities [6]. From the CWE cross section, out of 158 vulnerabilities, 150 vulnerabilities were successfully mapped with SW91's defect categories. This was a manual one to one mapping. In my initial mapping it was not possible to find a suitable category from SW91 for the following eight vulnerabilities from the CWE cross section:

173: Improper Handling of Alternate Encoding	486: Comparison of Classes by Name
175: Improper Handling of Mixed Encoding	502: Deserialization of Untrusted Data
222: Truncation of Security-relevant Information	798: Use of Hard-coded Credentials
434: Unrestricted Upload of File with Dangerous Type	323: Reusing a Nonce, Key Pair in Encryption

The one to one mapping was reviewed by the SW91 development team who paid particular attention to the eight vulnerabilities that could not be mapped. The team members checked for the possibility of mapping with SW91's existing defect categories and there was a discussion on adding new defect categories and changing the name of a defect category, in order to ensure that all vulnerabilities could be mapped to a suitable defect category.

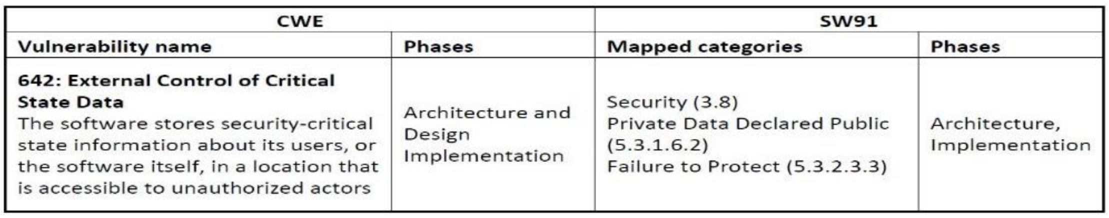
Out of the eight vulnerabilities which could not initially be mapped, five vulnerabilities were mapped with newly added defect categories or defect categories that required name changes. Three vulnerabilities were assigned to an existing defect category. This one to one mapping established that all of the CWE cross section vulnerabilities could be mapped to at least one defect category from SW91.

From the completed one to one mapping, a subset of vulnerabilities was selected by an experienced team member. The selected subset included eighteen vulnerabilities from all phases of the software development life cycle. One to many mapping was conducted for those eighteen vulnerabilities. In one to many mapping, each vulnerability was mapped with possible different defects categories from SW91. The purpose of the one to many mapping was to show the usability of SW9. The initial one to many mappings was conducted by me. Then the SW91 development team reviewed the mapping.

In the validation process out of eighteen mappings, five mappings were accepted by the team without any changes. Three mappings were changed by adding additional defect categories into the existing mapping. Two mappings were changed by adding additional defect categories and deleting some mapped categories. Three mappings were changed by adding additional defect categories including the reason for why those categories were selected. Two mappings were changed by deleting few mapped categories from the mapped defect categories. One mapping was changed by deleting defect category and including the reason for why other categories were selected. Finally two mapping were totally deleted and replaced with new mapped categories.

For example vulnerability 642: External Control of Critical State Data was mapped with the following two categories from SW91 by me. The third defect category was added by the team members when they were validating the one to many mapping:

1. **Failure to Protect (5.3.2.3.3):** Software permits access to an object that should be protected (for security reasons rather than for coherency), assuming the design is correct. **Fehler! Textmarke nicht definiert.**
2. **Private Data Declared Public (5.3.1.6.2):** An object is declared as public when it should be private. The object may be accessible to functions that should not use it, creating an unintended dependency or security vulnerability**.**Fehler! Textmarke nicht definiert.
3. **Security (3.8):** The defects are related to security issues in the architecture, such as compromising of sensitive information, choosing an inappropriate authentication protocol, not using access control, or communication integrity. It may also include the use of unsigned software and the introduction of unknown changes after the software is deployed. Note that many issues related to security involve requirements inadequacies, and many security vulnerabilities might be caused by poor design or implementation activities. Capturing all the causes and contributing factors for a security defect should involve identifying possible Requirement Defects (2.*), Design Defects (4.*) and Implementation Defects (5.*) rather than categorizing every failure associated with security as a Security (3.8) defect. **Fehler! Textmarke nicht definiert.**



CWE		**SW91**	
Vulnerability name	**Phases**	**Mapped categories**	**Phases**
642: External Control of Critical State Data The software stores security-critical state information about its users, or the software itself, in a location that is accessible to unauthorized actors	Architecture and Design Implementation	Security (3.8) Private Data Declared Public (5.3.1.6.2) Failure to Protect (5.3.2.3.3)	Architecture, Implementation

Figure 1: CWE Mapping**Fehler! Textmarke nicht defi-**

This one to many mapping shows how a user can select multiple different defect categories from SW91 to map to a particular vulnerability. This subset of mappings was added as an annex in SW91. This CWE mapping was conducted as part of the validation of SW91. This work was carried out to determine the defect categories and their coverage in SW91 when compared with publicly available vulnerabilities. This mapping also shows difficulties in reliability mapping data, with different people coming up with different mappings. The next section explains another part of the validation carried out with empirical data from a medical device software development company.[2]

5.2 Mapping SW91 with data from a medical device software company

As a part of the validation process, we contacted a medical device software development company to request empirical data to map with SW91. Company A from Ireland develops medical device software and web-based applications. The benefits of defect classification schemes and the need for a defect classification scheme in the medical device software industry were explained to the management of company A. Data was obtained from company A. The following data has been used in this mapping:

1 Defects
2. Software Design Specification
3. User Requirement Specification (URS)
4. Risks
5. Testing Protocols

The first four data sets were mapped with SW91 defect categories. Figure 2 displays the mappings of data from company A to SW91 defect categories. In mapping A of Figure 2 the defects from company A mapped to nineteen distinct defect categories from SW91. Some slight changes of wording were observed between the defects from the defect data and SW91 defect categories. For example, a defect from the defect data for a “function X” was described as “Units not converting correctly”. This defect was included in the mapping between SW91 defect category and received defect as “Type Conversion” for “function X”.

Then, in mapping B of Figure 2 control flow diagrams from the software design specification document were mapped with eighteen distinct defect categories from SW91. The software design specification document clearly explained the software and the constraints of the system. In addition, control flow diagrams in the software design specification document described the functionality of the system step by step. In this mapping, all elements from the control flow diagrams were mapped into defect categories from architectural defects, design defects, and implementation defects in SW91. Since SW91 uses a hierarchical structure of defects, it was easy to jump into the relevant defect category at the appropriate level. For example, if a control flow diagram has a processing step containing a statement “Count <1”, then searching for a relevant defect category from the implementation defects is straight forward rather than searching for defects from other phases of software development such as requirement defects or maintenance defects. Here the following defect categories were assigned to the above processing step “Count <1”:

- Mixed Sign
- Use Before Check
- Invalid Path
- Operator

The URS document includes forty-two requirements. In mapping C of Figure 2, forty requirements were mapped with thirty-eight distinct defect categories from SW91. Each requirement from the URS document has associated prioritized risks. Despite the URS documents and software design specification documents being prepared for the company’s own use, it was possible to map them with SW91.

A separate mapping of the testing protocols and SW91 defect categories was not performed because the testing protocols are already linked with the software design specification and the user requirements from URS document.

[2] This capture is from the draft version of the standard that was out for public comment.
This is not intended to represent the final version of the standard.

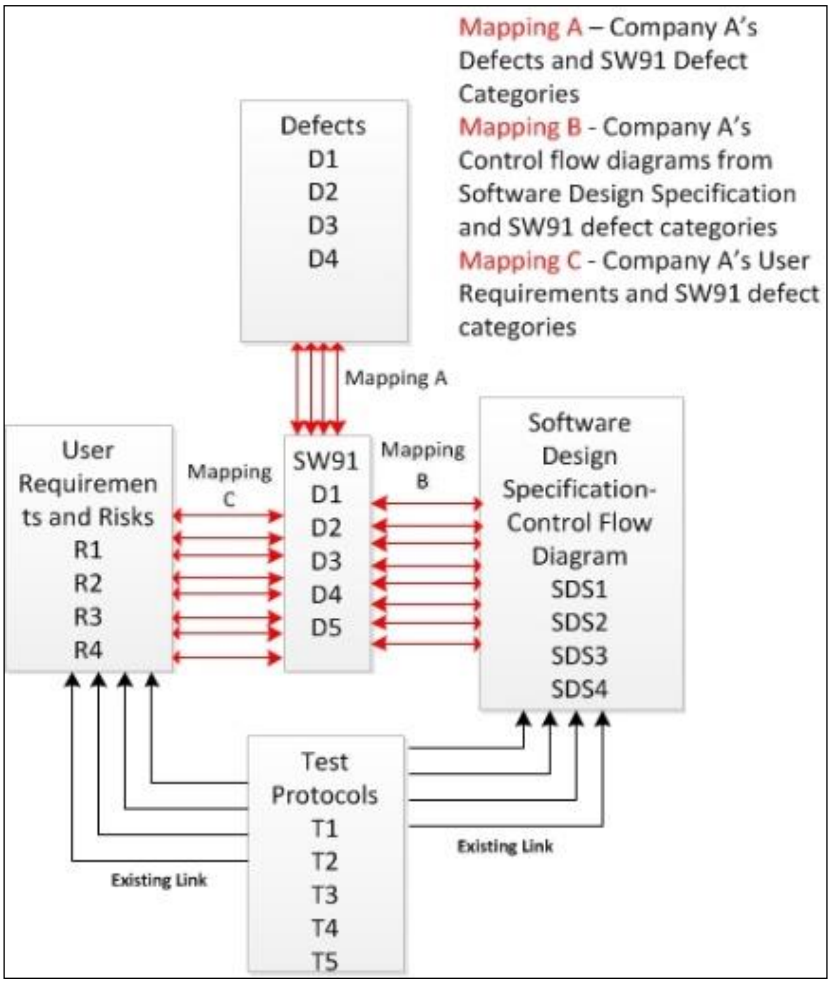

Mapping A – Company A's Defects and SW91 Defect Categories
Mapping B - Company A's Control flow diagrams from Software Design Specification and SW91 defect categories
Mapping C - Company A's User Requirements and SW91 defect categories
Defects D1 D2 D3 D4
Mapping A
User Requirements and Risks R1 R2 R3 R4
Mapping C
SW91 D1 D2 D3 D4 D5
Mapping B
Software Design Specification-Control Flow Diagram SDS1 SDS2 SDS3 SDS4
Test Protocols T1 T2 T3 T4 T5
Existing Link
Existing Link

Figure 2: Mapping company A's data to SW91

So the defect categories from SW91 were used in both mapping B and C can be directly linked with the respective the testing protocols

From this mapping, five common SW91 defect categories have been identified from all three mappings A, B and C. In this approach, whenever the requirements have been gathered and company A does this mapping, it will enable them to see the possible defect categories for each requirement. This type of mapping also allows goal oriented test cases to be written, consistent with the taxonomy based testing approach. Those goal oriented test cases will be based on the requirements and respective mapped defect categories from SW91. Execution of these goal oriented test cases will save time finding defects when a test case fails. This mapping will improve software quality by identifying defects at an earlier stage of software development such as identified common five defects.Since company A has detailed control flow diagrams, mapping each stage of the control flow diagram with SW91 defect categories will help to minimize defects at the development phase. When we have the anticipated defect categories for every stage, developers can work to avoid those defects. Quality assurance engineers run tests to find the mapped defect categories. This will minimize the time to find the defects and it will help to prevent defects at the earliest possible phase of software development. Company A has risks for every requirement in their URS document. Those risks are prioritized by severity. If every requirement is mapped with defect categories from SW91, the risks can be used to prioritize which defects should be fixed first.

As we discussed in Section 5, in terms of the validation of a defect classification scheme, the reliability of SW91 can be observed here. Normally, the reliability of a defect classification scheme is determined by the mapping of the same defects by different people. If different people map the same defects with the same defect categories from a classification scheme, then it is decided that the defect classification scheme has good reliability. Here, the same defects from three different documents including URS, control flow diagram from software design specification document and defects from defect data mapped to the same categories in SW91. Due to the confidentiality of the data from company A, we are unable to detail all the mappings here. This section explained how an initial empirical validation was carried out with data from a medical device software development company. Future work in this research will use taxonomy based testing as another method of validating SW91. Next section presents an explanation of taxonomy based testing. Section 7 explains plans for future work involving taxonomy based testing.

6. *Taxonomy based testing*

Defect taxonomies can be used in testing [3]. Creating the test cases for the defect categories from a defect taxonomy gives better test coverage [3,15]. Michael Felderer and Armin Beer have conducted significant research on defect taxonomy-supported testing (DTST) [9]. They stated, "Defect taxonomies can be applied to control the design of tests and the quality of releases to keep testing manageable although time and resources in projects are limited". In their research, a novel process of system testing using a defect taxonomy has been proposed and implemented. A case study was used to explain how a taxonomy can be integrated into the standardized test process defined by the ISTQB. The proposed test process contains five steps. The first four steps of the DTST process were integrated into the first step of the ISTQB test process called "Test Planning and Control". The next section explains future work with taxonomy based testing.

7. *Future work*

To continue the validation of SW91, our future work will focus on taxonomy based testing in a medical device software development company. This taxonomy based testing will consider the following points in terms of the validation of SW91:

- The efficiency of SW91
- The reliability of SW91
- Useful analyses enabled by SW91
- Defect coverage

Defect data will be requested from medical device software companies. This data will initially be used to check the defect coverage and reliability of SW91. After gathering other necessary data for taxonomy based testing, requirements will be mapped with SW91 defect categories. Test cases will be generated based on those mapped requirements. During the testing process, test cases generated from mapped requirements with SW91 defect categories will be executed and the results will be observed.

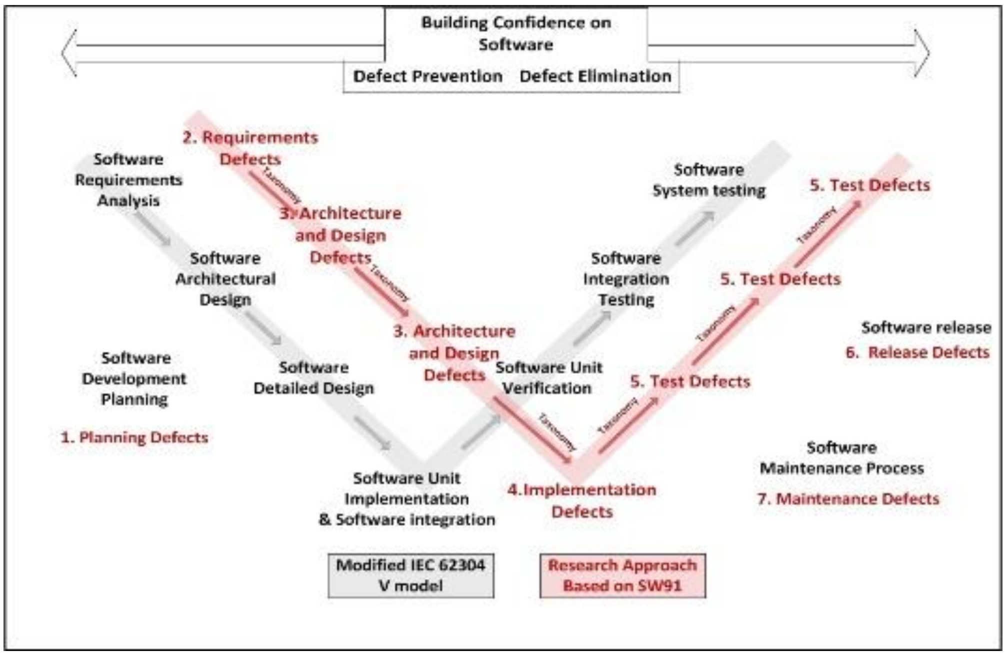

Building Confidence on Software
Defect Prevention
Defect Elimination
Software Requirements Analysis
2. Requirements Defects
Taxonomy
3. Architecture and Design Defects
Software Architectural Design
Software Development Planning
1. Planning Defects
Software Detailed Design
3. Architecture and Design Defects
Software Unit Implementation & Software integration
4.Implementation Defects
Software Unit Verification
5. Test Defects
Software Integration Testing
5. Test Defects
Software System testing
5. Test Defects
Software release
6. Release Defects
Software Maintenance Process
7. Maintenance Defects
Modified IEC 62304 V model
Research Approach Based on SW91

Figure 3: Modified IEC 62304 V model and taxonomy based testing

This method of validation will be used to assess the quality of SW91 in terms of efficiency, reliability, performance of useful analyses and defect coverage. Since SW91 includes defect categories for all phases of the software development lifecycle, taxonomy based testing will be used to examine the efficiency of SW91 in finding defects at an earlier stage of the medical device software development lifecycle. When it comes to the reliability of SW91, if a statistically significant number of quality assurance engineers at a medical device software company mapped the same given defects with same defect categories this will demonstrate the reliability. **Fehler! Verweisquelle konnte nicht gefunden werden.** explains the modified V model from IEC 62304 and how SW91 defect categories link with each phase of medical device software development. At the end of the taxonomy based testing, if SW91 helped to increase the test efficiency and helped to reduce similar software defects in future phases, this will be considered as the validation of SW91 in terms of useful analyses. If SW91 covers all identified defects from requirements capture to the final system, this will be considered a validation of defect coverage. Section 8 details the summary and conclusions of this paper.

8. *Summary and Conclusion*

This paper explained software quality problems in medical device software industries and why quality assurance practices may fail to identify defects. The benefits of defect taxonomies were outlined with the empirical examples from the literature. The necessity for a domain specific defect taxonomy in safety critical domains was also explained. The development of a new defect classification for health software (SW91) is underway to address this need in the medical device domain. Validation methods for defect taxonomies from the literature were also presented. As a validation of this newly developed defect classification for health software, two mapping were completed. Firstly, with CWE's vulnerabilities and, secondly, with data from a medical device company. These mappings examined the reliability and the defect coverage of SW91. Finally, our future work will utilize taxonomy based testing to validate the efficiency, reliability, enabling of useful analyses and defect coverage of SW91. Taxonomy based testing will also improve the software quality in medical device software.

9. *ACKNOWLEDGMENTS*

This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

10. *Literature*

1. Association for the Advancement of Medical Instrumentation. About AAMI. Retrieved 22 December 2016 from http://www.aami.org/membershipcommunity/content.aspx?ItemNumber=1292&navItemNumber=2906
2. Association for the Advancement of Medical Instrumentation. 2016. Classification of Defects in Health Software. 40. Retrieved from http://www.aami.org/standards/downloadables/aamirevf.pdf
3. Rex Black. 2008. *Advanced Software Testing - Vol. 2*. Rocky Nook Inc, Santa Barbara.
4. Norm Bridge and Corinne Miller. 1998. Orthogonal Defect Classification Using Defect Data to Improve Software Development. *Software Quality* 3, 1: 1–8.
5. Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Bonnie K. Ray, and Diane S. Moebus. 1992. Orthogonal Defect Classification-A Concept for In-Process Measurements. *IEEE Transactions on Software Engineering* 18, 11: 943–956. https://doi.org/10.1109/32.177364
6. CWE. CWE - CWE-884: CWE Cross-section (2.9). Retrieved 11 January 2017 from https://cwe.mitre.org/data/definitions/884.html
7. FDA. 2014. Medical Device Recall Report FY2003 to FY2012. 1–20.
8. Michael Felderer and Armin Beer. 2013. Using defect taxonomies for requirements validation in industrial projects. In *Requirements Engineering Conference (RE)*, 296–301. https://doi.org/10.1109/RE.2013.6636733
9. Michael Felderer and Armin Beer. 2013. Using defect taxonomies to improve the maturity of the system test process: Results from an industrial case study. In *SWQD 2013*, 125–146. https://doi.org/10.1007/978-3-642-35702-2_9
10. Bernd Freimut, Christian Denger, and Markus Ketterer. 2005. An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management. In *International Software Metrics Symposium*, 10–19. https://doi.org/10.1109/METRICS.2005.10
11. Bernd Freimut. 2001. *Developing and Using Defect Classification Schemes*. Retrieved from http://en.scientificcommons.org/20203575
12. IEEE-SA Standard Board. 1993. IEEE Standard Classification for Software Anomalies. 32.

13. IEEE-SA Standard Board. 2010. IEEE Standard Classification for Software Anomalies (Revision of IEEE Std 1044- 1993). 15.

14. International Software Testing Qualifications Board. *Standard glossary of terms used in Software Testing*.

15. Ravishankar K. Iyer, Zbigniew Kalbarczyk, Jaishankar Raman, and Jai Raman. 2013. Analysis of safety-critical computer failures in medical devices. *IEEE Security and Privacy Magazine 11*, 14–26. https://doi.org/10.1109/MSP.2013.49

16. Cem Kaner, Jack Falk, and Hung Quoc Nguyen. 1993. Testing Computer Software Second Edition APPENDIX: COMMON SOFTWARE ERRORS. . 1–89.

17. Ali A Karahroudy and M H N Tabrizi. 1996. Software Defect Taxonomy , Analysis and Overview. May.

18. Diane Kelly and Terry Shepard. 2001. A Case Study in the Use of Defect Classification in Inspections. *conference of the Centre for Advanced Studies on Collaborative research*.

19. Ning Li, Zhanhuai Li, and Xiling Sun. 2010. Classification of software defect detected by black-box testing: An empirical study. In *WCSE 2010*, 234–240. https://doi.org/10.1109/WCSE.2010.28

20. Robyn R. Lutz and Ines Carmen Mikulski. 2004. Empirical analysis of safety-critical anomalies during operations. *IEEE Transactions on Software Engineering* 30, 3: 172–180. https://doi.org/10.1109/TSE.2004.1271171

21. Raymond Madachy and Barry Boehm. 2008. *ODC COQUALMO - A Software Defect Introduction and Removal Model using Orthogonal Defect Classification*. Retrieved from http://csse.usc.edu/TECHRPTS/2008/usc-csse-2008-817/usc-csse-2008-817.pdf

22. Marcelo M. Manhães, Maria Claudia P. Emer, and Laudelino C. Bastos. 2014. Classifying Defects in Software Maintenance to Support Decisions Using Hierarchical ODC. *Computer on the Beach*: 283–292.

23. Niklas Mellegård, Miroslaw Staron, and Fredrik Törner. 2012. *A light-weight defect classification scheme for embedded automotive software and its initial evaluation*. https://doi.org/10.1109/ISSRE.2012.15

24. PTC. Software Development for Medical Devices. 1–9. Retrieved 8 April 2016 from https://www.ptc.com/~/media/Files/PDFs/ALM/Integrity/Software_Development_for_Medical_Devices.pdf?la=en

25. Robert B. Grady. 1992. *Practical software metrics for project management and process improvement*. Prentice Hall PTR.

26. P N Robillard and T Francois-Brosseau. 2007. Saying, ‘I Am Testing,’ is Enough to Improve the Product: An Empirical Study. In *ICCGI 2007*, 5. https://doi.org/10.1109/ICCGI.2007.54

27. Nuno Silva and Marco Vieira. 2014. Experience report: Orthogonal classification of safety critical issues. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*: 156–166. https://doi.org/10.1109/ISSRE.2014.25

28. Lisa K. Simone. 2013. Software-related recalls: An analysis of records. *Biomedical Instrumentation and Technology* 47, 6: 514–522. https://doi.org/10.2345/0899-8205-47.6.514

29. Lisa Simone and Daniel Rubery. 2014. Lisa Simone and Daniel Rubery: A Tower of Babel with Medical Device Software Failures. *AAMIBlog*, 1. Retrieved 24 January 2017 from https://aamiblog.org/2014/10/10/lisa-simone-and-daniel-rubery-a-tower-of-babel-with-medical-device-software-failures/

30. Jeff Tian. 1990. *Software quality engineering*. John Wiley & Sons, Inc, Dallas, TX. https://doi.org/10.1016/0950-5849(90)90039-T

31. U.S. Food and Drug Administration. Recalls, Market Withdrawals, &amp; Safety Alerts - Background and Definitions. Retrieved 3 February 2017 from https://www.fda.gov/Safety/Recalls/ucm165546.htm

32. Diego Vallespir, Fernanda Grazioli, and Juliana Herbert. 2009. A framework to evaluate defect taxonomies. In *XV Argentine Congress of Computer Science*.

11. *Author CVs*

Hamsini Ketheswarasarma Rajaram
Hamsini Ketheswarasarma Rajaram is a Doctoral Researcher in the Regulated Software Research Centre. She graduated with Honors in Bachelor Science in Information Technology (Software Engineering) in the year 2011, at Sheffield Hallam University, UK. She then completed her Master of Science degree in Bioinformatics in 2015, IBMBB University of Colombo, Sri Lanka. Currently, her research focus is on validating medical device software defect taxonomy.

John Loane
Dr. John Loane is a lecturer at the Dundalk Institute of Technology. He is a researcher in the Regulated Software Research Centre and formerly a researcher at the NetwellCASALA research centre. He has received EU FP7 research funding to develop an Indicator-based Interactive Decision Support and Information Exchange Platform for Smart Cities.

Silvana Togneri Mac Mahon
Dr Silvana Togneri Mac Mahon is a Postdoctoral Researcher in the Regulated Software Research Centre. She is currently working as a postdoctoral researcher on a Lero project which focuses on the development of a Medical Device Software Development Framework. She has acted as international project leader, author and editor for the development of ISO/TR 80001-2-7 and is currently involved in the revision of the IEC 80001-1 standard.

Fergal MC Caffery
Dr Fergal Mc Caffery is a Lecturer with Dundalk Institute of Technology. He is the leader of the Regulated Software Research Group in Dundalk Institute of Technology and a member of Lero. He has been awarded Science Foundation Ireland funding through the Stokes Lectureship and Principal Investigator Programmes to research the area of software process improvement for the medical device domain. Additionally, he has received EU FP7 research funding to improve the effectiveness of embedded software development environments for the medical device industry.