Automated Prediction of Defect Severity Based on Codifying Design Knowledge Using Ontologies

Martin Iliev[1], Bilal Karasneh[2], Michel R. V. Chaudron[3]
Leiden Institute of Advanced Computer Science
Leiden University
Leiden, The Netherlands
[1]miliev@liacs.nl, [2]bkarasne@liacs.nl, [3]chaudron@liacs.nl

Edwin Essenius
Technical Software Engineering
Logica Nederland B. V.
Rotterdam, The Netherlands
edwin.essenius@logica.com

***Abstract*—Assessing severity of software defects is essential for prioritizing fixing activities as well as for assessing whether the quality level of a software system is good enough for release. In filling out defect reports, developers routinely fill out default values for the severity levels. The purpose of this research is to automate the prediction of defect severity. Our aim is to research how this severity prediction can be achieved through reasoning about the requirements and the design of a system using ontologies. In this paper we outline our approach based on an industrial case study.**

***Keywords*-severity; defect; design; ontology; automatic classification**

I. INTRODUCTION

Software goes through a testing phase which aims to find the problems users might experience before the software goes into actual use. According to the IEEE Standard Classification for Software Anomalies [2], the cause of a software problem is called a software defect. In order to remove the problems, the defects need to be fixed. We use well established standards (including the IEEE Standard Computer Dictionary [1] and the IEEE Standard in [2]) for defining these terms. The classification in [2] defines a defect as: (1) a fault if it is encountered during software execution (thus causing a failure); (2) not a fault if it is detected by inspection or static analysis and removed prior to executing the software.

In [1] a fault is defined as an incorrect step, process, or data definition in a computer program, while a failure represents the inability of a system or component to perform its required functions within specified performance requirements. The dictionary relates all these terms to one another by distinguishing between a human action (a mistake), its manifestation (a hardware or software fault), the result of the fault (a failure), and the amount by which the result is incorrect (the error). Hence, a software defect is the reason for producing an incorrect or unexpected result in a computer program or system, or it causes it to behave in unintended ways.

Therefore, in order to deploy a high quality software product, it needs to be tested first. Defects found during the testing phase need to be fixed within a specific time constraint – before the deployment date. A software team needs to decide on the order in which to fix these defects. It is a common practice to assign severity levels (SLs) to the defects to differentiate between their impacts on the software. The severities of defects represent the different levels of negative impact a defect will have on the deployment of a software product. For example, a severity level "showstopper" is assigned to defects which prevent the release of the software system and immediate attention is required. It is clear, then, that defects must be assigned the correct SLs. The assignment of SLs to defects is specific for every software system or company and is done manually, usually by test analysts according to their expertise. However, it is often the case that a defect is assigned the default severity level, which typically is medium. A user might not agree with the assignment of default severities and might want some defects to be fixed sooner than others. To address this problem, we are conducting research in the area of how to predict the severity of defects using the knowledge of the design of the software and incorporating the user requirements while decreasing the workload of the software architects and the test analysts. This means that we will use the user specifications to assign SLs that will reflect what is important not only for the developers but also for the users. With this paper, we would like to illustrate our approach towards devising a method for automatically predicting the severity of defects found during testing at the system level. Such a method would be especially useful for large software systems which have many defects. In achieving our goal, we use Artificial Intelligence (AI) techniques, namely ontologies, reasoning and automatic classification, in order to capture software defects through an ontology and automatically reason about the defects and their SLs.

The rest of this paper is organized as follows. Section 2 presents a brief discussion of related work. The explanation of the used approach is in Section 3. Section 4 presents the intermediate results. The threats to validity are presented in Section 5. Section 6 concludes the paper. Finally, a discussion of the future research is presented in Section 7.

II. RELATED WORK

It is known that the severity levels assigned to defects are used to find out what is the impact of that defect on the deployment of the software. However, different software projects assign different severity levels to their defects. More importantly, why a specific defect is assigned one severity and not another and whether both the developers of the

software product and its users agree on the assignment of the severity levels are areas that still need more attention.

A new and automated method which assists the test engineer in assigning SLs to defect reports is presented in the paper by Menzies and Marcus [3]. The authors have named the method SEVERIS (SEVERity ISsue assessment) and it is based on text mining and machine learning techniques applied to existing sets of defect reports. The paper presents a case study on using SEVERIS with data from NASA's Project and Issue Tracking System (PITS). The case study results indicate that SEVERIS is a good predictor for issue SLs, while it is easy to use and efficient. The idea behind our research is similar to the study in [3] – an automated method for predicting what SLs should be assigned to defects. However, we base our method on the design of the software system (product) in order to incorporate the user specifications when deciding what severity level to assign to a defect so that in the end, the user satisfaction with the quality of the deployed software product will rise.

Zhou and Leung [4] investigate the accuracy of the fault-proneness predictions of six widely used object-oriented design metrics with particular focus on how accurately they predict faults when taking fault severity into account. Their results indicate that most of these design metrics are statistically related to fault-proneness of classes across fault severity and that the prediction capabilities of the investigated metrics greatly depend on the severity of faults. The authors use logistic regression and machine learning methods for their empirical investigation. In our research, we focus on predicting the severity levels of defects using AI techniques such as ontologies and automatic classification. This is achieved by developing an ontology and classifying the defects input in it using predefined reasoning rules.

Additional motivation for our work comes from the research conducted by Suffian [5] who establishes a defect prediction model for the testing phase using Six Sigma methodology. The author's aim is to achieve zero-known post release defects of the software delivered to the end users. At the end of his research, the author states that his work focuses on predicting the total number of defects regardless of their severity or the duration of the testing activities and that future effort can focus on improving the defect prediction model to predict defect severity in the testing phase. Therefore, our study represents an extension to the research in [5] since we aim at predicting the SLs of defects which have been found during the testing phase of the software development cycle. Moreover, our study makes use of defects' attributes as defined in [2] to develop a method that will be applicable to many software projects.

Overall, the contribution of our study is the following. (1) We use the knowledge of the design of the software system (product) to incorporate the user specifications. This way, the severity levels assigned to the defects will reflect what is important not only according to the developers but also according to the users. (2) We use ontologies and ontology reasoning to automatically classify the defects input in the ontology into the severity levels predefined in it. (3) We use attributes and their values from a well-established IEEE standard in order to describe the defects and their SLs in the ontology. This way, the ontology will be applicable to any software project and useful for many people such as software architects, developers, test analysts (under the condition that they will also use that standard).

III. APPROACH

The approach is divided in three parts: data collection, data analysis and conversion, and data classification.

A. Data Collection

This case study was conducted at Logica, in the company's office in Rotterdam, the Netherlands. The data represent defects from the testing phase of a project. In order to collect relevant and useful data, at least basic knowledge of the project in question was required. To gain such knowledge, the project was studied using its documentation – design documents, UML diagrams, user manual, test documents (providing insights about the issue management system and the SLs). After that, the issue management system was used to extract a sample of 33 defects based on the project knowledge and the recommendations of the designers, the developers and the test analyst working on the project. This subset was selected to include defects from each severity level yet limit the amount of defects because of time constraints. Table I presents details about the number of fixed defects according to the project's SLs. Then, interviews were conducted with the same people to get detailed information about the selected defects and to verify that they are a representative subset (almost one third) of all defects fixed in the latest version of the system.

B. Data Analysis and Conversion

The detailed information about the 33 defects from the previous step includes the following: the SLs of the defects, the causes for the defects, the types of the defects, the reasons for assigning a specific severity level to a defect and the ways through which the defects were found. Since this information is project-specific, the standard in [2] is used to convert the project-specific information about the defects into the defect attributes and their values defined in this standard. The attributes used are the following: severity, effect, type, insertion activity and detection activity. This conversion resulted in a table that contains the defect IDs together with the values of the attributes from the standard

TABLE I. NUMBER OF FIXED DEFECTS ACCORDING TO THE SEVERITY LEVELS FROM THE PROJECT

Severity Level	Number of Fixed Defects		
	In all versions of the system	*In the latest version of the system*	*Chosen for this case study*
Minor	85	12	5
Medium	301	93	17
Severe	47	10	10
Showstopper	6	1	1
Total	439	116	33

assigned to each defect based on its detailed information. In order to get an idea how the table looks, Table II shows only the first three defects after the conversion step (the rest of the table is omitted because of space concerns).

In addition, we have decided to use three SLs for the defects in the ontology. Although the IEEE standard provides five SLs (blocking, critical, major, minor and inconsequential), there are very small distinctions between blocking and critical SLs and between minor and inconsequential SLs (with respect to the impact of a defect with such a severity level on program flow). For this reason, we have combined these SLs together. We have defined the relation in Table III between the ontology SLs and the SLs from the classification in [2]. The table also shows the relation with the original SLs used in the project (given in Table I). According to the project's definitions, there is a small distinction between showstopper and severe SLs so they are combined together.

C. Data Classification

In our approach, software defects are captured through an ontology. For building the ontology, a specific ontology editor is employed – Protégé (http://protege.stanford.edu). The Protégé-OWL plugin [6] is also used since it supports the Web Ontology Language (OWL). The OWL sub-language used is OWL-DL, which is based on Description Logics (DL) according to [6]. We chose OWL-DL because it is possible to automatically compute the classification hierarchy and check for inconsistencies in an ontology that conforms to OWL-DL and our ontology conforms to it.

Once this decision was made, we created classes for the defects, classes for the attributes from the standard in [2] and their respective subclasses. We also created properties describing the relations between the chosen defects and the attributes from [2] used in the ontology. After that, we input in the ontology the converted information about the defects.

As a last step in the process, we defined the reasoning rules for classifying the defects into the categories major-, medium- and minor severity level. The defect classification rules mimic the reasoning patterns of the designers. In order to incorporate the user requirements, the rules give more weight to defects inserted during the requirements and design phases than during coding and configuration phases.

Rule 1 (R1) defines the necessary and sufficient conditions for a defect with major severity level. It consists of five sub-rules and a defect must satisfy all of them to be assigned that severity level. They are the following:
(R1.1) Defect;
(R1.2) (isInserted only (InDesign or InRequirements)) or ((isInserted only (InCoding or InConfiguration)) and (hasEffectOnNumber min 3)) or ((isInserted only (InCoding or InConfiguration)) and (hasType min 2));
(R1.3) hasEffectOnNumber min 2;
(R1.4) hasType only (Data or Interface or Logic);
(R1.5) isDetected only (FromSupplierTesting or FromCoding).

These sub-rules mean the following: an entity is assigned major severity level if and only if it is: (R1.1) a defect; (R1.2) inserted during the design or the requirements, or inserted during the coding or the configuration and affecting at least three values of attribute Effect (which represent quality properties) or at least two values of attribute Type; (R1.3) affecting at least two of the values of attribute Effect (quality properties); (R1.4) affecting one or more of the values data, interface or logic of attribute Type; (R1.5) detected during the coding or the supplier testing phase. The rules and the numbers were chosen manually based on the pattern in the data while keeping them as general as possible.

Rule 2 (R2) defines the necessary and sufficient conditions for a defect with medium severity level. It consists of three sub-rules and a defect must satisfy all of them to be assigned that severity level. They are:
(R2.1) Defect;
(R2.2) (not DefectWithMajorSL) or ((isInserted only (InCoding or InConfiguration)) and (hasEffectOnNumber exactly 2) and (hasType only Logic));
(R2.3) not DefectWithMinorSL.

These sub-rules mean the following: an entity is assigned medium severity level if and only if it is: (R2.1) a defect; (R2.2) not a defect with major severity level or it is inserted during the coding or the configuration and is affecting a fixed number of values of attributes Effect and Type (the second part classifies the boundary cases between major and medium SLs which were not classified using earlier rules); (R2.3) not a defect with minor severity level.

Rule 3 (R3) defines the necessary and sufficient conditions for a defect with minor severity level. It consists of two sub-rules and a defect must satisfy both of them to be assigned that severity level. They are the following:
(R3.1) Defect;
(R3.2) hasEffectOnNumber max 1.

TABLE II. THREE DEFECTS CONVERTED INTO THE ATTRIBUTES OF THE IEEE STANDARD CLASSIFICATION IN [2]

Defect ID	**Attributes**				
	Severity	***Effect***	***Type***	***Insertion Activity***	***Detection Activity***
440	Blocking	Functionality; security; performance; serviceability	Data; interface	Design	Supplier testing
318	Critical	Usability; performance	Logic	Coding	Supplier testing
333	Critical	Functionality; performance	Logic	Design	Supplier testing

TABLE III. THE RELATION BETWEEN THE SEVERITY LEVELS

Severity Levels (SL)		
Used in the ontology	***From the IEEE Standard Classification***	***From the project used in this case study***
MajorSL	Blocking; critical	Showstopper; severe
MediumSL	Major	Medium
MinorSL	Minor; inconsequential	Minor

These sub-rules mean the following: an entity is assigned minor severity level if and only if it is: (R3.1) a defect; (R3.2) affecting at most one of the values of attribute Effect (which represent quality properties).

In order to represent explicitly the relations between the defects, the SLs and the quality properties mentioned in the rules above, we created a schematic view of these relations presented in Fig. 1, which shows only a selection of the attributes. Then, using the formal semantics, the created classes and properties and the rules for the severity levels, we automatically classified all defects into the three SLs. Figure 2 shows the ontology classification process as a function that takes the input for the ontology (shown in Fig. 1) and uses the defined rules to produce the output from the ontology (also shown in Fig. 1). The results are presented in the next section.

IV. RESULTS

The final step in the process of achieving the results is the automatic classification. The main point is to use the OWL-DL reasoner to perform automatic classification and determine which defects have major, medium or minor severity level in the ontology. The chosen reasoner for the automatic classification of the defects is Pellet [7]. The choice fell upon it because it is an open-source Java-based OWL-DL reasoner and it provides functionalities to check consistency of ontologies, classify the taxonomy, etc. Based on the defined rules, the reasoner classified all defects input in the ontology into the three severity levels.

After the classification results were present, we compared them with the results from the data conversion step, given in Table II. Since both the ontology results and the table's contents conform to the IEEE standard in [2], a comparison was possible (using the defined relation in Table III). The comparison between the two classifications is summarized in Table IV. The ontology classified 58% of the defects in the same SLs as originally. The other defects were classified in different SLs by the ontology compared to Table II. More specifically, 21% of all defects were classified one

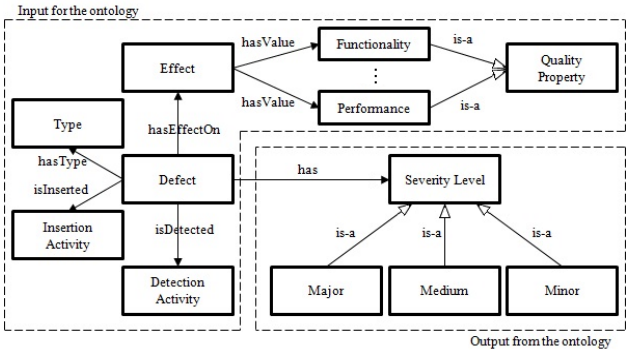

Figure 1. A schematic view of the relations between defects, quality properties and severity levels used in the ontology.

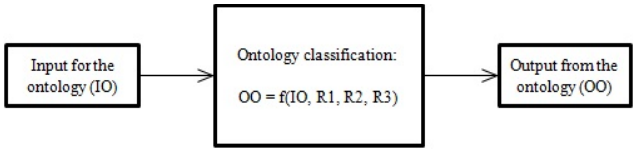

Figure 2. An abstract view of the ontology classification process.

severity level higher and 21% of all defects were classified one severity level lower by the ontology than the original classification. The reason is the following: the classification performed in the ontology is concerned with taking into account the user's point of view while preserving the developer's point of view when considering which defects are important for fixing and which are not (as opposed to not taking into account the user's point of view at all). Therefore, some defects related to the design of and the requirements for the software are classified one severity level higher according to the results from the ontology compared to the results from Table II. This way, these defects will be given a greater chance of being fixed for the next release which will satisfy more users of the software product.

Moreover, some of the differences between the two classifications come from the fact that there are defects assigned the default severity level (usually medium) by the people working on the project without paying much attention whether this is the correct severity level or not. So, Table II also contains such defects. Since the ontology classifies all defects, some of the defects assigned the default severity level in the table are assigned major, medium or minor severity level by the ontology. Hence, each defect is assigned a specific severity level and no default SLs are used.

Table V shows the above-mentioned results using a confusion matrix. The rows represent the results from the original classification while the columns show the classification results obtained by the ontology.

TABLE IV. SUMMARY OF THE COMPARISON BETWEEN THE TWO CLASSIFICATIONS

Severity Levels Used in the Ontology	**Percentage of All Defects**		
	Classified in the same SL by the ontology	***Classified one SL higher by the ontology***	***Classified one SL lower by the ontology***
MajorSL	25%	Not Applicable	9%
MediumSL	18%	21%	12%
MinorSL	15%	0%	Not Applicable
Total	58%	21%	21%

TABLE V. SUMMARY OF THE RESULTS USING A CONFUSION MATRIX

		Ontology Classification		
		MajorSL	***MediumSL***	***MinorSL***
Original Classification	***MajorSL***	8	3	0
	MediumSL	7	6	4
	MinorSL	0	0	5

In the end, it is worth mentioning that the results from this case study were presented to two software architects from the company familiar with the project used in the case study. In their opinion the results satisfy the expectations that an automatic classification of defects into predefined severity levels is possible and the results from it are satisfactory for an initial case study.

V. THREATS TO VALIDITY

This section presents the threats to validity in our study.

Firstly, this is a single case study and, therefore, its results cannot be generalized.

The second threat is related to the selection of the project for our case study. This project was chosen out of a set of projects performed at the company based on the extensiveness of its documentation and the availability of the people who have worked on it. The availability of documentation may indicate a more than average level of formality of the development process.

The last threat that we have defined is the fact that the data in the repository may contain more default values for the severity levels than should truthfully be the case. Consequently, the statistics should not be considered the ideal validation of the prediction.

VI. CONCLUSIONS

The goal of this research is to devise a method for automatically predicting severity of defects found during the testing phase of the whole software system. Moreover, the basis of the prediction is the design of the system in order to incorporate the user requirements in the process. To achieve this goal software defects are captured through an ontology. This enables automated reasoning about the defects and their severity levels. The initial version of the ontology was developed using the Protégé ontology editor and the OWL-DL language.

In order to have a uniform framework for the attributes of the defects, the IEEE Standard Classification in [2] was used and the defects' detailed information was converted into the values of the attributes from this standard. This step is essential in order to have the possibility to apply the ontology to various software projects.

The main parameter that affects the classification is the number of quality properties that is impacted by a defect. There are inconsistencies between the ontology classification and the original classification of the defects used in the project. These inconsistencies can be contributed to the fact that our research defines unique rules for the severity levels in the ontology as explained in the previous sections. Another reason for the inconsistencies is that no default severity levels are used in the ontology while default severities have been used in the original classification.

Overall, automated reasoning about defects and their severity levels could be used to suggest a classification of the defects in a software system to be confirmed by its users. Moreover, it would aid in the testing phase by decreasing the workload of the test analysts. Automated reasoning about designs, requirements, defects and severity levels could open up a new realm of possibilities for advanced Computer-Aided Software Engineering (CASE) tools.

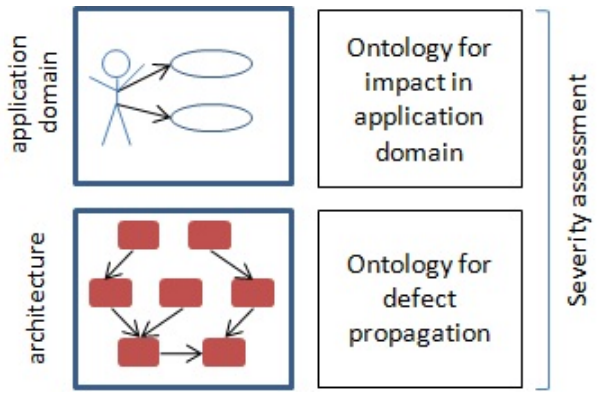

Figure 3. Possible future work direction.

VII. FUTURE WORK

Our future work will include extending the number of severity levels to five as given in [2]. We will perform a validation case study through which we will test the generality of the classification.

We would also like to increase the level of automation of reasoning by focusing on defect propagation that links defects found at unit-level to use cases at the system level. A graphical representation of such future research is given in Fig. 3. The proposed severity assessment will be based on the impact found via defect propagation and the importance of the use cases that are impacted in the application domain.

REFERENCES

[1] *IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries*, IEEE Std 610-1991, doi:10.1109/IEEESTD.1991.106963.

[2] *IEEE Standard Classification for Software Anomalies*, IEEE Std 1044-2009, doi:10.1109/IEEESTD.2010.5399061.

[3] T. Menzies and A. Marcus, "Automated Severity Assessment of Software Defect Reports," IEEE Int. Conf. on Software Maintenance, Beijing, 2008, pp. 346-355, doi:10.1109/ICSM.2008.4658083.

[4] Y. Zhou and H. Leung, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults," IEEE Trans. Softw. Eng., vol. 32, no. 10, Oct. 2006, pp. 771-789, doi:10.1109/TSE.2006.102.

[5] M. D. B. M. Suffian, "Defect Prediction Model for Testing Phase," M.S. thesis, Faculty Comput. Sci. Inform. Syst., Univ. Teknologi Malaysia, Johor, Malaysia, 2009.

[6] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, "A Practical Guide to Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools Edition 1.0," Univ. Manchester, Manchester, UK, Aug. 27, 2004.

[7] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A Practical OWL-DL Reasoner," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5, no. 2, June 2007, pp. 51-53, doi:10.1016/j.websem.2007.03.004.