Proposing and empirically validating change impact analysis metrics

Arvanitou, Elvira Maria

Based on: E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, and N. Tsiridis — "*Change Impact Analysis at the Requirements Level for Test Case Prioritization*", IEEE Transactions on Software Engineering, (Under Review).

Chapter 7 – A Metric for Requirements Change Proneness

Test case prioritization entails identifying the tests that need to be executed to ensure the correctness of the software before deployment. In particular, teams need to execute tests for the requirements that have been modified in the current development iteration and for those that can potentially be affected by these changes (even indirectly). In this paper, we propose a method for assessing the probability of one requirement to be affected by a change in another requirement (i.e., requirements ripple effect), as an indicator of its priority to be tested. The method considers the change history of the involved requirements (i.e., their co-change due to the specification of the change request), 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), leading to the calculation of the Requirements Ripple Effect Measure (R2EM). To validate the proposed metric, we conducted an industrial case study. The results suggested that the method is able to prioritize test cases at a satisfactory level, and that ripple effects between requirements are mostly caused by overlapping contracts (dealing with the same entities) and ripple effects between classes implementing related requirements.

7.1 Motivation

Automated code testing is usually performed through regression testing that guarantees the execution of an adequate number of test cases. However, performing all the required regression tests is not realistic, due to resource and time limitations. Thus, there is a need for efficient prioritization of test cases, so that the execution of a subset of all available tests can achieve adequate test coverage of the source code, ensuring the identification of a high ratio of underlying defects.

The test cases that need to be prioritized at the end of development iterations are those associated with the requirements that were changed or added during the iteration. As a consequence, change impact analysis between requirements

needs to be performed (Rahman et al., 2014). In particular, a requirement might need to be re-tested, due to *ripple effects* (i.e., changes that occur due to changes in other artifacts) from other requirements. For example, while bug-fixing a requirement, source code associated with other requirements might be affected and require re-testing. Consider the example illustrated in Figure 7.1.a, where a change in requirement Req-1, requires the execution of test cases TC1 to TC3, but also TC4 to TC11, since it directly or indirectly affects four other requirements (i.e., Req-2, Req-3, Req-4, and Req-5).

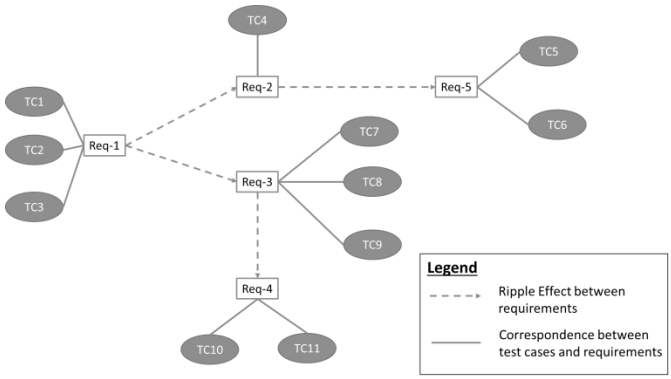

Figure 7.1.a: Propagation of Bugs among Requirements

The reasons that can lead to ripple effects between requirements can be categorized into two main categories, as described below (for more details see Chapter 7.3):

(a) ***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 expressed or in terms of a similar underlying idea of what the system should be able to perform. For example, the requirements "*The system shall support the management of library items*" and "*The system shall provide means to handle books and journals within the library*" are overlapping since both books and journals could be considered as library items.

(b) ***Dependencies between implementations of requirements***— This if further categorized into:

(b1) ***Overlap in requirements implementations***—the implementations of two or more conceptually independent requirements overlap, e.g., under a common set of classes;

(b2) ***Instability of requirements implementations due to source code ripple effects***—the classes that implement a specific requirement might emit changes, due to structural dependencies to classes implementing other requirements.

Given the existence of ripple effects between requirements, we argue that each pair of requirements can be assigned a metric score that characterizes the probability with which one requirement might need to change, due to changes occurring to the other requirement. The larger this score, the larger the probability that changes are made and that new defects are introduced in the implementation of the affected requirement. Therefore, the probability of a ripple effect to occur, can act as an indicator for test case prioritization: based on the requirements that have changed, we can determine which additional requirements to test. The aforementioned probability depends on the strength of the relationship between the requirements. For example, a set of requirements whose implementations are overlapping by 80% is more prone to produce a ripple effect compared to a set of requirements with 20% implementation overlap (assuming that the rest of the parameters are the same).

Therefore, the goal of this paper is to propose a method for calculating the requirements ripple effect probability, which can act as an indicator for testing a requirement, due to a change in other requirements. In particular, the method calculates the ***Requirements Ripple Effect Metric (R2EM)*** for each pair of systems requirements, which can be used as an indicator of test case prioritization (see Chapter 7.3). The validity of the metric is evaluated in an industrial setting, involving two medium-size systems, by considering expert opinions (i.e., quality mangers and developers).

The organization of the rest of the paper is as follows: In Chapter 7.2 we present related work, whereas in Chapter 7.3 we present in detail the rationale of the proposed approach. In Chapter 7.4, we discuss the industrial case study design, whereas in Chapter 7.5 we present the results of the study. Next, in Chapter 7.6 we discuss the main findings, and in Chapter 7.7 the threats to validity. Finally, in Chapter 7.8 we conclude the paper.

7.2 Related Work

In literature, one can identify studies that aim at quantifying ripple effects or assessing change proneness and stability (the quality attributes that are related to the ripple effect) at different levels of granularity and development phas-

es. According to a recent mapping study on design-time quality attributes, change proneness and instability have been quantified by eight measures at the implementation level (Yau and J. Collofello, 1985), six at the design level (Black, 2008), but none at the requirements level (Arvanitou et al., 2017a). Nevertheless, in the literature there are some measures of requirements change proneness, which however only work after-the-fact (i.e., they are not predicting change proneness, but measure it after the change has occurred—e.g., number of times a requirement has changed)—see Chapter 7.2.3.

In Chapter 7.2.1, we summarize indirect related work, by presenting studies that attempt to quantify code and design instability, i.e., the ability of the software to remain unchanged, regardless of the changes, occurring to other parts of the system. Next, in Chapter 7.2.2, we present our own previous work on ripple effects and change proneness. We reuse several aspects and notations from our previous work in this paper; we note that our earlier work applies at the class and package level and the only commonality with the proposed approach is the targeted quality attributes, i.e., instability and change proneness. In Chapter 7.2.3, we present studies that deal with change impact analysis at the level of requirements. Finally, in Chapter 7.2.4, we present the main contributions of this paper, compared to related work.

7.2.1 Design and Source Code Change Proneness

In the early '80s Yau and Collofello proposed some measures for *design* and *source code stability*. Both measures were considering the probability of an actual change to occur, the complexity of the changed module, the scope of the used variables, and the relationships between modules (1985). In a more recent study (2007), Black, proposed an approach for calculating complexity-*weighted total variable definition propagation for a module*, based on the model proposed by Yau and Collofello. The approach calculates complexity metrics, coupling metrics, and control flow metrics, and their combination provides the proposed ripple effect measure (Black, 2008). An *empirical study* conducted by Elish and Rine (2003), investigated the *ability of existing metrics* to assess the design stability of classes. The results suggested that coupling metrics (CBO and RFC) are the optimum assessors of class stability, i.e., the reciprocal of the ripple effect. However, the used stability measure was not the actual one, but estimation based on dependencies and attribute sharing. Finally, on a different direction, Alshayeb and Li (2005) propose a system *design instability measure*

that quantifies the actual changes that have been performed from one version of a system to the other.

7.2.2 Our Earlier Work on Change Proneness

In our earlier work on change impact analysis (Ampatzoglou et al., 2015; Arvanitou et al., 2015; Arvanitou et al., 2017b; Arvanitou et al., 2017c), we use a common high-level approach that is instantiated at different levels of granularity and characteristics of the specific artifacts. This high-level approach is based on the fact that any software artifact can change for internal or external events: ***internal*** events refer to changes that occur to the artifact due to maintenance activities that need to be made directly to the artifacts, and ***external*** events refer to changes that propagate from changes to other artifacts of the system. Assuming that the artifact needs to change regardless of which of the two events is triggered, the probability of the artifact to change is calculated as the joint probability of events. In case an artifact can change in the occurrence of two external events (e.g., a class depends upon two classes), change proneness is calculated as follows:

$$\mathbf{CP} = \textit{JointProbability}\ \{P(\text{internal}), P(\text{external}_1), P(\text{external}_2)\}$$

On the one hand, ***internal probability to change*** is estimated based on historical data, as the percentage of commits in which the artifact has changed. On the other hand, the calculation of ***external probability to change*** is more sophisticated. External probability to change is calculated as a joint probability of two factors: (a) the probability of the artifact that emits the change, to actually change (i.e., its ***internal probability to change***), and (b) the probability that the change propagates from one artifact to another through a dependency (namely ***propagation factor***, or ripple effect measure—REM). The calculation of REM is performed in a different way for each type of artifact and development phase.

$$P(A{:}\text{external}_B) = P(A \mid B) \cdot P(B)$$

$P(A \mid B)$ is the **propagation factor** between module B and A (i.e., the probability that a change made in B is emitted to A).

$P(B)$ refers to the **internal probability** of changing module B.

To this end, Ampatzoglou et al. (2015) proposed a way to calculate REM at detailed-design level, and validated the accuracy of the metric through an inde-

pendent study (Arvanitou et al., 2015). Later on, this metric has been applied in the change proneness formula (described above), and the ability of the derived metric to assess change proneness has been validated (Arvanitou et al., 2017b). In the latest study of this series, Arvanitou et al. (2017c), tailored the method to fit the architecture level, and subsequently validated the accuracy of the method. All validations have been performed on Open Source Software (OSS) systems, by applying the 1061 IEEE Standard for Software Quality Metrics (IEEE-1061, 1998). As a control group for comparison, existing coupling metrics at class and package levels have been used.

7.2.3 Change Impact Analysis on Requirements

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. 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. Although, the proposed metric is able of measuring change proneness of a requirement, it is considered an after-the-fact measurement.

Additionally, Loconsole (2002) investigated the ability of 10 requirements management measures to predict the stability and volatility of requirements and change requests. The proposed measures have been validated, however, in a non-empirical manner. Furthermore, Rahman et al. (2014), investigated the reasons (risk factors as mentioned in the paper), that can lead to requirements change. 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.

Finally, Lee et al. (2010) proposed the combination of traceability matrices and design artifacts for performing change impact analysis. The main idea of the paper is similar to ours, since they share the same goal and exploit the link between requirements and later development phase artifacts. However, the approach of Lee et al. lacks empirical validation and is performed on artifacts that are not always available, e.g. Design-Structure-Matrix (DSM). For the sake of completeness, our method depends only on the existence of a list of requirements and a source code versioning system (e.g. Git)—see Chapter 7.3.1.

7.2.4 Contributions of this Study

Compared to the related work presented in this chapter, the current study is the first one which:

- ***models ripple effects among requirements***, and assesses the probability of those effects to occur through a metric;
- proposes and empirically validates a ***metric for requirements change proneness***;
- ***applies the aforementioned metric for test case prioritization***, based on change impact analysis of requirements; and
- ***uses industrial experts' opinion for validating*** the results.

7.3 Requirements Change Proneness Metric

In this paper, we tailor the generic method for assessing software artifact change proneness (see Chapter 7.2.2), so as to fit the requirements level, by calculating the Requirements Ripple Effect Metric (R2EM). Specifically, we present the proposed method In Chapter 7.3.1, an illustrative example in Chapter 7.3.2, and finally the tool-chain that we used for applying the method in Chapter 7.3.3.

7.3.1 Proposed Method

In this chapter, we describe how we calculate the two pillars of the R2EM calculation model i.e., ***internal probability to change*** and ***external probability to change*** (see Chapter 7.2.2).

Regarding ***internal probability to change***, we exploit the version control history of the project. In particular, we reuse the metric proposed by Concero et al. (2012), which calculates the Percentage of Commits in which a specific Requirement has Changed (PCRC). In particular, 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 (since we cannot assume traceability between requirements and source code), but the commit message. Thus, we assume that the process of the organization that uses the method, 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. Finally, many studies aim at the improvement of the quality of commit messages, so as to be even more descriptive (McIntosh et al., 2014; Cortés-Coy et al., 2014). Therefore, the applicability of R2EM can be enhanced by applying the aforementioned methods to improve the quality of commit messages and particularly strengthen their connection to requirements.

Concerning the ***external probability to change (ripple effect)***, we identified three main reasons for change propagation among requirements (see Chapter 1): namely: (a) *conceptually related requirements*, (b1) *requirements with overlapping implementations*, and (b2) *requirements with structurally dependent implementations*. Regarding the first case, the tendency of requirements to co-change due to overlapping contracts (i.e., related to the same concept—the *SimilarTo* requirement dependency as proposed by Dahlstedt and Persson (2005)) has already been discussed in the literature. According to Zhang et al.(2014), requirements that deal with the same data, are highly probable to co-change. In particular, Zhang et al. validated with industrial stakeholders that if the data is to be changed, all the related similar functions may be changed too (Zhang et al., 2014). The second case (b1 and b2), i.e. requirements co-change due to source code dependencies, can be illustrated by several approaches that link requirements and implementation (Ali et al., 2013). For example, Kagdi et al. (2007) 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., belong to the same requirement. As another example, consider Class–Responsibility–Collaboration (CRC) cards (Beck and Cunningham, 1989; Fowler, 2003[33]). In particular, all pairs of responsibilities that are noted in the same CRC card have an overlapping implementation in the specific class (second case); whereas the existence of source code ripple effects can take place through collaborating classes in the CRC (third case).

[33] CRC cards are design elements in agile software development that are developed for every class of the system. Apart from the class name, the card includes the responsibilities of the class (i.e., the requirements that it will implement), and the classes with which it collaborates.

The aforementioned cases of ***external probability to change*** are further discussed below in terms of a simple example system and are visually represented in Figure 7.3.1.a. The example system concerns managing 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 for simplicity, 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.

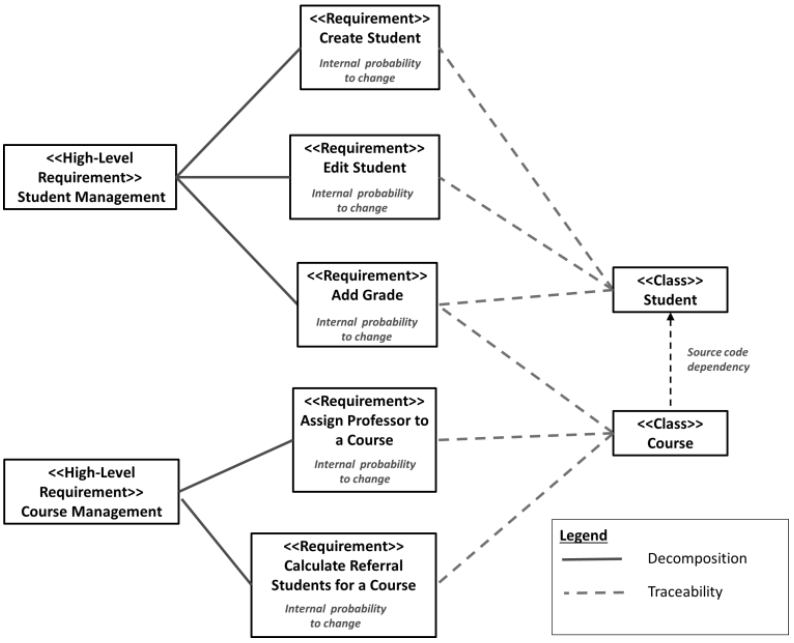

Figure 7.3.1.a: Example System for the External Probability to Change

External probability to change due to Conceptually Overlapping requirements (PCO). Some requirements are co-changing frequently because they are conceptually related, e.g., because they work on the same entity or they perform the same action on different entities. For example, requirements Create Student and Edit Student, are highly likely to co-change, 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 re-tested, so as to ensure the correctness of the validation.

External probability to change due to Overlapping requirements Implementations (POI). Some requirements, although they might not be related in terms of belonging to the same high-level requirement, they might have overlapping implementations. For 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 tests for the implementation of both requirements need to be executed.

External probability to change due to Ripple Effects at the source code level (PRE). 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 resit 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-testing.

As explained in Chapter 7.2.2, the external probability to change is calculated by multiplying the internal probability by the propagation factor. Thus, the external probabilities to change due to overlapping contracts, overlapping implementations, or source code ripple effects are each calculated through the product of their respective ***internal probability to change*** and the ***probability of change propagation***. As ***internal probability to change*** we use the PCRC, for the requirement that emits the change. To calculate each ***probability of change propagation*** (for simplicity we therefore refer to this as *propagation factor*), the following calculations are conducted:

- The probability of one requirement (e.g., ReqA) to change due to conceptual ***overlap with another requirement*** (e.g., ReqB)—$PCO_{ReqA\text{-}ReqB}$—is assessed by using as propagation factor the percentage of past commits, in which two requirements have co-changed. We note, that for this calculation, we do not need the calculation of internal probabilities if we calculate the percentage over the complete commit history, since both calculations would lead to the same results. Thus, we opt for the simplest solution (see Chapter 7.3.2 for more details).

- The probability of one requirement (e.g., ReqA) to change due to ***overlapping implementations with another requirement*** (e.g., ReqB)—$POI_{ReqB \rightarrow ReqA}$—is assessed by using as propagation factor the percentage of classes implementing ReqB, which are related to ReqA. The reason is that the percentage of shared classes between ReqB and ReqA reflects to a satisfactory degree the extent of overlapping implementation as classes are often at the level of granularity around which development teams organize functional requirements. To guarantee the independence of propagation due to overlapping implementations to propagation due to overlapping contracts, we omit from this calculation the commits that two or more requirements are co-changing.
- The probability of one requirement (e.g., ReqA) to change due to ***ripple effects at implementation level with another requirement*** (e.g., ReqB)—$PRE_{ReqA\text{-}ReqB}$—is assessed by using as propagation factor the Ripple Effect Measures (Ampatzoglou et al., 2015; Arvanitou et al., 2015) for all classes implementing ReqA and are not involved in the implementation of ReqB. In particular, we examine all pairs of classes that are not considered in the propagation due to overlapping contracts calculation (guaranteeing the independence of propagation due to overlapping implementations and propagation due to ripple effect), and investigate if they are structurally dependent, leading to direct propagation due to ripple effects or indirect propagation due to ripple effects . The calculation of ripple effects at the code level is discussed in detail in the original studies (i.e., Ampatzoglou et al., 2015; Arvanitou et al., 2015). However, to support the independent reading of this manuscript we demonstrate its usage in Chapter 7.3.2.

Having completed the calculation of the aforementioned probabilities, R2EM metric can be calculated as the joint probability of propagation due to overlapping contracts, propagation due to overlapping implementations, direct propagation due to ripple effects, and indirect propagation due to ripple effects.

7.3.2 Illustrative Example

To illustrate the application of the previously described methodology, we consider a system with 3 requirements (R1, R2, and R3) that are implemented in 10 classes, throughout a version history of 10 commits. A sample commit log is presented in Table 7.3.2.a. Based on the above, the probability to change due to overlapping contracts is as follows:

- ***$PCO_{R1\text{-}R2}$ is 10%***, due to commit #6. As discussed in the previous chapter the calculation of $PCO_{R1\text{-}R2}$ would be the same, even if we used the analytical calculation: propagation factor 50% (R1 co-changes with R2 in commit #6, but not in #1), and its internal probability to change is 20%. By multiplying the internal probability to change with the propagation factor, we would again reach a PCO that equals 10%.
- ***$PCO_{R1\text{-}R3}$ is 0%***, since they have not co-changed in the history of the project
- ***$PCO_{R2\text{-}R3}$ is 20%***, due to commits #4 and #9

Table 7.3.2.a: Illustrative Example Commit History

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

To calculate the probability to change due to overlapping implementations, first each requirement should be mapped to the classes in which it is implemented. As noted before, in this list, we do not include commits for which requirements co-change (this is part of the calculation of PCO). Thus, the list is as follows: $ImplementationSet_{R1}$ = {C1, C2, C3}, $ImplementationSet_{R2}$ = {C3, C4, C5, C6}, and $ImplementationSet_{R3}$ = {C6, C7, C8, C9, C10}. Thus, POI is calculated as follows:

- ***$POI_{R1 \rightarrow R2}$ is 3.3%***. The propagation factor is 33%, since the implementation set has one common class (i.e., C3), and the requirement that emits the change is implemented in three classes. Whereas, the internal probability to change of R1 is 10% (see commit #1). POI is calculated as the multiplication of internal probability and propagation factor.
- ***$POI_{R2 \rightarrow R1}$ is 5%***. The propagation factor is 25%, since the implementation

set has one common class (i.e., C3), and the requirement that emits the change is implemented in four classes. Whereas, the internal probability to change of R2 is 20% (see commits #2 and #7).

- ***$POI_{R1 \rightarrow R3}$ is 0%***, since their implementations do not share classes (i.e., propagation factor 0%).
- $POI_{R3 \rightarrow R1}$ ***is 0%***, since their implementations do not share classes (i.e., propagation factor 0%).
- ***$POI_{R2 \rightarrow R3}$ is 5%***. The propagation factor is 25%, since the implementation set has one common class (i.e., C6), and the requirement that emits the change is implemented in four classes. Whereas, the internal probability to change of R2 is 20% (see commits #2 and #7).
- ***$POI_{R3 \rightarrow R2}$ is 8%***. The propagation factor is 20%, since the implementation set has one common class (i.e., C6), and the requirement that emits the change is implemented in five classes. The internal probability to change of R3 is 40% (see commits #3, #5, #8, and #10).

To calculate REM at the class level, we need to take into account the following parameters, which are later synthesized using the following formula:

$$\text{REM} = \frac{NDMC+NOP+NPrA}{NOM+NA}$$

NDMC: Number of distinct method calls from class *A* to class B (super class method invocations for the case of generalization)

NOP: Number of polymorphic methods in class *B* (valid only for generalization)

NPrA: Number of protected attributes in class *B* (valid only for generalization)

NOM: Number of methods in class *B*

NA: Number of attributes in class *B* (valid only for generalization)

The illustrative (they cannot be deducted from the class diagram—see Figure 7.3.2.a) metrics for each class are presented in Table 7.3.2.b. Each row of the table represents one class (the one that emits the change), whereas the rows represent the classes with which they interact. The internal probability to change for every class is presented in the column named PCCC (Percentage of Commits in which a Class has Changed), and is calculated from Table 7.3.2.a. Also, let us suppose that the only inheritance relationships are between class C4 and C5 from C1, and C10 from C7.

Table 7.3.2.b: Illustrative Example Class Dependencies

Class ID	PCCC	NOP	NPrA	NOM	NA	NDMC
C1	0.2	2	1	4	4	C2(2), C3(2)
C2	0.1	0	0	5	2	C7(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)
C7	0.1	1	0	8	1	C8(2)
C8	0.3	0	0	2	2	C9(4), C10(2)
C9	0.1	0	0	10	3	C2(2), C10(1)
C10	0.3	0	0	6	6	C6(1)

By considering that the number of combinations that are required for examining all pairs of requirements is too high (100 DPRE and 100 IPRE scores need to be calculated[34]), for the rest of this chapter, we focus on requirements R1 and R2, and in particular the probability that changes from R1 propagate to R2. To this end, we need to calculate $DPRE_{C1\rightarrow C4}$, $DPRE_{C2\rightarrow C4}$, $DPRE_{C1\rightarrow C5}$, $DPRE_{C2\rightarrow C5}$, $DPRE_{C1\rightarrow C6}$, and $DPRE_{C2\rightarrow C6}$. We note that C3 is not considered in this process, since it is common for both requirements. Based on Table 7.3.2.b and the formula for REM calculation, the following scored have been calculated:

$$DPRE_{C1\rightarrow C4} = \frac{NDMC+NOP+NPrA}{NOM+NA} * PCCC_{C1} = \frac{0+2+1}{4+4} * 0.2 = 7.5\%$$

$$DPRE_{C2\rightarrow C4} = \frac{NDMC+NOP+NPrA}{NOM+NA} * PCCC_{C2} = \frac{1+0+0}{5+2} * 0.1 = 1.4\%$$

$$DPRE_{C1\rightarrow C5} = \frac{NDMC+NOP+NPrA}{NOM+NA} * PCCC_{C1} = \frac{1+2+1}{4+4} * 0.2 = 10.0\%$$

$$DPRE_{C2\rightarrow C5} = \frac{NDMC+NOP+NPrA}{NOM+NA} * PCCC_{C2} = \frac{0+0+0}{5+2} * 0.1 = 0.0\%$$

$$DPRE_{C1\rightarrow C6} = \frac{NDMC+NOP+NPrA}{NOM+NA} * PCCC_{C1} = \frac{2+0+0}{4+4} * 0.2 = 5.0\%$$

$$DPRE_{C2\rightarrow C6} = \frac{NDMC+NOP+NPrA}{NOM+NA} * PCCC_{C2} = \frac{2+0+0}{5+2} * 0.1 = 2.8\%$$

$DPRE_{R1\rightarrow R2}$=JointProbability{$DPRE_{C1\rightarrow C4}$,$DPRE_{C2\rightarrow C4}$,$DPRE_{C1\rightarrow C5}$ $DPRE_{C2\rightarrow C5}$, $DPRE_{C1\rightarrow C6}$, $DPRE_{C2\rightarrow C6}$ }= 4.3%

[34] We note that all the calculations are automated through the tool support that we provide (see Chapter 3.3).

In order to calculate the probability to change due to indirect ripple effects, we visualized the relationships between classes in Figure 7.3.2.a. In this figure we denote method calls with dashed-arrows, and generalization relationships with solid lines. As we can observe for Figure 7.3.2.a, the only indirect ripple effect between classes that implement requirements R1 and R2 can be caused by a change occurring in C2 that might be emitted to C9 which in turn can be emitted to C4. To calculate IPRE we need to calculate $REM_{C2 \rightarrow C9}$ and $REM_{C9 \rightarrow C4}$. These values will be multiplied with $CCPM_{C2}$ and $CCPM_{C9}$.

$$IPRE_{C2 \rightarrow C4} = \frac{NDMC+NOP+NPrA}{NOM+NA} * PCCC_{C2} * \frac{NDMC+NOP+NPrA}{NOM+NA} * PCCC_{C9}$$

$$= \frac{1+0+0}{5+2} * 0.1 * \frac{2+0+0}{10+3} * 0.1 = 0.02\%$$

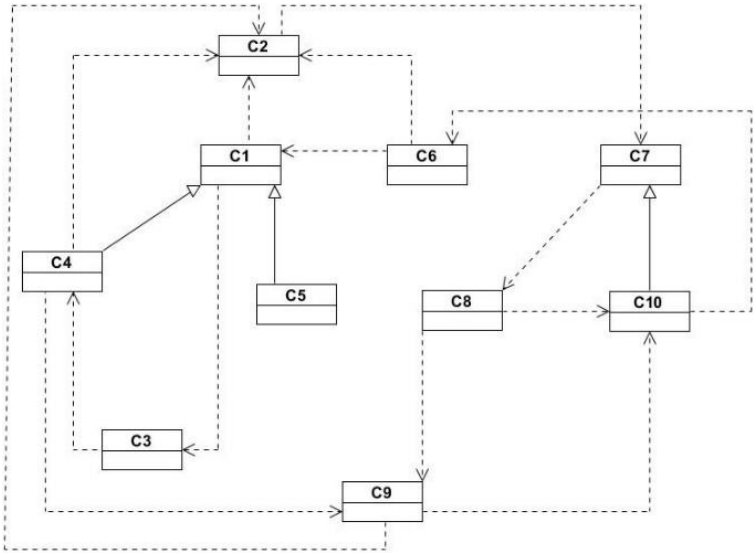

Figure 7.3.2.a: Class Diagram for the illustrative example

$R2EM_{R1 \rightarrow R2}$= JointProbability{PCO,POI,DPRE,IPRE}=16.72%

7.3.3 Proposed Tool-Chain

To automate the calculation of all the aforementioned probabilities, we have developed a sequence of tools during our earlier work (see Figure 7.3.3.a) that calculate the probabilities for each pair of requirements of the projects.

First, we use the git Repository in order to: (a) clone it and export the source code, and (b) produce the git-log to document the commit history. Next, the commit history has been used as an input for calculating PCCC (Arvanitou et al., 2017b). 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 propagation factor 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. The tool is available online along with all the other tools that comprise the aforementioned tool-chain[35].

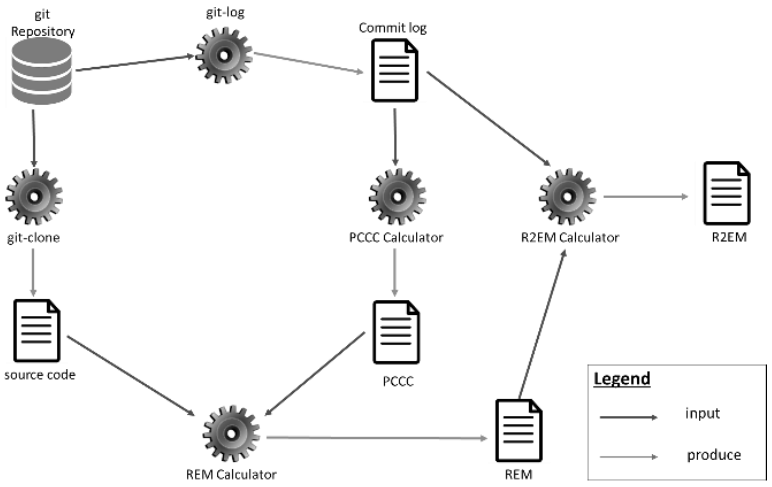

Figure 7.3.3.a: Proposed Tool-Chain

7.4 Case Study Design

In this chapter we present the design of the industrial case study. The case study has been performed within a Small-Medium Enterprise (SME) in Greece, which is considered a national key-player in mobile and web development. This case study is designed and reported according to the linear-analytic structure template suggested by Runeson et al. (2012). In particular in the next chapters we present the four parts of our research design, i.e., research questions (see Chapter 7.4.1), case selection (see Chapter 7.4.2), data collection (see Chapter 7.4.3), and analysis (see Chapter 7.4.4).

[35] http://www.cs.rug.nl/search/uploads/Resources/R2EM_Calculator.rar

7.4.1 Research Questions

According to the above-mentioned goals we have derived two research questions (RQ) that will guide the case study design and the reporting of the results:

RQ$_1$: *How frequently do the events that can lead to ripple effects at the requirements level, occur in practice?*

Through this research question we investigated the propagation factors discussed in Chapter 7.3 (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). As an outcome of this research question we will provide a ranked list of these factors in terms of importance, and a discussion on the statistical significance of the differences.

RQ$_2$: *Is R2EM able to* efficiently *prioritize requirements testing?*

This research question will explore the efficiency of the proposed requirements ripple effects metric to support test case prioritization. R2EM should be able to prioritize the requirements to be tested, in a similar way to the intuition of the practitioners. To investigate this, we will contrast the ranking provided by R2EM to the ranking provided by practitioners. The ability of the method to accurately predict the expert opinion of software engineers can be useful for two reasons: (a) the test case prioritization through a tool can scale much better than expert opinion, and (b) the method can guide inexperienced developers, who are not able to reach the level of understanding of experts in the field.

For both research questions, in addition to the aforementioned quantitative assessment, we will complement the findings with quotes and explanations provided by practitioners (qualitative assessment).

7.4.2 Case Selection

This study is a holistic multiple case study that has been conducted in OTS, an SME in Greece. OTS is a key-player in Greece mobile and web development sector, and has special interest in requirements management approaches, such as requirements-to-code traceability and change impact analysis on requirements. In this study as cases and corresponding units of analysis we consider the requirements of the systems under study. As case study participants we selected 9 software engineers that are currently working on the maintenance and evolution of the two systems under study:

- ***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 for 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. The main requirements of this system deal with the main CRUD[36] actions on these entities. Additionally, we identified many requirements that were horizontal to these entities, and merged all of them under the common term: System-wide requirements. Therefore, 25 requirements have been investigated (4 CRUD actions x 6 subsystems + 1 representing all system-wide requirements). We note that system-wide requirements are expected to have very high proneness to ripple effects, since by nature they are overlapping with many other requirements.
- ***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 between 13/01/2016 and 17/01/2017. The system can be decomposed into 8 main sub-systems, each one managing the following entities: (a) Birth, (b) Death, (c) Marriage, (d) Namegiving, (e) Partnership, (f) Citizen, (g) Reports, and (h) Temporal Triggers. Similarly as before, CRUD actions on these entities have been considered as our cases. Therefore, 33 (4 x 8 + 1) requirements have been investigated (again, by considering system-wide requirements).

The requirements are coded using the name of the system, the first letter of the entity and the first letter of the CRUD operation. For example, requirement that Reads an Account in the YDATA system is named as: YDATA-AR.

7.4.3 Data Collection

To answer the research questions mentioned in Chapter 7.4.1, we executed the tool-chain described in Chapter 7.3.3, and obtained change impact data of all studied requirements for both systems. In Table 7.4.3.a, we present the percentage of commits, in which each requirement has changed (PCRC). From Table 7.4.3.a, we omit requirements that are not encountered in the commit history, as their modification does not make sense (e.g., a bill that was created will never be deleted).

[36] CRUD: Create / Read / Update / Delete

Table 7.4.3.a: Requirements Change Frequency

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

System	Requirements		PCRC
	ID	Name	
CREGAPI (Cont.)	CR-BU	Birth Update	0,59%
	CR-PD	Political Delete	0,59%
	CR-CD	Citizen Delete	0,47%
	CR-DU	Death Update	0,35%
	CR-DR	Death Read	0,24%
	CR-MD	Marriage Delete	0,24%
	CR-NR	Naming Read	0,24%
	CR-NU	Naming Update	0,24%
	CR-PR	Political Read	0,24%
	CR-BD	Birth Delete	0,12%

Our dataset consists of 421 rows (190 from YDATA and 231 from CREGAPI) that represent all pairs of requirements in Table 7.4.3.a (we develop pairs only within systems). For each pair, we have recoded the following information, which will be used for answering RQ_1:

- ***FromReq***: The ID of the requirement that might triger a ripple effect;
- ***ToReq***: The ID of the requirement that receives the ripple effect;
- ***PCO***: The probability of ToReq to change because of overlaps with FromReq;
- ***POI***: The probability of ToReq to change because of overlaps of its implementation with FromReq;
- ***DPRE:*** The probability of ToReq to change because of source code direct ripple effects from the implementation of FromReq
- ***IPRE:*** The probability of ToReq to change because of source code indirect ripple effects from the implementation of FromReq
- ***R2EM***: The probability of ToReq to change, due to changes occuring in FromReq. This value is the joint probability of propagation due to overlapping contracts, propagation due to overlapping implementations, direct propagation due to ripple effects, and indirect propagation due to ripple effects.

To answer RQ_2, we will need to contrast the aforementioned data with the expert opinions of software engineers in OTS. To collect the data required for our study, we conducted a workshop with 9 industrial practitioners, working

for the company. The participants have been involved in the original construction and/or maintenance of the two projects. The workshop comprised of:

- *Structured interviews*. According to Runeson et al. (2012) 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). 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 researchers 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 the researchers 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).

Due to the limited time that experts were available, it was not possible to validate all 421 pairs of requirements; therefore we had to make a selection of pairs. To do this, we first applied the proposed method (see Chapter 7.3.1) to calculate 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 7.4.3.a)—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 more important starting points for change impact analysis: requirements that are rarely modi-

fied are probably not easy to remember since they are not used regularly.

The list of selected requirements is presented in the data extraction forms in Appendix B. For each one of these requirements, we listed all other requirements and asked participants to evaluate how important they believe they are to re-test the latter because of changes in the former (1=min-5=max). In total, 76 questions were asked for YDATA (4 requirements paired with 19 others) and 84 for CREGAPI (4 requirements paired with 21 others). For assessing the evaluators' agreement, we used inter-rater reliability calculated through the intra-class correlation coefficient (ICC) (Field, 2013). The reliability for the YDATA project has been calculated as 80.0% and as 77.4% for CREGAPI. The workshop organization and the questions used in the interviews and focus group are presented in Appendix B. The ICC for each requirement is presented in Table 7.4.3.b. It can be observed that, in most of the cases, the participants were consistent with their answers; the only exception is Statement Create, which we discuss separately while interpreting the results.

Table 7.4.3.b. Intra-Class Correlation

System	Requirements		ICC
	ID	Name	
YDATA	YDATA-AR	Bill Read	87.4%
	YDATA-LC	Alert Create	51.1%
	YDATA-SC	Statement Create	57.9%
	YDATA-PC	Payment Create	90.9%
CREGAPI	CR-CC	Citizen Create	81.5%
	CR-BC	Birth Create	72.9%
	CR-MU	Marriage Update	80.3%
	CR-NC	Namegiving Create	72.9%

7.4.4 Data Analysis

In this chapter, we present the data analysis process that has been used for answering the research questions described in Chapter 7.4.1.

Ripple Effect Factors: To answer RQ_1 we conduct the following four levels of analysis:

- *Descriptive Statistics for the Dataset*. We will provide the list of the most change-prone requirements, due to ripple effects from both projects;
- *Comparison of Different Propagation Factors*. We will present descriptive

statistics for all factors: i.e., mean, mix, max, and standard deviation for the propagation due to overlapping contracts, propagation due to overlapping implementations, direct propagation due to ripple effects, and indirect propagation due to ripple effects variables. Next, we will perform hypothesis testing to check the existence of differences in the four factors (Field, 2013).

Effectiveness of R2EM for Requirements Test Prioritization: To answer RQ_2, we perform four types of analysis, as dictated by the 1061-1998 IEEE Standard on Software Measurement (IEEE-1061, 1998)—see Chapter 4.4. We note that from the aforementioned standard, we have excluded tracking, since no data on the evolution of requirements were available and predictive power, because in case of univariate analysis the R value of a linear regression equals the value of Pearson correlation coefficient.

7.5 Results

In this chapter we present the results of our study organized by research question. In this chapter we provide initial interpretations, whereas an overall discussion of the results of all research questions is provided in Chapter 7.6.1.

First, we present some descriptive statistics of our dataset. In particular, in Tables 7.5.a and 7.5.b, we present the top-10 ripple effects in each one 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 the high co-change of the two requirements and the overlap of their implementations. One of the participants 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"*. Additionally, regarding CREGAPI, we can observe that the Citizen entity is dominant among the ripple-effect prone requirements. The centrality of the role of Citizen has been vividly explained one anticipant 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"*.

Additionally, many system-wide requirements have been identified in the top most ripple-effect-prone ones (especially for the CREGAPI project). This finding can be explained that due to the nature of the system, many changes are

occurring in all entities of the system. One participant explained this as follows: "*changing the citizen identifier column from the ID card number to the Social Security Number, will affect all entities (e.g., Marriage, Death, Birth Certificate etc.)*". In the next chapters we answer the research questions and provide more detailed comparisons.

Table 7.5.a: Top-10 Ripple Effects for YDATA System

From	To	R2EM	PCO	POI	DPRE	IPRE
YD-BR	YD-PC	47.98%	29.89%	39.06%	26.84%	0.50%
YD-BR	YD-HR	47.61%	14.80%	39.00%	34.25%	1.07%
YD-HR	YD-UR	47.61%	8.03%	41.81%	32.55%	0.11%
YD-HR	YD-SC	47.56%	11.29%	41.90%	30.50%	0.16%
YD-HR	YD-UU	47.14%	5.80%	39.45%	34.54%	0.43%
YD-SC	YD-HR	47.19%	0.91%	39.44%	33.83%	8.10%
YD-BR	YD-SC	46.60%	13.97%	38.44%	29.47%	0.29%
YD-BR	YD-UR	46.26%	3.57%	37.88%	33.23%	0.43%
YD-UR	YD-HR	46.46%	13.71%	38.07%	29.53%	0.15%
YD-BR	YD-BU	46.07%	5.19%	38.97%	30.09%	0.10%

Table 7.5.b: Top-10 Ripple Effects for CREGAPI System

From	To	R2EM	PCO	POI	DPRE	IPRE
CR-CC	CR-CU	31.78%	2.65%	19.04%	13.33%	0.14%
CR-CU	CR-CR	29.15%	1.36%	15.05%	15.23%	0.26%
CR-CC	CR-CR	25.03%	1.11%	13.91%	11.72%	0.24%
CR-CU	CR-CD	22.16%	0.00%	11.17%	12.38%	0.00%
CR-CC	CR-CD	21.25%	0.00%	12.39%	10.11%	0.00%
CR-CR	CR-CD	17.37%	0.00%	9.33%	8.87%	0.00%
CR-CC	CR-MC	14.87%	0.00%	10.09%	5.30%	0.01%
CR-CC	CR-DC	14.72%	0.65%	10.42%	4.18%	0.00%
CR-MC	CR-PC	14.64%	2.31%	9.00%	3.98%	0.00%
CR-CC	CR-BC	12.89%	0.00%	8.28%	5.02%	0.00%

7.5.1 Ripple Effect Factors (RQ_1)

In this chapter, we present the results on the comparison of the various factors that can trigger ripple effects. From Table 7.5.1.a we can observe that *Overlapping Implementations (POI)* is the factor that is mostly responsible for the emission of ripple effects, followed by *Direct Ripple Effects* and then *Overlapping Contracts*. An interesting observation that can be made by comparing the results between the two projects (Table 7.5.1.a), is that this ranking is consistent in both projects. Additionally, the YDATA system presents higher ripple-effect proneness on average, compared to CREGAPI. This outcome is probably due to the density of the system (smaller size, but similar number of requirements). The ripple effect propagation factor with the highest average probability for each system is denoted with grey cell shading.

Table 7.5.1.a: Descriptive Statistics for Ripple Effect Factors

		Min	Max	Mean	SDev
CREGAPI	PCO	0.00%	20.59%	0.212%	1.178%
	POI	0.00%	43.35%	2.603%	5.179%
	DPRE	0.00%	32.42%	1.487%	3.352%
	IDPRE	0.00%	0.26%	0.007%	0.029%
YDATA	PCO	0.00%	29.89%	1.547%	3.350%
	POI	0.00%	41.90%	14.262%	11.307%
	DPRE	0.00%	34.54%	8.353%	8.315%
	IDPRE	0.00%	16.60%	0.551%	2.020%

To investigate if the aforementioned mean values are representing significant differences or if they are affected by outliers, we analyzed the variance of the variables through the ANOVA test. For both systems, ANOVA indicated that the four factors lead to different probabilities of ripple effect scores. To bilaterally compare the factors, we have performed a Wilcoxon Rank test. The results are presented in Tables 7.5.1.b and 7.5.1.c. The first column of both tables presents the compared factors, the second and the third columns demonstrate in how many cases each factor is higher (Neg. Ranks suggest that the 2nd factor is higher, etc.). The last two columns of each row represent the results of the Wilcoxon Rank test (Z and sig.).

Table 7.5.1.b: Hypothesis testing CREGAPI (N=506)

		N	Mean Rank	Z	Sig.
POI – PCO	Neg. Ranks	4	158.75	-18.452	0.000
	Pos. Ranks	460	233.14		
	Ties	42			
DPRE – PCO	Neg. Ranks	16	209.94	-15.633	0.000
	Pos. Ranks	375	195.41		
	Ties	115			
IPRE – PCO	Neg. Ranks	77	90.12	-6.010	0.000
	Pos. Ranks	54	31.61		
	Ties	375			
DPRE – POI	Neg. Ranks	387	250.35	-14.989	0.000
	Pos. Ranks	76	138.57		
	Ties	43			
IDRE – POI	Neg. Ranks	464	232.50	-18.671	0.000
	Pos. Ranks	0	.00		
	Ties	42			
IDRE - DPRE	Neg. Ranks	385	193.00	-17.004	0.000
	Pos. Ranks	0	.00		
	Ties	121			

Table 7.5.1.c: Hypothesis testing YDATA (N=418)

		N	Mean Rank	Z	Sig.
POI – PCO	Neg. Ranks	0	.00	-18.452	0.000
	Pos. Ranks	412	206.50		
	Ties	6			
DPRE – PCO	Neg. Ranks	15	173.10	-15.633	0.000
	Pos. Ranks	390	204.15		
	Ties	13			

		N	Mean Rank	Z	Sig.
IPRE – PCO	Neg. Ranks	224	184.06	-6.010	0.000
	Pos. Ranks	102	118.35		
	Ties	92			
DPRE – POI	Neg. Ranks	408	209.43	-14.989	0.000
	Pos. Ranks	5	9.10		
	Ties	5			
IDRE – POI	Neg. Ranks	412	206.50	-18.671	0.000
	Pos. Ranks	0	.00		
	Ties	6			
IDRE - DPRE	Neg. Ranks	403	203.94	-17.004	0.000
	Pos. Ranks	2	14.00		
	Ties	13			

Similarly as before, the results are consistent among projects, and all differences have proven to be statistically significant.

The propagation factors rank from more to less frequent as follows:

- Overlap of requirements implementations
- Direct ripple effects on requirements implementations
- Overlap in requirements contracts (i.e., history of co-change)
- Indirect ripple effects on requirements implementations

The aforementioned ordering is statistically significant.

Since factors that represent requirements' implementation relations are already fine-grained, for the rest of the chapter we further investigate conceptual dependency factors. In the context of persistent storage, CRUD operations CRUD refers to the major actions performed on system entities. Thus, we can assume two types of conceptual relationships: relationships due to working on the same entity (same first letter in requirements ID), and relationship due to performing the same action on different entities (same second letter in requirements ID). In Table 7.5.1.d, we present descriptive statistics on the propagation factors and the R2EM metric, for the aforementioned relations (plus the System-Wide requirements introduced in Chapter 7.4.2). We note that in

this research question, we treat the complete dataset as one, and do not report per project. The relation with the most intense ripple effects is denoted with grey cell shading.

Table 7.5.1.d: 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%
	System-Wide	0.41%	63.34%	20.64%	14.92%
PCO	Same Entity	0.00%	19.58%	1.14%	2.95%
	Same Action	0.00%	17.07%	0.85%	2.46%
	System-Wide	0.00%	20.59%	1.09%	2.69%
POI	Same Entity	0.12%	38.64%	9.36%	10.57%
	Same Action	0.00%	41.81%	8.10%	10.60%
	System-Wide	0.12%	43.35%	14.18%	10.88%
DPRE	Same Entity	0.00%	31.76%	5.58%	7.14%
	Same Action	0.00%	34.25%	4.85%	7.49%
	System-Wide	0.00%	32.42%	7.95%	7.07%
IPRE	Same Entity	0.00%	7.43%	0.15%	0.80%
	Same Action	0.00%	2.26%	0.07%	0.25%
	System-Wide	0.00%	16.60%	1.65%	3.79%

Based on the findings of Table 7.5.1.d, system-wide requirements are more probable to trigger ripple effects, followed by change propagation between requirements working on the same entity or performing the same action. The fact that system-wide requirements are the ones that trigger ripple effects is considered intuitive, since they are by definition related to many requirements. Regarding the rest, one participant explained this 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 operations. On the other hand, activity-related requirements are also prone to co-change, 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 operations are almost always maintained in the same time or under a common transaction*".

Regarding change propagation factors, the only one for which System-Wide requirements are not the most ripple-effect-prone is propagation due to overlapping contracts. This finding suggests that the majority of requirements tend to co-change with others working on the same entity.

We further investigate the statistical significance of the aforementioned observations. The results of the corresponding Mann-Whitney tests are reported in Table 7.5.1.e. We can observe that most differences are statistically significant. However, the differences between *Same Entity* and *Same Action* relations are not statistically significant, and therefore, they should be treated as being similarly ripple-effect-prone. The only exception to this is the PCO metric, suggesting that relations between requirements working on the *Same Entity* are more prone to ripple effects, due to overlapping contracts.

Table 7.5.1.e: Hypothesis Testing

Metric	Relationship	Mean Rank	Z	Sig.
R2EM	Same Entity	155.56	-1.293	0.196
	Same Action	141.99		
	Same Entity	76.84	-3.909	0.000
	System-Wide	107.36		
	Same Action	121.57	-5.815	0.000
	System-Wide	182.79		
PCO	Same Entity	162.13	-2.459	0.014
	Same Action	138.73		
	Same Entity	90.12	-0.252	0.801
	System-Wide	92.01		
	Same Action	132.14	-2.719	0.007
	System-Wide	158.25		
POI	Same Entity	155.41	-1.272	0.203
	Same Action	142.07		
	Same Entity	77.86	-3.626	0.000
	System-Wide	106.17		
	Same Action	122.91	-5.393	0.000
	System-Wide	179.68		

Metric	Relationship	Mean Rank	Z	Sig.
DRPE	Same Entity	153.52	-1.002	0.316
	Same Action	143.01		
	Same Entity	79.40	-3.626	0.000
	System-Wide	104.40		
	Same Action	124.34	-4.944	0.000
	System-Wide	176.36		
IRPE	Same Entity	156.24	-1.546	0.122
	Same Action	141.65		
	Same Entity	77.35	-3.891	0.000
	System-Wide	106.76		
	Same Action	122.31	-5.951	0.000
	System-Wide	181.06		

The ranking of requirements relations, in terms of ripple-effect-proneness, excluding ripple effects due to overlapping contracts, is as follows: *System-wide Requirements are more probable to emit ripple effects compared to requirements working on the Same Entity or Action*.
Regarding ripple effects due to overlapping contracts, the ranking differs as follows: Requirements Working on the Same Entity and System-wide Requirements are more probable to emit ripple effects compared to Requirements performing the Same Action on a different Entity.

7.5.2 R2EM Efficiency for Testing Prioritization (RQ_2)

In this chapter, we present the results on evaluating the efficiency of the proposed metric in assessing the priority of test cases. In Table 7.5.2.a, we present the correlation for the complete dataset as a whole, and per project (***correlation*** and ***consistency***). Based on the results we can observe that for both projects, the ranking of the method is strongly correlated (coeff. > 0.6 (Marg et al., 2014)) to the opinions of software engineers. Therefore, adequate correlation and consistency is achieved. As expected, the consistency criterion (Spearman correlation coefficients) is achieved at a better degree compared to correlation (Pearson correlation coefficients), since the two variables are not measured at the same scale. Nevertheless, the difference is rather small (almost negligible).
Regarding reliability at the project level, the results on the CREGAPI project

are better compared to YDATA, an observation that can be explained due to the smaller size of the specific project. Nevertheless, the fact that the difference is small suggests that the method is scalable, since doubling up the number of requirements and classes, costs less than 1% of efficiency. Thus, the consistency and correlation assessments can be considered reliable at the project level.

Table 7.5.2.a: R2EM Validation

Requirements	Pearson		Spearman	
	Coeff.	Sig.	Coeff.	Sig.
Complete Dataset	59.8%	0.000	60.6%	0.000
CREGAPI	57.1%	0.000	63.4%	0.000
YDATA	61.3%	0.000	62.6%	0.000

In Table 7.5.2.b, we present the correlation of the proposed metrics with the priority that the software engineers have assigned at the requirements level. We further observe that requirements can be divided into two main categories, denoted with grey- and white-cell shading.

Table 7.5.2.b: Validation per Requirement

System	Requirements	Pearson		Spearman	
		Coeff.	Sig.	Coeff.	Sig.
YDATA	Bill Read	71.9%	0.035	78.7%	0.012
	Alert Create	62.1%	0.013	61.3%	0.015
	Statement Create	63.7%	0.008	69.4%	0.003
	Payment Create	75.1%	0.037	76.8%	0.017
CREGAPI	Citizen Create	85.7%	0.000	78.4%	0.000
	Birth Create	79.7%	0.000	58.0%	0.019
	Marriage Update	44.8%	0.002	66.4%	0.000
	Namegiving Create	47.5%	0.086	54.5%	0.044

One the one hand, the requirements with the grey-cell shading are those for which the method proves to be the most accurate. This efficiency can be explained by the fact that participants consider the specific requirements to have straightforward ripple effects. For example, consider the following statements from participants about pairs of requirements that have been both ranked very high from our method and deemed as 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 relationship 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 relationship*."
- (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."*

On the other hand, the requirements with lower efficiency (55% - 69%) are those for which the participants had contradictory opinions between themselves as well. For example, regarding YDATA the agreement of participants on the requirements affected by a change in the way alerts are created is 51.1% (see Table 7.4.3.b), while the correlation of the proposed method to the average expert opinion is 61.3%. A possible explanation for the deviation is the way software engineers perform requirements testing prioritization as provided by the lead software engineer of CREGAPI: "*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*".

Regarding discriminative power, in Figure 7.5.2.a, we present the boxplots for each level of ripple effects. The boxes correspond to the 50% of the R2EM scores for each category, whereas the line in the box the median value.

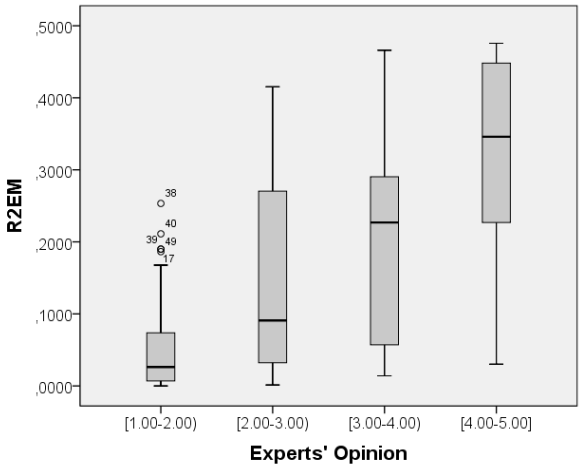

Figure 7.5.2.a: Discriminative Power of R2EM

The box plots demonstrate that the mean values of R2EM get substantially higher for pairs of requirements that have been evaluated as ripple effect-prone, by experts. To statistically explore the differences that are visualized in Figure 7.5.2.a, we performed Analysis of Variance (ANOVA) and the Bonferroni post-hoc test. The ANOVA confirmed the existence of statistically significant differences among the categories (F: 22.666 and sig: 0.00). By a pairwise comparison between categories, we have observed that all differences are statistically significant, except from [2.00 – 3.00) and [3.00 – 4.00). Thus, the method is able to accurately discriminate requirements that have been evaluated as very low (1) to low (2), from medium (2-4), and high (4) to very high (5). The results of the ANOVA are very similar in both projects: GREGAPI (F: 9.05, sig: 0.00) and YDATA (F: 11.95, sig: 0.00). Thus, the discriminative power of the R2EM metric has been validated, with respect to this criterion, as well.

The ranking that R2EM provides for test case prioritization is strongly correlated (62%-63%) to the expert opinion of software engineers. The scalability of the proposed approach has been positively evaluated, since doubling up the size of the systems in terms of requirements and source code size, resulted in only 1% decrease in the correlation strength.

7.6 Discussion

7.6.1 Interpretation of Results

The case study reported in this paper had two main goals: (***g1***) understanding the most important factors that can lead to ripple effects among requirements, and (***g2***) the validation of the proposed metric.

Regarding (***g1)*** a two-level analysis has been performed: (a) an analysis based on the ripple effect factors (i.e., overlapping contracts, overlapping implementations, and structural dependencies of these implementations), and (b) an analysis on the requirements relations (e.g., are ripple effects more common among requirements working on the same entity). Based on our results, the following main observations have been reached:

- ***requirements implementations vs. contracts***. The implementation of requirements appears to be more important with respect to ripple effects compared to the requirement contract. In particular, the probability to co-change, due to overlapping implementations (POI) is the highest ripple effect factor, followed by DPRE (probability due to direct ripple effects at implementation level). On the other hand, co-change of requirements due to their similarity has proven to be sparse (1.1% - 3.3%), compared to POI (5.1% – 11.3%) and DPRE (3.3% – 8.3%). It is expected that this observation will be more evident for denser systems, i.e., when many requirements are concentrated in a few classes.
- ***system-wide requirements and central entities***. In both examined systems, we have observed that system-wide requirements are responsible for the emission of most ripple effects. This outcome is expected in the sense that they are meant to touch upon many other requirements. Additionally, some central entities (e.g., the Citizen in the municipality application) have been identified, and any change in such requirements is highly probable to affect many parts of the system.
- ***entity- vs. action-related requirements.*** Requirements affecting the same entity are more probable to co-change (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 DPRE). Nevertheless, both types have been

validated as important by the software engineers. Finally, the results confirmed our intuition to merge all other types of relations, since even combined, they are statistically producing less ripple effects compared to the other types.

Regarding (***g2***) the results of the empirical validation suggested that the proposed metrics ***R2EM*** is a valid assessor of requirements ripple effect, and ***can serve as a means for test case prioritization***. In particular, R2EM exhibited a strong correlation with expert's opinion (approx. 60%) in both Spearman and Pearson correlation. Also, statistically significant discriminative power can be achieved by using this metric. In principle, the aforementioned correlation is stronger for pairs of requirements whose relationship is stronger according to practitioners (i.e., a high-level of agreement on high practitioners' values).

7.6.2 Implications for Researchers & 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 in requirements testing prioritization. To ease the adoption of the proposed methods, based on the suggestions of our case study's participants, we encourage the integration of the tool or any similar approach in the IDE that each company is using. Second, for cases in which the proposed tool-chain is not applicable (e.g., not Java, or no Git for version control), we are able to guide test case prioritization, based on the findings of the empirical analysis on requirements ripple effect factors and relationships. In particular, we advise practitioners to prioritize test cases, 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 testing 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 test case prioritization through a longitudinal case study is required. For such a case, a company should use the suggestions of the tool for a long period and evaluate: (a) the required testing effort, and (b) the number of bugs identified, when test cases are prioritized, based on the proposed suggestions. 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 sys-

tems would be required. Finally, an interesting extension scenario would be to tailor the proposed metrics (from REM to R2EM) to non-object-oriented paradigms, for example, by considering files or folders as units of analysis.

7.7 Threats to Validity

In this chapter, we present and discuss potential threats to the validity of our case study (Runeson et al., 2012). Internal validity is not considered, since we have not dealt with causal relations.

7.7.1 Construct Validity

A possible threat to construct validity is related to the accuracy of the proposed approach 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 approach, we note that the definition of the proposed metric is clear and well-documented (see Chapter 7.3), whereas the used tools have been thoroughly tested, before deployment, in a large number of open source projects (see Chapter 7.3.3).
Regarding the approach, 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 Chapter 7.6). 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 the method, and therefore, we opted to omit commits in which more than one requirement has co-changed.

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 7.4.3.b). Specifi-

cally, we observed that for both systems high agreement between developers occur. A detailed discussion on this issue is presented in Chapter 4.3, focusing on specific pairs of requirements with lower levels of agreement.

7.7.2 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 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.

7.7.3 External Validity

Concerning external validity, a potential threat to generalization is the possibility that performing the study on different requirements otf 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. Finally, 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.

7.8 Conclusion

Change impact analysis at the requirements level can prove extremely useful for test case prioritization, in the sense that the test cases that need to be exe-

cuted at the end of each release 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 overlapping contracts 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 analyzed the source code and the commit history of two industrial products, recorded the strength of relationships 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 factors 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.