A Metric for Quantifying the Ripple Effects among Requirements

Elvira-Maria Arvanitou[1], Apostolos Ampatzoglou[1(*)], Alexander Chatzigeorgiou[1], Paris Avgeriou[2], and Nikolaos Tsiridis[3]

e.arvanitou@uom.edu.gr, a.ampatzoglou@uom.edu.gr, achat@uom.edu.gr, paris@rug.nl, ntsiridis@gmail.com

[1] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

[2] Faculty of Science and Engineering, University of Groningen, Groningen, Netherlands

[3] Research and Development Department of OTS, Thessaloniki, Greece

[(*)] Corresponding Author

Abstract—During software maintenance, it is often costlier to identify and understand the artifacts that need to be changed, rather than to actually apply the change. In addition to identifying the artifacts related to the change per se, one needs also to identify the artifacts that are changed due to ripple effects. In this paper, we focus on ripple effects and propose a metric for assessing the probability of one requirement to be affected by a change in another requirement (i.e., requirements ripple effect). We focus on the requirements level, since most maintenance tickets (that stem from the customer) are captured in natural language and therefore are more naturally mapped to requirements, rather than source code. The proposed metric—the Requirements Ripple Effect Measure (R2EM)—is calculated by considering the conceptual overlap between the involved requirements (through their past co-change), the parts of the code in which they are implemented (i.e., their overlapping implementations), and the underlying dependencies of the source code (i.e., ripple effects between classes). We note that despite the involvement of source code artifacts in the calculation of R2EM, this metric is considered as a requirements' level one, since the unit of analysis is pairs of software requirements. To validate the proposed metric, we conducted an industrial case study, on two enterprise applications of an SME. The study design involved both quantitative and qualitative data, and input was given by 9 practitioners. The results suggested that R2EM is able to identify ripple effects between requirements at a satisfactory level, and those effects are mostly caused by overlapping implementations and source code ripple effects of these implementations.

Keywords— metrics; change impact analysis; requirements; maintenance

1. Introduction

During software maintenance, one of the most challenging activities, is to *identify* the software artifacts (e.g., requirements, design, source code, etc.) that need to be maintained (Queille et al. 1994). This identification process typically involves two steps:

- ***Identifying directly affected artifacts***. The maintenance ticket (e.g., a bug report, a feature request, etc.) is examined, in order to infer the affected artifacts, such as the requirement, the design artifact, and eventually the source code modules that need to be updated.
- ***Identifying indirectly affected artifacts***. Due to structural or conceptual reasons, a change in a software artifact might emit changes to other (seemingly disconnected) artifacts, in a form of *ripple effect*. Such ripple effects are typically studied through *Change Impact Analysis (CIA)* (Kretsou et al., 2021). The importance of the ripple effect phenomenon, as a factor that increases maintenance costs is highlighted by Galorath (2008) and Chen and Huang (2009), who suggest that maintenance costs increase by up to 75% if the software has a high risk of ripple effects.

In this paper we focus on ripple effects (and the corresponding change impact analysis) at the requirements level, so that we can identify which requirements might be affected by a maintenance ticket pertaining to another requirement. According to Antoniol et al. (2000), maintenance activities, which are initiated by end-users

are usually specified in natural language. Therefore, it is easier to map these maintenance tickets to requirements, rather than other artifacts (Antoniol et al. 2000).

Our goal is to propose a metric (see Section 3) termed ***Requirements Ripple Effect Metric (R2EM***) for assessing the probability of requirements ripple effects—i.e., the probability of one requirement to change, due to a change in another requirement. Calculating R2EM for all pairs of requirements can be ***useful in practice,*** since it can ***save maintenance time:*** practitioners will need less time to identify which source code artifacts will potentially need to be updated, re-tested, and re-deployed, due to their relation to the affected requirements. More specifically, when a certain requirement is updated, R2EM can be consulted to identify requirements that might need to be updated as well. Next, following traces from requirements-to-code, a list of source code artifacts can be identified that might need to be changed as well. This results in a more precise maintenance activity rather than intuitively looking for potentially affected artifacts, which could be assisted by regression testing—identifying parts of the code "*that break*".

The validity of the proposed metric is evaluated in an industrial setting, involving two medium-size systems, by considering expert opinions (i.e., quality managers and developers). In the case study, we first analyse the two systems and calculate R2EM for all pairs of requirements. At the same time, we ask practitioners to rank the requirements that are prone to be affected by a change in another requirement. Subsequently, we explore if the ranking based on R2EM and the participants' expert opinion are consistent. The organization of the rest of the paper is as follows: In Section 2 we present related work, whereas in Section 3 we present in detail the proposed metric. In Section 4, we discuss the industrial case study design, whereas in Section 5 we present the results of the study. Next, in Section 6 we discuss the main findings, and in Section 7 the threats to validity. Finally, in Section 8 we conclude the paper.

2. Related Work

In this section, we present studies related to change impact analysis at the requirements levels. We note that despite the fact that the proposed approach relies on requirements-to-code traceability, the proposed approach is not itself a traceability approach. Therefore, in this section we do not discuss studies related to software artifact traceability. The interested reader can refer to secondary studies on software artifact traceability (Charalampidou et al., 2020).

Nejati et al. (2016) presented an approach to automatically identify the impact of requirements changes on system design using Systems Modelling Language (SysML) models. The approach has two main steps: for a given change, the method gets a set of estimated impacted model elements, by identifying the design elements (inter-block structural relations) that are reachable from the changed requirement, and then, they rank the resulting set of elements according to a quantitative measure obtained using Natural Language Processing (NLP) techniques. Moreover, the measure is computed, by applying NLP to the textual information of the elements. To validate the approach, the authors have performed an industrial case study for evaluation purposes. The results suggest that (by using the approach) software engineers need to inspect on average only 4.8% of the entire design to identify the actually-impacted elements. The main difference of this work compared to our study is that this work focuses on the identification of the impact of requirements changes to system design, whereas in our study we perform change impact analysis from the requirements to the source-code level.

Goknil et al. (2014) proposed an approach for change impact analysis in requirements models. The approach uses formal semantics of requirements relations (e.g., requires, refines, etc.) and requirements change types (e.g., add, update, delete, etc.). The basis of this work was provided in a previous study, by the same group (Goknil et al. 2008). For example, when comes a request to "delete the R1", there are 4 cases: (a) R1 contains R2 and R3; (b) R1 refines R2; (c) R1 requires R2; and (d) R1 conflicts R2. Additionally, the authors extended their tool namely TRIC with features for change impact analysis at requirements level (i.e., proposing and

propagating changes, displaying inconsistent proposed changes, implementing proposed changes in the requirements model, and predicting the impact of proposed changes). More specifically, the tool automatically determines the change propagation paths, checks the consistency of the changes, and suggests alternatives for implementing the changes. The authors illustrated their approach and their tool with a Course Management System example. The results of the study suggest that none of the industrial requirements management tools support change impact alternatives and consistency checking of changes. Additionally, Goknil et al. (2014) determined some of the false positive impacts that usually occur in the industrial tools, by providing change alternatives with impact prediction. The main differences compared to our study is that this work focuses only at the requirements level using formal semantics, without taking into account the change history of the requirement and other elements, such as source code or design elements.

Conejero et al. (2012) investigated the relations between crosscutting concerns and requirements maintainability. In particular, the authors studied the correlation between crosscutting properties and requirements changeability and stability; stability is defined as the quality attribute that refers to the extent to which a software system is resistant to change (ISO/IEC 9126-1 2001). As a proxy of requirements stability, the authors have used the number of times, in which a requirement has changed along the history of the system. The authors performed an empirical study in order to identify the relation between modularity properties (namely tangling, scattering and crosscutting) and maintainability quality attributes at the requirements level using three software product lines. The results of the study suggest that the presence of crosscutting properties negatively affects to changeability and stability at requirements level. Although the proposed metric is able to measure change proneness of a requirement, it is considered an after-the-fact measurement.

Arora et al. (2015) suggested an approach based on NLP for analysing the impact of change in Natural Language requirements. More specifically, the approach detects the phrases in the requirements statements, extracts the tokens of these phrases, and computes similarity scores for the extracted tokens. To enable phrase-level analysis of changes, they cast these change operations as additions and deletions of phrases. Next, they calculate for every requirements statement, a normalized matching score given a propagation condition. The matching score is computed bottom-up, from atomic to composite expressions. The authors have implemented their approach in a prototype tool, namely NARCIA (NAtural language Requirements Change Impact Analyser). The evaluation of this approach has performed in two industrial case studies using 14 change scenarios. The results of this study suggest that across the change scenarios in their case studies, the author could detect 99% (105 / 106) of the impacted requirements through phrasal analysis. The difference of this study, compared to ours, is that this study focuses only on CIA in natural language requirements specifications.

Furthermore, Rahman et al. (2014), investigated the reasons (risk factors as mentioned in the paper), that can lead to requirements change. In order to identify the risk factors, the authors performed both theoretical and practical approaches. Regarding theoretical, the authors reviewed of previous studies in the literature, whereas regarding empirical, they performed a focus group interview with 7 practitioners from software industry. The results of the study suggested that changes can arise due to people, processes, product internal changes, and hardware infrastructure. The difference of this study to ours is that in our work, we go one step further than Rahman et al., since we do not only explore the factors that can lead to changes, but also quantify them.

Hassine et al. (2005) provided a change impact analysis approach for requirements using use case (UC) maps. Use Case Maps (UCMs) (Dahlstedt and Persson 2005) have been introduced to capture and integrate functional requirements in terms of causal scenarios representing behavioral aspects at a higher level of abstraction, providing stakeholders with guidance and reasoning about the system-wide functionalities and behaviour. The aim of this study is to present an approach that applies both scenario and component-based dependency analysis techniques and the UCM forward slicing approach to identify change impacts at the requirement level. Dependencies between UC scenarios are used to identify the impacted scenarios. The authors performed a case study on

a telephony system to illustrate the applicability of their approach. The main difference of this approach to ours is that Hassine et al. (2005) focuses only on requirements (ignoring relations at other levels), as well as, it assumes the existence of UCMs in existing software systems.

Compared to the related work presented in this section, the current study is the first one which: (a) ***quantifies ripple effects among requirements***, and assesses the probability of those effects to occur through a metric that relies on both source code and requirements history; (b) receives a ***simple input*** that is usually existent in practice (i.e., Git repository and commit comments), without relying on sparse inputs (such as UCMs); and (c) ***uses industrial experts' opinion for validating*** the results.

3. Requirements Ripple Effect Metric (R2EM)

3.1 Requirements Ripple Effect Metric (R2EM)

In our earlier work (Ampatzoglou et al. 2015; Arvanitou et al. 2015; Arvanitou et al. 2017a; Arvanitou et al. 2017b) we proposed a common high-level approach for performing CIA, by calculating the probability of an artifact to change due to ripple effects. To calculate the probability of one artifact to change due to a ripple effect, we use the joint probability formula, since two events need to co-occur: (a) the artifact that emits the ripple effect needs to change; and (b) the change actually ripples to the affected artifact, since the ripple effect is a probabilistic event itself. Next, we describe how the probability of the two aforementioned events to occur can be calculated at the requirements level.

Probability of a Requirement to Change (i.e., Trigger for a possible ripple effect). To make this assessment, we exploit the version control history of the project. In particular, we reuse the metric proposed by Conejero et al. (2012), which calculates the Percentage of Commits in which a specific Requirement has Changed (PCRC). We calculate the metric at commit level, i.e., we compute the percentage of commits, in which a specific requirement has changed. To be able to calculate this metric, a detailed commit message is required that allows tracking the requirement(s) that are being affected. We note, that for this mapping we are not using the committed code, but we only rely on the commit message. Thus, we assume that the process of the organization that uses the metric, imposes that developers commit a message that explicitly states the affected requirements or the issue (e.g., when using an issue tracking system) that has initiated the commit. Commit messages have been widely used in research as accurate descriptors of changes occurred in the software: according to Spinellis et al. (2009) in FreeBSD, all commit messages provide a reference to the id of the change request, whereas Buse and Weimer (2010) suggested that approximately 66% of commit messages are informative enough to understand the change in the requirement.

Probability of the Ripple Effect to Occur. The assessment of probability to change due to ripple effect depends on the strength of the dependency between the requirements. Therefore, the first step for assessing this probability is to build a catalogue (although not exhaustive) of the kinds of dependencies between requirements that are able to emit ripple effects[1]. Along with the presentation of each kind of dependency, we also describe the metric for assessing the probability that this dependency will generate a ripple effect P(Y|X), through an exemplar system that is visually represented in Figure 1. The example system concerns the management of students and courses in a university. For simplicity, we consider that the system is object-oriented (in a different case, classes could have been substituted with files), and that high-level requirements are formed based on groups of requirements that work on the same entity. Also, we consider two main entities, namely Student and Course, while the course grade is considered as an attribute of a student for a specific course.

Conceptual Dependencies between Requirements. According to Dahlstedt and Persson (2005) two requirements are related in situations where one requirement is similar to or overlapping with another in terms of how it is

[1] We note that we cannot claim that this list is exhaustive. However, we have not identified any other type of dependency from the case study. Nevertheless, other types of dependencies may exist, so we have added a relevant threat to validity.

expressed or in terms of a similar underlying idea of what the system should be able to perform. Additionally, Zhang et al. (2014) suggest that requirements dealing with the same data, are highly prone to ripple effects. In particular, Zhang et al. (2014) validated with industrial stakeholders that if the data is to be changed, all similar functions may be changed too. In our example, the requirements `Create Student` and `Edit Student`, are highly likely to produce a ripple effect, since they are both related to the *Student* entity. Therefore, if the email address of the `Student` needs to be validated (e.g., includes the '@' symbol and a dot) in a future version of the system, the implementation of both requirements will need to be co-maintained, so as to ensure the correctness of the validation. The probability of this dependency to produce a ripple effect is assessed through the ***Probability to change due to Conceptually Overlapping requirements (PCO)*** metric. $PCO_{Y \rightarrow X}$ is assessed by using the percentage of past commits, in which the two requirements (Y and X) have co-changed.

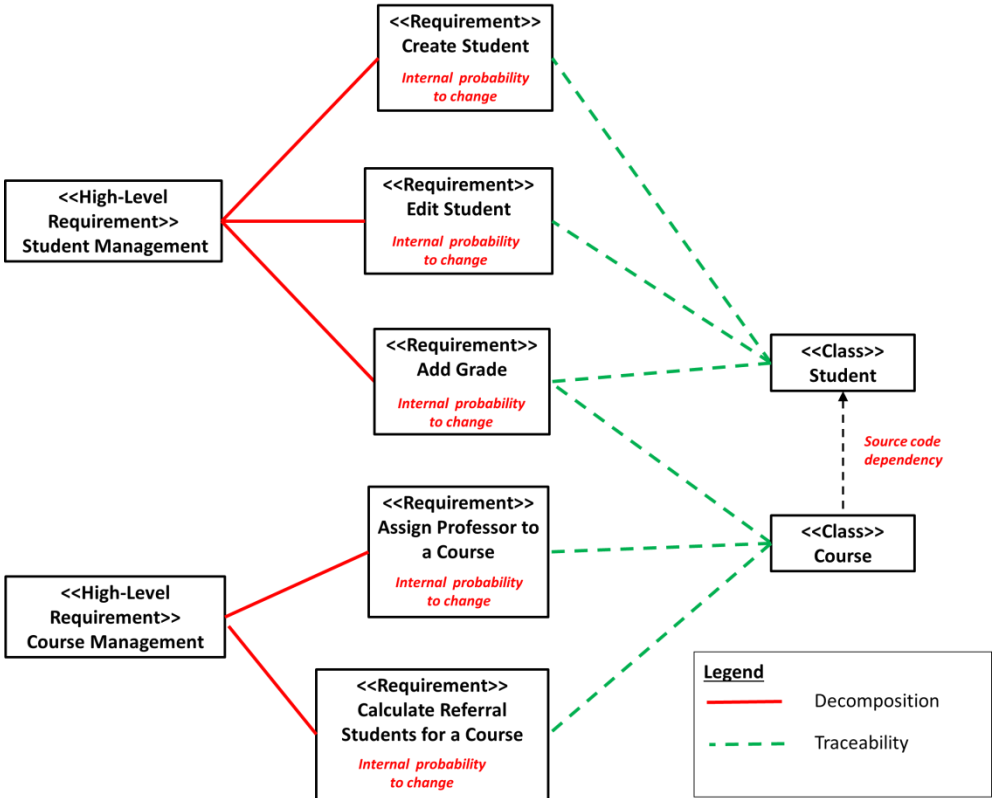

Figure 1. Example System for the External Probability to Change

Dependencies between Implementations of Requirements. This category emerged based on several approaches that link requirements and implementation (Ali et al. 2013). For example, Kagdi et al. (2009) suggest that if two or more source code artifacts (e.g., files) tend to co-change for a long time in the history of the project, they are highly probable to be conceptually related, e.g., they belong to the same requirement. This category is decomposed into two sub-categories:

- *Dependencies due to Overlap in Requirements Implementations*. The source code implementation of two or more requirements includes a set of common artifacts (e.g., classes, files, etc.). For example, considering Class–Responsibility–Collaboration (CRC) cards (Beck and Cunningham 1989; Fowler 2003), all pairs of responsibilities that are noted in the same CRC have an overlapping implementation in the specific class. By focusing on the example, even though the requirements `Add Grade` and `Assigning Professor to a Course`, are part of different high-level requirements (`Student` and `Course Management`, respectively) they are probably sharing at least one common implementation (e.g., the `Course` class). Thus, when changing the `Course` class, the implementation of both requirements might have to be maintained. The probability of this dependency to produce ripple effect is assessed through the ***Probability to change due to***

Overlapping requirements Implementations (POI) metric. This probability ($POI_{Y \rightarrow X}$) is assessed by the percentage of shared classes in the implementation of Y and X requirements. To guarantee the independence of PCO and POI, we omit from this calculation the commits that two or more requirements are co-changing.

Dependencies due to Source Code Ripple Effects of Requirements Implementations. The classes that implement a specific requirement might emit changes, due to structural dependencies to classes implementing other requirements. Based on the example of Figure 1, although some requirements might not have overlapping implementations, the classes in which they are implemented, might be structurally dependent (e.g., class `Course` holds an array of `Student` objects, so as to be aware of which students are enrolled in it). In this case, a change in class `Student` can potentially emit changes to `Course`, for example if the signature of the method that fetches the list of students that have to re-sit the course exam. Therefore, if the implementation of any of the `Student Management` requirements changes, then also the implementations of the `Course Management` requirements need re-maintaining. The probability of ripple effect based on this dependency is assessed through the ***Probability to change due to Ripple Effects at the source code level (PRE)***. This probability ($PRE_{Y \rightarrow X}$) is assessed by using the union probability of all classes implementing Y to ripple changes to classes that are involved in the implementation of X, through source code dependencies. For assessing the probability of a single dependency to produce a source-code ripple effect, we use the Ripple Effect Metric (REM) (Arvanitou et al. 2017a). In particular, we examine all pairs of classes that are not considered in the PCO calculation (guaranteeing the independence of PCO and PRE), and investigate if they are structurally dependent. The calculation of REM is as follows:

$$REM_{(A \rightarrow B)} = \frac{NDMC(A \rightarrow B) + NOP(B) + NPrA(B)}{NOM(B) + NA(B)} * PCCC(A)$$

NDMC: Number of distinct methods' calls from class A to class B

NOP: Number of polymorphic methods in class B

NPrA: Number of protected attributes in class B

NOM: Number of methods in class B

NA: Number of attributes in class B

PCCC: Percentage of commits in which class a has changed[2]

Upon the calculation of the aforementioned probabilities, R2EM can be calculated as described below:

$$R2EM = \text{Joint Probability } \{PCRC, \text{Union Probability } \{PCO, POI, PRE\}\}$$

3.2 Illustrative Example

To illustrate the calculation of R2EM, we consider a system with 3 requirements (R1, R2, and R3) that are implemented in 10 classes, throughout a version history of 10 commits (as presented in Table 1). We illustrate the calculation of R2EM for the pair R1→R2, i.e., the probability of R2 to change, due to changes in R1. The PCRC for R1 equals 20% (i.e., the possible trigger for a ripple effect), since R1 changes in 2 (out of 10) commits.

Table 1. Illustrative Example Commit History

	Changed	
Commit	**Requirements**	**Classes**
1	R1	C1, C2, C3
2	R2	C3, C4, C6
3	R3	C6, C7
4	R2, R3	C3, C8

[2] We note that for REM, we refer to the frequency of past changes through PCCC, whereas for R2EM through PCRC

Commit	Changed	
	Requirements	Classes
5	R3	C6, C8, C9
6	R1, R2	C1, C3
7	R2	C3, C5
8	R3	C6, C8, C10
9	R2, R3	C4, C5, C10
10	R3	C10

Based on the above, the ***probability*** of R2 ***to change due to conceptual overlap*** with R1 ***($PCO_{R1 \rightarrow R2}$)*** is 10%—1 out of 10 commits. From Table 1, we can observe that the two requirements co-change only in commit #6 (i.e., they exist in the same row in the second column).

To calculate the ***probability*** of R2 ***to change due to overlapping implementation*** with R1 ***($POI_{R1 \rightarrow R2}$)***, first each requirement should be mapped to the classes in which it is implemented: $\text{ImplementationSet}_{R1} = \{C1, C2, C3\}$ and $\text{ImplementationSet}_{R2} = \{C3, C4, C5, C6\}$. Thus, ***$POI_{R1 \rightarrow R2}$ is 33%*** since the implementation set has one common class (C3), and the requirement R1 that emits the change is implemented in three classes.

To calculate the ***probability*** of R2 ***to change due to ripple effects*** at classes implementing R1 ***($PRE_{R1 \rightarrow R2}$)***, we first need to calculate REM (Arvanitou et al. 2017a) at the class level. The illustrative (they cannot be deducted from the class diagram—see Figure 2) metrics for each class are presented in Table 2. Each row of the table represents one class (the one that emits the change), whereas the columns represent the quality metrics that are synthesized in the REM metric. The internal probability to change for every class is presented in the column named PCCC (Percentage of Commits in which a Class has Changed)—based on Table 1. Also, let us suppose that the only inheritance relationships are between class C4 and C5 from C1. The value in the parenthesis in the NDMC column refers to the number of distinct methods' calls from one class to the other.

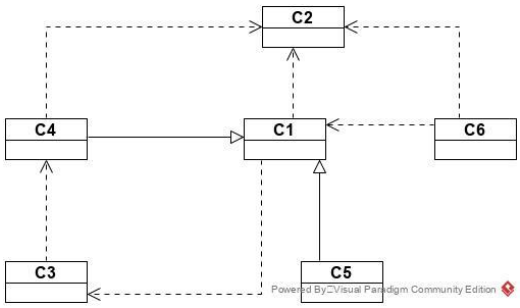

Figure 2. Class Diagram for the illustrative example

Class	PCCC	NOP	NPrA	NOM	NA	NDMC
C1	0.2	2	1	4	4	C2(2), C3(2)
C2	0.1	0	0	5	2	C4(6)
C3	0.5	0	0	3	3	C4(2)
C4	0.2	0	0	6	3	C2(1), C9(2)
C5	0.2	0	0	4	2	C1(1)
C6	0.4	0	0	2	1	C1(2), C2(2)

Table 2. Illustrative Class Metrics

To calculate ***$PRE_{R1 \rightarrow R2}$*** we need to calculate $\text{PRE}_{C1 \rightarrow C4}$, $\text{PRE}_{C2 \rightarrow C4}$, $\text{PRE}_{C1 \rightarrow C5}$, $\text{PRE}_{C2 \rightarrow C5}$, $\text{PRE}_{C1 \rightarrow C6}$, and $\text{PRE}_{C2 \rightarrow C6}$. The pairs are created by making pairs from $\text{ImplementationSet}_{R1}$ to $\text{ImplementationSet}_{R2}$. We note that C3 is not considered in this process, since it is common for both requirements.

$$\text{PRE}_{C1 \rightarrow C4} = \frac{0+2+1}{4+4} * 0.2 = 7.5\% \qquad \text{PRE}_{C2 \rightarrow C4} = \frac{1+0+0}{5+2} * 0.1 = 1.4\% \qquad \text{PRE}_{C1 \rightarrow C5} = \frac{1+2+1}{4+4} * 0.2 = 10.0\%$$

$$\text{PRE}_{C2 \rightarrow C5} = \frac{0+0+0}{5+2} * 0.1 = 0.0\% \qquad \text{PRE}_{C1 \rightarrow C6} = \frac{2+0+0}{4+4} * 0.2 = 5.0\% \qquad \text{PRE}_{C2 \rightarrow C6} = \frac{2+0+0}{5+2} * 0.1 = 2.8\%$$

$$PRE_{R1 \rightarrow R2} = \text{Union Probability } \{\text{PRE}_{C1 \rightarrow C4}, \text{PRE}_{C2 \rightarrow C4}, \text{PRE}_{C1 \rightarrow C5}, \text{PRE}_{C2 \rightarrow C5}, \text{PRE}_{C1 \rightarrow C6}, \text{PRE}_{C2 \rightarrow C6}\} \approx 26\%$$

$$\text{R2EM}_{R1 \rightarrow R2} = \text{Joint Probability } \{\text{PCRC, Union Probability } \{\text{PCO}_{R1 \rightarrow R2}, \text{POI}_{R1 \rightarrow R2}, \text{PRE}_{R1 \rightarrow R2}\}\} =$$

$$\text{Joint Probability } \{20\%, \text{Union Probability } \{10\%, 33\%, 26\%\}\} = \text{Joint Probability } \{20\%, 69\%\} \approx 75\%$$

3.3 Proposed Tool-Chain

To automate the calculation of all the aforementioned probabilities, we have extended a sequence of tools during our earlier work (see Figure 3) that calculate the probabilities for each pair of requirements of the projects. First, we use the git Repository and two commands (clone and log) in order to: (a) clone the repository and export the source code, and (b) produce the log documenting the commit history. Next, the commit history is used as an input for calculating PCCC (Arvanitou et al. 2017a). The produced document along with the source code is provided as an input to the REM Calculator tool (Arvanitou et al. 2015). REM Calculator produces a file that records the ripple effect probability at the class level. As a final step of the process and for the purposes of this study, we also developed the R2EM Calculator tool. The tool is command line and receives as input: (a) the commit history and (b) the REM document. The tool is available online along with all the other tools that comprise the aforementioned tool-chain[3].

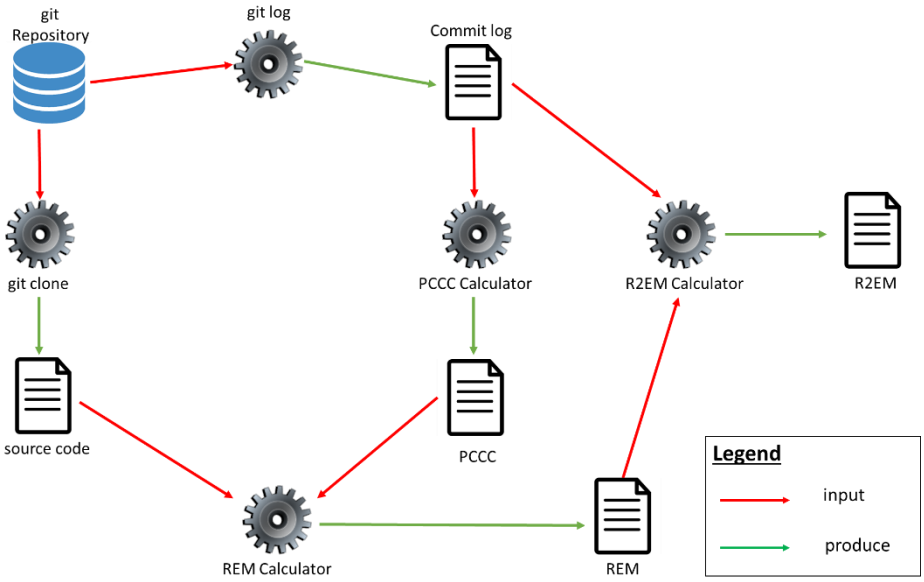

Figure 3. Used Tool-chain for Calculating R2EM

4. Case Study Design

In this section we present the design of the industrial case study that we conducted to validate R2EM, within a Small-Medium Enterprise (SME). The study is designed and reported according to Runeson et al. (2009).

4.1 Research Questions

In this section, we present the research questions of our case study. RQ_1 is exploratory, aiming at providing some insights on the phenomenon, whereas RQ_2 corresponds to the validation of R2EM. To answer the RQs, we used a quantitative approach complemented with quotes and explanations provided by the practitioners. We note that due to the lack of an automated, generic-enough tool (i.e., a tool that uses as input artifacts that exist in the industrial partner), we were not able to perform a comparative study to demonstrate the effectiveness of R2EM. To this end, we have performed a user study that is able to identify the agreement of R2EM to expert opinion.

RQ_1: *What types of dependencies are more prone to generate ripple effects?*
Through this research question we explore the three kinds of dependencies among requirements that can lead to ripple effects (i.e., *conceptually overlapping requirements*, *overlapping implementations*, *ripple effects at the source code level*), so as to explore their occurrence in practice (i.e., their average probability to occur). The nature of RQ_1 is exploratory, since answering it can provide an insight on the ripple effects phenomenon at the requirements level. As an outcome of this research question, we provided a ranked list of these three types of

[3] https://users.uom.gr/~a.ampatzoglou/aux_material/RCPM_Calculator.rar

requirements dependencies in terms of importance, and a discussion on the statistical significance of the differences. This outcome can help to improve the proposed metric: if for example one of the dependency types yields similar probabilities to the other, then it can be considered redundant and removed from the calculation of R2EM. Furthermore, the outcome can unveil which kind of dependency is more probable to generate more ripple effects; thus, deserve further investigation as well as more attention from practitioners.

RQ$_2$: *Is R2EM able to identify which requirements are affected by ripple effects?*

This research question will explore the effectiveness of the proposed metric in identifying ripple effects between requirements. R2EM should be able to prioritize the requirements to be maintained, in a similar way to the intuition of the practitioners. To investigate this, we considered the probability of various requirements to be affected, given a change in a selected number of other requirements[4]. This probability was obtained through the R2EM metric and the potentially affected requirements were ranked accordingly. Finally, we contrasted the ranking provided by R2EM to the ranking provided by practitioners. The ability of the metric to accurately predict the expert opinion of practitioners can be useful for two reasons: (a) if there is a large number of requirements, an automatically calculated metric, such as the proposed one can scale much better than the one relying purely on expert opinion; and (b) the metric can guide inexperienced developers, who are not able to reach the level of understanding of experts in the field.

4.2 Case Selection

This study is a holistic multiple case study that has been conducted in an SME in Greece. As cases and corresponding units of analysis of the study, we consider the requirements of the systems under investigation. As case study participants we selected 9 software engineers that are currently working on the maintenance and evolution of the two systems under investigation, which are briefly described below:

- **YDATA** deals with customer management and billing of the national water supplier. It consists of 651 classes (45K lines of code) that have been developed and maintained with 384 commits between 03/03/2015 and 03/03/2017. The system can be decomposed into 6 main sub-systems, each one managing the following entities: (a) *Hydrometers*, (b) *Bills*, (c) *Users*, (d) *Consumption Statements*, (e) *Payments*, and (f) *Alerts to Users*.
- **CREGAPI** deals with managing the register office of cities. It consists of 1,473 classes (100K lines of code) that have been developed and maintained for 851 commits. The system can be decomposed into 8 main sub-systems, each one managing the following entities: (a) *Birth*, (b) *Death*, (c) *Marriage*, (d) *Name-giving*, (e) *Partnership*, (f) *Citizen*, (g) *Reports*, and (h) *Temporal Triggers.*

In traditional information systems (such as our two cases), a large proportion of requirements relies on Create / Read / Update / Delete (CRUD) operations on the entities that the information system handles (González-Aparicio et al. 2016; Kaur and Rani 2015; Truica et al. 2015; Basso et al. 2016). We therefore focus on requirements related to CRUD operations. For YDATA we investigated a sample of 24 requirements, corresponding to the 4 CRUD actions for each of the aforementioned 6 entities (hydrometers, bills, etc.). Similarly, for CREGAPI we considered a sample of 32 requirements for the 8 mentioned entities. The requirements are coded using the name of the system, the first letter of the entity and the first letter of the CRUD action. For example, the requirement that "*Reads a Bill*" in the YDATA is named as: YDATA-BR (see Table 3).

4.3 Data Collection

To answer the research questions mentioned in Section 4.1, we executed the tool-chain described in Section 3.3, and obtained change impact data of all studied requirements for both systems. In Table 3, we present the percentage of commits, in which each requirement has changed (PCRC). From the studied samples, we have omitted requirements that have not changed in the commit history.

[4] The way the requirements are selected is discussed in Section 4.2

Table 3. Requirements Change Frequency

System	Requirements		PCRC	System	Requirements		PCRC
	ID	**Name**			**ID**	**Name**	
CREGAPI	CR-CR	Citizen Read	6,23%	YDATA	YDATA-BR	Bill Read	26,11%
	CR-CU	Citizen Update	5,99%		YDATA-HR	Hydrometer Read	15,67%
	CR-CC	Citizen Create	5,88%		YDATA-LC	Alert Create	13,06%
	CR-MC	Marriage Create	2,94%		YDATA-SC	Statement Create	11,23%
	CR-BC	Birth Create	2,12%		YDATA-UU	User Update	11,23%
	CR-DC	Death Create	2,12%		YDATA-UR	User Read	10,44%
	CR-MU	Marriage Update	1,65%		YDATA-PC	Payment Create	9,92%
	CR-PC	Political Create	1,65%		YDATA-BC	Bill Create	7,31%
	CR-MR	Marriage Read	1,29%		YDATA-HC	Hydrometer Read	6,01%
	CR-PU	Political Update	1,18%		YDATA-UC	User Create	6,01%
	CR-NC	Naming Create	0,71%		YDATA-BU	Bill Update	4,18%
	CR-BR	Birth Read	0,59%		YDATA-SR	Statement Read	3,39%
	CR-BU	Birth Update	0,59%		YDATA-SU	Statement Update	3,39%
	CR-PD	Political Delete	0,59%		YDATA-HU	Hydrometer Update	2,35%
	CR-CD	Citizen Delete	0,47%		YDATA-SD	Statement Delete	2,09%
	CR-DU	Death Update	0,35%		YDATA-PR	Payment Read	2,09%
	CR-DR	Death Read	0,24%		YDATA-HD	Hydrometer Delete	0,78%
	CR-MD	Marriage Delete	0,24%		YDATA-PU	Payment Update	0,52%
	CR-NR	Naming Read	0,24%		YDATA-LR	Alert Read	0,52%
	CR-NU	Naming Update	0,24%		YDATA-CC	Connection Create	0,26%
	CR-PR	Political Read	0,24%				
	CR-BD	Birth Delete	0,12%				

Our dataset consists of 421 rows (190: YDATA and 231: CREGAPI) that represent all pairs of requirements in Table 3. For each pair, we have recorded the following information:

- ***From / To Req***: The `ID` of the requirement that triggers / receives the ripple effect;
- ***PCO / POI / PRE***: The probability of `ToReq` (requirement affected by ripple effect) to change due to changes in the `FromReq` (originating requirement)—a variable for each type of requirements dependency;
- ***R2EM***: The assessed total probability of `ToReq` to change, due to changes occurring in `FromReq`.

Additionally, since based on the study design, we needed to contrast R2EM to the expert opinions of practitioners in OTS, we conducted a workshop with 9 industrial practitioners. The participants have been involved in the original construction and/or maintenance of the two projects. The workshop comprised two parts:

- *Structured interviews*. According to Runeson et al. (Runeson et al. 2009) structured interviews consist of a number of open and/or closed questions and can be similar to questionnaire-based surveys. For the needs of our study, we asked a set of closed questions (in some cases followed by an open question for explanation purposes). The questions were of the following form: "*Please denote how probable you believe it is to change requirements X if you perform a change to the Y requirement, due to a ripple effect [Likert Scale: Very Low – Very High]*". Due to the technical nature of the questions, the participants received the questions on paper and they were asked to write down their answers after working on the respective tasks. The first and the second authors were present during the whole process, so the method can be compared to a supervised questionnaire-based survey (Kitchenham and Pfleeger 1996). The presence of authors in the room

aimed at eliminating the disadvantages of simply distributing a questionnaire, such as the ability for participants to ask for clarifications.

- *Focus group*. During the focus group the answers provided during the structured interviews were discussed, giving the opportunity to clarify potential differences of opinion or disagreements between the participants. The focus group was conducted after the participants had submitted their completed questionnaire, so that they would not be biased when filling in the questionnaires. Additionally, during the focus group, we discussed with the participants the reasoning behind their choices and the role of types of requirements in their change impact (i.e., same entities different actions vs. same actions on different entities).

The workshop organization and the questions used in the interviews and focus group are presented in Appendix A. Due to the limited time that participants were available, it was not possible to validate all pairs of requirements; therefore, we had to make a selection of pairs. To do this, we first calculated R2EM for the two projects YDATA and CREGAPI. Using the calculated metrics (R2EM), we selected a set of ranked pairs of requirements per project (ordered by the probability of the ripple effect to occur). Specifically, we selected four requirements from each project (based on their change frequency, see Table 3)—two frequently changing, one with medium and one with low frequency of change. Selecting requirements from different levels of change frequency, allows our results to be more representative. We preferred to select two frequently changing requirements, since we deem them important starting points for CIA: requirements that are rarely modified are probably not easy to remember since they are not used regularly. The selected requirements are presented in Table 4. For each one of these requirements, we listed all other requirements and asked participants to evaluate how probable they believe they are to change the latter because of maintenance request related to the former (1=min-5=max). In total, 76 questions were asked for YDATA (4 requirements paired with 19 others—the remaining ones from Table 3) and 84 for CREGAPI (4 requirements paired with 21 others—the remaining ones from Table 3). To assess the evaluators' agreement, we used two-way mixed inter-ratter reliability calculated through the intra-class correlation coefficient (ICC) (Field 2013). ICC is a descriptive statistic that can be used as a reliability measure, in the sense that it describes how strongly units (*in our case*: evaluations from different experts) in the same group (*in our case*: for the same pairs of requirements) resemble each other (*in our case*: share a common opinion on the ripple effect proneness). The reliability for the YDATA project has been calculated as 80.0% and as 77.4% for CREGAPI. The ICC for each requirement is presented in Table 4. It can be observed that, in most of the cases, the participants were consistent with their answers (sig. <0.01)—the outcome for ICC is statistically significant; the only exception is Statement Create, which we discuss separately while interpreting the results.

Table 4. Intra-Class Correlation

System	From Requirement		ICC	sig.
	ID	Name		
YDATA	YDATA-BR	Bill Read	87.4%	.00
	YDATA-LC	Alert Create	51.1%	.11
	YDATA-SC	Statement Create	57.9%	.01
	YDATA-PC	Payment Create	90.9%	.00
CREGAPI	CR-CC	Citizen Create	81.5%	.00
	CR-BC	Birth Create	72.9%	.00
	CR-MU	Marriage Update	80.3%	.00
	CR-NC	Name-giving Create	72.9%	.00

The compiled dataset on the end of the process is summarized in Table 5. The first column represents the 8 selected FromReqs, whereas the second column the corresponding ToReqs. The third column contains the calculated R2EM score for the pairs (e.g., for the first row the R2EM probability for a change in YDATA-BR to ripple to YDATA-HR). Finally, the fourth column lists the prevalent perception of practitioners on the probability of the ToReq to change due to a ripple effect, because of a change in the FromReq (e.g., for the first row the experts' opinion in the probability for a change in YDATA-BR to ripple to YDATA-HR); this column is calculated as the Mode value of all participants, regarding the corresponding pair of requirements.

Table 5. Dataset for validating R2EM (RQ_2)

From	To	R2EM	Experts' Opinion
YDATA-BR	YDATA-HR	X[1, 1]	Y[1, 1]
YDATA-BR	YDATA-LC	X[1, 2]	Y[1, 2]
…			
YDATA-BR	YDATA-CC	X[1, 19]	Y[1, 19]
…			
CR-CC	CR-CR	X[20, 1]	X[20, 1]
CR-CC	CR-CU	X[20, 2]	X[20, 2]
…			
CR-CC	CR-BD	X[20, 21]	Y[20, 21]
…			

4.4 Data Analysis

To answer the two RQs posed in Section 4.1, we analysed the data as follows. To answer **RQ_1 on the types of dependencies prone to generate ripple effects**, we first created *Descriptive Statistics for the Dataset*. Specifically, we provided the list of the most change-prone requirements, due to ripple effects from both projects. Subsequently we performed a *Comparison among the three Types of Requirements Dependencies* (i.e., conceptual overlapping, overlapping implementations, and source code ripple effects). To this end, we presented descriptive statistics (i.e., mean, mix, max, and standard deviation) for the probability of the three kinds of requirements dependencies to produce ripple effects. Next, we performed hypothesis testing to check the existence of statistically significant differences among them (Field 2013). To provide even more insights in which requirements tend to be affected by ripple effects, we performed a second level analysis, considering the CRUD operations on the main entities for each system (Bills, Accounts, etc. for YDATA, and Births, Deaths, etc. for CREGAPI). Specifically, we considered two types of conceptual relations: *relations due to working on the same entity* (same first letter in requirements ID), and *relations due to performing the same (CRUD) action on different entities* (same second letter in requirements ID).

*To answer **RQ2 on the validation of the R2EM Metric***, we performed a consistency analysis by investigating the ability of the metric to accurately rank a set of components, based on their levels of quality. Note that we consider the notion of consistency as defined in the IEEE-1061 standard (2009). To assess consistency validity of R2EM, we calculate the Spearman correlation between the expert assessment and R2EM scores, as presented in Table 5 organized by FromReq. The reported correlation coefficient stands for the ability of R2EM to accurately (based on experts' opinion) rank the ToReq, with respect to their probability to change due to a ripple effect, cause by the FromReq.

5. Results

In this section we present the results of our study organized by research question. In addition to the actual results, we provide some initial interpretations of the results, whereas an overall discussion of the results is pro-

vided in Section 6.1. First, we present some descriptive statistics. In particular, in Tables 6 and 7, we present the top-10 ripple effects in each of the examined systems. For example, regarding the YDATA system, we can observe that a change in the way that a Payment is created (`YD-PC`) has a probability of approximately 48% to ripple to the way that a Bill is read (`YD-BR`); this is mostly because of their conceptual (PCO) and implementation (POI) overlap. One of the practitioners confirmed this finding: "*it is obvious that whenever we receive a change in the way that a payment is created, we will need to check the way that we read the bill statement*".

Table 6. Intra-Class Correlation for YDATA

To	From	R2EM	PCO	POI	PRE
YD-BR	YD-PC	47.98%	29.89%	39.06%	26.84%
YD-BR	YD-HR	47.61%	14.80%	39.00%	34.25%
YD-HR	YD-UR	47.61%	8.03%	41.81%	32.55%
YD-HR	YD-SC	47.56%	11.29%	41.90%	30.50%
YD-HR	YD-UU	47.14%	5.80%	39.45%	34.54%
YD-SC	YD-HR	47.19%	0.91%	39.44%	33.83%
YD-BR	YD-SC	46.60%	13.97%	38.44%	29.47%
YD-BR	YD-UR	46.26%	3.57%	37.88%	33.23%
YD-UR	YD-HR	46.46%	13.71%	38.07%	29.53%
YD-BR	YD-BU	46.07%	5.19%	38.97%	30.09%

Table 7. Intra-Class Correlation for CregAPI

To	From	R2EM	PCO	POI	PRE
CR-CC	CR-CU	31.78%	2.65%	19.04%	13.33%
CR-CU	CR-CR	29.15%	1.36%	15.05%	15.23%
CR-CC	CR-CR	25.03%	1.11%	13.91%	11.72%
CR-CU	CR-CD	22.16%	0.00%	11.17%	12.38%
CR-CC	CR-CD	21.25%	0.00%	12.39%	10.11%
CR-CR	CR-CD	17.37%	0.00%	9.33%	8.87%
CR-CC	CR-MC	14.87%	0.00%	10.09%	5.30%
CR-CC	CR-DC	14.72%	0.65%	10.42%	4.18%
CR-MC	CR-PC	14.64%	2.31%	9.00%	3.98%
CR-CC	CR-BC	12.89%	0.00%	8.28%	5.02%

For the case of CREGAPI, we observe that the Citizen entity is dominant among the ripple-effect prone requirements. The centrality of the role of Citizen has been vividly explained by one participant as follows: "*all transactions are based on the citizen entity; citizens are born, given a name, getting married, and eventually die. All the certificates issued for these actions are related to the citizen. Thus, any change on the citizen affects the whole of the system"*.

5.1 Proneness to Ripple Effects for each Kind of Requirements Dependencies (RQ_1)

In this section, we present the results on the comparison of the three kinds of requirements dependencies that can trigger ripple effects. From Table 8 we can observe that *Overlapping Implementations (POI)* is the kind of dependency that is mostly responsible for the emission of ripple effects (denoted with grey cell shading), followed by *Code Ripple Effects (PRE)* and then *Conceptual Overlapping (PCO)*. An interesting observation that can be made by comparing the results between the two projects (Table 8), is that this ranking is consistent in both projects. Additionally, the YDATA system presents on average a higher ripple-effect proneness, compared to CREGAPI. This is probably due to the smaller size of the system, but with a similar number of requirements.

Table 8. Descriptive Statistics on the R2EM for each kind of Requirements Dependency

	CREGAPI				YDATA			
Metrics	**Min**	**Max**	**Mean**	**SDev**	**Min**	**Max**	**Mean**	**SDev**
PCO	0.00%	20.59%	0.212%	1.178%	0.00%	29.89%	1.547%	3.350%
POI	0.00%	43.35%	2.603%	5.179%	0.00%	41.90%	14.262%	11.307%
PRE	0.00%	32.42%	1.487%	3.352%	0.00%	34.54%	8.353%	8.315%

To investigate if the aforementioned mean values are representing significant differences, we analysed the variance of the variables through ANOVA. For both systems, ANOVA indicated that the three kinds of requirements dependencies lead to different probabilities of ripple effect scores. To bilaterally compare the kind of dependencies, we have performed a Wilcoxon Rank test. The results are presented in Table 9. The first column of Table 9 presents the compared kind of dependencies, the second, the third, and the seventh columns demonstrate in how many cases each kind of dependency is higher (Neg. Ranks suggest that the 2nd kind is higher, etc.). Col-

umns 5, 6, 9 and 10 represent the results of the Wilcoxon Rank test (Z and sig.). Similarly, the results are consistent among projects, and all differences have proven to be statistically significant.

Table 9. Hypothesis Testing for CREGAPI (N=506) and YDATA (N=418)

		CREGAPI				YDATA			
		N	**Rank**	**Z**	**Sig.**	**N**	**Rank**	**Z**	**Sig.**
POI – PCO	Neg. Ranks	4	158.75	-18.452	.00	0	.00	-18.452	.00
	Pos. Ranks	460	233.14			412	206.50		
	Ties	42				6			
PRE – PCO	Neg. Ranks	16	209.94	-15.633	.00	15	173.10	-15.633	.00
	Pos. Ranks	375	195.41			390	204.15		
	Ties	115				13			
PRE – POI	Neg. Ranks	387	250.35	-14.989	.00	408	209.43	-14.989	.00
	Pos. Ranks	76	138.57			5	9.10		
	Ties	43				5			

Next, we focus on the conceptual relations among requirements, based on the CRUD categorization (as discussed in Section 4.4). In Table 10, we present descriptive statistics on the *kind of requirements dependencies* and the R2EM metric, for the aforementioned relations: working on the same entity and performing the same action (CRUD) on different entities. We note that in this analysis, we treat the complete dataset as a whole. The relation with the most intense ripple effects is denoted with grey cell shading.

Table 10. Requirements Relations—Descriptive Statistics

Metric	**Relation**	**Min**	**Max**	**Mean**	**SDev**
R2EM	Same Entity	0.12%	46.19%	13.18%	13.91%
	Same Action	0.00%	47.61%	10.96%	13.41%
PCO	Same Entity	0.00%	19.58%	1.14%	2.95%
	Same Action	0.00%	17.07%	0.85%	2.46%
POI	Same Entity	0.12%	38.64%	9.36%	10.57%
	Same Action	0.00%	41.81%	8.10%	10.60%
PRE	Same Entity	0.00%	31.76%	5.58%	7.14%
	Same Action	0.00%	34.25%	4.85%	7.49%

Based on the findings of Table 10, requirements working on the same entity are more probable to trigger ripple effects. However, the hypothesis testing suggested that this result is not statistically significant. This observation was discussed by one practitioner as follows: "*On the one hand, working on the same entity inevitably creates ripple effects, since a change in the number of fields in an entity affects all CRUD actions. On the other hand, activity-related requirements are also prone to ripple effect, since in many cases there is a sequence in the actions: the birth of a person leads to the creation of a citizen and the creation of a birth certificate. These two Create actions are almost always maintained in the same time or under a common transaction*".

The ranking of the kind of requirements dependencies from more to less frequent is as follows (the ranking is statistically significant): (a) Dependencies due to Overlap in Requirements Implementations (assessed through ***POI***); (b) Dependencies due to code ripple effects of Requirements Implementations (assessed through ***PRE***); and (c) Conceptual Dependencies between Requirements (assessed through ***PCO***).

5.2 Validation of the R2EM Metric (RQ_2)

In this section, we present the results on evaluating the efficiency of the proposed metric in assessing pairs of requirements, with respect to their probability to emit a ripple effect. In Table 11, we present the correlation for the complete dataset as a whole, and per project. Based on the results we can observe that for both projects, the

ranking of the metric is strongly correlated (coeff. > 0.6 (Marg et al. 2014)) to the opinions of practitioners. Therefore, adequate correlation is achieved. Regarding reliability at the project level (i.e., if the two projects have similar results), the results on the CREGAPI project are better, compared to YDATA, an observation that can be explained due to the smaller size of the YDATA project. Nevertheless, the fact that the difference is small suggests that the metric is scalable, since doubling up the number of requirements and classes, costs less than 1% correlation strength. Thus, the consistency assessments can be considered reliable at the project level.

Table 11. Consistency and Reliability of R2EM

Requirements	Spearman	
	Coeff.	Sig.
Complete Dataset	60.6%	0.000
CREGAPI	63.4%	0.000
YDATA	62.6%	0.000

In Table 12, we present the correlation of R2EM with the priority that practitioners assigned to the pairs of requirements (we remind that only a subset of requirements has been explored as part of RQ_2, due to time limitations—see Section 4.3). We further observe that requirements can be divided into two main categories, denoted with grey- and white-cell shading (using the 0.7 threshold for strong correlations (Marg et al. 2014)). On the one hand, the pairs of requirements with the grey-cell shading are those for which R2EM proves to be the most accurate. This efficiency can be explained by the fact that practitioners consider the specific pairs of requirements to have straightforward (or highly probable) ripple effects. For example, consider the following statements about pairs of requirements that have been ranked very high from R2EM and deemed as having almost certain ripple effects by practitioners:

- (From: `Hydrometer Read`, To: `Statement Create`). "*To automatically create a statement, the system has to read the data from a smart hydrometer. Therefore, any change in the way that input is received from the device, may emit changes in the way that a statement is initialized*".
- (From: `Bill Read`, To: `Bill Update`). "*The relation here, is due to the use of the common entity. In particular, if the fields that characterize a bill change, then both requirements, will need to be updated. This is more or less a bi-directional relation*".
- (From: `Birth Create`, To: `Citizen Create`). "*The two requirements are heavily coupled, in the sense that a citizen is created upon his/her birth. Therefore, any change that is made on the fields that we use to declare a birth is automatically transferred to the newly created citizen*".

Table 12. Validation of R2EM per Requirement

System	From Requirement of the Pair	Spearman	
		Coeff.	Sig.
YDATA	Bill Read	78.7%	0.012
	Alert Create	61.3%	0.015
	Statement Create	69.4%	0.003
	Payment Create	76.8%	0.017
CREGAPI	Citizen Create	78.4%	0.000
	Birth Create	58.0%	0.019
	Marriage Update	66.4%	0.000
	Namegiving Create	54.5%	0.044

On the other hand, the pairs of requirements exhibiting a lower correlation with R2EM (55% - 69%) are those for which the practitioners had contradictory opinions between themselves as well. For example, regarding YDATA the agreement of practitioners on the requirements affected by a change in the way alerts are created is 51.1% (see Table 4) while the correlation of R2EM to the average expert opinion is 61.3%. A possible explana-

tion for the deviation is the way that practitioners perceive requirements ripple effects: "*Different people perceive each case in a different way, either because they have in mind different parts of the system (not all of us work on all parts of the system, although we have a generic idea of what each requirement has to do with), or because we consider different extension scenarios, based on our most recent experiences*".

The ranking that R2EM provides, with respect to the proneness of a pair of requirements to emit / receive a ripple effect, is strongly correlated (62%-63%) to the expert opinion of practitioners. The difference between the two studied systems indicates potential for R2EM to scale, since the system that was half the size of the other (in terms of requirements and source code size) had only 1% less correlation strength.

6. Discussion

Interpretation of Results. The case study reported in this paper had two main goals: (***g1***) assessing the kind of requirements dependencies that are more probable to produce ripple effects; and (***g2***) the validation of the proposed metric. Regarding (***g1***) two types of analysis were performed: (a) based on the kind of requirements dependencies (i.e., *conceptual overlapping*, *overlapping implementations*, and *ripple effects of these implementations*), and (b) based on the requirements relations (e.g., are ripple effects more common among requirements working on the same entity?). Based on our results, we observe the following:

- ***Requirements implementations vs. conceptual relevance***. The implementation of requirements appears to be more important with respect to ripple effects compared to the requirement contract. In particular, the probability to change due to a ripple effect, because of overlapping implementations (POI) has the highest probability to produce ripple effects, followed by PRE (probability due to ripple effects at implementation level). On the other hand, ripple effects between requirements due to their conceptual overlap (PCO) has proven to be rarer (1.1% - 3.3%), compared to POI (5.1% – 11.3%) and PRE (3.3% – 8.3%). These findings can be considered reasonable in the sense that conceptual overlap among requirements is a more abstract type of dependency compared to dependencies in the actual source code. This is good news in terms of preventing ripple effects: excessive dependencies among requirement implementations can be potentially avoided adhering to the Single Responsibility Principle, whereas conceptually related requirements are often imposed by the problem domain and cannot be avoided.
- ***Central entities***. In both examined systems, we have observed that some central entities (e.g., the Citizen in the municipality application) have been identified, and any change in requirements related to such entities, is highly probable to affect many parts of the system. We argue that high coupling poses a big risk for requirements specification as well, in the sense that requirements associated with many other requirements or with key elements of the architecture, are highly probable to be the cause of ripple effects. Breaking down 'god' requirements to finer-grained and less coupled ones, can in theory contribute to less change propagation among them.
- ***Entity- vs. action-related requirements***. Requirements affecting the same entity are more probable to produce a ripple effect (PCO) compared to requirements performing the same action. However, both requirements relation types seem to have a similar probability to experience ripple effects, due to implementation issues (POI and PRE). Nevertheless, both types have been validated as important by the practitioners.

Regarding (***g2***) the results of the empirical validation suggested that the proposed metric R2EM is a valid assessor of requirements ripple effect, and can serve as a means for predicting requirements ripple effects. In particular, R2EM exhibited a strong correlation with experts' opinion (approx. 60%). Also, statistically significant discriminative power can be achieved by using this metric. In principle, the aforementioned correlation is stronger for pairs of requirements whose relation is stronger according to practitioners (i.e., a high-level of agreement on high values between practitioners). The validity of the R2EM metric suggests that both the probability of a re-

quirement change happening and the propagation of changes among requirements are factors that practitioners deem as important for the criticality of requirements.

Implications for Researchers and Practitioners. Based on the above, we can derive some advice for *practitioners*. First, for cases in which the proposed tool-chain is applicable, we encourage them to use the suggested toolset so as to guide them along software maintenance. To ease the adoption of R2EM, based on the suggestions of our case study participants, we encourage the integration of the tool or any similar approach in the IDE that each company is using. Second, in case the proposed tool-chain is not applicable (e.g., not Java, or no Git for version control), we are still able to guide software maintenance, based on the findings of the empirical analysis on the proneness of each kind of requirements dependencies on ripple effect. In particular, we advise practitioners to inspect for co-maintenance, based on entity- and then activity-similarity. Additionally, we encourage practitioners to identify the central entities in the systems that they maintain, since for them, high maintenance effort would be required, due to massive ripple effects.
On the other hand, some interesting future work opportunities have been identified for *researchers*. First, the ability of R2EM to successfully guide maintenance activities through a longitudinal case study is required. For such a case, researchers working with an industry partner could use the suggestions of the tool for a long period and evaluate: (a) the required maintenance effort; (b) the maintenance efficiency (e.g., number of bugs identified, effort needed for new features, etc.) when serving maintenance tickets, (c) the effectiveness of R2EM (i.e., if the correct artifacts that need to be changed are identified) based on the proposed suggestions.

Additional validation could be performed by exploring the applicability of R2EM in terms of intuitiveness and usefulness. Second, there is a need to assess the predictive power and the tracking ability of the proposed metrics, since we were not able to validate them in the proposed setting. Third, replications with different programming languages, and version control systems would be required. Finally, an interesting extension scenario would be to tailor the proposed metric to non-object-oriented paradigms, for example, by considering files or folders as units of analysis.

7. Threats to Validity

In this section, we present and discuss potential threats to the validity of our case study (Runeson et al. 2009). Internal validity is not considered, since we have not dealt with causal relations.

Construct Validity. A possible threat to construct validity is related to the accuracy of the proposed metric and the developed tool-chain to assess requirements ripple effect. Such a threat is classified as construct validity in the sense that inaccurate results might lead to measuring a different phenomenon than the one originally intended to investigate. Concerning the rationale of the metric's calculation, we note that its definition is clear and well-documented (see Section 3), whereas the used tools have been thoroughly tested, before deployment, in a large number of open-source projects (see Section 3.3). Regarding the metric, we consider this threat mitigated in the sense that the provided empirical validation suggested that the proposed measure is an accurate assessor of requirement ripple effects (see Section 5). By further focusing on each probability, we acknowledge as a threat the fact that for POI, we only consider part of the requirements evolution (i.e., those in which only one requirement is changing). This decision might lead in losing traces between requirements and source code. However, we note again that this decision has been made so as to guarantee the independence of PCO and POI. We believe that threatening the independence of two parameters would be more severe for the validity of R2EM, and therefore, we opted to omit commits in which more than one requirement have changed due to ripple effects. Another threat to construct validity stems from the need to calculate the Percentage of Commits in which a specific Requirement has Changed (PCRC) based on commit messages in order to track the requirement(s) that are being affected. Lack of proper messages implies that PCRC will not be accurately calculated.

Nevertheless, an experienced developer would be able to identify the associated requirements even from the commit files if he wishes to apply the proposed methodology.

Moreover, the case study participants may have a different background and experience on specific requirements and thus influence the ranking that they performed. To avoid this threat, we involved four and five employees respectively for each project, who were all familiar with a large portion of the system. However, it is possible that participants have a different perspective of ripple effects, due to the different parts of the code-base that they maintain. To mitigate this risk, we calculated their agreement rate (see Table 4). Specifically, we observed that for both systems high agreement between participants occurs. A detailed discussion on this issue is presented in Section 4.3, focusing on specific pairs of requirements with lower levels of agreement.

Reliability. With regard to reliability, we consider any possible researchers' bias, during the data collection and data analysis process. The design of the study concerning data collection does not contain threats, since the material provided to the participants included the source code of the company and rankings of requirement-pairs, as they have been created automatically by a tool. Additionally, the researchers themselves were not required to interpret the results at any point, since the participants were answering the tasks on paper. Moreover, with respect to the data analysis process: (a) although the quantitative part is not subject to bias, in the sense that statistical analysis has performed; the analysis has been independently performed by the first two authors and the results have been cross-checked (b) with respect to qualitative analysis, potential threats to reliability have been to some extent mitigated since two researchers were involved in the process, aiming at double checking the work performed and thus reducing the chances of reliability threats.

External Validity. Concerning external validity, a potential threat to generalization is the possibility that performing the study on different requirements of different companies might affect results of the assessment. Thus, results cannot be generalized to large-scale systems and domains other than enterprise applications. Additionally, in this study we investigated projects written in Java due to the corresponding tool limitations. Therefore, the results cannot be generalized to other languages, e.g., C++. Moreover, we note that our results are not applicable to non-object-oriented systems, since our definition of ripple effects at source code level applies only in this programming paradigm. Furthermore, our metric is not applicable for projects that are not hosted in version control management systems, since the calculation of PCCC requires access to the complete development history of the project. Finally, with the respect to the provided toolchain, we need to note that its applicability cannot be guaranteed to all systems, in the sense that it considers a specific type of annotating the link between requirements and code artifacts, through commit messages. However, this does not threaten the applicability of the method, in the sense that given any kind of traceability between requirements and code, the first step of the toolchain can be replaced, with an adequate tool support.

8. Conclusions

Change impact analysis at the requirements level can prove extremely useful during software maintenance, in the sense that the artifacts that need to be changed, due to a maintenance ticket, are not only those associated with updated requirements, but also those that have been potentially affected due to ripple effects. Despite the existence of some metrics at the design and source code level on the quantification of the ripple effect, existing literature lacks such metrics at the level of requirements. In this paper, we introduce such a metric (namely R2EM) by considering several scenarios that can lead to ripple effects between requirements, such as conceptual overlapping and structural dependencies between their implementations. The metric has been validated in an industrial setting, based on the guidelines for metric validation provided by the 1061-1998 IEEE Standard. In particular, we analysed the source code and the commit history of two industrial products, recorded the strength of relations between requirements, and contrasted them with experts' opinion. The results of the study suggested that the proposed metric is capable of assessing requirements ripple effects at a satisfactory level, and that the most common reason for ripple effects between requirements lies at the implementation level. The results of the

study, including both the metric per se (and accompanying tool), and the empirical findings on the kinds of requirements dependencies that can lead to requirements ripple effects are expected to be useful in both academia and software development industry, since many useful implications to researchers and practitioners have been extracted.

Aknowledgements

This work was financially supported by the action "Strengthening Human Resources Research Potential via Doctorate Research" of the Operational Program "Human Resources Development Program, Education and Lifelong Learning, 2014-2020", implemented from State Scholarship Foundation (IKY) and co-financed by the European Social Fund and the Greek public (National Strategic Reference Framework (NSRF) 2014 – 2020).

References

1061-1998: IEEE Standard for a Software Quality Metrics Methodology. *IEEE Standards. IEEE Computer Society*. reaffirmed Dec. 2009.

Ali, N., Jaafar, F., & Hassan, A. E. (2013). Leveraging historical co-change information for requirements traceability. *20th Working Conference on Reverse Engineering (WCRE' 13)*. Germany.

Ampatzoglou, A., Chatzigergiou, A., Charalampidou, S., & Avgeriou, P. (2015). The Effect of GoF Design Patterns on Stability: A Case Study. *Transactions on Software Engineering*. IEEE. 41 (8). pp. 781 – 802.

Antoniol, G., Canfora, G., Casazza, G. &De Lucia, A. (2000). Identifying the starting impact set of a maintenance request: a case study. *4th European Conference on Software Maintenance and Reengineering*. Zurich. Switzerland.

Arora, C., Sabetzadeh, M., Goknil, A., Briand, L. C., & Zimmer, F. (2015). Change impact analysis for Natural Language requirements: An NLP approach. *23rd International Requirements Engineering Conference (RE)*. Ottawa.

Arvanitou, E. M., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (2015). Introducing a ripple effect measure: a theoretical and empirical validation. *9th International Symposium on Empirical Software Engineering and Measurement (ESEM '15)*. IEEE. China.

Arvanitou, E. M., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (2017). A Method for Assessing Class Change Proneness. *21st International Conference on Evaluation and Assessment in Software Engineering (EASE '17)*. ACM. Sweden.

Arvanitou, E. M., Ampatzoglou, A., Tzouvalidis, K., Chatzigeorgiou, A., Avgeriou, P., & Deligiannis, I. (2017). Assessing Change Proneness at the Architecture Level: An Empirical Validation. *1st International Workshop on Emerging Trends in Software Design and Architecture (WETSoDA '17)*. Nanjing. China.

Basso, F. P., Pillat, R. M., Oliveira, T. C., Roos-Frantz, F., & Frantz, R. Z. (2016). Automated design of multi-layered web information systems. *Journal of Systems and Software*. 117. pp. 612-637.

Beck, K., & Cunningham, W. (1989). A laboratory for teaching object oriented thinking. *Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '89)*. pp. 1-6. USA.

Buse, R. P. L., & Weimer, W. R. (2010). Automatically documenting program changes. *International conference on Automated software engineering (ASE '10)*. pp. 33-42. Belgium.

Charalampidou S., Ampatzoglou A., Karountzos E., Avgeriou P (2020). Empirical studies on software traceability: A mapping study. *Journal of Software: Evolution and Process*, 32 (11). Wiley and Sons.

Chen, J.-C. & Huang, S.-J. (2009). An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software*. 82(6). pp. 981–992.

Conejero, J. M., Figueiredo, E., Garcia, A., Hernández, J., & Jurado, E. (2012). On the relationship of concern metrics and requirements maintainability. *Information and Software Technology*. Elsevier, 54 (2). pp. 212-238.

Dahlstedt, A. G. & Persson, A. (2005). Requirements Interdependencies: State of the Art and Future Challenges. *Engineering and Managing Software Requirements*. Springer. pp 95-116.

Field, A. (2013). Discovering Statistics using IBM SPSS Statistics. *SAGE Ltd.*

Fowler, M. (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language. *Addison-Wesley Professional*. 3rd Edition.

Galorath, D. D. (2008). Software total ownership costs: development is only job one. *Software Tech News*. 11(3).

Goknil, A., Kurtev, I. & van den Berg, K. (2008). Change impact analysis based on formalization of trace relations for requirements. *4th ECMFA Traceability Workshop*.

Goknil, A., Kurtev, I., van den Berg, K. & Spijkerman, W. (2014). Change impact analysis for requirements: A metamodeling approach. *Information and Software Technology*. 56 (8). pp. 950-972.

González-Aparicio, M. T., Younas, M., Tuya, J. & Casado, R. (2016). A New Model for Testing CRUD Operations in a NoSQL Database. *30th International Conference on Advanced Information Networking and Applications (AINA)*, Crans-Montana.

Hassine, J., Rilling, J., Hewitt, J., & Dssouli, R. (2005). Change impact analysis for requirement evolution using use case maps. *8th International Workshop on Principles of Software Evolution (IWPSE'05)*. Lisbon. Portugal.

ISO/IEC 9126-1. (2001) Software engineering - Product quality (Part 1: Quality model), Geneva, Switzerland.

Kagdi, H., Maletic, J. & Sharif, B. (2009). Mining software repositories for traceability links. *15th International Conference on Program Comprehension (ICPC '07)*. IEEE. pp. 145 –154.

Kaur, K., & Rani, R. (2015). Managing Data in Healthcare Information Systems: Many Models, One Solution. *Computer*. 48 (3). pp. 52-59.

Kitchenham, B. & Pfleeger, S. L. (1996). Software quality: the elusive target. *IEEE Software*. IEEE. 13 (1). pp. 12 – 21.

Krestou M., Arvanitou E. M., Ampatzoglou A., Deligiannis I., and Gerogiannis V., "Change Impact Analysis: A Systematic Mapping Study", Journal of Systems and Software, Elsevier, vol. 173, March 2021.

Marg, L., Luri, L.C., O'Curran, E. & Mallett, A. (2014). Rating Evaluation Methods through Correlation. *1st Workshop on Automatic and Manual Metrics for Operational Translation Evaluation (MTE '14)*. Reykjavik. Iceland.

Nejati, S., Sabetzadeh, M., Arora, C., Briand, L. C. & Mandoux, F. (2016). Automated Change Impact Analysis between SysML Models of Requirements and Design. *24th International Symposium on Foundations of Software Engineering (FSE'16)*, Seattle, USA.

Queille, J-P., Voidrot, J-F., Wilde, N., & Munro, M. (1994). The Impact Analysis Task in Software Maintenance: A Model and a Case Study. *International Conference on Software Maintenance*, Victoria, Canada.

Rahman, M. A., Razali, R. & Singh, D. (2014). A Risk Model of Requirements Change Impact Analysis. *Journal Of Software*. 9(1). pp. 76-81.

Runeson, P., Höst, M., Rainer, A. & Regnell, B. (2009). Case Study Research in Software Engineering: Guidelines and Examples. *John Wiley and Sons*. Inc.

Spinellis, D., Gousios, G., Karakoidas, V., Louridas, P., Adams, P. J., Samoladas, I. & Stamelos, I. (2009). Evaluating the Quality of Open Source Software. *Electronic Notes in Theoretical Computer Science (ENTCS)*. ACM. 233 (3). pp. 5-28.

Truica, C., Radulescu, F., Boicea, A., & Bucur, I. (2015). Performance Evaluation for CRUD Operations in Asynchronously Replicated Document Oriented Database. *20th International Conference on Control Systems and Computer Science*. Bucharest.

Zhang, H., Li, J., Zhu, L., Jeffery, R., Liu, Y., Wang, Q. & Li, M. (2014). Investigating dependencies in software requirements for change propagation analysis. *Information and Software Technology*. Elsevier. 56 (1). pp. 40-53.