Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Evolution-aware Product-Line Reliability Analysis

Tobias Astoni Sena

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientador
Prof. Dr. Vander Ramos Alves

Brasília
2021

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

Sena, Tobias Astoni
SS474e Evolution-aware Product-Line Reliability Analysis / Tobias Astoni Sena; orientador Vander Ramos Alves. -- Brasília, 2021.
70 p.

Dissertação (Mestrado - Mestrado em Informática) -- Universidade de Brasília, 2021.

1. Software Product Lines. 2. Reliability. 3. Feature Family-based Analysis. 4. Software Evolution. I. Alves, Vander Ramos, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Evolution-aware Product-Line Reliability Analysis

Tobias Astoni Sena

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Prof. Dr. Vander Ramos Alves (Orientador)
CIC/UnB

Profa. Dra. Genaina Nunes Rodrigues
CIC/UnB

Prof. Dr. Pierre-Yves Schobbens
Université de Namur

Prof. Dr. Ricardo Pezzuol Jacobi
Coordenador do Programa de Pós-graduação em Informática

Brasília, 19 de novembro de 2021

Dedicatória

Aos que resistem em prol da ciência.

Agradecimentos

À Mykaella, minha esposa, por todo companheirismo, amor e apoio oferecido durante essa jornada.

Aos meus pais, Sebastiana e Francisco, e meus irmãos, Gabriel e Baltazar, por sempre incentivarem meus estudos durante toda a minha vida.

Ao meu orientador, Prof. Vander Alves, que não poupou esforços, tempo e dedicação para me auxiliar na pesquisa. A gestão e coordenação realizada no grupo de pesquisa foi de extremo profissionalismo e a contribuição final neste trabalho é inestimável.

Aos colegas do grupo de pesquisa Thiago Castro, André Lanna, Leopoldo Teixeira, Breno Bortolli, Danilo Caldas, Igor Correia, Junier Amorim and Sven Apel, por suas colocações valiosas e contribuições diretas neste trabalho.

Por fim, à Fundação de Apoio à Pesquisa do Distrito Federal (FAPDF), cujo apoio parcial da SEI 00193-00000926/2019-67 contribuiu para execução desta pesquisa.

Meu muito obrigado.

Resumo

Contexto: Como qualquer sistema de software, as linhas de produtos de software evoluem. Ainda assim, a maioria do estado da arte das técnicas de análise de linha de produto não considera esse fato e executa a análise do zero em cada etapa da evolução. No caso da análise de confiabilidade, isso significa que, dependendo do cenário de evolução, os cálculos para as partes não afetadas do software são refeitos obtendo os mesmos resultados parciais. Isso desperdiça recursos computacionais, o que é especialmente problemático porque essas análises são demoradas, dado o desafio de lidar com o problema de explosão de estado combinado com a variabilidade inerente às linhas de produtos.

Objetivo: Propomos um método implementado na ferramenta ReAnaE para realizar análises incrementais de confiabilidade da linha de produtos, em que os resultados da análise e artefatos são reutilizados sempre que possível ao longo do histórico de evolução da linha de produtos.

Método: ReAnaE potencializa os esforços de análise anteriores, armazenando etapas de análise intermediárias e traçando cenários de evolução para primitivas computacionais da análise que afetam esses artefatos. A análise de impacto resultante facilita a reutilização consistente de artefatos de análise anteriores e a atualização daqueles afetados pelo cenário de evolução em questão.

Resultados: ReAnaE tem um desempenho melhor em termos de tempo e espaço do que a ferramenta ReAna, alcançando melhorias de até 10 vezes para linhas de produtos maiores, o que resulta em melhorias de até uma ordem de magnitude no número de variantes que podem ser analisadas.

Conclusão: ReAnaE melhora em relação ao estado da arte em análise de confiabilidade de linha de produtos, tornando possível analisar modelos mais complexos de forma eficiente.

Palavras-chave: Confiabilidade, Linha de produtos de software, Análise Feature-Family-based, Evolução de software

Abstract

Context: As any software system, software product lines evolve. Still, most state-of-the-art product-line analysis techniques do not consider this fact and perform analysis from scratch in each evolution step. In the case of reliability analysis, this means that, depending on the evolution scenario, computations for unaffected parts of the software are redone obtaining the same partial results. This wastes computational resources, which is especially problematic since these analyses are time-consuming, given the challenge of coping with the state explosion problem compounded with the variability inherent to product lines.

Objective: We propose a method implemented in the ReAnaE tool to perform incremental product-line reliability analysis, in which analysis results and artifacts are reused whenever possible across the evolution history of the product line.

Method: ReAnaE leverages previous analysis efforts by storing intermediate analysis steps and by tracing evolution scenarios to computational primitives of the analysis affecting these artifacts. The resulting impact analysis facilitates consistently reusing previous analysis artifacts and updating the ones affected by the evolution scenario at hand.

Results: ReAnaE has a better performance in terms of both time and space than the state-of-the-art tool ReAna, achieving up to 10-fold improvements for larger product lines, which results in up to an order of magnitude improvement in the number of variants that can be analyzed.

Conclusion: ReAnaE improves over the state of the art in product-line reliability analysis, making it possible to efficiently analyze more complex models.

Keywords: Reliability, Software Product Lines, Feature-Family-based Analysis, Software Evolution

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A software product line is a family of software systems that share a common set of technical core assets (code, models, etc.), with preplanned extensions and variations to address the needs of specific customers or markets [2]. To model commonalities and variabilities, software product line engineering leverages the concept of *features*—i.e., prominent or distinctive user-visible aspects, qualities, or characteristics of a software system [3]. So, different software products (also known as *variants*) can be generated from the common set of assets according to a given feature selection. Software product lines are able to reduce production costs, increase quality, and decrease time-to-market [4], which makes them well accepted in both industry [5, 6, 7, 8, 9, 10, 11, 12] and academia [2, 4, 13, 14, 15].

The number of possible products of a product line increases exponentially with the number of available features, making it infeasible to quality-check each product individually. Thus, ensuring the quality of all variants is time-consuming and error-prone. This fact has motivated researchers to devise product-line analysis techniques that leverage existing commonality to avoid redundant effort [16, 1]. These techniques can reduce the computational effort of analyzing variants that co-exist—sometimes called *variability in space* [17].

1.1 Problem Statement

As with other kinds of software systems, product lines evolve, which amounts to *variability in time* [17]. Existing product line analysis techniques [18] are mostly unaware of *variability in time*. Therefore, whenever a product line evolves (e.g., a new feature is added), the analysis is performed again over the new set of products. This way, assets and variants that remain the same in both evolution steps, called revisions, of the product line are redundantly analyzed.

In the same fashion, there is already evidence from studies that use evolution-aware tools for analysis, such as the REVISER tool, which updates the results of static analyses for a broad class of data flow analysis for each evolution step [19]. REVISER, however, does not consider *variability in space*.

There has been an effort to bridge the gap between *variability in time* and *variability in space*, for instance, in the form of higher-order deltas [20] and 175% modeling [21], but there is still a gap in simultaneously dealing with both dimensions, not only for reliability analysis techniques.

Problem Statement: The feature-family-based reliability analysis is not evolution-aware.

1.2 Proposed Solution

To close this gap, we present a strategy to perform *evolution-aware* reliability analysis of software product lines. For this purpose, we build on an existing feature-family-based product-line reliability analysis technique [1]. Our evolution-aware product line analysis consists of identifying evolved assets, then tracing the impact of each change to intermediate analysis steps and corresponding results. The evolved assets are analyzed, and the results are combined with the ones that were considered unchanged. This consolidation is performed using the same algorithms that are employed when analyzing the product line from scratch. Hence, we are able to deal with *variability in time* using product-line analysis techniques designed to cope with *variability in space* [17].

To evaluate our evolution-aware product line analysis, we implement it in the tool ReAnaE, which is an evolution-aware extension of ReAna [1]. We empirically compare ReAna and ReAnaE in four different evolution scenarios for six subject product lines. Our results indicate that, overall, ReAnaE has a better performance in terms of both time and space, achieving up to 10-fold improvements for larger product lines, and up to an order of magnitude improvement in the number of variants that can be analyzed.

1.3 Contributions

In summary, our contributions are as follows:

- We present an evolution-aware method for feature-family-based reliability analysis of product lines.

- We design and implement the method in the publicly available tool ReAnaE.[1]
- We report on an empirical study comparing the evolution-aware ReAnaE with a tool that handles only variability in space.

1.4 Outline

The remainder of this work is organized as follows:

- Chapter 2 lays the fundamental concepts for the research presented.
- Chapter 3 refines the research problem statement, scoping each type of software product line evolution scenario that was studied.
- Chapter 4 presents the method for evolution-aware product-line reliability analysis and introduces ReAnaE, an extension of the ReAna tool, capable of applying the proposed method.
- Chapter 5 presents the empirical study comparing the evolution-aware method with the original method.
- Finally, Chapter 6 presents the conclusion, the limitations, related work, and future work.

[1] https://dspl4c2.github.io/tool

Chapter 2

Background

This chapter presents fundamental concepts related to our work and important for the understanding of the problem.

2.1 Software Product Lines

Software product lines provide a form of mass customization by enabling the construction of software products based on reusable components [4]. By reusing parts to build products, software product line engineering is able to increase quality while reducing development costs and production time.

Managing variability is a key principle in software product line engineering. Such variability is typically represented by features, which are graphically represented by a tree called feature diagram, which models the relationships and constraints among features [3]. A product line with n features gives rise to up to 2^n distinct products. Each product is generated by choosing a valid feature selection, which respects the relationships and constraints from the feature model.

Variability is implemented in assets that are associated with features or combinations thereof. In the *annotation-based* approach, one uses special annotations throughout the code to associate snippets with features [4]. It is a simpler technique to be performed and supported by almost all programming environments (e.g., `#ifdef`'s in C/C++). The *composition-based* approach keeps the code of each feature in separate units, such as modules or containers, and the final product is created by composing the units, according to the mapping between features and units. A product line can be defined as a triple (FM, CK, AB), consisting of a feature model FM, an asset base AB (either composition- or annotation-based), and a configuration knowledge CK, which relates features to assets or parts thereof [22].

Analyzing a product line is a non-trivial task. As each product has different characteristics, it is important to establish analysis techniques that can analyze the product line as a whole. Applying traditional software analysis techniques to each individual product is often infeasible, considering that the number of products might grow exponentially with the number of features.

2.2 Product-Line Reliability Analysis

Dependability is an integrating concept tha encompasses as attributes availability, reliability, safety, integrity, and maintainability [23]. Some attributes, such as availability, security, and reliability, are inherently probabilistic. In these cases, the concept of absolute correctness is replaced by bounds on the probability that certain behavior may occur [24, 23]. We considered software reliability from a user's perspective. Such user-oriented reliability of a software program in a given user environment is defined as the probability that the program will give the correct output with a typical set of input data from that user environment [25].

To analyze the behavior of product lines, we can use probabilistic models that are variational [26]. Markov chains are commonly used to specify probabilistic behavior. A Discrete-Time Markov Chain (DTMC) is a kind of Markov Chain where each transition take values in a discrete space. We resort to Parametric Markov Chains (PMC), which extend DTMCs with the capacity to represent variable transition probabilities. The transition probabilities are defined at modeling time, representing the possible behavior of the system. These variables can be leveraged to represent product line variability [27, 16].

Product line analyses aim at determining properties that are valid for all products. Thüm et al. [18] classify product line analyses techniques along three dimensions: *product-based* (one analyzes every software product individually), *feature-based* (all domain artifacts implementing a certain feature are analyzed in isolation), and *family-based* (operates only on domain artifacts and incorporates the knowledge about valid feature combinations). Product-based strategies perform reliability analysis on products derived from behavioral diagrams, which is infeasible for a large number of products. In a feature-based analysis, the artifacts of each feature are analyzed in isolation, and family-based analysis seeks to analyze the commonality between the features of the entire product line, incorporating existing knowledge between them. These strategies can be combined during the analysis, as in the case of the *feature-family-based* analysis, where the first step partially analyzes the features in isolation and then composes the resulting information, generating the analysis of the entire product line.

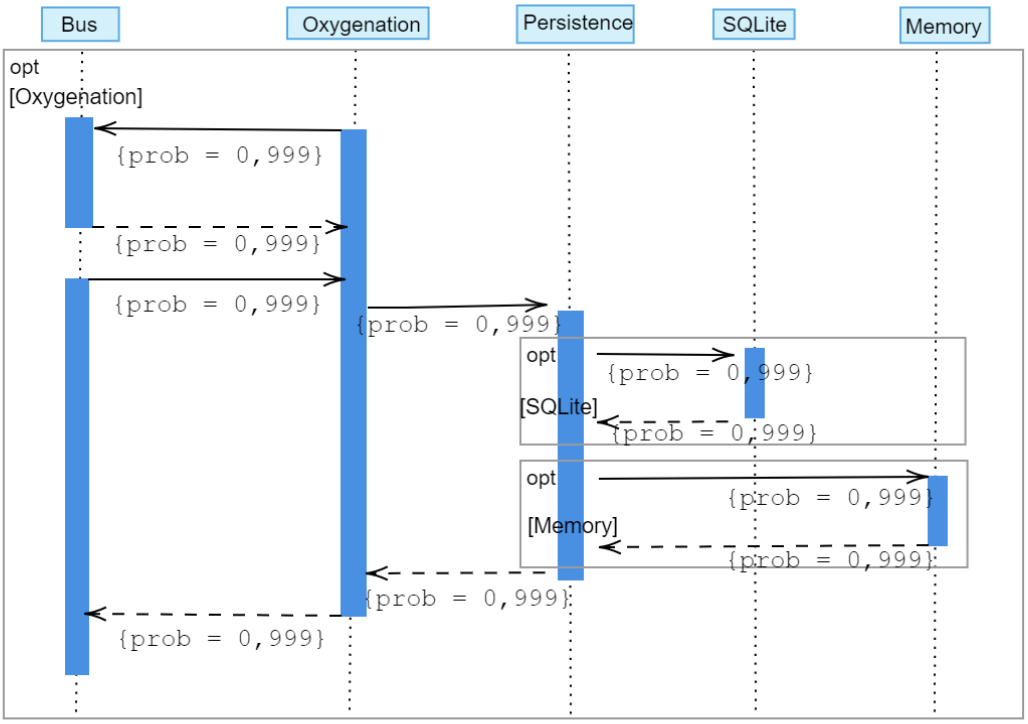

Figure 2.1: Oxygenation Sequence Diagram.

2.3 Feature-Family-Based Reliability Analysis

To tame the exponential blowup of the configuration space, Lanna et al. [1] proposed a feature-family-based reliability analysis technique, implemented in the ReAna tool. The method employs a divide-and-conquer strategy in which pre-computed reliabilities of individual behavioral model fragments associated to features are combined to compute the reliability of the whole product line in a single pass [1]. To achieve this, ReAna performs three main steps: transformation, feature-based analysis, and family-based analysis.

As a running example, let us consider the Body Sensor Network Dynamic Product Line [28]. Feature *Oxygenation* refers to functionality where a sensor captures vital signal from a patient, informing the control bus of any signals received. Optionally, data can be persisted in according to features *Sqlite* or *Memory*, depending on the choice when (re-)configuring the product. Figure 2.1 presents an excerpt of the sequence diagram for the optional behavior of analyzing oxygenation data. Interactions between components are annotated with probabilistic values denoting their success rates. Additionally, the diagram is annotated with variability by means of optional fragments whose guard conditions are propositional expressions over features, representing presence conditions [29].

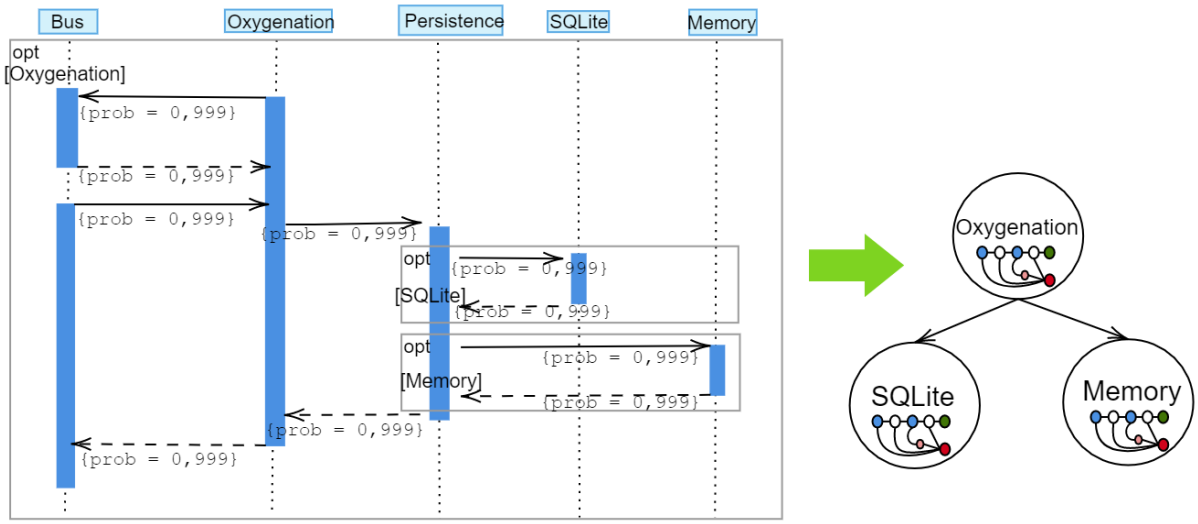

Figure 2.2: Transformation step.

Transformation step. ReAna first converts the behavioral models and their inherent variability shown in Figure 2.1 into a dedicated data structure called Runtime Dependency Graph (RDG) [1]. The RDG is a representation of the product line, which contains configurability information with its probabilistic behavior. Figure 2.2 shows the result of the transformation step. Each RDG node has a corresponding stochastic model (FDTMC) [27] associated with the behavioral models, and each edge represents dependency between the latter. In the Figure 2.2, the `Oxygenation` fragment depends on the `SQLite` and `Memory` fragments.

Feature-based step. This step is performed by analyzing the FDTMC of each RDG node individually, taking a compositional approach according to the existing dependencies between the nodes. To analyze each FDTMC, we use a parametric model checker, such as PRISM [30] or PARAM [31]. Then, each RDG node is augmented with the corresponding reliability expression. Figure 2.3 shows the feature-based step being applied to the RDG resulting from the transformation step. The `SQLite` and `Memory` nodes, which have no dependents, have their expressions defined only by constants, while the `Oxygenation` node has its expression with variables that depend on other nodes.

Family-based step. To carry out the family-based analysis, ReAna lifts the expressions generated per feature by the model checker using a variational data structure, the Algebraic Decision Diagram (ADD) [32]. This way, when the variables of an expression resulting from the feature-based step are substituted by an ADD, the reliability of the entire product line is computed by a bottom-up evaluation of the ADDs along the RDG guided by the constraints imposed by the feature model [1]. Figure 2.4 shows the resulting ADD from such evaluation. Dashed lines represent the absence of a feature. The absence

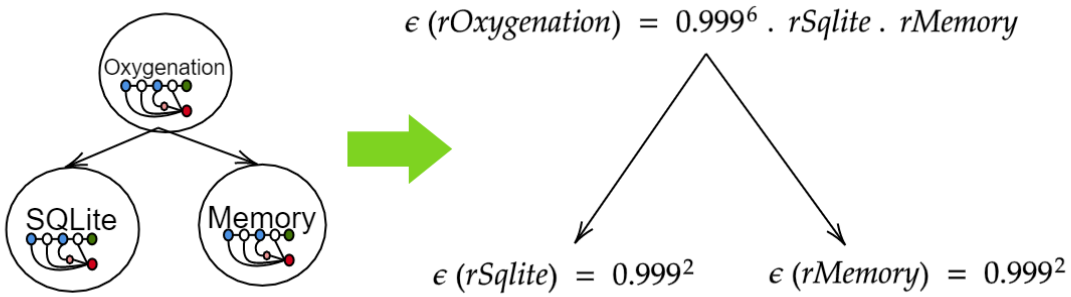

Figure 2.3: Feature-based step.

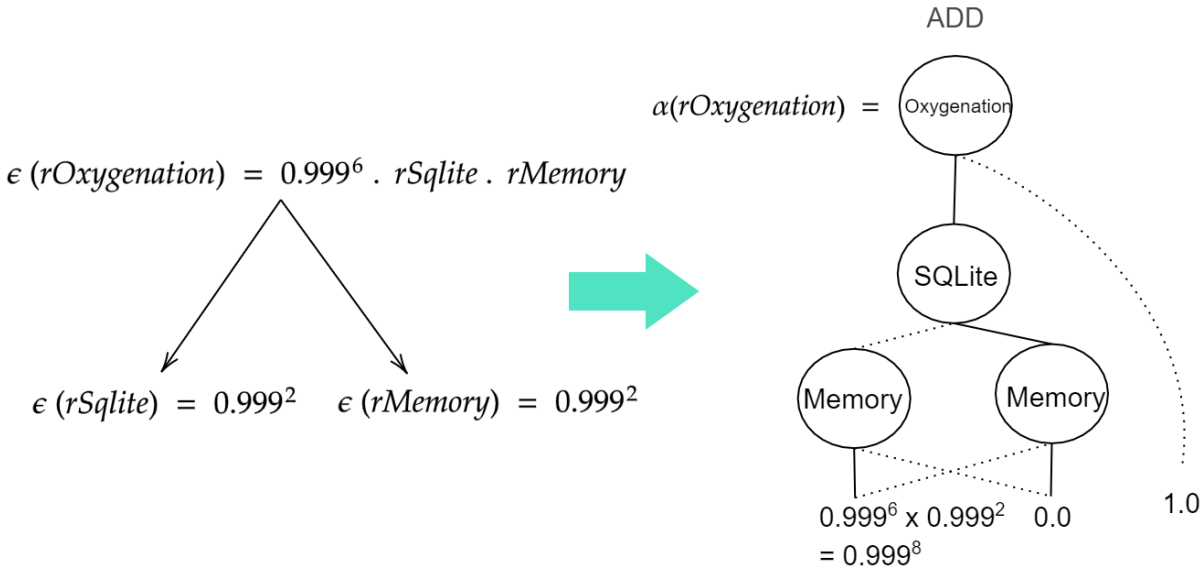

Figure 2.4: Family-based step.

of the *Oxygenation* feature results in the neutral value for multiplication 1. Choosing the *Sqlite* and *Memory* features leads to a value of 0. Since they are mutually exclusive, choosing both features would lead to an invalid feature selection. All paths leading to a non-zero terminal are valid configuration.

Chapter 3

Problem Statement

State-of-the-art product-line reliability analysis methods do not address evolution. If one applies such analysis and the model evolves, one has to perform once again all computational analysis steps. Nevertheless, since some parts of the model might not be affected by the evolution, and given that some reliability analyses exploit compositionality, there is an untapped potential for reuse of intermediate analysis results. In this chapter, we scope this problem with the help of four evolution scenarios.

3.1 Evolution Scenarios

A product line can evolve in various ways, such as adding new features, changing dependencies, or modifying internal behavior [33]. We can decompose complex evolution scenarios into primitive ones, involving addition, removal, and update of problem and solution space artifacts as well as the mapping between them [34, 35]. We concentrate on feature, message, and fragment addition as well as changes in presence conditions, since these are observed to occur frequently (as identified in previous work that have considered the evolution of highly configurable systems, such as the Linux kernel [36, 33, 37]). In what follows, we describe and illustrate these scenarios as well as outline a corresponding change impact analysis in the reliability analysis computed by ReAna, pinpointing optimization opportunities.

We illustrate these evolution scenarios starting from the sequence diagram in Figure 2.1. ReAna generates the RDG given in Figure 3.1 (after the transformation step in Figure 2.2) and the expressions in Figure 2.3 (after the feature-based step).

New feature In the first evolution scenario, we add the new optional *File* feature and its corresponding behavioral model fragment to the product line, as shown in Figure 3.3. When analyzing the product line obtained in this evolution scenario, ReAna first yields

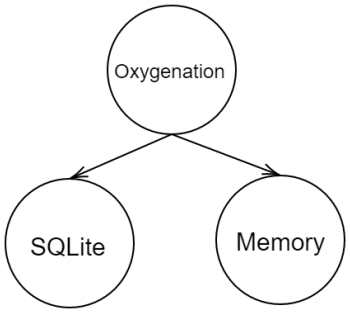

Figure 3.1: RDG of feature *Oxygenation*.

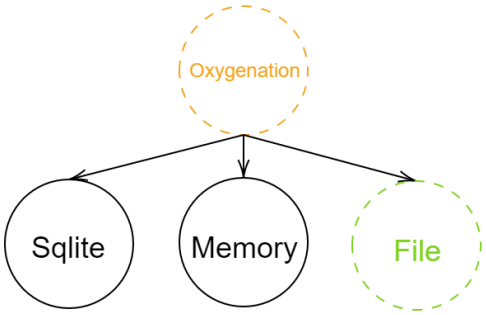

Figure 3.2: RDG with *File* feature added and the impact on the parent node.

the RDG shown in Figure 3.2, which shows that the new feature modifies *Oxygenation*'s node dependencies.

Figure 3.4 illustrates the feature-based analysis step. Comparing it with Figure 2.3, we see that the reliability expression for *Oxygenation* changed from Expression 3.1 to Expression 3.2:

$$0.999^6 \cdot rSqlite \cdot rMemory \tag{3.1}$$

$$0.999^6 \cdot rSqlite \cdot rMemory \cdot \mathbf{rFile} \tag{3.2}$$

However, the expressions for nodes *Sqlite* and *Memory* are not affected. Thus, running ReAna with the entire evolved product line as input unnecessarily recomputes the expressions and ADDs related to *Sqlite* and *Memory*, which are exactly the same as in the previous version.

Addition of messages In the second evolution scenario, we add two new messages to the behavioral diagram of the *Oxygenation* feature, as shown in Figure 3.5. In this case, analyzing the evolved model results in an RDG with the same topology as the original, but with a new model in the *Oxygenation* node. As a result, the expression generated

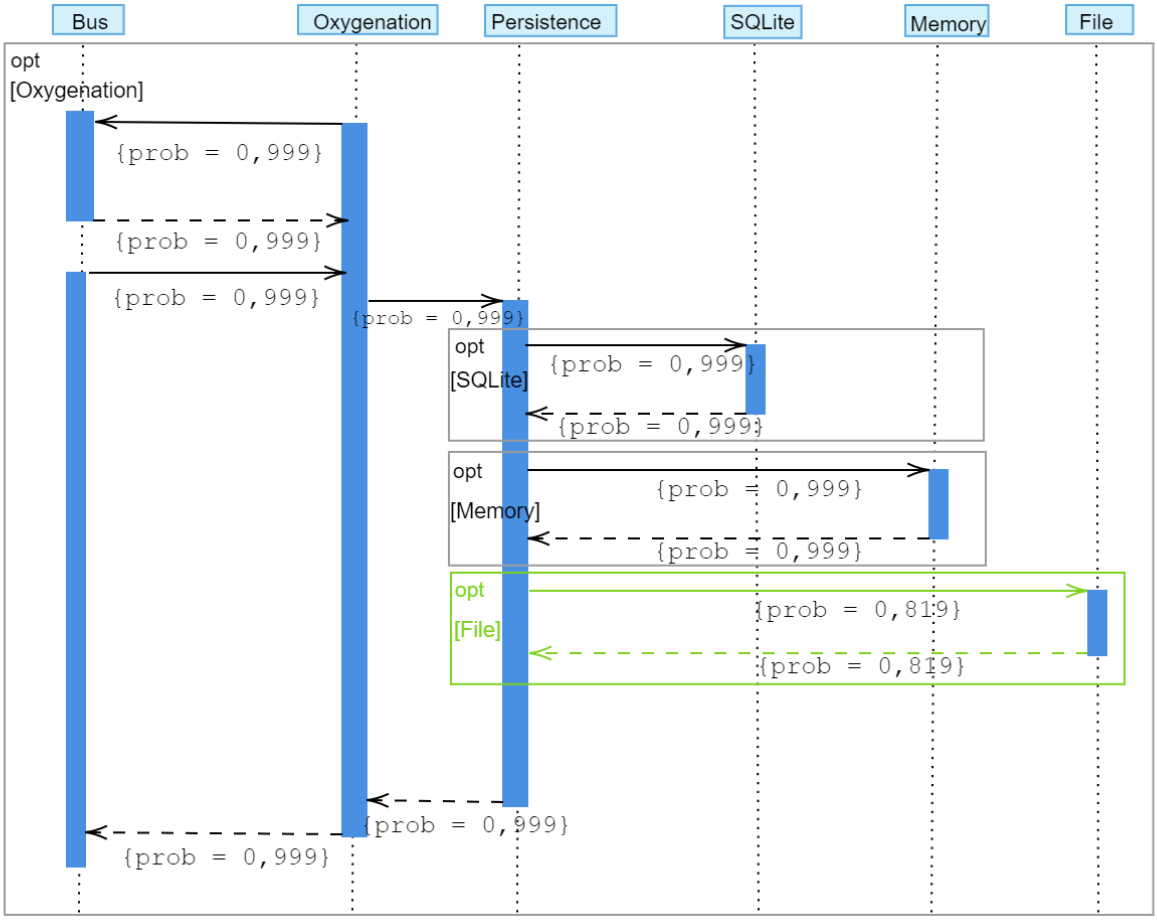

Figure 3.3: Sequence diagram with feature *File* added.

in the feature-based step remains the same for the *Memory* and *Sqlite* nodes, but the expression for the *Oxygenation* node changes from Expression 3.3 to Expression 3.4:

$$0.999^{6} \cdot rSqlite \cdot rMemory \tag{3.3}$$

$$\mathbf{0.999^{8}} \cdot rSqlite \cdot rMemory \tag{3.4}$$

Applying ReAna after this evolution scenario results not only in computing the new expression for *Oxygenation* and evaluating it, but also in unnecessarily *recomputing* and evaluating the expressions for *Memory* and *Sqlite*.

Addition of a behavioral fragment For the third evolution scenario, consider the inclusion of a fragment in the behavioral model of the *Oxygenation* feature. In this case, there is no addition of a new feature to the product line with its associated behavioral model, but just a fragment in a behavioral model of an existing feature. There is a change

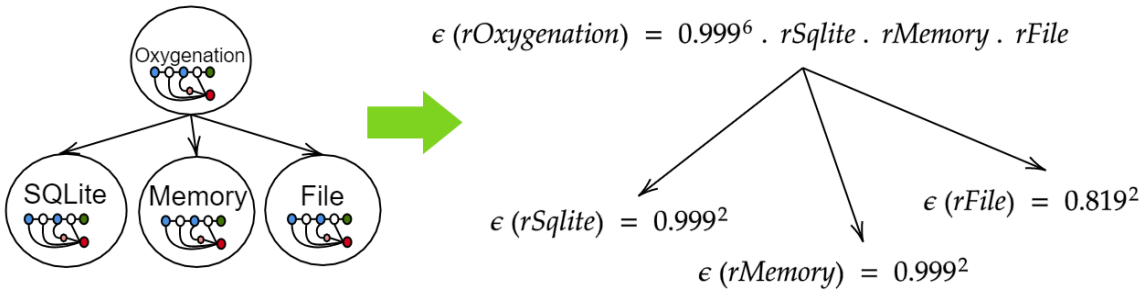

Figure 3.4: Feature-based step with new *File* feature added.

in the RDG topology, however the added node does not represent a new feature. Figure 3.6 shows the behavioral diagram with a new fragment and Figure 3.7 shows the feature-based step applied in this scenario, yielding the expression $0.999^6 \cdot rSqlite \cdot rMemory \cdot rNewFrag$. Applying ReAna after this evolution scenario implies unnecessary analysis. There is no need to *recomputing* the expression of *Sqlite* and *Memory*.

This scenario is similar to the first evolution scenario. However, although the number of nodes in the RDG increases, the size of the product line configuration space remains the same. This requires a smaller variational data structure during the family-based step.

Change in presence condition Finally, the fourth evolution scenario changes the guard condition of a specific fragment in a sequence diagram. This scenario does not change the RDG topology or the expressions generated by the feature-based step, preserving the original scenario as Figure 2.3. However, the family-based step recomputes the ADDs of features that had the presence condition changed. Applying ReAna after this evolution scenario results in unnecessarily computing the expression of all RDG nodes, as there was no change in the behavioral models of the product line.

Summary In all scenarios, part of the expressions to be computed and evaluated remain the same with respect to the original model. The family-based analysis step is performed based on expressions that are generated along the RDG structure. Figure 3.8 shows a comparison of nodes impacted on the RDG for the different evolution scenarios. Depending on the modifications generated in the RDG, applying a new feature-family-based analysis to the entire model entails that a significant part of the calculations have the same result as the analysis made in previous models. Thus, there is computational waste resulting from the fact that the analysis is not evolution-aware.

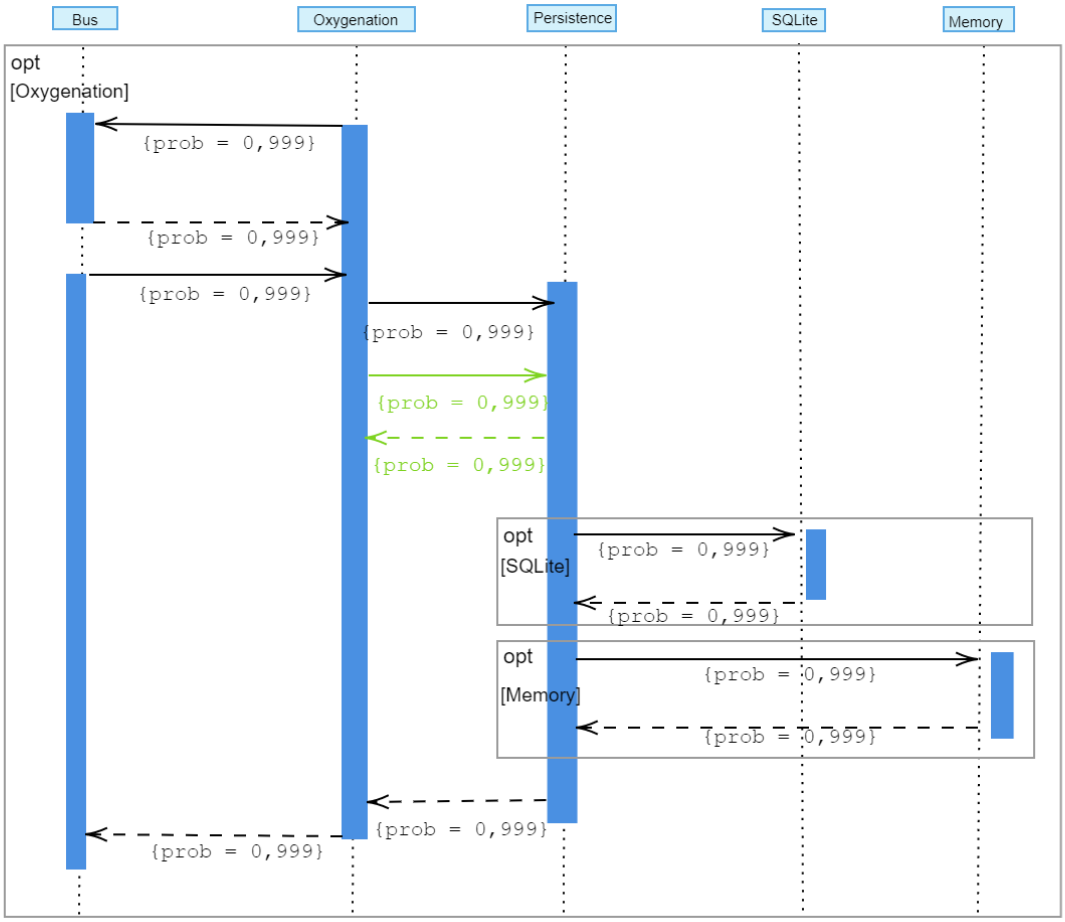

Figure 3.5: Sequence diagram with new messages added.

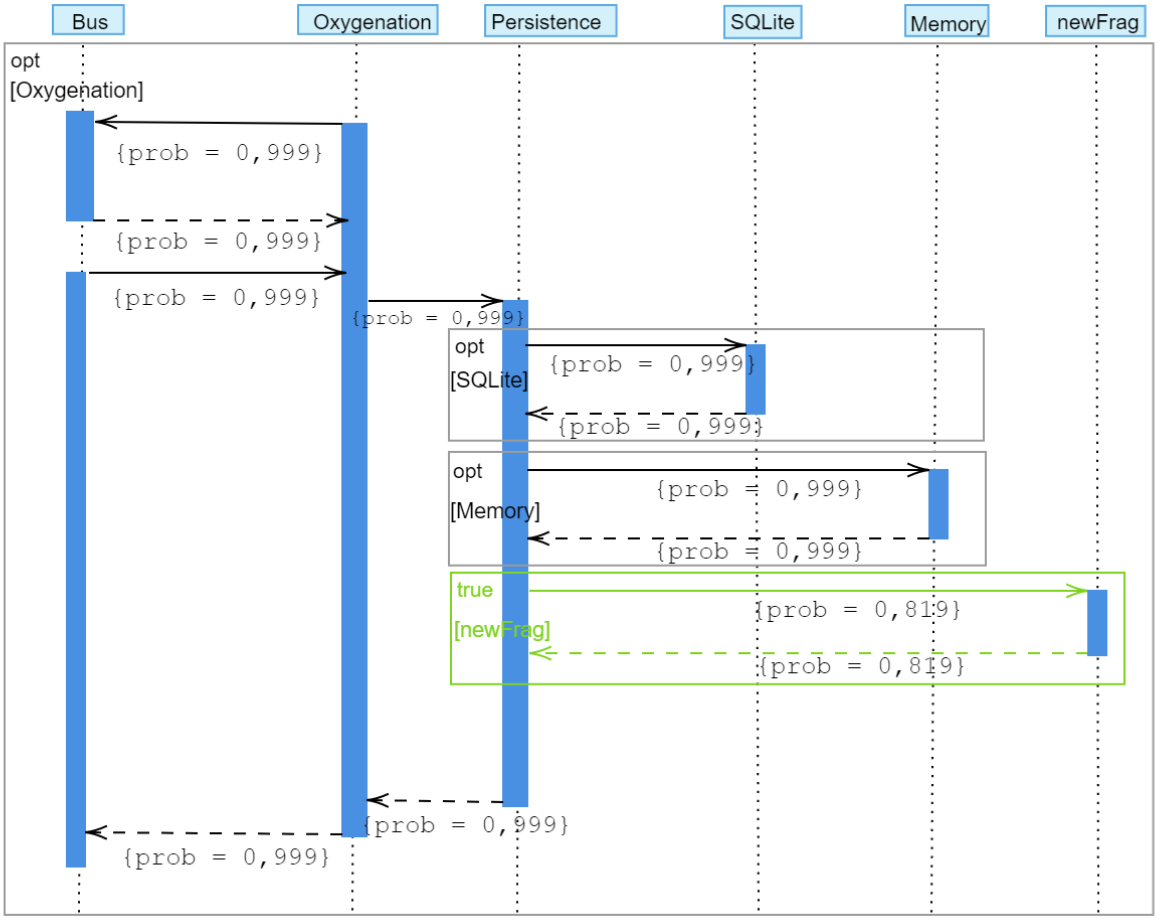

Figure 3.6: Sequence Diagram with new fragment.

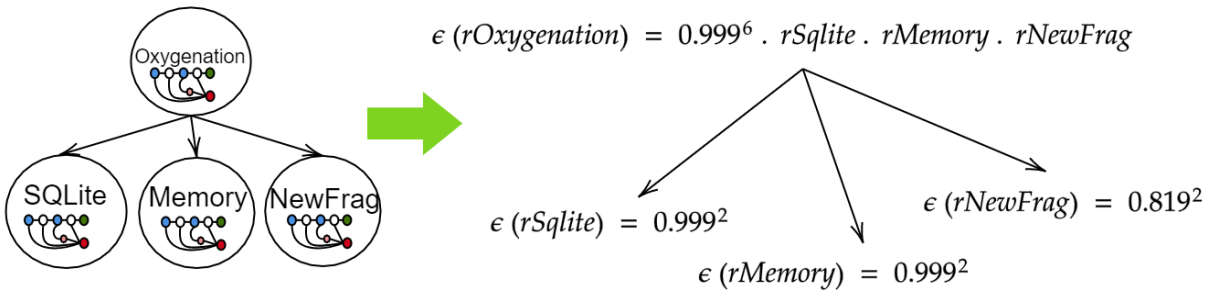

Figure 3.7: Feature-based step with new fragment in *Oxygenation* feature.

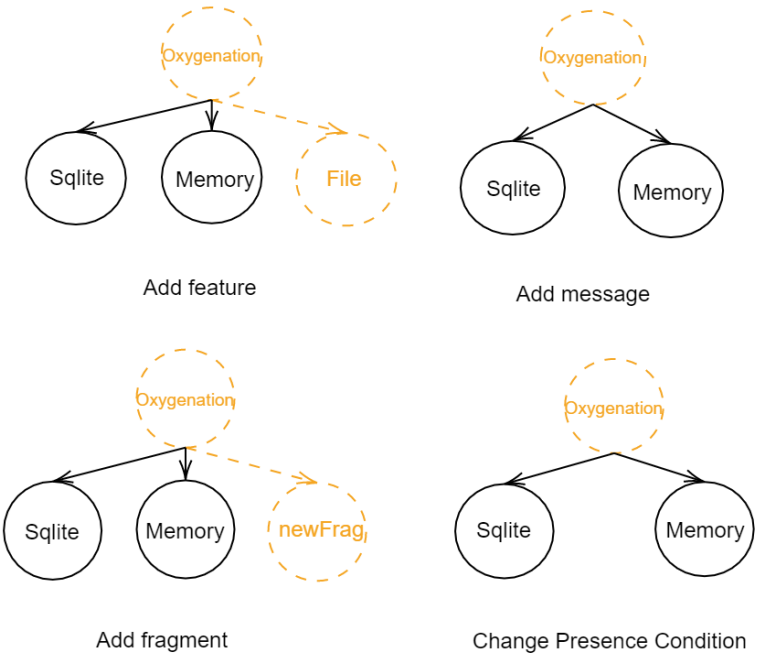

Figure 3.8: Comparison of the impact of each evolution scenario on the RDG.

Chapter 4

Method and Supporting Tool

In this chapter, we present a method for performing evolution-aware feature-family-based reliability analysis and the supporting tool ReAnaE. Initially, Section 4.1 overviews the method, and Section 4.2 provides its functional description. We also present ReAnaE´s logical view (Section 4.3), its architecture (Section 4.4), and its implementation (Section 4.5).

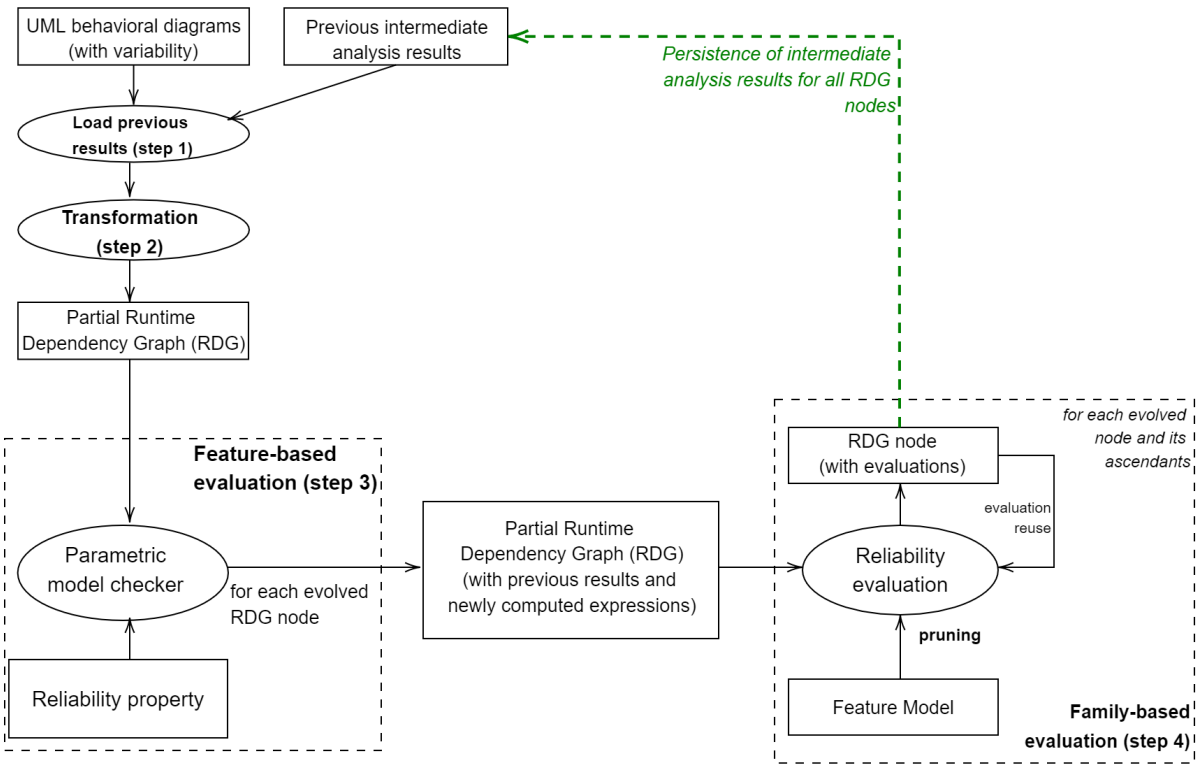

Figure 4.1: Evolution-aware feature-family-based reliability analysis of product lines (adapted from Lanna et al. [1]).

4.1 Method Overview

Our proposed method avoids redundant computation by reusing reliability analysis results from previous versions of the product line. It extends the original variability-aware reliability analysis method proposed by Lanna et al. [1] to make it evolution-aware. Similar to the original, our extended method is a feature-family-based analysis. Figure 4.1 provides an overview of the method, which adds a preparation step before the three phases described in Section 2.3.

When a product line is analyzed for the first time, the preparation step is skipped. In this case, our method works almost exactly as the one by Lanna et al. [1]. The only difference lies in the last step (step 4 – *family-based evaluation*), in which the ADDs corresponding to the partial reliability values for each RDG node are persistently stored. Whenever the product line in question evolves, these intermediate results are available to speed up the analysis of its new revision. The first step (i.e., preparation) of the evolution-aware reliability analysis consists of loading the results obtained from the previous revision. After that, the input behavioral models are transformed into an RDG as they would in the original method (step 2 – *transformation*). However, before proceeding to the feature-based evaluation (step 3), we determine which RDG nodes were impacted by evolution (according to the scenarios and corresponding impact analyses presented in Section 3.1) and mark them as such. Then, each node in a path from the RDG root to a marked node is also marked, to account for indirect impact. So, during the feature-based step, *only* the FDTMCs corresponding to impacted nodes go through parametric model checking. This saves analysis time by not computing reliability expressions that would be unchanged. Finally, the family-based step is performed to recursively compute reliability ADDs. At this point, the computation for a marked node may require the reliability ADD for a node that was *not* impacted by evolution. Whenever that happens, we reuse existing data (step 1), instead of proceeding recursively. This way, our method saves analysis time by avoiding unnecessary operation of ADDs.

It is worth noting that, during the family-based step, the technique proposed by Lanna et al. [1] reuses the reliability ADDs computed for RDG nodes on which more than one other node depends. Our evolution-aware method leverages the same memoization strategy to reuse reliability ADDs computed for previous *revisions* of unchanged nodes. Thus, the evolution-aware reliability analysis method handles variability in both space *and* time using similar mechanisms.

Moreover, the main difference between the two analysis methods is that, in our evolution-aware method, the actual memoization data structure is pre-populated with ADDs that were previously persisted. Hence, we may consider that the analytical complexity of our extended method is the same as the one reported by Lanna et al. [1]. This

means that differences in performance are expected to arise from practical issues, such as disk access and ADD ordering (cf. Chapter 5).

4.2 Functional Description of the Method

We now present a functional formulation of our method. In the following, we denote the evolution-aware reliability analysis by function ϕ', which takes as input a behavioral model m in RDG form and an evolution function δ according to the scenarios described in Chapter 3. It outputs an ADD representing the reliability of the model resulting from evolving m with δ:

$$\phi'(m,\delta) = \begin{cases} \phi(m) & \delta = \mathbf{id} \\ \phi(m^\delta) & deps(r_m) = \emptyset \\ \phi^{ind}\Big(r_m^\delta, map\big(\phi'(\cdot,\delta), deps(r_m)\big)\Big) & \bigvee\{message(\delta), pc(\delta)\} \\ \phi^{ind}\Big(r_m^\delta, \phi(\delta) : map\big(\phi'(\cdot,\delta), deps(r_m)\big)\Big) & \bigvee \left\{ \begin{array}{l} newFeature(\delta), \\ newFragment(\delta) \end{array} \right\} \\ \phi^{ind}\Big(r_m, map\big(\phi'(\cdot,\delta), deps(r_m)\big)\Big) & otherwise \end{cases}$$

We use m^δ to denote application of the evolution function δ on model m, and r_m^δ to denote the root of this model. Function ϕ is the feature-family-based analysis computed by ReAna, and ϕ^{ind} denotes its node-wise definition, receiving as arguments a behavioral model at a node and a list of reliability ADDs from each dependency of that node. It then performs the feature-based step at this node, resulting in expression e, followed by the family-based step, evaluating e with the list of ADDs. We overload $\phi(\delta)$ to denote $\phi(m')$ if δ adds a new fragment or feature m'. For conciseness, we denote a lambda abstraction $\lambda s \to f(s)$ by $f(\cdot)$. Moreover, colons are used to denote *prepending* an element as the head of a list—that is, $a : b$ denotes the list where a is the head and b is the tail.

The case-wise definition of ϕ' examines the evolution function in relation to the node under application. The two base cases happen when the evolution is the identity function or the current node has no dependencies. The next two cases examine whether the evolution applies one of the four evolution scenarios to the current node. Finally, we examine the case where the evolution applies to some node that is not the current one, in which the function is simply mapped to each of the dependencies.

ReAnaE's soundness is stated as follows. For every behavioral model m and evolution function δ, we have that $\phi'(m, \delta) = \phi(m^{\delta})$. In other words, ReAnaE and ReAna yield the same result. The intuition behind this fact is that ϕ' unfolds ϕ along the evolved sub-models of m (cf. cases 3–6 of the definition of ϕ') until it can reuse previous analyses, eventually bootstrapping from ϕ (cases 1 and 2 of the definition of ϕ'). The proof follows by induction on the well-founded relation induced by m and case analysis on δ. This result is contingent on the correctness of the impact analysis discussed in Chapter 3.

4.3 ReAnaE´s Logical View

ReAnaE performs evolution-aware feature-family-based reliability analysis of software product lines from feature and UML behavioral models. ReAnaE takes as input a UML activity diagram representing the coarse-grained behavior, a set of UML sequence diagrams, representing the fine-grained behavioral of each activity, and a feature model described in conjunctive normal form (CNF). The behavioral models used were created by SPL-Generator tool[1] and the conjunctive normal form by FeatureIDE [38]. The behavioral models can also be created in MagicDraw tool[2]. As output, the tool computes the ADD representing the reliability of all products in the product line and prints the reliability of all possible configurations.

Listing 4.1 shows an excerpt of a behavioral model used as input to the tool. The model is in XML and contains information about the activity diagram (Lines 4-18), sequence diagram (Lines 24-34), messages (Lines 26-30), and guard conditions (Lines 25 and 32) present in the product line.

Listing 4.1: Excerpt from a behavioral model of BSN product line

```
1 <SplBehavioralModel name="translatedBSN">
2     <ActivityDiagram name="AD_SPL_0">
3         <Elements>
4             <ActivityDiagramElement name="Start node" type="StartNode"/>
5             <ActivityDiagramElement name="Capture" type="Activity">
6                 <RepresentedBy seqDiagName="Capture"/>
7             </ActivityDiagramElement>
8             <ActivityDiagramElement name="End node" type="EndNode"/>
9             <ActivityDiagramElement name="Reconfiguration" type="
    Activity">
10                <RepresentedBy seqDiagName="Reconfiguration"/>
11            </ActivityDiagramElement>
```

[1] https://github.com/SPLMC/spl-generator/
[2] https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/

```
12          <ActivityDiagramElement name="DecisionNode_0" type="
    DecisionNode"/>
13          <ActivityDiagramElement name="QoSChange" type="Activity">
14              <RepresentedBy seqDiagName="QoSChange"/>
15          </ActivityDiagramElement>
16          <ActivityDiagramElement name="Situation" type="Activity">
17              <RepresentedBy seqDiagName="Situation"/>
18          </ActivityDiagramElement>
19      </Elements>
20      <Transitions>
21          ...
22      </Transitions>
23   </ActivityDiagram>
24   <SequenceDiagrams>
25      <SequenceDiagram guard="true" name="QoSChange">
26          <Message name="" probability="0.999" source="Mock lifeline"
    target="Lifeline_0" type="asynchronous"/>
27          <Message name="getQoSRequired" probability="0.999" source="
    Mock lifeline" target="Lifeline_0" type="synchronous"/>
28          <Message name="returnQoSRequired" probability="0.999" source
    ="Mock lifeline" target="Lifeline_0" type="synchronous"/>
29          <Message name="notifyChange" probability="0.999" source="
    Mock lifeline" target="Lifeline_0" type="synchronous"/>
30          <Message name="replyNotify" probability="0.999" source="Mock
     lifeline" target="Lifeline_0" type="synchronous"/>
31      </SequenceDiagram>
32      <SequenceDiagram guard="Memory" name="n6">
33          ...
34      </SequenceDiagram>
35      ...
36   ...
```

The other input to ReAnaE is the feature model in CNF. Figure 4.2 shows the input for BSN. In this way, it is possible to determine during the analysis which product configurations are valid for the analyzed product line.

```
Root && (Root || !Monitoring) && (Root || !Storage) && (Monitoring || !Root) &&
(Storage || !Root) && (SensorInformation || !Oxygenation) && (SensorInformation ||
!PulseRate) && (SensorInformation || !Temperature) && (SensorInformation || !Position)
&& (SensorInformation || !Fall) && (Oxygenation || PulseRate || Temperature ||
Position || Fall || !SensorInformation) && (Storage || !Sqlite) && (Storage || !Memory)
&& (Sqlite || Memory || !Storage) && (!Sqlite || !Memory) && (Monitoring ||
!SensorInformation) && (Monitoring || !Sensor) && (SensorInformation || !Monitoring)
&& (Sensor || !Monitoring) && (Sensor || !SPO2) && (Sensor || !ECG) && (Sensor ||
!TEMP) && (Sensor || !ACC) && (SPO2 || ECG || TEMP || ACC || !Sensor) &&
(!Oxygenation || SPO2) && (!PulseRate || SPO2 || ECG) && (!Fall || ACC) &&
(!Position || ACC) && (!Temperature || TEMP) && True && !False
```

Figure 4.2: CNF input for BSN.

The output is an ADD representing the reliability of the entire product line, and the list of all possible product configurations and their reliability values.

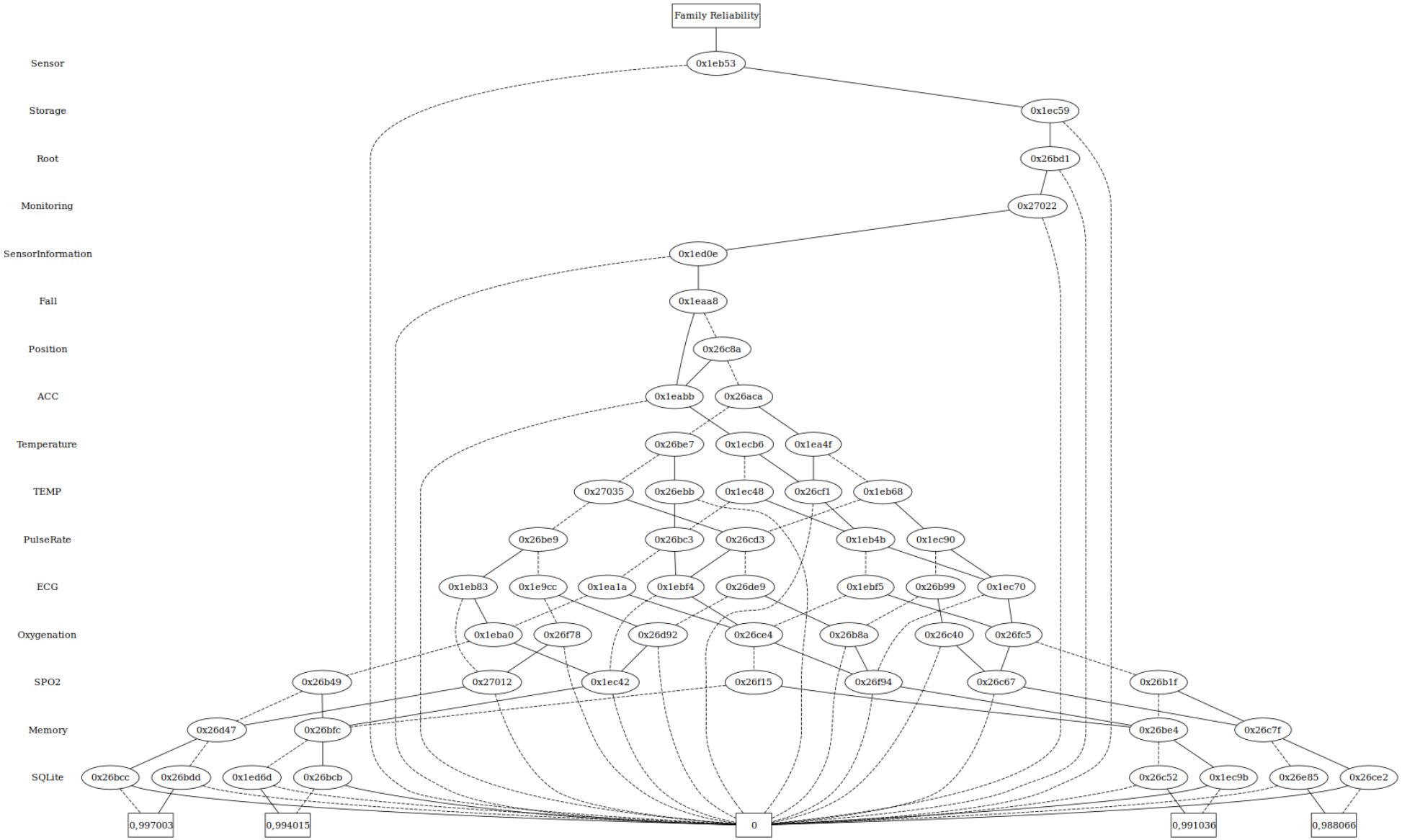

Figure 4.3: `Capture` fragment ADD.

Figure 4.3 shows the ADD of the `Capture` fragment, which represents an intermediate calculation performed during the BSN analysis. The ADD is topologically ordered and going from the root to the leaf choosing the presence (continuous line) or not (dotted line) of the fragment determines the reliability of the chosen configuration.

To carry out the evolution-aware analysis, ReAna also needs to perform the reading and persistence of the ADD. Data reading is performed if the analysis is an evolved model. In this way, at the beginning of the ReAnaE execution, if the model under analysis is an evolved model, the intermediate ADD used in previous analyses are read from disk. Data persistence is performed at the end of the ReAnaE execution. In this way, all ADD resulting from the analysis are stored on disk.

4.4 ReAnaE´s Architecture

As an extension of ReAna, three new components were added to ReAnaE, for it to be able to address variability in time: data reading, data persistence, and new feature-family-based analysis.

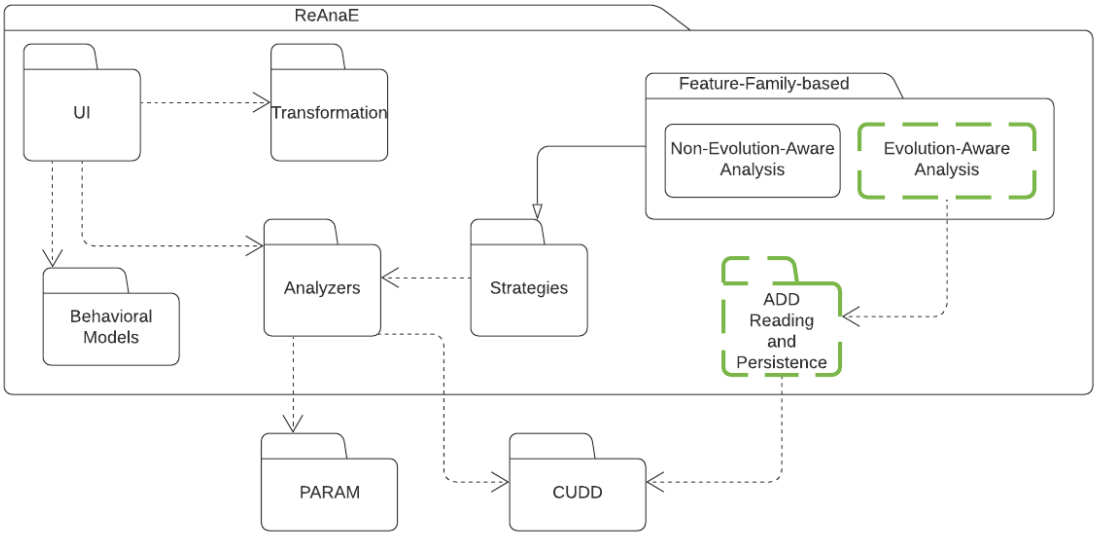

Figure 4.4: UML Package Diagram of ReAnaE.

The handling of ADD is performed by the `CUDD` library, present at ReAna. To perform reading and persistence, methods from the `CUDD` library are also used. The new feature-family-based analysis implements the `IReliabilityAnalysisResults` interface present in `Strategies` package of ReAna. During evolution-aware analysis, all RDG nodes that were impacted in the evolution step are first identified. For this, from the list of fragments that have undergone modifications, a depth-first search in RDG is performed to determine

all nodes that depend on them, as they will also need to be recalculated as impacted nodes. For those who were not impacted, the corresponding ADDs read from disk is reused. For impacted nodes, a new analysis will be computed. Figure 4.4 shows ReAnaE´s packet diagram. In dashed line are the new modules that were added in ReAna so that it was possible to perform the evolution-aware analysis.

ReAna implements different analysis strategies for product lines. In this work, we used the feature-family-based analysis because it obtained the best results, according to the study by Lanna et. al [1]. The feature-family-based analysis implements the `IReliabilityAnalysisResults` interface. To implement the method proposed by this research in Section 4.1, it was necessary to implement the evolution-aware part. This analysis depends on the `ADD Reading and Persistence` package, which had to be added to ReAna, as shown in Figure 4.4. In addition, in order to develop it, we used the DDDMP package native from CUDD to manipulate ADD.

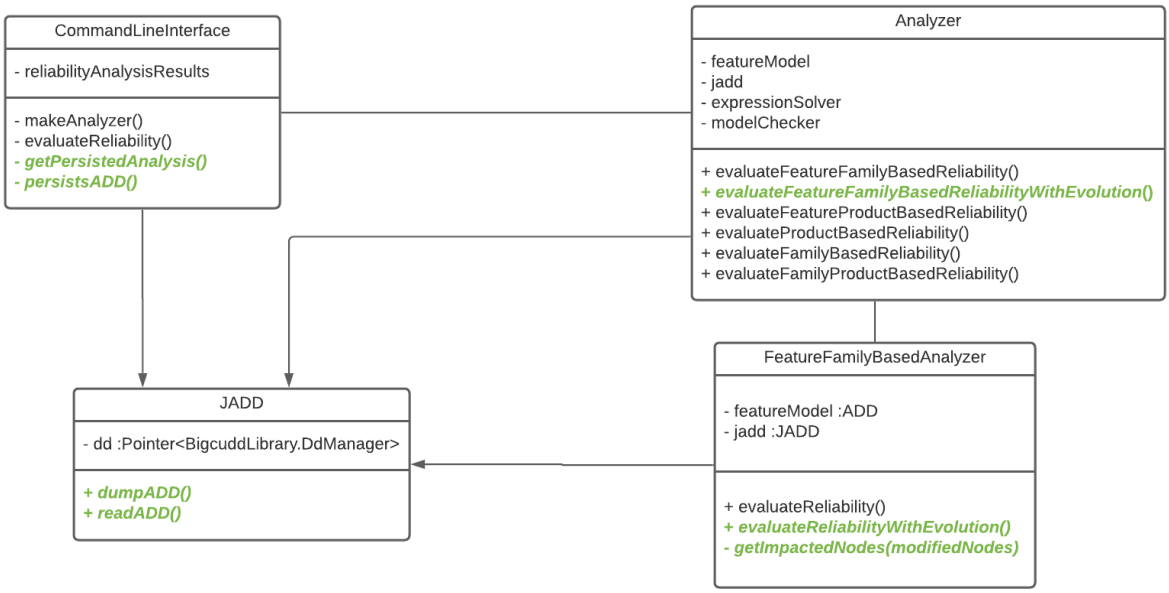

Figure 4.5: UML Class Diagram of ReAnaE.

Figure 4.5 shows a class diagram of the main elements introduced in ReAnaE to allow the construction of the evolution-aware method. We highlighted in green the main functions that needed to be created in the tool. The `CommandLineInterface` contains ReAnaE's main class. It initializes the instrumentations for key analysis metrics, time and memory collection. From it, the `Analyzer` class is started, which is mainly responsible for organizing all the analyses to be done. It orchestrates the analysis tasks and initializes the main components used: `jadd`, created to manage and manipulate ADD through the CUDD library; `featureModel`, which has all the ADD variables that will be used; `expressionSolver`, which encapsulates the model checker used in the analy-

sis and is responsible for solving the expression resulting from it. The `Analyzer` class has been extended to manage evolution-aware feature-family-based analysis as well. The `CommandLineInterface` calls two of the main methods created, responsible for reading the persisted analysis data and for persisting the analysis results. For this, the `JADD` class, responsible for the ADD operations, needed to be expanded, adding DDDMP package methods to perform the read and dump operations of the ADD structure. The `FeatureFamilyBasedAnalyzer` class is responsible for orchestrating the feature-family-based analysis. It has been extended to support evolution-aware analysis as well. When the analysis deals with an evolved model, the `evaluateReliabilityWithEvolution` method will be responsible for performing the analysis. In addition, it is responsible for checking all nodes impacted by evolution, through the evolution primitives applied in the models.

4.5 ReAnaE´s Implementation

ReAnaE was developed in Java and the source code is open and publicly available[3]. To be able to perform the evolution-aware analysis, the key extensions to ReAna are related to reading, analysis and persistence of ADDs.

Listing 4.2: Method to read a persisted ADD

```
1  public ADD readADDpreviousAnalysis(String fileName) {
2      Pointer<?> input = CUtils.fopen(fileName, CUtils.ACCESS_READ);
3
4      IntValuedEnum<BigcuddLibrary.Dddmp_VarMatchType> varMatchMode =
       BigcuddLibrary.Dddmp_VarMatchType.DDDMP_VAR_MATCHIDS;
5      int mode = BigcuddLibrary.DDDMP_MODE_TEXT;
6      Pointer<Byte> file = Pointer.pointerToCString(fileName);
7      Pointer<DdNode> node = BigcuddLibrary.Dddmp_cuddAddLoad(dd,
8          varMatchMode,
9          null,
10         null,
11         null,
12         mode,
13         file,
14         input);
15
16     CUtils.fclose(input);
17     ADD retrievedADD = new ADD(dd, node, variableStore);
18     return retrievedADD;
19 }
```

[3]https://dspl4c2.github.io/tool

Listing 4.2 shows the method created to read a ADD persisted in previous analyses. This method is called at the beginning of ReAnaE´s execution, if it is an evolved model. To read the persisted data, we use DDDMP, a native package from the CUDD library. To call DDDMP (Lines 7-14), we pass as parameters: variable `dd`, the pointer responsible for managing the library (Line 7); variables `varMatchMode` and `mode`, the type flags used (Lines 8 and 12); variable `file`, the persisted ADD file name (Line 13); and variable `input`, the type of access (Line 14). With the returned file, we call the original ReAna function responsible for creating an ADD (Line 17).

Listing 4.3: Main method of ReAnaE's evolution-aware analysis

```
1  public IReliabilityAnalysisResults evaluateReliabilityWithEvolution(
   RDGNode node, ConcurrencyStrategy concurrencyStrategy, String
   idChangedFragment, Map<String, ADD> previousAnalysis) throws
   CyclicRdgException {
2
3      List<RDGNode> dependencies = getImpactedNodesDFS(node,
   idChangedFragment, previousAnalysis);
4
5      // Feature-based analysis
6      List<Component<String>> expressions = firstPhase.
   getReliabilityExpressions(dependencies, concurrencyStrategy);
7
8      // Lift expressions to ADD operators
9      List<Component<Expression<ADD>>> liftedExpressions = expressions
   .stream()
10             .map(helper::lift)
11             .collect(Collectors.toList());
12
13     // Family-based analysis
14     ADD reliability = newSolveFromMany(liftedExpressions,
   previousAnalysis);
15
16     return new ADDReliabilityResults(result);
17 }
```

Listing 4.3 shows the evolution-aware feature-family-based method. This method is called every time the analyzed model has already been analyzed before. Otherwise, ReAna's feature-family-based analysis is called.

The first step of the analysis is to compute the nodes impacted by evolution (Line 3). according to the evolution scenarios described in Chapter 3. The method makes an depth-first search of the RDG and finds all nodes that were impacted by the evolution scenario and return them. Following our method proposed in Section 4.1, the analysis will

only be computed on the impacted nodes. For the rest, we use the calculations performed in previous analyzes that were persisted on disk, retrieved as seen in Listing 4.2.

For the feature-based step (Line 6), it was not necessary to change the ReAna analysis method. The difference is that, instead of calculating the expressions of all nodes, only the list of nodes that were impacted by evolution are passed as an argument. In this step, using PARAM, a parametric model checking analysis of the reliability present in the FDTMC of each node is performed.

After generating the expressions in the feature-based step, the method lifts each expression to perform arithmetic operations over variational data (Line 9), in this case the chosen structure is the ADD.

Finally, the family-based step (Line 14) computes an ADD representation of node reliability, evaluating each of the reliability expressions. However, for evolution-aware analysis, the reliability expressions will only be those of the impacted nodes.

Listing 4.4: Method to persist the ADD

```
1   public void dumpADD(String functionName, ADD add, String fileName) {
2   Pointer<?> output = CUtils.fopen(fileName, CUtils.ACCESS_WRITE);
3
4   Pointer<Byte> ddname;
5   if (functionName == null || functionName.isEmpty())
6     ddname = null;
7   else
8     ddname = Pointer.pointerToCString(functionName);
9
10  String[] orderedVariableNames = variableStore.getOrderedNames();
11  BigcuddLibrary.Dddmp_cuddAddStore(dd,
12      ddname,
13      add.getUnderlyingNode(),
14      Pointer.pointerToCStrings(orderedVariableNames),
15      null,
16      BigcuddLibrary.DDDMP_MODE_TEXT,
17      BigcuddLibrary.Dddmp_VarInfoType.DDDMP_VARIDS,
18      Pointer.pointerToCString(fileName),
19      output);
20  CUtils.fclose(output);
21  }
```

At the end of this analysis, we have an ADD with reliability values representing the entire product line, which was calculated taking advantage previous analyzes. The ADD of existing fragments will be persisted before the completion of the ReAnaE execution, so that they can be used in future evolutions of the models. Listing 4.4 shows the code to perform the persistence of the ADD generated during the analysis. The core of the

method is the call to the CUDD library to dump the ADD (Lines 11-19). In addition to the flags requested by the function (Lines 16-17), the function receives the ADD information that is persisted: variable `ddname` (Line 8) contains the ADD name; variable `add` is the contents of the ADD; variable `orderVariableNames` (Line 10) is an array with the name of all ADD variables; and variable `fileName` is the name of the text that will be written. After executing the method, the ADD will be persisted in text mode and can be read in a new analysis.

Chapter 5

Evaluation

In this chapter, we describe an experiment conducted to assess the proposed evolution-aware product-line reliability analysis implemented by ReAnaE. First, Section 5.1 defines the experiment. Section 5.2 presents its planning. Section 5.3 then reports results and performs an analysis. Section 5.4 discusses the findings resulting from the results, and Section 5.5 discusses threats to validity. The replication package of the experiment is publicly available.[1]

5.1 Definition

We use the Goal Question Metric (GQM) method [39] to structure the experiment. Table 5.1 synthesizes the goal of our investigation.

Table 5.1: GQM goal

Purpose	Assess
Issue	Correctness and Performance
Object	Evolution-aware product-line reliability analysis
Viewpoint	Software engineer
Context	Evolving model-based software product lines

To achieve the goal, considering the state-of-the-art evidence on reliability analysis, we choose as a baseline the analysis implemented by ReAna [1]. We address performance in terms of time and space. The corresponding questions and metrics are as follows:

- **Question 1**: Does the proposed evolution-aware method *yield the same results* as the analysis implemented by ReAna ?

[1] https://dspl4c2.github.io/empirical

  - **Metric M1.1**: Raw difference between the reliability values for each configuration in both analyses.

- **Question 2:** Is the proposed evolution-aware technique *faster* than the analysis implemented by ReAna?
  - **Metric M2.1:** Execution time of the analysis.

- **Question 3**: Does the proposed evolution-aware technique require less space than the analysis implemented by ReAna?
  - **Metric M3.1:** Memory consumption of the analysis.

Correctness (Question 1) was empirically assessed by comparing the output of both approaches for a random subset of configurations of a given evolved version of a product line. Each valid configuration given as input yields a value in the closed Real interval [0,1], denoting the reliability of the corresponding product. Metric M1.1 is calculated by comparing the output of the evolution-aware feature-family reliability analysis and the output of its non-evolution-aware counterpart. A formal proof of correctness is outside the scope of this work and is regarded as future work.

To address Questions 2 and 3, we measured the time and space required by each approach. For the time measure (M2.1), we considered the wall-clock time spent during analysis, including the time to persist the ADD at the end. Thus, for the evolution-aware analysis, the time taken to read the data persisted in the previous evolution and the time spent to persist the data after the current analysis is included in the total analysis time. For the space measure (M3.1), we considered the peak memory usage during the analysis.

5.2 Planning

5.2.1 Hypothesis Formulation

The dependent variables of our experiment are analysis time and space consumption. The independent variable is the analysis method, one treatment being the proposed evolution-aware method implemented by ReAnaE and the other the method implemented by ReAna. To address our research questions, we define two hypotheses:

Null Hypothesis 1 (H_0): There is no difference between ReAnaE's and ReAna's reliability value.

Alternative Hypothesis 1 (H_1): ReAnaE yields a different reliability value than ReAna.

Null Hypothesis 2 (H_0): There is no difference between ReAnaE's and ReAna's analysis time.

Alternative Hypothesis 2 (H_1): ReAnaE's analysis time is different than ReAna's.

Null Hypothesis 3 (H_0): There is no difference between ReAnaE's and ReAna's memory consumption.

Alternative Hypothesis 3 (H_1): ReAnaE's memory consumption is different than ReAna's.

5.2.2 Subject System Selection

We analyze six different product lines and 20 evolved versions thereof according to four evolution scenarios (Chapter 3). We have selected these product lines because previous studies have analyzed them [27, 40, 1], their variability models were available, and they have distinct and comprehensive characteristics, both in behavior and in the number of existing features. In addition, we take advantage of the description of the feature and behavioral models available in the work by [1]. Table 5.2 shows the characteristics of each product line model in terms of the number of features, products, and behavioral fragments.

Table 5.2: Initial version of product lines used for empirical evaluation.

	# Features	# Products	# Behavioral fragments
EMail [41]	10	40	15
MinePump [42]	11	128	28
BSN [28]	16	298	19
Lift [43]	10	512	11
InterCloud [44]	54	110592	56
TankWar [41]	144	4.21×10^{18}	88

The TankWar and InterCloud product lines have a greater number of features and product configurations. These are two lines that require greater computational effort for analysis. The other product lines, although with a similar amount of existing features, have very different structures. The BSN product lines have mandatory, or, optional, and alternative features, as well as greater depth in the feature model structure. The Lift product line has only optional features, all of which are children of the root. The Email line, on the other hand, has only one mandatory feature and the other optional children

of the root. Finally, the MinePump product line has mandatory, optional, and alternative features.

For this work, the evolution of the software product line occurs in the models. We stress each evolution scenario to observe the behavior of the proposed analysis method. As our behavioral models are composed of features, fragments, messages and presence conditions, we chose each one of them to be evolved, in order to handle as much as possible real scenarios. For each of the original six subject product lines, we have applied different evolution scenarios (cf. Chapter 3):

1. In Evolution Scenario 1, each one of the 20 evolution steps adds an optional feature and a corresponding behavioral fragment. This way, each evolution step doubles the size of the product line's configuration space.

2. Evolution Scenario 2 adds messages to an existing behavioral fragment such that, at each evolution step, 10 messages with random success probabilities are added to a fragment belonging to a node below the root of the RDG. The source and destination lifelines are also chosen at random.

3. Evolution Scenario 3 adds a fragment with a guard condition equal to `True` to a pre-existing behavioral fragment at each evolution step. The added fragment contains 10 messages, each one with random success probabilities. The source and destination lifelines of the messages in this fragment were also chosen randomly.

4. Lastly, Evolution Scenario 4 evolves the given product line by changing the presence condition of a behavioral fragment. It changes the fragment's presence condition by either *strengthening* it (increasing conjunction of feature atoms) or *weakening* it (increasing disjunction of feature atoms).

5.2.3 Experiment Design and Analysis Procedures

We applied each treatment 10 times to each subject system (original version of the product line and 20 corresponding evolved versions) in all evolution scenarios. For each version, we analyzed the results using statistical tests to address any biased results or possible outliers. We applied standard tests for equality of the pairs of samples. If both samples were normally distributed with equal variance, we applied a t-test; if they had different variance, we applied Welch's t-test. Whenever one of the data sets, at least, was not normally distributed, we applied Mann-Whitney U test. The significance level for all tests was 0.01.

5.2.4 Instrumentation and Operation

To obtain the analysis time data, we used the `System.nanoTime()` method from the Java standard library, reporting the times before and after the execution of the treatments. This way, we are able to determine the total analysis time as the difference between the time at the end of the run and the time at the start of the run. Memory usage was collected by the CUDD 2.5.1 library[2] during ADD manipulation. This value represents the peak RAM used when performing the analysis. The `diff` command is used to verify the correctness of the executions, by comparing the reliability values of the analyzes.

In terms of preparation, we selected 21 behavioral models and corresponding feature models from each product line (representing the starting state and each of the 20 successive evolutions). These artifacts were used as input for each of the treatments, which were packaged into executable `.jar` files. Shell scripts carried out execution according to the experiment design. After each run, data regarding time and memory consumption were persisted on disk. Before starting the analysis of another product line, the machine was restarted to avoid any bias related to the use of swap memory. In cases where some kind of error occurred on the machine, causing a significant increase in execution time, we repeated the analysis for the 2 treatments, to ensure sufficient precision in the results. We ran the experiment on an Intel Core i7-8565U 1.80GHz machine with 16 GB of RAM and 2 GB of swap memory, running 64-bit Ubuntu 18.04.

5.3 Results and Analysis

In Section 5.3.1, we present the correctness assessment of the proposed analysis method. In Sections 5.3.2 to 5.3.5, we present the performance results of each of the proposed evolution scenarios. All analysis data are available in the Appendix A.

5.3.1 Correctness

In principle, Question 1 could be answered by comparing the reliability result of all possible feature configurations for all models. However, for Evolution Scenario 1, as the quantity of products in each product line grows exponentially, it is infeasible to compare all outputs generated by the two treatments. Thus, M1.1 metric was calculated in two steps. The first one was to compare corresponding leaf nodes of the ADDs resulting from the reliability analyses, and check whether the necessary condition that their reliability values matched, which they did. Second, we randomly selected a sample of 50 feature configurations and compared the reliability values. This was done for all models, and

[2]https://davidkebo.com/source/cudd_versions/cudd-2.5.1.tar.gz

the result was the same for the evolution-aware technique and its non-evolution-aware counterpart. The results generated and the script for reproducing the analysis can be found in the repository[3].

For Evolution Scenarios 2, 3 and 4, except for TankWar product line, it was possible to compare the reliability values for all possible generated products. The result of the comparison between the evolution-aware treatment and the non-evolution-aware counterpart was the same in all cases, confirming that the proposed method obtained indeed the same results as the original method. All analysis data are available in Appendix A (A.1 through A.6).

5.3.2 Evolution Scenario 1

Null Hypotheses 2 and 3 were rejected for most models. The statistical tests rejected Null Hypothesis 2 in 84.31% of the cases, and in 75.58% of these, the evolution-aware analysis was faster. Figure 5.1 shows the result of the time consumption to analyze the original and evolved models of Lift, Email, BSN, MinePump, InterCloud, and TankWar. The horizontal axis represents the evolution of each model, while the vertical axis represents the time consumption in seconds to perform the model analysis, on a logarithmic scale.

With respect to memory consumption, Null Hypothesis 3 was rejected in 97,05% of the tests, and in 86.86% of these, the evolution-aware analysis consumed less memory. Figure 5.2 shows the result of the memory consumption to analyze the same product lines. The horizontal axis represents the evolution of each model, while the vertical axis represents the memory consumption in megabytes to perform the model analysis, on the logarithmic scale.

In this evolution scenario, analysis time and memory consumption of both treatments exhibited exponential growth on the evolved models, which is related to doubling the configuration space at each evolution step. Still, the evolution-aware treatment was able to analyze more evolution steps in the Email, InterCloud, MinePump, and TankWar than the original treatment. In addition, there was a substantial reduction in analysis time in models with a greater number of features, which are the InterCloud (54 features) and TankWar (144 features) product lines.

Findings for Evolution Scenario 1: The evolution-aware technique was more efficient in the analysis time in 63.72% of the analysis runs and in the memory consumption in 84.30% of the analysis runs.

[3]htttp://reanaEresults

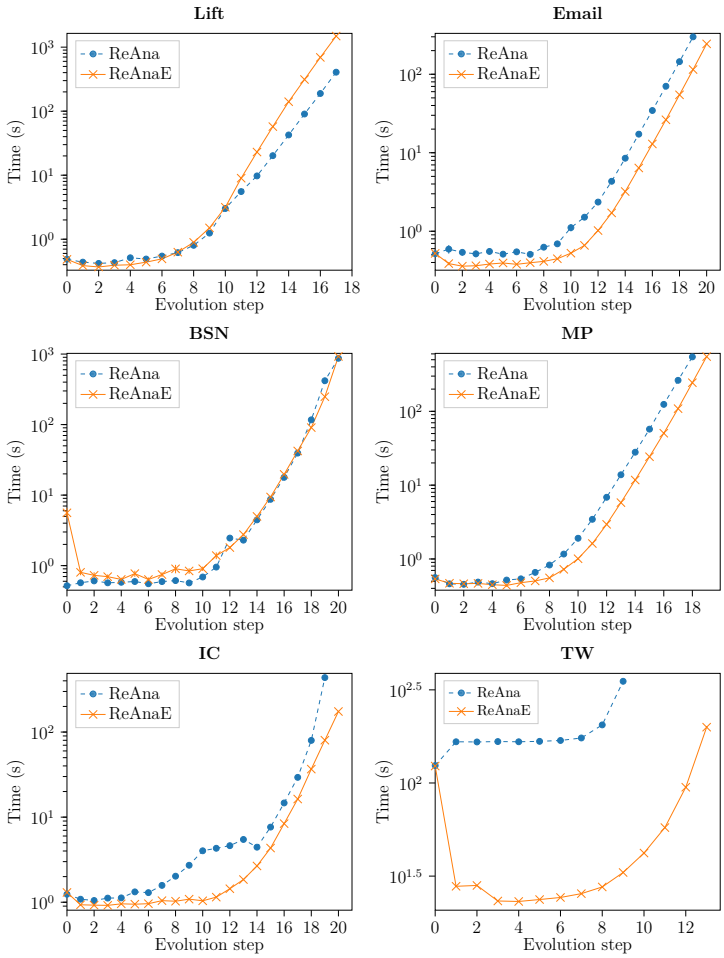

Figure 5.1: Analysis time for *Evolution Scenario 1*.

5.3.3 Evolution Scenario 2

Null Hypotheses 2 and 3 were rejected for most models. The statistical tests rejected Null Hypothesis 2 in 98.33% of the cases, and in 100% of these, the evolution-aware analysis was faster than its counterpart. Figure 5.3 shows the result of the time consumption to analyze the subject systems. The horizontal axis represents the evolution of each model, whereas the vertical axis represents the time consumption in seconds to perform the model analysis, on a linear scale.

Null Hypothesis 3 was rejected in 100% of the cases, and in 100% of these the evolution-aware treatment consumed less memory in the analysis than its counterpart. Figure 5.4 shows the result of the memory consumption to analyze the subject systems. The horizontal axis represents the evolution of each model, whereas the vertical axis represents the memory consumption in megabytes to perform the model analysis, on the linear scale.

In contrast to Evolution Scenario 1, Evolution Scenario 2 maintained a linear growth in the analysis time for the smaller product lines and an almost constant analysis time for

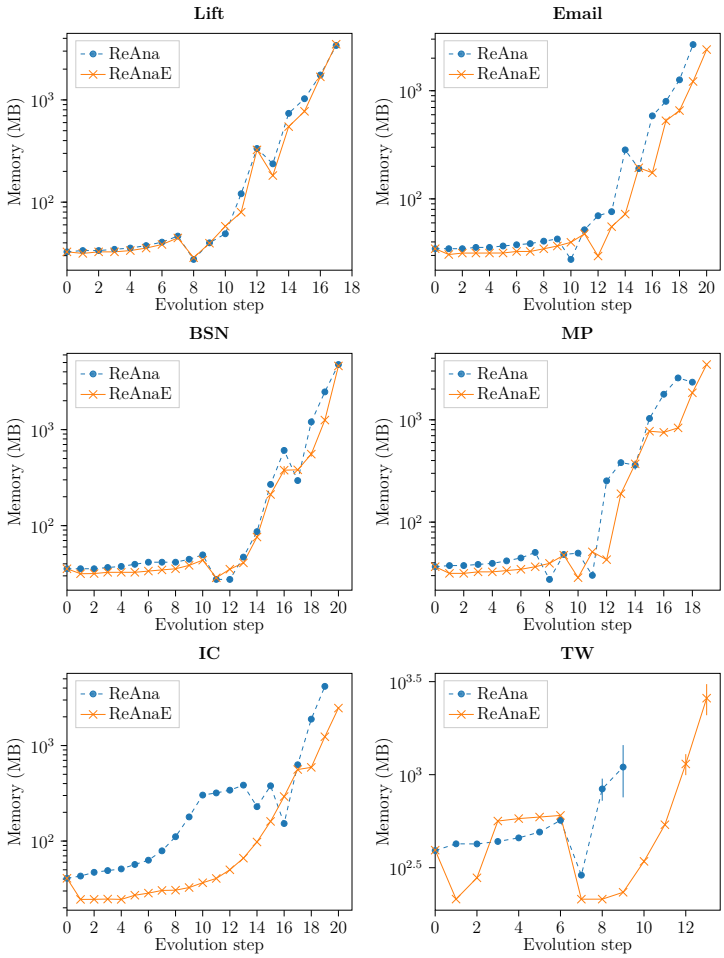

Figure 5.2: Memory consumption for *Evolution Scenario 1*.

the TankWar product line. In most models, the analysis time was below 1 second, except for TankWar, whose analysis can take over a minute. As this scenario does not increase the number of features or the number of products, the most pronounced results appears in product lines that were originally large.

> *Findings for Evolution Scenario 2:* The evolution-aware technique was more efficient with respect to time and memory consumption in 98.33% and 100% of the analyses, respectively.

5.3.4 Evolution Scenario 3

Null Hypotheses 2 and 3 were rejected in most models. The statistical tests rejected Null Hypothesis 2 in 90% of the cases, and in 100% of these, the evolution-aware analysis was faster than its counterpart (cf. Figure 5.5). Null Hypothesis 3 was rejected in 99.16% of

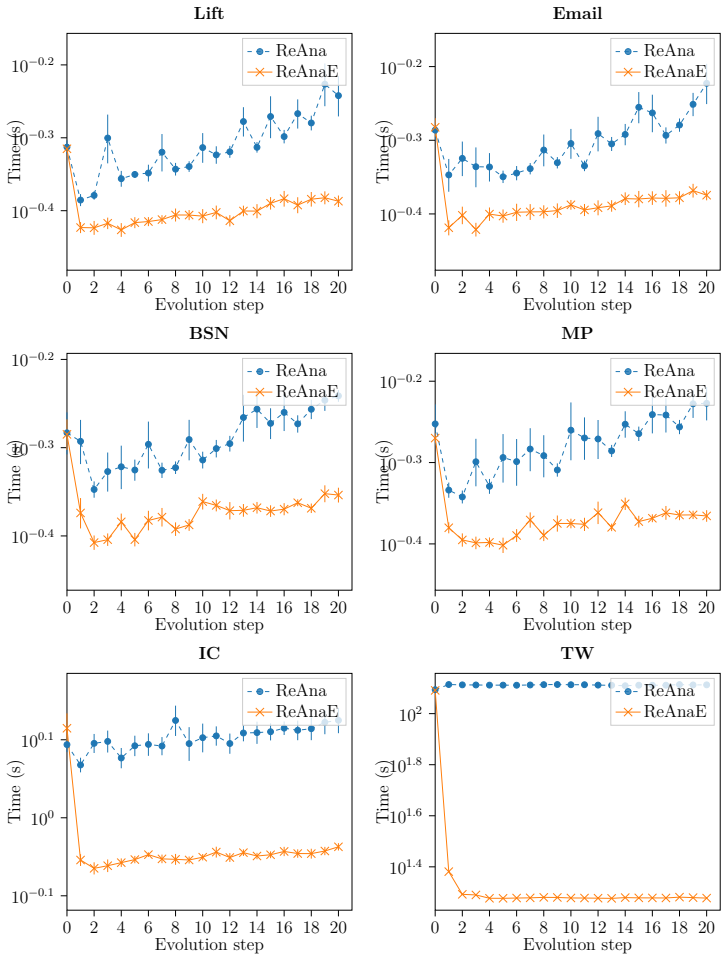

Figure 5.3: Analysis time for *Evolution Scenario 2*.

the cases, and in 100% of these the evolution-aware consumed less memory in the analysis than its counterpart (cf. Figure 5.6).

This evolution scenario also shows a linear growth in the analysis time for the smaller product lines and an almost constant analysis time for the TankWar product line. As in Evolution Scenario 2, in most models, the analysis time was below 1 second, taking minutes only in TankWar. As this scenario also does not increase the number of features or the number of products, the most pronounced results appears in product lines that were originally large.

Findings for Evolution Scenario 3: The evolution-aware technique was more efficient with respect to time and memory consumption in 90% and 99.16% of the analyzes, respectively.

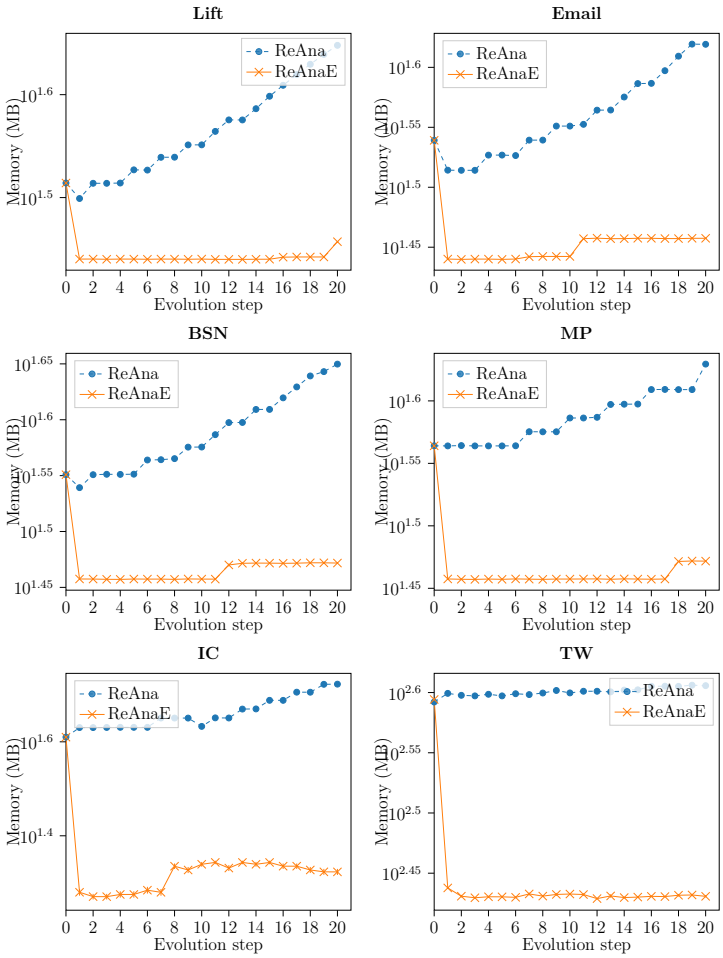

Figure 5.4: Memory consumption for *Evolution Scenario 2*.

5.3.5 Evolution Scenario 4

After running the analyses on both scenarios, Null Hypotheses 2 and 3 were rejected in most models, which we explain next.

Strengthening scenario: The statistical tests rejected Null Hypothesis 2 in 67.94% of the cases, and in 100% of these, the evolution-aware analysis was faster than its counterpart (cf. Figure 5.7). Null Hypothesis 3 was rejected in 100% of the cases, and in 100% of these the evolution-aware consumed less memory in the analysis than its counterpart (cf. Figure 5.8).

Weakening scenario: The statistical tests rejected the Null Hypothesis 2 in 64.10% of the cases, and in 100% of these, the evolution-aware analysis was faster than its counterpart (cf. Figure 5.9). Null Hypothesis 3 was rejected in 100% of the cases, and in 100% of these the evolution-aware consumed less memory in the analysis than its counterpart (cf. Figure 5.10).

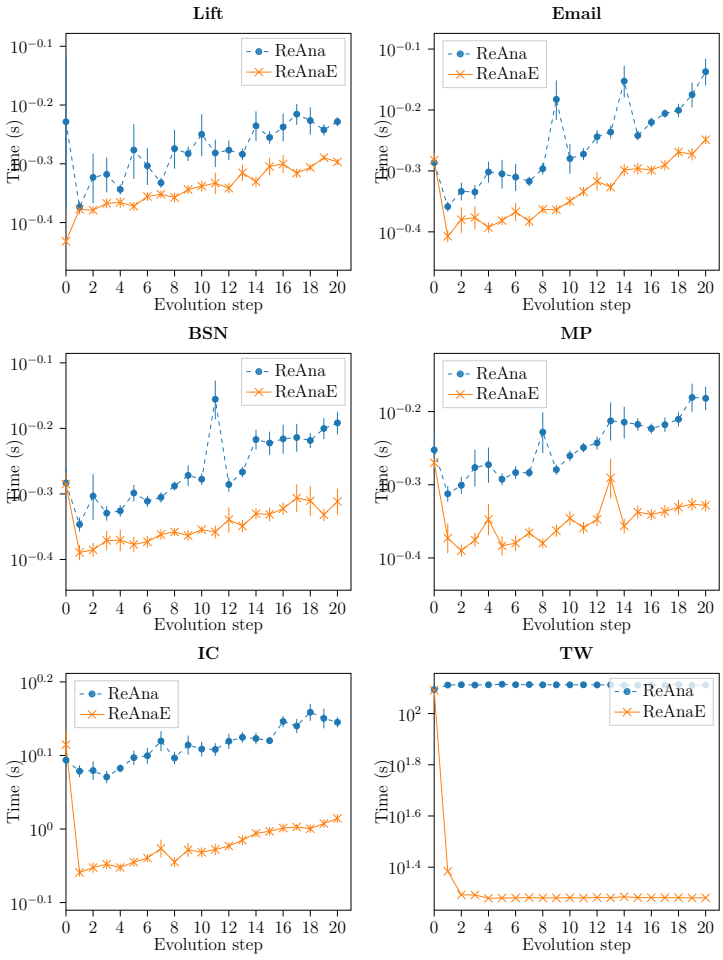

Figure 5.5: Analysis time for *Evolution Scenario 3*.

Findings for Evolution Scenario 4: The evolution-aware technique was more efficient with respect to time in 67.94% of the strengthening scenarios analyzed and in 64.10% of the weakening scenarios analyzed. Regarding memory consumption, the evolution-aware technique was more efficient in 100% of the cases.

5.4 Discussion

Our empirical results show that, in most analysis scenarios, the evolution-aware method was more efficient than the original method, both in terms of execution time and memory consumption. To better understand the results, we further instrumented the source code to highlight additional timing information, such as disk access and time spent in the feature-based and family-based steps. We found that the computational gain of ReAnaE arises from both the feature-based and family-based steps. In the product lines with larger

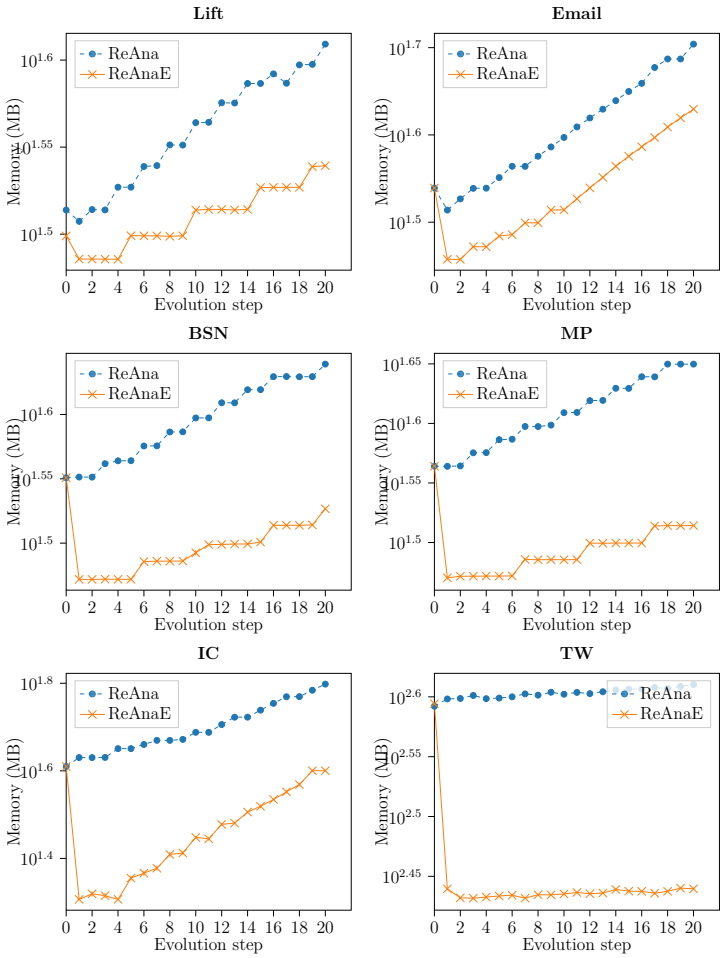

Figure 5.6: Memory consumption for *Evolution Scenario 3*.

configuration spaces, most of the analysis time savings occurred in the family-based step. In the product lines with smaller configuration spaces, most of the analysis time savings occurred in the feature-based step. In situations where ReAna outperformed ReAnaE, the difference can be explained by the additional time needed to store and retrieve ADDs, considering that the time in the feature-based and family-based steps were similar.

Table 5.3 provides a comparison of the analysis time spent on the latest evolved model for each product line and all evolution scenarios. This way, it is possible to understand the cause of the difference in the time spent in the feature-based and family-based steps. For each model, the table shows (a) the total number of nodes in the RDG, (b) the number of nodes impacted by the evolution step, (c) the time spent in the feature-based step in both treatments, and (d) the time spent in the family-based step in both treatments.

In Evolution Scenario 1, where an optional feature is added in each evolution step, the product line configuration space doubles with each evolution, which makes ADD operations progressively slower. Consequently, the family-based step dominates analysis time, as it is apparent in the last two columns of Table 5.3. Since evolution-aware anal-

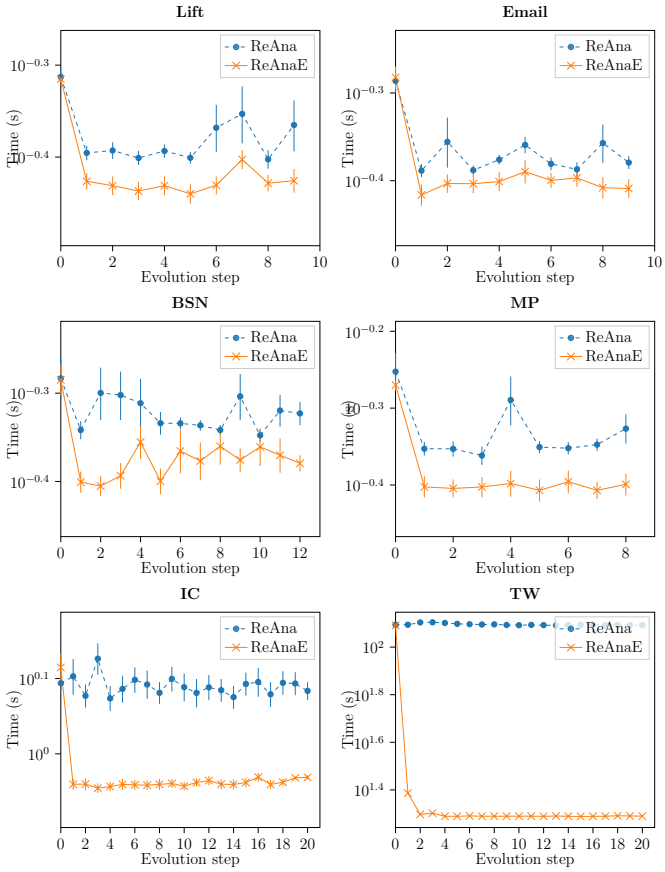

Figure 5.7: Analysis time for *Evolution Scenario 4* with strengthening.

ysis performs the family-based step only on impacted nodes, the overall analysis time decreases.

For the remaining evolution scenarios, no features are added. In these scenarios, the biggest gains in analysis time were observed in the feature-based step. As the product lines originally had a small configuration space, except for TankWar, the time spent in the family-based step to calculate the ADD was in the order of milliseconds, and so forth for the successive evolutions. In this case, the evolution-aware treatment reduced most of the analysis time in the feature-based step, by avoiding generating the parametric reliability expression for all nodes, taking advantage of previously performed calculations. This represents the biggest savings in analysis time, as we can see by comparing columns *ReAna's feature-based step* and *ReAnaE's feature-based step* of Table 5.3. In the case of TankWar, the saving was greater in the family-based step due to its larger configuration space.

In the majority of cases, ReAnaE achieved significantly better results, both in terms of analysis time and memory consumption, which shows the potential of the evolution-aware method. The extent of the gains of using the method depends on the product line: with larger configuration spaces, it is possible to obtain greater benefits, as we have

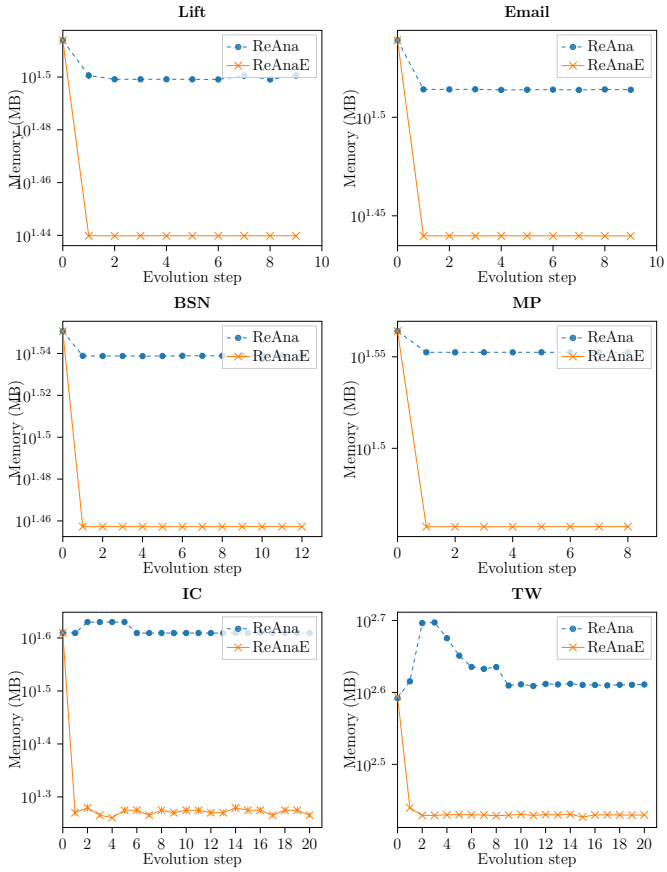

Figure 5.8: Memory consumption for *Evolution Scenario 4* with strengthening.

observed for InterCloud and TankWar.

5.5 Threats to Validity

A possible threat to internal validity would be the fact that we do not account for the time spent in some relevant tasks of the analysis method, such as reading the previous results and rebuilding the RDG structure, which could lead to an underestimated time penalty. To mitigate this threat, both treatments were instrumented such that the timer starts right at the beginning of the analysis and stops just after the end of persistence of the ADDs generated during the analysis. This instrumentation captures the characteristics of each treatment during the whole analysis method.

The way in which the product lines evolved might threaten internal validity, since all evolution steps follow the same pattern of evolving an RDG node just below the root node. We followed this pattern to focus on the evolution effects to the analysis method and to isolate the results from interference of other factors such as RDG node depth level and the reuse of results by more than one RDG node (reuse sharing). However, since the

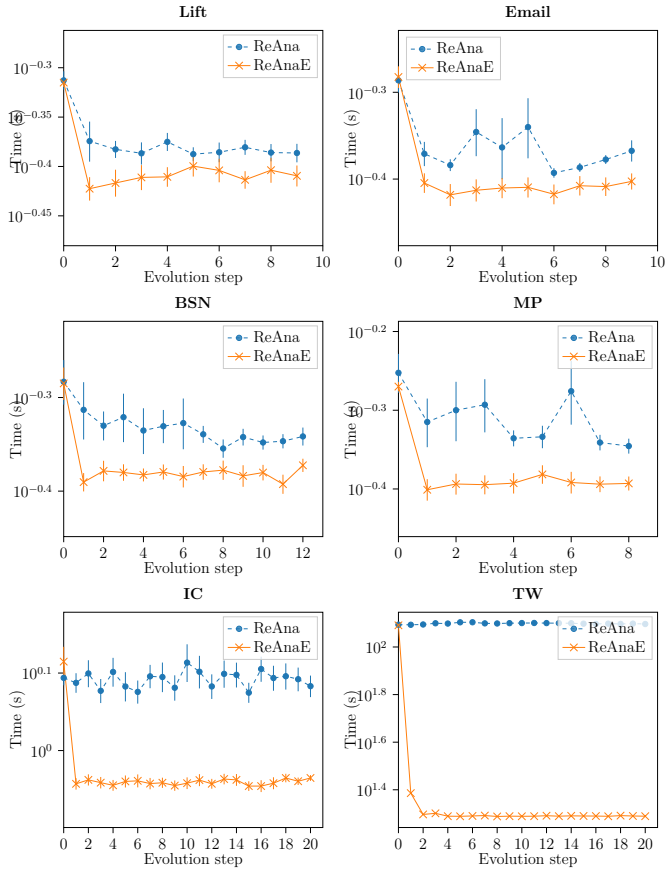

Figure 5.9: Analysis time for *Evolution Scenario 4* with weakening.

evolution scenarios were not evaluated at different positions of the RDG structure, the results may not reflect the full picture of the analysis method.

As a corollary of the previous threat, our implementation might be overfitting to the particular evolution scenarios considered, since we knew beforehand which are their changes at the RDG structure and their impacts to the analysis. There might be other evolution scenarios that impact the RDG structure and the analysis method other than the scenarios considered in this evaluation. Thus, as the evolution scenarios set is not complete, we can not assert the current implementation of the analysis method will perform similarly to other evolution types, but the ones that we analyzed are nevertheless scenarios that are relevant in practice.

A threat to internal validity would be an incidental and undesired treatment interaction between different analysis runs due to high cache memory usage required for the analysis method. To mitigate this threat, the system was rebooted by the end of the evaluation of a product line and its evolution steps. Thereby, we seek to ensure all resources were accordingly released and all analyses start with the same use of system resources.

Finally, a threat to external validity arises from the selection of the subject systems. To mitigate this threat, we selected systems commonly used by the community as benchmarks

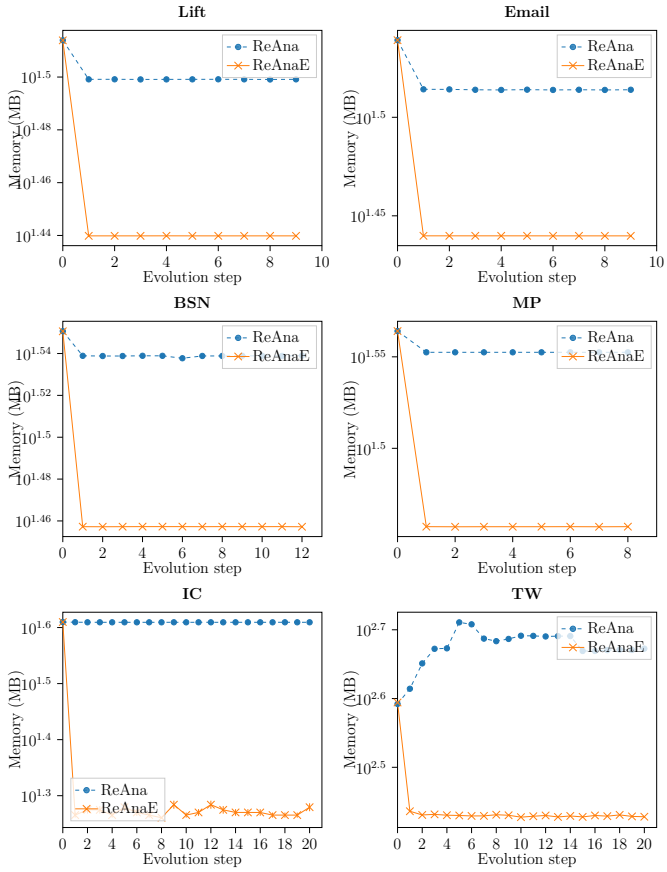

Figure 5.10: Memory consumption for *Evolution Scenario 4* with weakening.

to evaluate variability-aware model checking techniques [42, 43]. To mitigate the risk of our approach not being generalizable, we applied it to further product lines (InterCloud and TankWar), whose configuration spaces resemble the ones of real-world applications.

Table 5.3: Feature-based step (FeBS) and family-based step (FaBS) analysis time comparison

SPL	Model	Total nodes	Impacted nodes	ReAna's FeBS (ms)	ReAnaE's FeBS (ms)	ReAna's FaBS (ms)	ReAnaE's FaBS (ms)
Evol. Scenario 1							
Lift	17	27	3	124	26	371463	380891
Email	19	33	3	164	21	283492	98446
BSN	20	36	3	124	21	855445	885695
MinePump	18	41	3	177	20	560607	213660
InterCloud	19	70	3	215	18	354910	50506
TankWar	9	87	3	272	20	169580	6869
Evol. Scenario 2							
Lift	20	11	2	129	10	4	1
Email	20	14	2	138	11	8	3
BSN	20	16	2	150	11	8	6
MinePump	20	24	2	161	15	13	4
InterCloud	20	52	2	522	12	27	3
TankWar	20	79	2	273	9	1758	314
Evol. Scenario 3							
Lift	20	31	3	149	32	12	7
Email	20	34	3	234	120	14	3
BSN	20	36	3	181	29	16	8
MinePump	20	44	3	184	26	18	6
InterCloud	20	72	3	329	108	41	4
TankWar	20	99	3	305	18	1734	327
Evol. Scenario 4 - Strengthening							
Lift	9	11	1	56	9	7	2
Email	9	14	1	69	11	8	2
BSN	12	16	1	85	11	12	6
MinePump	8	23	1	101	13	19	4
InterCloud	20	52	1	180	10	30	4
TankWar	20	79	1	225	9	1867	322
Evol. Scenario 4 - Weakening							
Lift	9	11	1	56	10	7	1
Email	9	14	1	100	10	9	3
BSN	12	16	1	75	10	18	6
MinePump	8	23	1	96	14	15	4
InterCloud	20	52	1	163	11	28	2
TankWar	20	79	1	226	9	2683	314

Chapter 6

Conclusion

We proposed an evolution-aware software product line analysis method that is capable of performing reliability analysis by taking advantage of intermediate calculations from previous analysis runs. In particular, through evolution primitives, which define possible evolution scenarios for the product line, the method identifies which parts were impacted, this way avoiding unnecessary calculations during the analysis.

To evaluate this method, we have developed an extension of the ReAna tool, called ReAnaE, for feature-family-based reliability analysis of product lines. At the end of an analysis run, ReAnaE persists the data from the intermediate calculations. In a subsequent evolution-step, we identify which calculations will not be needed and retrieve the information that was persisted to use during the analysis. In this way, our method is able to work with variation in time in product line analysis.

For the empirical evaluation, we compared the original feature-family-based analysis method (ReAna) and our new evolution-aware extension (ReAnaE). For this purpose, we analyzed six different product lines using an experiment setup that implements both analysis methods, comparing running time and memory consumption. In most of the evaluation scenarios, the evolution-aware method was able to perform faster and less memory-consuming analyses, even considering the extra time required to read and persist the data. ReAnaE was able to analyze a larger number of evolution-steps in certain product lines, for which the complete analysis effort otherwise exceeded the available resources.

Overall, we believe our method and analysis tool are a step towards improved efficiency of the feature-family-based reliability analysis.

6.1 Limitations

Although our method has achieved expressive results in most of the empirical analyzes performed, there are some limitations.

Types of evolution primitives. Our evolution scenarios were based on 4 different types of evolution primitives: add feature; add message; add fragment; and change presence condition. However, some other evolution primitives can be defined to make it closer to the real scenario, such as deleting features, deleting messages, deleting fragments, change order of messages, change components names and so on.

Pre-defined evolution primitives. Each evolution scenarios was pre-defined before the analysis and it was known to ReAnaE the part of the behavioral model that was modified in evolution. Thus, the tool cannot automatically identify which part of the model has gone through any of the evolution primitives.

Transformation steps. Our method reuses previous analyzes for the feature-family-based reliability analysis. However, some preparatory steps for performing the analysis are required by the tool, such as the transformation step for creating the RDG. In this phase, there is no optimization to reuse the analysis done previously.

Evolution scenarios. Product line models maintained a evolution pattern by adding features, messages, or fragments directly to the root. The evolution primitives applied at other levels of depth in the RDG bring different impacts on the analysis.

6.2 Related Work

Family-based analyses are inherently monolithic and expensive to perform when evolution is considered. This work tackles the challenge of combining variability in time and space by approaching it from the *feature-family-based analyses* perspective. In what follows, we discuss related work that considers the evolution dimension in product line analysis and beyond.

Thüm et al. [17] discussed the existing techniques and challenges regarding analysis with variation in time and space. They defined variation in space as being the variants that must coexist and variation in time as the fact that there are several revisions of software that are being replaced. The authors discussed the existing challenges of approaches that deal with variation in space to be able to analyze variation in time, and vice versa. Our proposed method addresses this problem by using an analysis that deals with variation in space, in the case feature-family-based, to deal with variation in time, identifying change impacts on evolved models and only performing analysis in the modified parts, updating the analysis result where necessary.

Our work is based on the ReAna tool, a model-based approach for reliability analysis of software product lines developed by Lanna et al. [1]. ReAna performs a feature-family-based analysis relying on a graph structure to represent the behavioral models. A parametric analysis is performed on each node of the model to later lift the expressions for an ADD structure, on which the tool is able to determine the reliability for the entire product line. Our work builds on this strategy, making it capable of performing an evolution-aware analysis of the product line, reducing the computational effort of performing an analysis from scratch with each evolution of the model.

Ghezzi and Sharifloo [16] also proposed a model-based approach to product line analysis for non-functional properties. To avoid product-by-product analysis, they proposed a compositional technique that reuses the intermediate results to calculate the overall properties. The behavior of the product line is organized through a tree, in which the nodes are related to expressions resulting from the analysis performed by a parametric model checker. To determine the reliability of the product as a whole, a bottom-up analysis of the tree is performed, where the root result represents the product reliability. The strategy can be considered feature-product-based, since it divides the behavioral models into small units to later evaluate them and obtain the reliability of a product. This approach differs from our proposal, since it does not perform a feature-family-based analysis and, more importantly, it is not able to handle variability in time as the model evolves.

Lity et al. introduced higher-order delta modeling [20, 21], which aims at combining concepts from variability modeling with evolution concepts. The key idea is to provide an integrated modeling approach that includes variability and evolution. As a consequence, the evolution history is captured by the modeling formalism and can be leveraged to provide change impact analysis [20]. The work acts in a compositional way, and establishes possible evolution operators, similar to what we present in our evaluation. Lity et al. extend this work by proposing the 175% model formalism [21]. In the context of annotative product lines, 150% models embed the entire variability space into a single model to leverage single system analysis for the context of variable systems. The 175% model embeds evolution over time into such models. A bidirectional transformation between higher-order delta models and 175% models is also presented. The formalism was not extended to the context of reliability analysis, so a proper comparison remains as future work.

Schröter et al. [45] proposed the notion of feature model interfaces. The main concept is that complex feature models can be organized as a set of smaller feature models that expose interfaces, thus allowing to perform the analysis on these parts separately and recompose complete analysis results from these parts. As a consequence, if a change is local, it is possible to simply re-analyze this subpart and re-compose the analysis results.

However, different from our work, this concept only considers analysis at the feature model level. They prove compositionality properties and also perform an empirical study using an industrial feature model. The study illustrates the potential for compositional analysis, since in more than half of the considered evolution scenarios, the feature model interface is sufficient to reduce the need for computing analyses over the entire feature model.

Angerer et al. [46] propose a configuration-aware change impact analysis approach. The approach embeds variability information into system dependence graphs, to enable identifying which products might be impacted by a change. The results show that computing the change impact is not expensive, and that it is important to have such an approach especially in complex systems. For instance, the results show that, on average, 5 configuration options (features) are affected by a change. In extreme cases, there might be over 30 configuration options to reason, which could be difficult to reason about, without tool support. This approach differs from our work as it seeks to determine the impact of an evolution through the use of standard control flow and data flow analysis in the source code. Furthermore, the search for the impact generated on evolution would be interested in the associated maintenance tasks and not in non-functional requirements analysis such as reliability.

In the context of evolution-aware analysis that does not tackle variability, Artz and Bodden [19] proposed REVISER, an approach to update the data flow inter-procedural analysis in incremental programs changes. It traces the changes made to the program by comparing the control-flow graph of two versions. Then, it identifies the modified locations and propagates the change by updating the analysis results. This way, it is able to reduce computational effort, considering that the data update does not need to be done on the whole program, but only on the modified parts. REVISER, however, does not consider variability, which is inherent in product lines. Our proposed method is similar with respect to Reviser's concept of propagating analysis updates only in parts affected by the evolution. Furthermore, it is capable of handling the variability present in the models.

6.3 Future Work

Our method and the developed tool have some limitations. In future work, some improvements could be made:

Automatic evolution identification. To perform the evolution-aware analysis, the label of the included fragment is standardized in the evolved models, and thus the ReAnaE can identify which node was impacted. We intend to provide a method of com-

parison between the evolved model and previous models, so that the evolution primitives can be identified automatically, regardless of labels, or previous knowledge by the tool, or amounts of evolution primitives.

Method generalization. At the moment, the proposed method works with an evolution-aware analysis for the feature-family-based strategy. However, the method of reuse of intermediate calculations during the analysis also applies to other strategies, given that the re-analysis of unmodified parts of a software is a common problem. In this way, we aim to generalize the proposed method to cope with different analysis strategies [18].

Formal proof. We also plan to formalize the method, proving its soundness using a proof assistant. We would like to prove the following theorem, based on Equation 4.2: for every compositional model m and evolution function δ, we have that $\phi'(m, \delta) = \phi(m^{\delta})$. In other words, the evolution-aware analysis and the original analysis yield the same result when evolving compositional model m with evolution function δ.

References

[1] Lanna, André, Thiago Castro, Vander Alves, Genaina Rodrigues, Pierre Yves Schobbens, and Sven Apel: *Feature-family-based reliability analysis of software product lines.* Information and Software Technology, 94:59–81, 2017, ISSN 0950-5849. x, 1, 2, 6, 7, 16, 17, 24, 29, 31, 48

[2] Clements, Paul and Linda Northrop: *Software Product Lines: Practices and Patterns.* Addison-Wesley Professional, 2001. 1

[3] Kang, K. C., S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson: *Feature-oriented domain analysis (FODA) feasibility study.* Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990. 1, 4

[4] Apel, Sven, Don Batory, Christian Kstner, and Gunter Saake: *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer Publishing Company, Incorporated, 2013, ISBN 3642375200. 1, 4

[5] Weiss, David M.: *The product line hall of fame.* In *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings*, page 395. IEEE Computer Society, 2008. `https://doi.org/10.1109/SPLC.2008.56`. 1

[6] Li, Harry C., Shriram Krishnamurthi, and Kathi Fisler: *Modular verification of open features using three-valued model checking.* Autom. Softw. Eng., 12(3):349–382, 2005. `https://doi.org/10.1007/s10515-005-2643-9`. 1

[7] Bastos, Jonatas Ferreira, Paulo Anselmo da Mota Silveira Neto, Pádraig O'Leary, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira: *Software product lines adoption in small organizations.* Journal of Systems and Software, 131:112 – 128, 2017, ISSN 0164-1212. `http://www.sciencedirect.com/science/article/pii/S0164121217300997`. 1

[8] Berger, Thorsten, Ralf Rublack, Divya Nair, Jo Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski: *A survey of variability modeling in industrial practice.* pages 7–14, January 2013. 1

[9] Ignaim, Karam and João M. Fernandes: *An industrial case study for adopting software product lines in automotive industry: An evolution-based approach for software product lines (evoa-spl).* In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B*, SPLC '19, page 183–190, New

York, NY, USA, 2019. Association for Computing Machinery, ISBN 9781450366687. https://doi.org/10.1145/3307630.3342409. 1

[10] Linden, Frank J. van der, Klaus Schmid, and Eelco Rommes: *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.* Springer-Verlag, Berlin, Heidelberg, 2007, ISBN 3540714367. 1

[11] Villela, Karina, Adeline Silva, Tassio Vale, and Eduardo Santana de Almeida: *A survey on software variability management approaches.* In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, page 147–156, New York, NY, USA, 2014. Association for Computing Machinery, ISBN 9781450327404. https://doi.org/10.1145/2648511.2648527. 1

[12] Rhein, Alexander Von, JöRG Liebig, Andreas Janker, Christian Kästner, and Sven Apel: *Variability-aware static analysis at scale: An empirical study.* ACM Trans. Softw. Eng. Methodol., 27(4), November 2018, ISSN 1049-331X. https://doi.org/10.1145/3280986. 1

[13] Heradio, Ruben, Hector Perez-Morago, David Fernández-Amorós, Francisco Javier Cabrerizo, and Enrique Herrera-Viedma: *A bibliometric analysis of 20 years of research on software product lines.* Inf. Softw. Technol., 72:1–15, 2016. https://doi.org/10.1016/j.infsof.2015.11.004. 1

[14] El-Sharkawy, Sascha, Nozomi Yamagishi-Eichler, and Klaus Schmid: *Metrics for analyzing variability and its implementation in software product lines: A systematic literature review.* Information and Software Technology, 106, August 2018. 1

[15] Åkesson, Jonas, Sebastian Nilsson, Jacob Krüger, and Thorsten Berger: *Migrating the android apo-games into an annotation-based software product line.* In Berger, Thorsten, Philippe Collet, Laurence Duchien, Thomas Fogdal, Patrick Heymans, Timo Kehrer, Jabier Martinez, Raúl Mazo, Leticia Montalvillo, Camille Salinesi, Xhevahire Tërnava, Thomas Thüm, and Tewfik Ziadi (editors): *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*, pages 19:1–19:5. ACM, 2019. https://doi.org/10.1145/3336294.3342362. 1

[16] Ghezzi, Carlo and Amir Molzam Sharifloo: *Model-based verification of quantitative non-functional properties for software product lines.* Inf. Softw. Technol., 55(3):508–524, 2013. https://doi.org/10.1016/j.infsof.2012.07.017. 1, 5, 48

[17] Thüm, Thomas, Leopoldo Teixeira, Klaus Schmid, Eric Walkingshaw, Mukelabai Mukelabai, Mahsa Varshosaz, Goetz Botterweck, Ina Schaefer, and Timo Kehrer: *Towards efficient analysis of variation in time and space.* In Cetina, Carlos, Oscar Díaz, Laurence Duchien, Marianne Huchard, Rick Rabiser, Camille Salinesi, Christoph Seidl, Xhevahire Tërnava, Leopoldo Teixeira, Thomas Thüm, and Tewfik Ziadi (editors): *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume B, Paris, France, September 9-13, 2019*, pages 69:1–69:8. ACM, 2019. https://doi.org/10.1145/3307630.3342414. 1, 2, 47

[18] Thüm, Thomas, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake: *A classification and survey of analysis strategies for software product lines.* ACM Comput. Surv., 47(1):6:1–6:45, 2014. https://doi.org/10.1145/2580950. 1, 5, 50

[19] Arzt, Steven and Eric Bodden: *Reviser: efficiently updating ide-/ifds-based data-flow analyses in response to incremental program changes.* In Jalote, Pankaj, Lionel C. Briand, and André van der Hoek (editors): *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 288–298. ACM, 2014. https://doi.org/10.1145/2568225.2568243. 2, 49

[20] Lity, Sascha, Matthias Kowal, and Ina Schaefer: *Higher-Order Delta Modeling for Software Product Line Evolution.* In *Proceedings of the 7th International Workshop on Feature-Oriented Software Development*, FOSD 2016, pages 39–48, New York, NY, USA, 2016. ACM, ISBN 978-1-4503-4647-4. http://doi.acm.org/10.1145/3001867.3001872. 2, 48

[21] Lity, Sascha, Sophia Nahrendorf, Thomas Thüm, Christoph Seidl, and Ina Schaefer: *175% Modeling for Product-Line Evolution of Domain Artifacts.* pages 27–34, 2018. 2, 48

[22] Borba, Paulo, Leopoldo Teixeira, and Rohit Gheyi: *A theory of software product line refinement.* Theoretical Computer Science, 455:2–30, October 2012, ISSN 03043975. 4

[23] Avizienis, A., J. C. Laprie, B. Randell, and C. Landwehr: *Basic concepts and taxonomy of dependable and secure computing.* IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, 2004. 5

[24] Grunske, Lars: *Specification patterns for probabilistic quality properties.* pages 31–40, January 2008. 5

[25] Cheung, Roger C.: *A user-oriented software reliability model.* IEEE Trans. Software Eng., 6:118–125, 1980. 5

[26] Walkingshaw, Eric, Christian Kästner, Martin Erwig, Sven Apel, and Eric Bodden: *Variational data structures: Exploring tradeoffs in computing with variability.* In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, Onward! 2014, pages 213–226, 2014. 5

[27] Rodrigues, Genaína Nunes, Vander Alves, Vinicius Nunes, André Lanna, Maxime Cordy, Pierre-Yves Schobbens, Amir Molzam Sharifloo, and Axel Legay: *Modeling and verification for probabilistic properties in software product lines.* In *16th IEEE International Symposium on High Assurance Systems Engineering, HASE 2015, Daytona Beach, FL, USA, January 8-10, 2015*, pages 173–180. IEEE Computer Society, 2015. https://doi.org/10.1109/HASE.2015.34. 5, 7, 31

[28] Pessoa, Leonardo, Paula Fernandes, Thiago M. Castro, Vander Alves, Genaína Nunes Rodrigues, and Hervaldo Carvalho: *Building reliable and maintainable dynamic software product lines: An investigation in the body sensor network domain.* Information

and Software Technology, 86:54–70, 2017. https://doi.org/10.1016/j.infsof.2017.02.002. 6, 31

[29] Czarnecki, Krzysztof and Krzysztof Pietroszek: *Verifying feature-based model templates against well-formedness ocl constraints.* In *Proceedings of the 5th International Conference on Generative Programming and Component Engineering*, GPCE, 2006. 6

[30] Kwiatkowska, Marta, Gethin Norman, and David Parker: *Prism 4.0: Verification of probabilistic real-time systems.* Volume 6806, pages 585–591, July 2011, ISBN 978-3-642-22109-5. 7

[31] Hahn, Ernst Moritz, Holger Hermanns, Björn Wachter, and Lijun Zhang: *Param: A model checker for parametric markov models.* In Touili, Tayssir, Byron Cook, and Paul Jackson (editors): *Computer Aided Verification*, pages 660–664, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg, ISBN 978-3-642-14295-6. 7

[32] Bahar, R.Iris, Erica Frohm, Charles Gaona, Gary Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi: *Algebraic decision diagrams and their application.* Formal Metthods in System Design, 10:171–206, April 1997. 7

[33] Kröher, Christian, Lea Gerling, and Klaus Schmid: *Identifying the Intensity of Variability Changes in Software Product Line Evolution.* In *Proceedings of the 22Nd International Systems and Software Product Line Conference - Volume 1*, SPLC '18, pages 54–64, New York, NY, USA, 2018. ACM, ISBN 978-1-4503-6464-5. http://doi.acm.org/10.1145/3233027.3233032. 9

[34] Marques, Maíra, Jocelyn Simmonds, Pedro O. Rossel, and María Cecilia Bastarrica: *Software product line evolution: A systematic literature review.* Information and Software Technology, 105:190 – 208, 2019, ISSN 0950-5849. http://www.sciencedirect.com/science/article/pii/S0950584918301848. 9

[35] Quinton, Clément, Rick Rabiser, Michael Vierhauser, Paul Grünbacher, and Luciano Baresi: *Evolution in dynamic software product lines: challenges and perspectives.* In Schmidt, Douglas C. (editor): *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, pages 126–130. ACM, 2015. https://doi.org/10.1145/2791060.2791101. 9

[36] Passos, Leonardo Teixeira, Leopoldo Teixeira, Nicolas Dintzner, Sven Apel, Andrzej Wasowski, Krzysztof Czarnecki, Paulo Borba, and Jianmei Guo: *Coevolution of variability models and related software artifacts - A fresh look at evolution patterns in the Linux kernel.* Empirical Softw. Engg., 21(4):1744–1793, 2016. 9

[37] Gomes, Karine, Leopoldo Teixeira, Thayonara Alves, Márcio Ribeiro, and Rohit Gheyi: *Characterizing safe and partially safe evolution scenarios in product lines: An empirical study.* In *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems*, VaMoS, 2019. 9

[38] Thüm, Thomas, Christian Kästner, Fabian Benduhn, Jens Meinicke, G. Saake, and Thomas Leich: *Featureide: An extensible framework for feature-oriented software development*. Sci. Comput. Program., 79:70–85, 2014. 19

[39] Basili, Victor R., Gianluigi Caldiera, and H. Dieter Rombach: *The goal question metric approach*. In *Encyclopedia of Software Engineering*. Wiley, 1994. 29

[40] Classen, A., M. Cordy, P. Schobbens, P. Heymans, A. Legay, and J. Raskin: *Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking*. IEEE Transactions on Software Engineering, 39(8):1069–1089, 2013. 31

[41] University of Magdeburg, Otto von Guericke: *SPL2go*. Available at `http://spl2go.cs.ovgu.de/`, 2011. Accessed: 2016-01-27. 31

[42] Kramer, J., J. Magee, M. Sloman, and A. Lister: *CONIC: an integrated approach to distributed computer control systems*. Computers and Digital Techniques, IEE Proceedings E, 130(1):1–, January 1983. 31, 44

[43] Plath, Malte and Mark Ryan: *Feature integration using a feature construct*. Science of Computer Programming, 41(1):53–84, September 2001. 31, 44

[44] Leite, Alessandro Ferreira, Vander Alves, Genaína Nunes Rodrigues, Claude Tadonki, Christine Eisenbeis, and Alba Cristina Magalhaes Alves de Melo: *Dohko: an autonomic system for provision, configuration, and management of inter-cloud environments based on a software product line engineering method*. Cluster Computing, 20(3):1951–1976, 2017. `https://doi.org/10.1007/s10586-017-0897-1`. 31

[45] Schröter, Reimar, Sebastian Krieter, Thomas Thüm, Fabian Benduhn, and Gunter Saake: *Feature-Model Interfaces: The Highway to Compositional Analyses of Highly-Configurable Systems*. pages 667–678, 2016. 48

[46] Angerer, Florian, Andreas Grimmer, Herbert Prähofer, and Paul Grünbacher: *Configuration-Aware Change Impact Analysis*. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 385–395, Washington, DC, USA, 2015. IEEE Computer Society, ISBN 978-1-5090-0025-8. `https://doi.org/10.1109/ASE.2015.58`. 49

Appendix A

Analysis data for all evolution steps of all product lines

In the following tables, RO denotes the original reliability analysis (ReAna) and RE denotes the evolution-aware reliability analysis (ReAnaE). Runtime analyses are in seconds and memory consumption in megabytes. Boldface indicates that the treatment is superior and statistically significantly in the model analyzed. Standard deviation in shown in parentheses.

Table A.1: Evolution Scenario 1 (adding new feature): BSN

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
BSN 0	0.52 (0.07)	0.6 (0.3)	–	35.57	35.56	–
BSN 1	0.6 (0.1)	0.8 (0.2)	–	35.59	**31.57**	Large
BSN 2	0.6 (0.1)	0.7 (0.2)	–	35.56	**31.70**	Large
BSN 3	0.6 (0.1)	0.7 (0.2)	–	36.66	**32.66**	Large
BSN 4	0.6 (0.1)	0.6 (0.1)	–	37.64	**32.66**	Large
BSN 5	0.6 (0.2)	0.8 (0.2)	–	39.57	**32.66**	Large
BSN 6	0.55 (0.07)	0.6 (0.1)	–	41.62	**33.64**	Large
BSN 7	0.6 (0.1)	0.7 (0.2)	–	41.62	**34.61**	Large
BSN 8	0.6 (0.2)	0.9 (0.4)	–	41.62	**35.54**	Large
BSN 9	**0.57 (0.04)**	0.8 (0.2)	Large	44.66	**38.58**	Large
BSN 10	**0.69 (0.09)**	0.9 (0.2)	Large	49.62	**43.70**	Large
BSN 11	1.0 (0.1)	1.4 (0.5)	–	**27.55**	28.66	Large
BSN 12	2.46 (0.09)	**1.8 (0.3)**	Large	**27.54**	35.20	Large
BSN 13	**2.3 (0.03)**	2.8 (0.2)	Large	47.05	**40.97**	Large
BSN 14	**4.5 (0.1)**	5.0 (0.3)	Large	86.63	**76.11**	Large
BSN 15	**8.7 (0.1)**	9.4 (0.2)	Large	268.91	**211.16**	Large
BSN 16	**17.9 (0.3)**	19.7 (0.3)	Large	608.02	**378.20**	Large
BSN 17	**39.4 (0.6)**	42.3 (0.6)	Large	**295.00**	381.41	Large
BSN 18	117.0 (4.0)	**90.7 (0.5)**	Large	1203.29	**555.84**	Large
BSN 19	419.0 (6.0)	**249.0 (7.0)**	Large	2472.90	**1258.10**	Large
BSN 20	**870.0 (10.0)**	930.0 (10.0)	Large	4766.50	4607.50	–

Table A.2: Evolution Scenario 1 (adding new feature): Email

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Email 0	0.52 (0.05)	0.52 (0.05)	–	34.61	34.61	–
Email 1	0.6 (0.2)	**0.39 (0.06)**	Large	34.62	**30.59**	Large
Email 2	0.5 (0.2)	**0.36 (0.01)**	Large	34.62	**31.54**	Large
Email 3	0.5 (0.2)	0.36 (0.02)	–	35.59	**31.55**	Large
Email 4	0.6 (0.1)	**0.38 (0.04)**	Large	35.59	**31.57**	Large
Email 5	0.5 (0.1)	**0.39 (0.05)**	Large	36.69	**31.60**	Large
Email 6	0.5 (0.1)	**0.379 (0.008)**	Large	37.65	**32.66**	Large
Email 7	0.51 (0.06)	**0.4 (0.02)**	Large	38.59	**32.66**	Large
Email 8	0.6 (0.1)	**0.41 (0.01)**	Large	40.66	**34.58**	Large
Email 9	0.7 (0.2)	**0.45 (0.02)**	Large	42.58	**36.66**	Large
Email 10	1.1 (0.4)	**0.53 (0.01)**	Large	**27.56**	39.68	Large
Email 11	1.5 (0.3)	**0.66 (0.01)**	Large	51.80	**47.54**	Large
Email 12	2.4 (0.1)	**1.02 (0.08)**	Large	69.87	**29.62**	Large
Email 13	4.33 (0.08)	**1.71 (0.08)**	Large	76.18	**55.31**	Large
Email 14	8.5 (0.1)	**3.21 (0.1)**	Large	283.84	**72.34**	Large
Email 15	17.2 (0.4)	**6.4 (0.05)**	Large	**190.83**	193.90	Large
Email 16	34.5 (0.4)	**12.94 (0.04)**	Large	585.92	**174.48**	Large
Email 17	70.3 (1.0)	**26.31 (0.1)**	Large	797.90	**527.69**	Large
Email 18	144.7 (0.8)	**54.7 (0.2)**	Large	1261.30	**655.70**	Large
Email 19	300.0 (6.0)	**114.8 (0.5)**	Large	2671.90	**1217.90**	Large
Email 20	–	244.0 (2.0)	–	–	2402.90	–

Table A.3: Evolution Scenario 1 (adding new feature): InterCloud

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
IC 0	1.24 (0.09)	1.3 (0.2)	–	40.73	40.73	–
IC 1	1.1 (0.1)	**0.93 (0.05)**	Large	43.01	**24.49**	Large
IC 2	1.05 (0.01)	**0.92 (0.09)**	Large	47.07	**24.53**	Large
IC 3	1.1 (0.1)	**0.92 (0.07)**	Large	49.07	**24.70**	Large
IC 4	1.12 (0.04)	**0.96 (0.07)**	Large	51.06	**24.50**	Large
IC 5	1.3 (0.2)	**0.95 (0.06)**	Large	56.86	**27.09**	Large
IC 6	1.3 (0.08)	**0.96 (0.08)**	Large	63.05	**28.50**	Large
IC 7	1.6 (0.1)	**1.0 (0.1)**	Large	79.05	**30.40**	Large
IC 8	2.02 (0.03)	**1.03 (0.06)**	Large	111.05	**30.60**	Large
IC 9	2.7 (0.1)	**1.09 (0.1)**	Large	179.04	**32.60**	Large
IC 10	4.0 (0.2)	**1.04 (0.08)**	Large	303.09	**36.61**	Large
IC 11	4.3 (0.1)	**1.1 (0.1)**	Large	319.18	**40.60**	Large
IC 12	4.6 (0.1)	**1.4 (0.1)**	Large	341.18	**50.02**	Large
IC 13	5.5 (0.1)	**1.8 (0.1)**	Large	385.17	**66.22**	Large
IC 14	4.43 (0.03)	**2.7 (0.2)**	Large	229.20	**97.81**	Large
IC 15	7.64 (0.06)	**4.3 (0.2)**	Large	379.19	**160.62**	Large
IC 16	14.68 (0.06)	**8.4 (0.1)**	Large	**152.60**	292.62	Large
IC 17	29.3 (0.3)	**16.3 (0.1)**	Large	629.39	**561.22**	Large
IC 18	80.0 (9.0)	**36.7 (0.2)**	Large	1891.32	**591.15**	Large
IC 19	440.0 (30.0)	**80.2 (0.3)**	Large	4170.30	**1237.50**	Large
IC 20	–	174.4 (0.9)	–	–	2468.70	–

Table A.4: Evolution Scenario 1 (adding new feature): Lift

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Lift 0	0.49 (0.03)	0.48 (0.02)	–	32.68	**32.68**	Medium
Lift 1	0.44 (0.08)	0.39 (0.06)	–	33.64	**31.55**	Large
Lift 2	0.42 (0.03)	**0.37 (0.01)**	Large	33.63	**32.66**	Large
Lift 3	0.43 (0.06)	0.39 (0.01)	–	34.59	**32.66**	Large
Lift 4	0.5 (0.1)	0.4 (0.01)	–	35.59	**33.62**	Large
Lift 5	0.49 (0.09)	0.44 (0.02)	–	37.62	**35.55**	Large
Lift 6	0.55 (0.06)	0.493 (0.008)	–	40.66	**38.58**	Large
Lift 7	0.62 (0.07)	0.63 (0.02)	–	46.58	**44.66**	Large
Lift 8	**0.8 (0.03)**	0.89 (0.01)	Large	**27.54**	28.66	Large
Lift 9	**1.25 (0.07)**	1.5 (0.02)	Large	40.03	**39.67**	Large
Lift 10	**3.02 (0.07)**	3.16 (0.03)	Large	**49.08**	58.11	Large
Lift 11	**5.6 (0.2)**	9.0 (0.1)	Large	120.86	**79.74**	Large
Lift 12	**9.75 (0.06)**	23.0 (0.2)	Large	334.87	**324.65**	Large
Lift 13	**20.2 (0.2)**	57.4 (0.3)	Large	237.15	**182.18**	Large
Lift 14	**42.5 (0.2)**	140.6 (0.8)	Large	738.81	**547.74**	Large
Lift 15	**90.3 (0.6)**	312.0 (2.0)	Large	1027.50	**773.30**	Large
Lift 16	**188.9 (0.8)**	692.0 (5.0)	Large	1751.50	**1681.10**	Large
Lift 17	**406.0 (3.0)**	1490.0 (7.0)	Large	**3405.70**	3502.30	Large

Table A.5: Evolution Scenario 1 (adding new feature): MinePump

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
MP 0	0.6 (0.1)	0.54 (0.02)	–	36.66	36.66	–
MP 1	0.5 (0.1)	0.47 (0.09)	–	37.62	**31.56**	Large
MP 2	0.45 (0.09)	0.46 (0.08)	–	37.65	**31.57**	Large
MP 3	0.5 (0.1)	0.47 (0.06)	–	38.60	**32.66**	Large
MP 4	0.46 (0.02)	0.45 (0.06)	–	39.55	**32.66**	Large
MP 5	0.52 (0.08)	0.44 (0.06)	–	41.81	**33.62**	Large
MP 6	0.54 (0.03)	**0.48 (0.04)**	Large	44.66	**34.60**	Large
MP 7	0.66 (0.09)	**0.5 (0.05)**	Large	50.58	**36.66**	Large
MP 8	0.83 (0.07)	**0.55 (0.03)**	Large	**27.54**	39.54	Large
MP 9	1.17 (0.06)	**0.72 (0.07)**	Large	48.30	**47.54**	Large
MP 10	1.91 (0.09)	**1.01 (0.08)**	Large	49.77	**28.66**	Large
MP 11	3.45 (0.07)	**1.6 (0.1)**	Large	**30.20**	51.22	Large
MP 12	6.9 (0.1)	**2.95 (0.08)**	Large	253.25	**42.99**	Large
MP 13	13.82 (0.08)	**5.8 (0.07)**	Large	381.75	**189.55**	Large
MP 14	27.9 (0.3)	**11.7 (0.1)**	Large	361.33	371.26	–
MP 15	57.36 (0.07)	**24.27 (0.1)**	Large	1030.91	**773.90**	Large
MP 16	124.0 (2.0)	**50.5 (0.4)**	Large	1777.89	**754.25**	Large
MP 17	263.0 (1.0)	**109.0 (2.0)**	Large	2567.70	**834.50**	Large
MP 18	546.0 (2.0)	**244.0 (5.0)**	Large	2334.10	**1837.30**	Large
MP 19	–	551.0 (10.0)	–	–	3472.70	–

Table A.6: Evolution Scenario 1 (adding new feature): TankWar

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
TW 0	124.0 (2.0)	123.3 (0.8)	–	391.16	393.05	–
TW 1	166.0 (1.0)	**27.9 (0.1)**	Large	425.05	**214.40**	Large
TW 2	166.1 (0.7)	**28.2 (0.6)**	Large	424.45	**279.16**	Large
TW 3	167.0 (1.0)	**23.2 (0.2)**	Large	**437.61**	563.40	Large
TW 4	166.5 (0.7)	**23.1 (0.2)**	Large	**457.40**	580.80	Large
TW 5	167.3 (0.8)	**23.7 (0.2)**	Large	**491.67**	592.14	Large
TW 6	169.1 (0.5)	**24.3 (0.1)**	Large	**568.54**	603.97	Large
TW 7	174.5 (1.0)	**25.5 (0.2)**	Large	288.41	**214.21**	Large
TW 8	205.0 (2.0)	**27.7 (0.1)**	Large	838.36	**214.32**	Large
TW 9	352.0 (3.0)	**33.1 (0.3)**	Large	1098.90	**233.34**	Large
TW 10	–	42.0 (0.2)	–	–	342.05	–
TW 11	–	57.6 (0.6)	–	–	538.45	–
TW 12	–	95.0 (2.0)	–	–	1141.91	–
TW 13	–	199.0 (3.0)	–	–	2577.16	–

Table A.7: Evolution Scenario 2 (adding new message): BSN

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
BSN 0	0.52 (0.09)	0.52 (0.07)	–	35.57	35.56	–
BSN 1	0.51 (0.09)	**0.42 (0.05)**	Large	34.62	**28.68**	Large
BSN 2	0.45 (0.03)	**0.39 (0.02)**	Large	35.56	**28.68**	Large
BSN 3	0.47 (0.08)	**0.39 (0.02)**	Large	35.59	**28.66**	Large
BSN 4	0.48 (0.09)	**0.41 (0.03)**	Medium	35.58	**28.66**	Large
BSN 5	0.47 (0.04)	**0.39 (0.02)**	Large	35.60	**28.68**	Large
BSN 6	0.51 (0.1)	**0.41 (0.03)**	Large	36.66	**28.68**	Large
BSN 7	0.47 (0.03)	**0.42 (0.03)**	Large	36.68	**28.68**	Large
BSN 8	0.48 (0.02)	**0.41 (0.02)**	Large	36.76	**28.66**	Large
BSN 9	0.51 (0.09)	**0.41 (0.02)**	Large	37.64	**28.69**	Large
BSN 10	0.49 (0.03)	**0.44 (0.03)**	Large	37.64	**28.68**	Large
BSN 11	0.5 (0.04)	**0.43 (0.02)**	Large	38.62	**28.68**	Large
BSN 12	0.51 (0.03)	**0.43 (0.03)**	Large	39.59	**29.52**	Large
BSN 13	0.5 (0.1)	**0.43 (0.02)**	Large	39.60	**29.63**	Large
BSN 14	0.55 (0.08)	**0.43 (0.02)**	Large	40.67	**29.64**	Large
BSN 15	0.53 (0.07)	**0.42 (0.02)**	Large	40.68	**29.63**	Large
BSN 16	0.55 (0.08)	**0.43 (0.02)**	Large	41.66	**29.63**	Large
BSN 17	0.53 (0.04)	**0.43 (0.01)**	Large	42.61	**29.63**	Large
BSN 18	0.55 (0.04)	**0.43 (0.02)**	Large	43.58	**29.66**	Large
BSN 19	0.57 (0.05)	**0.44 (0.03)**	Large	43.97	**29.65**	Large
BSN 20	0.57 (0.03)	**0.44 (0.03)**	Large	44.66	**29.64**	Large

Table A.8: Evolution Scenario 2 (adding new message): Email

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Email 0	0.52 (0.05)	0.52 (0.05)	–	34.61	34.61	–
Email 1	0.45 (0.07)	**0.38 (0.03)**	Large	32.69	**27.56**	Large
Email 2	0.47 (0.08)	**0.4 (0.03)**	Large	32.68	**27.54**	Large
Email 3	0.46 (0.09)	**0.38 (0.03)**	Large	32.68	**27.55**	Large
Email 4	0.46 (0.06)	**0.4 (0.03)**	Large	33.65	**27.55**	Large
Email 5	0.45 (0.03)	**0.4 (0.03)**	Large	33.65	**27.54**	Large
Email 6	0.45 (0.03)	**0.4 (0.03)**	Large	33.62	**27.56**	Large
Email 7	0.46 (0.03)	**0.4 (0.03)**	Large	34.63	**27.68**	Large
Email 8	0.49 (0.08)	**0.4 (0.03)**	Large	34.63	**27.70**	Large
Email 9	0.47 (0.03)	**0.4 (0.03)**	Large	35.57	**27.70**	Large
Email 10	0.5 (0.07)	**0.41 (0.02)**	Large	35.58	**27.70**	Large
Email 11	0.46 (0.02)	**0.4 (0.02)**	Large	35.70	**28.66**	Large
Email 12	0.51 (0.09)	**0.41 (0.03)**	Large	36.68	**28.68**	Large
Email 13	0.5 (0.03)	**0.41 (0.02)**	Large	36.69	**28.66**	Large
Email 14	0.51 (0.05)	**0.42 (0.03)**	Large	37.62	**28.66**	Large
Email 15	0.56 (0.09)	**0.42 (0.03)**	Large	38.61	**28.68**	Large
Email 16	0.5 (0.1)	**0.42 (0.03)**	Large	38.62	**28.68**	Large
Email 17	0.51 (0.04)	**0.42 (0.03)**	Large	39.57	**28.66**	Large
Email 18	0.53 (0.03)	**0.42 (0.03)**	Large	40.69	**28.66**	Large
Email 19	0.56 (0.06)	**0.43 (0.03)**	Large	41.64	**28.68**	Large
Email 20	0.6 (0.1)	**0.42 (0.02)**	Large	41.62	**28.68**	Large

Table A.9: Evolution Scenario 2 (adding new message): InterCloud

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
IC 0	1.24 (0.09)	1.3 (0.2)	–	40.73	40.73	–
IC 1	1.17 (0.08)	**0.88 (0.05)**	Large	42.69	**19.05**	Large
IC 2	1.2 (0.1)	**0.86 (0.05)**	Large	42.69	**18.64**	Large
IC 3	1.3 (0.1)	**0.87 (0.05)**	Large	42.71	**18.64**	Large
IC 4	1.2 (0.1)	**0.88 (0.03)**	Large	42.74	**18.85**	Large
IC 5	1.2 (0.1)	**0.88 (0.03)**	Large	42.74	**18.84**	Large
IC 6	1.2 (0.1)	**0.9 (0.02)**	Large	42.73	**19.24**	Large
IC 7	1.2 (0.1)	**0.89 (0.04)**	Large	44.73	**19.04**	Large
IC 8	1.3 (0.2)	**0.88 (0.05)**	Large	44.72	**21.64**	Large
IC 9	1.2 (0.2)	**0.88 (0.03)**	Large	44.72	**21.24**	Large
IC 10	1.3 (0.2)	**0.89 (0.03)**	Large	42.94	**21.85**	Large
IC 11	1.3 (0.1)	**0.9 (0.05)**	Large	44.77	**22.04**	Large
IC 12	1.2 (0.1)	**0.89 (0.04)**	Large	44.74	**21.44**	Large
IC 13	1.28 (0.1)	**0.9 (0.03)**	Large	46.74	**22.04**	Large
IC 14	1.3 (0.1)	**0.89 (0.03)**	Large	46.77	**21.84**	Large
IC 15	1.3 (0.1)	**0.9 (0.03)**	Large	48.78	**22.04**	Large
IC 16	1.3 (0.08)	**0.91 (0.04)**	Large	48.78	**21.64**	Large
IC 17	1.3 (0.1)	**0.9 (0.04)**	Large	50.77	**21.64**	Large
IC 18	1.3 (0.1)	**0.9 (0.04)**	Large	50.76	**21.24**	Large
IC 19	1.3 (0.1)	**0.91 (0.03)**	Large	52.76	**21.05**	Large
IC 20	1.3 (0.2)	**0.92 (0.03)**	Large	52.81	**21.05**	Large

Table A.10: Evolution Scenario 2 (adding new message): Lift

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Lift 0	0.49 (0.03)	0.48 (0.02)	–	32.68	**32.68**	Medium
Lift 1	0.41 (0.02)	**0.38 (0.02)**	Large	31.57	**27.56**	Large
Lift 2	0.42 (0.02)	**0.38 (0.03)**	Large	32.66	**27.57**	Large
Lift 3	0.5 (0.1)	0.38 (0.02)	–	32.66	**27.55**	Large
Lift 4	0.44 (0.03)	**0.37 (0.03)**	Large	32.68	**27.56**	Large
Lift 5	0.45 (0.01)	**0.38 (0.02)**	Large	33.66	**27.56**	Large
Lift 6	0.45 (0.04)	**0.38 (0.02)**	Large	33.64	**27.55**	Large
Lift 7	0.48 (0.09)	**0.39 (0.02)**	Large	34.61	**27.56**	Large
Lift 8	0.45 (0.03)	**0.39 (0.03)**	Large	34.62	**27.56**	Large
Lift 9	0.46 (0.02)	**0.39 (0.02)**	Large	35.59	**27.55**	Large
Lift 10	0.49 (0.07)	**0.39 (0.03)**	Large	35.59	**27.57**	Large
Lift 11	0.47 (0.04)	**0.4 (0.03)**	Large	36.68	**27.55**	Large
Lift 12	0.48 (0.03)	**0.39 (0.02)**	Large	37.64	**27.55**	Large
Lift 13	0.53 (0.08)	**0.4 (0.02)**	Large	37.64	**27.55**	Large
Lift 14	0.49 (0.03)	**0.4 (0.03)**	Large	38.59	**27.55**	Large
Lift 15	0.5 (0.1)	**0.41 (0.03)**	Large	39.69	**27.56**	Large
Lift 16	0.5 (0.03)	**0.41 (0.03)**	Large	40.67	**27.68**	Large
Lift 17	0.54 (0.08)	**0.41 (0.03)**	Large	41.65	**27.70**	Large
Lift 18	0.53 (0.04)	**0.41 (0.03)**	Large	42.62	**27.70**	Large
Lift 19	0.6 (0.1)	**0.41 (0.03)**	Large	43.57	**27.70**	Large
Lift 20	0.6 (0.1)	**0.41 (0.02)**	Large	44.47	**28.66**	Large

Table A.11: Evolution Scenario 2 (adding new message): MinePump

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
MP 0	0.6 (0.1)	0.54 (0.02)	–	36.66	36.66	–
MP 1	0.46 (0.03)	**0.42 (0.02)**	Large	36.66	**28.68**	Large
MP 2	0.45 (0.03)	**0.4 (0.02)**	Large	36.68	**28.66**	Large
MP 3	0.5 (0.1)	**0.4 (0.02)**	Large	36.66	**28.66**	Large
MP 4	0.47 (0.03)	**0.4 (0.02)**	Large	36.66	**28.68**	Large
MP 5	0.5 (0.1)	**0.4 (0.03)**	Large	36.66	**28.66**	Large
MP 6	0.5 (0.1)	**0.41 (0.02)**	Large	36.66	**28.68**	Large
MP 7	0.5 (0.1)	0.43 (0.03)	–	37.63	**28.67**	Large
MP 8	0.51 (0.1)	**0.41 (0.02)**	Large	37.62	**28.66**	Large
MP 9	0.49 (0.03)	**0.42 (0.03)**	Large	37.62	**28.68**	Large
MP 10	0.5 (0.1)	**0.42 (0.02)**	Large	38.59	**28.68**	Large
MP 11	0.5 (0.1)	**0.42 (0.02)**	Large	38.59	**28.68**	Large
MP 12	0.54 (0.09)	**0.43 (0.04)**	Large	38.64	**28.69**	Large
MP 13	0.52 (0.03)	**0.42 (0.01)**	Large	39.56	**28.66**	Large
MP 14	0.56 (0.07)	**0.45 (0.02)**	Large	39.58	**28.69**	Large
MP 15	0.54 (0.03)	**0.42 (0.02)**	Large	39.59	**28.68**	Large
MP 16	0.57 (0.09)	**0.43 (0.02)**	Large	40.66	**28.67**	Large
MP 17	0.57 (0.09)	**0.43 (0.03)**	Large	40.67	**28.68**	Large
MP 18	0.55 (0.04)	**0.43 (0.02)**	Large	40.66	**29.62**	Large
MP 19	0.59 (0.07)	**0.43 (0.02)**	Large	40.66	**29.65**	Large
MP 20	0.59 (0.09)	**0.43 (0.02)**	Large	42.61	**29.64**	Large

Table A.12: Evolution Scenario 2 (adding new message): TankWar

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
TW 0	124.0 (2.0)	123.3 (0.8)	–	391.16	393.05	–
TW 1	130.0 (1.0)	**24.1 (0.1)**	Large	397.72	**274.11**	Large
TW 2	130.0 (2.0)	**19.6 (0.1)**	Large	396.24	**269.81**	Large
TW 3	129.0 (2.0)	**19.5 (0.2)**	Large	395.80	**269.01**	Large
TW 4	129.0 (2.0)	**18.9 (0.1)**	Large	397.01	**269.52**	Large
TW 5	129.0 (2.0)	**18.91 (0.07)**	Large	395.79	**269.49**	Large
TW 6	129.0 (2.0)	**19.0 (0.1)**	Large	397.38	**269.23**	Large
TW 7	129.0 (2.0)	**19.0 (0.1)**	Large	396.74	**270.97**	Large
TW 8	130.0 (2.0)	**19.1 (0.4)**	Large	397.98	**269.85**	Large
TW 9	130.0 (2.0)	**19.1 (0.1)**	Large	399.88	**270.73**	Large
TW 10	130.0 (1.0)	**19.0 (0.2)**	Large	398.03	**270.93**	Large
TW 11	129.8 (0.9)	**19.0 (0.2)**	Large	399.32	**270.67**	Large
TW 12	129.3 (1.0)	**18.9 (0.1)**	Large	399.30	**268.58**	Large
TW 13	129.0 (1.0)	**18.9 (0.08)**	Large	398.83	**269.94**	Large
TW 14	128.6 (0.7)	**19.0 (0.2)**	Large	400.13	**269.13**	Large
TW 15	129.2 (0.7)	**19.0 (0.1)**	Large	400.35	**269.34**	Large
TW 16	129.0 (1.0)	**19.0 (0.1)**	Large	403.06	**269.73**	Large
TW 17	129.0 (1.0)	**19.0 (0.1)**	Large	403.13	**269.62**	Large
TW 18	130.0 (0.7)	**19.1 (0.1)**	Large	403.00	**270.31**	Large
TW 19	129.5 (0.7)	**19.0 (0.2)**	Large	403.88	**270.45**	Large
TW 20	130.0 (1.0)	**18.95 (0.09)**	Large	403.56	**269.76**	Large

Table A.13: Evolution Scenario 3 (adding new fragment): BSN

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
BSN 0	0.52 (0.09)	0.52 (0.07)	–	35.57	35.56	–
BSN 1	0.45 (0.03)	0.41 (0.03)	–	35.60	**29.65**	Large
BSN 2	0.5 (0.1)	**0.41 (0.03)**	Medium	35.60	**29.65**	Large
BSN 3	0.47 (0.04)	0.43 (0.05)	–	36.47	**29.65**	Large
BSN 4	0.47 (0.03)	0.43 (0.05)	–	36.66	**29.65**	Large
BSN 5	0.5 (0.05)	**0.42 (0.04)**	Large	36.66	**29.65**	Large
BSN 6	0.49 (0.03)	**0.42 (0.03)**	Large	37.64	**30.60**	Large
BSN 7	0.5 (0.03)	**0.43 (0.02)**	Large	37.65	**30.63**	Large
BSN 8	0.52 (0.03)	**0.44 (0.02)**	Large	38.60	**30.63**	Large
BSN 9	0.53 (0.06)	**0.43 (0.03)**	Large	38.61	**30.64**	Large
BSN 10	0.53 (0.03)	**0.44 (0.02)**	Large	39.58	**31.09**	Large
BSN 11	0.7 (0.2)	**0.44 (0.03)**	Large	39.58	**31.55**	Large
BSN 12	0.52 (0.04)	**0.46 (0.06)**	Large	40.66	**31.56**	Large
BSN 13	0.54 (0.03)	**0.45 (0.03)**	Large	40.66	**31.58**	Large
BSN 14	0.61 (0.07)	**0.47 (0.03)**	Large	41.62	**31.58**	Large
BSN 15	0.6 (0.08)	**0.47 (0.04)**	Large	41.63	**31.70**	Large
BSN 16	0.61 (0.1)	**0.48 (0.03)**	Large	42.61	**32.66**	Large
BSN 17	0.61 (0.09)	**0.49 (0.08)**	Large	42.61	**32.66**	Large
BSN 18	0.6 (0.05)	**0.49 (0.08)**	Large	42.60	**32.66**	Large
BSN 19	0.63 (0.07)	**0.47 (0.03)**	Large	42.61	**32.68**	Large
BSN 20	0.64 (0.08)	**0.49 (0.07)**	Large	43.57	**33.64**	Large

Table A.14: Evolution Scenario 3 (adding new fragment): Email

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Email 0	0.52 (0.05)	0.52 (0.05)	–	34.61	34.61	–
Email 1	0.44 (0.03)	**0.39 (0.03)**	Large	32.66	**28.68**	Large
Email 2	0.46 (0.05)	0.42 (0.06)	–	33.64	**28.67**	Large
Email 3	0.46 (0.04)	**0.42 (0.06)**	Large	34.59	**29.65**	Large
Email 4	0.5 (0.06)	**0.4 (0.02)**	Large	34.60	**29.64**	Large
Email 5	0.5 (0.09)	**0.42 (0.02)**	Large	35.58	**30.50**	Large
Email 6	0.49 (0.08)	**0.43 (0.05)**	Large	36.66	**30.61**	Large
Email 7	0.48 (0.03)	**0.41 (0.03)**	Large	36.66	**31.57**	Large
Email 8	0.51 (0.04)	**0.43 (0.02)**	Large	37.64	**31.59**	Large
Email 9	0.7 (0.2)	**0.43 (0.02)**	Large	38.59	**32.66**	Large
Email 10	0.53 (0.09)	**0.45 (0.03)**	Large	39.55	**32.68**	Large
Email 11	0.53 (0.04)	**0.46 (0.03)**	Large	40.67	**33.65**	Large
Email 12	0.57 (0.05)	**0.48 (0.05)**	Large	41.64	**34.63**	Large
Email 13	0.58 (0.04)	**0.47 (0.02)**	Large	42.62	**35.60**	Large
Email 14	0.7 (0.1)	**0.5 (0.04)**	Large	43.59	**36.67**	Large
Email 15	0.57 (0.03)	**0.51 (0.03)**	Large	44.66	**37.64**	Large
Email 16	0.6 (0.03)	**0.5 (0.03)**	Large	45.63	**38.60**	Large
Email 17	0.62 (0.03)	**0.51 (0.03)**	Large	47.57	**39.56**	Large
Email 18	0.63 (0.05)	**0.54 (0.03)**	Large	48.66	**40.66**	Large
Email 19	0.67 (0.1)	**0.53 (0.04)**	Large	48.66	**41.65**	Large
Email 20	0.7 (0.1)	**0.56 (0.03)**	Large	50.60	**42.63**	Large

Table A.15: Evolution Scenario 3 (adding new fragment): Lift

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Lift 0	0.6 (0.5)	**0.37 (0.009)**	Large	32.66	**31.56**	Large
Lift 1	0.42 (0.03)	0.42 (0.03)	–	32.18	**30.61**	Large
Lift 2	0.5 (0.1)	0.42 (0.02)	–	32.68	**30.61**	Large
Lift 3	0.5 (0.1)	0.43 (0.03)	–	32.66	**30.60**	Large
Lift 4	0.45 (0.03)	0.43 (0.03)	–	33.66	**30.60**	Large
Lift 5	0.5 (0.2)	**0.42 (0.02)**	Large	33.66	**31.57**	Large
Lift 6	0.5 (0.1)	0.44 (0.02)	–	34.59	**31.56**	Large
Lift 7	0.47 (0.03)	0.44 (0.02)	–	34.64	**31.56**	Large
Lift 8	0.5 (0.1)	**0.44 (0.03)**	Large	35.60	**31.54**	Large
Lift 9	0.52 (0.05)	**0.45 (0.03)**	Large	35.59	**31.57**	Large
Lift 10	0.6 (0.1)	0.46 (0.03)	–	36.66	**32.66**	Large
Lift 11	0.52 (0.09)	0.46 (0.06)	–	36.68	**32.68**	Large
Lift 12	0.53 (0.06)	**0.46 (0.03)**	Large	37.64	**32.68**	Large
Lift 13	0.52 (0.04)	0.48 (0.05)	–	37.63	**32.66**	Large
Lift 14	0.6 (0.1)	**0.47 (0.03)**	Large	38.62	**32.69**	Large
Lift 15	0.56 (0.04)	**0.5 (0.05)**	Large	38.62	**33.65**	Large
Lift 16	0.58 (0.1)	**0.5 (0.06)**	Large	39.11	**33.65**	Large
Lift 17	0.61 (0.08)	**0.48 (0.03)**	Large	38.62	**33.65**	Large
Lift 18	0.6 (0.1)	**0.49 (0.02)**	Large	39.57	**33.65**	Large
Lift 19	0.57 (0.04)	**0.51 (0.02)**	Large	39.60	**34.59**	Large
Lift 20	0.59 (0.03)	**0.5 (0.02)**	Large	40.67	**34.63**	Large

Table A.16: Evolution Scenario 3 (adding new fragment): IC

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
IC 0	1.24 (0.09)	1.3 (0.2)	–	40.73	40.73	–
IC 1	1.2 (0.07)	**0.87 (0.03)**	Large	42.69	**20.25**	Large
IC 2	1.2 (0.1)	**0.89 (0.05)**	Large	42.68	**20.85**	Large
IC 3	1.18 (0.07)	**0.9 (0.04)**	Large	42.71	**20.65**	Large
IC 4	1.21 (0.05)	**0.89 (0.03)**	Large	44.74	**20.25**	Large
IC 5	1.25 (0.09)	**0.9 (0.04)**	Large	44.75	**22.64**	Large
IC 6	1.26 (0.1)	**0.91 (0.04)**	Large	45.73	**23.24**	Large
IC 7	1.3 (0.1)	**0.94 (0.08)**	Large	46.73	**23.85**	Large
IC 8	1.25 (0.08)	**0.9 (0.04)**	Large	46.72	**25.65**	Large
IC 9	1.3 (0.1)	**0.94 (0.06)**	Large	46.97	**25.84**	Large
IC 10	1.28 (0.09)	**0.93 (0.04)**	Large	48.75	**28.05**	Large
IC 11	1.28 (0.08)	**0.94 (0.05)**	Large	48.74	**27.84**	Large
IC 12	1.32 (0.1)	**0.95 (0.03)**	Large	50.79	**30.04**	Large
IC 13	1.33 (0.07)	**0.97 (0.04)**	Large	52.79	**30.24**	Large
IC 14	1.33 (0.08)	**0.99 (0.04)**	Large	52.80	**32.05**	Large
IC 15	1.32 (0.05)	**0.99 (0.04)**	Large	54.78	**33.04**	Large
IC 16	1.4 (0.08)	**1.0 (0.04)**	Large	56.77	**34.24**	Large
IC 17	1.38 (0.1)	**1.01 (0.03)**	Large	58.78	**35.65**	Large
IC 18	1.4 (0.1)	**1.0 (0.04)**	Large	58.82	**37.04**	Large
IC 19	1.4 (0.1)	**1.02 (0.04)**	Large	60.83	**39.84**	Large
IC 20	1.4 (0.07)	**1.03 (0.04)**	Large	62.81	**39.84**	Large

Table A.17: Evolution Scenario 3 (adding new fragment): MP

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
MP 0	0.6 (0.1)	0.54 (0.02)	–	36.66	36.66	–
MP 1	0.49 (0.04)	**0.42 (0.06)**	Large	36.66	**29.54**	Large
MP 2	0.5 (0.04)	**0.41 (0.02)**	Large	36.68	**29.63**	Large
MP 3	0.53 (0.1)	**0.42 (0.03)**	Large	37.63	**29.63**	Large
MP 4	0.53 (0.09)	**0.45 (0.07)**	Medium	37.64	**29.64**	Large
MP 5	0.51 (0.03)	**0.41 (0.04)**	Large	38.60	**29.64**	Large
MP 6	0.52 (0.03)	**0.42 (0.03)**	Large	38.63	**29.65**	Large
MP 7	0.52 (0.02)	**0.43 (0.02)**	Large	39.59	**30.61**	Large
MP 8	0.6 (0.1)	**0.42 (0.02)**	Large	39.58	**30.60**	Large
MP 9	0.53 (0.03)	**0.43 (0.03)**	Large	39.70	**30.60**	Large
MP 10	0.55 (0.03)	**0.45 (0.03)**	Large	40.66	**30.60**	Large
MP 11	0.56 (0.03)	**0.44 (0.03)**	Large	40.68	**30.60**	Large
MP 12	0.57 (0.04)	**0.45 (0.03)**	Large	41.62	**31.60**	Large
MP 13	0.6 (0.1)	0.5 (0.1)	–	41.64	**31.59**	Large
MP 14	0.61 (0.1)	**0.44 (0.03)**	Large	42.62	**31.60**	Large
MP 15	0.61 (0.04)	**0.46 (0.03)**	Large	42.62	**31.60**	Large
MP 16	0.6 (0.03)	**0.46 (0.03)**	Large	43.58	**31.60**	Large
MP 17	0.61 (0.04)	**0.46 (0.03)**	Large	43.57	**32.66**	Large
MP 18	0.62 (0.04)	**0.47 (0.03)**	Large	44.66	**32.69**	Large
MP 19	0.66 (0.09)	**0.47 (0.03)**	Large	44.66	**32.68**	Large
MP 20	0.66 (0.08)	**0.47 (0.03)**	Large	44.66	**32.69**	Large

Table A.18: Evolution Scenario 3 (adding new fragment): TW

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
TW 0	124.0 (2.0)	123.3 (0.8)	–	391.16	393.05	–
TW 1	129.2 (0.8)	**24.2 (0.1)**	Large	396.72	**275.23**	Large
TW 2	129.6 (0.7)	**19.6 (0.05)**	Large	397.07	**270.53**	Large
TW 3	129.2 (0.7)	**19.57 (0.07)**	Large	399.38	**270.33**	Large
TW 4	129.5 (0.9)	**19.0 (0.1)**	Large	396.89	**270.92**	Large
TW 5	130.0 (1.0)	**19.04 (0.09)**	Large	397.43	**271.49**	Large
TW 6	129.6 (0.8)	**19.06 (0.08)**	Large	398.36	**271.88**	Large
TW 7	129.8 (0.9)	**19.1 (0.1)**	Large	400.54	**270.37**	Large
TW 8	129.5 (0.8)	**19.06 (0.08)**	Large	399.60	**272.13**	Large
TW 9	129.5 (0.5)	**19.06 (0.09)**	Large	401.80	**272.08**	Large
TW 10	129.5 (0.5)	**19.1 (0.1)**	Large	400.26	**272.55**	Large
TW 11	129.5 (0.5)	**19.05 (0.08)**	Large	401.67	**273.32**	Large
TW 12	129.0 (1.0)	**19.1 (0.1)**	Large	400.72	**272.68**	Large
TW 13	129.5 (0.6)	**19.1 (0.1)**	Large	402.19	**273.03**	Large
TW 14	129.0 (0.6)	**19.3 (0.2)**	Large	403.50	**274.93**	Large
TW 15	129.0 (0.9)	**19.1 (0.2)**	Large	404.24	**274.03**	Large
TW 16	129.3 (0.9)	**19.1 (0.1)**	Large	404.30	**273.90**	Large
TW 17	129.1 (0.8)	**19.09 (0.09)**	Large	405.51	**272.93**	Large
TW 18	130.0 (1.0)	**19.1 (0.08)**	Large	404.69	**273.95**	Large
TW 19	128.8 (0.8)	**19.05 (0.06)**	Large	406.38	**275.63**	Large
TW 20	129.4 (0.8)	**19.08 (0.09)**	Large	407.79	**275.25**	Large

Table A.19: Evolution Scenario 4 (change presence condition - strengthening): BSN

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
BSN 0	0.52 (0.09)	0.52 (0.07)	–	35.57	35.56	–
BSN 1	0.46 (0.03)	**0.4 (0.03)**	Large	34.61	**28.68**	Large
BSN 2	0.5 (0.1)	**0.39 (0.03)**	Large	34.61	**28.68**	Large
BSN 3	0.5 (0.1)	0.4 (0.04)	–	34.61	**28.68**	Large
BSN 4	0.5 (0.1)	0.44 (0.06)	–	34.61	**28.68**	Large
BSN 5	0.46 (0.04)	**0.4 (0.04)**	Large	34.61	**28.68**	Large
BSN 6	0.46 (0.02)	0.43 (0.08)	–	34.62	**28.68**	Large
BSN 7	0.46 (0.02)	**0.42 (0.07)**	Large	34.61	**28.68**	Large
BSN 8	0.46 (0.02)	**0.44 (0.06)**	Medium	34.62	**28.68**	Large
BSN 9	0.5 (0.09)	**0.42 (0.04)**	Large	34.63	**28.68**	Large
BSN 10	0.45 (0.02)	0.44 (0.07)	–	34.63	**28.68**	Large
BSN 11	0.48 (0.06)	0.43 (0.06)	–	34.64	**28.68**	Large
BSN 12	0.48 (0.05)	**0.42 (0.03)**	Large	34.64	**28.68**	Large

Table A.20: Evolution Scenario 4 (change presence condition - strengthening): Email

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Email 0	0.52 (0.05)	0.52 (0.05)	–	34.61	34.61	–
Email 1	0.41 (0.02)	0.38 (0.03)	–	32.68	**27.54**	Large
Email 2	0.44 (0.09)	0.4 (0.03)	–	32.68	**27.54**	Large
Email 3	0.41 (0.02)	0.4 (0.03)	–	32.68	**27.54**	Large
Email 4	0.42 (0.02)	0.4 (0.03)	–	32.66	**27.54**	Large
Email 5	0.44 (0.03)	0.41 (0.04)	–	32.67	**27.54**	Large
Email 6	0.42 (0.02)	0.4 (0.02)	–	32.67	**27.54**	Large
Email 7	0.41 (0.02)	0.4 (0.03)	–	32.66	**27.54**	Large
Email 8	0.44 (0.07)	0.39 (0.04)	–	32.68	**27.54**	Large
Email 9	0.42 (0.02)	0.39 (0.03)	–	32.67	**27.54**	Large

Table A.21: Evolution Scenario 4 (change presence condition - strengthening): InterCloud

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
IC 0	1.24 (0.09)	1.3 (0.2)	–	40.73	40.73	–
IC 1	1.3 (0.2)	**0.91 (0.05)**	Large	40.70	**18.63**	Large
IC 2	1.2 (0.1)	**0.91 (0.05)**	Large	42.69	**19.04**	Large
IC 3	1.3 (0.2)	**0.9 (0.04)**	Large	42.68	**18.43**	Large
IC 4	1.2 (0.1)	**0.9 (0.04)**	Large	42.69	**18.23**	Large
IC 5	1.2 (0.2)	**0.91 (0.05)**	Large	42.69	**18.83**	Large
IC 6	1.3 (0.2)	**0.91 (0.04)**	Large	40.70	**18.84**	Large
IC 7	1.2 (0.2)	**0.91 (0.04)**	Large	40.69	**18.43**	Large
IC 8	1.2 (0.1)	**0.91 (0.05)**	Large	40.69	**18.84**	Large
IC 9	1.3 (0.1)	**0.91 (0.04)**	Large	40.69	**18.63**	Large
IC 10	1.2 (0.2)	**0.91 (0.04)**	Large	40.69	**18.83**	Large
IC 11	1.2 (0.2)	**0.92 (0.04)**	Large	40.70	**18.84**	Large
IC 12	1.2 (0.1)	**0.92 (0.04)**	Large	40.69	**18.63**	Large
IC 13	1.2 (0.1)	**0.91 (0.04)**	Large	40.69	**18.63**	Large
IC 14	1.2 (0.1)	**0.91 (0.04)**	Large	40.69	**19.03**	Large
IC 15	1.2 (0.1)	**0.92 (0.04)**	Large	40.69	**18.84**	Large
IC 16	1.2 (0.2)	**0.93 (0.04)**	Large	40.69	**18.83**	Large
IC 17	1.2 (0.1)	**0.91 (0.04)**	Large	40.69	**18.43**	Large
IC 18	1.2 (0.1)	**0.92 (0.03)**	Large	40.69	**18.84**	Large
IC 19	1.2 (0.1)	**0.93 (0.01)**	Large	40.69	**18.84**	Large
IC 20	1.2 (0.1)	**0.93 (0.02)**	Large	40.69	**18.43**	Large

Table A.22: Evolution Scenario 4 (change presence condition - strengthening): Lift

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Lift 0	0.49 (0.03)	0.48 (0.02)	–	32.68	**32.68**	Medium
Lift 1	0.4 (0.02)	0.37 (0.02)	–	31.69	**27.55**	Large
Lift 2	0.4 (0.03)	0.37 (0.03)	–	31.58	**27.55**	Large
Lift 3	0.4 (0.02)	0.37 (0.03)	–	31.59	**27.55**	Large
Lift 4	0.4 (0.02)	0.37 (0.03)	–	31.59	**27.55**	Large
Lift 5	0.4 (0.02)	**0.36 (0.03)**	Large	31.58	**27.55**	Large
Lift 6	0.43 (0.08)	**0.37 (0.03)**	Large	31.58	**27.55**	Large
Lift 7	0.44 (0.1)	0.4 (0.03)	–	31.69	**27.55**	Large
Lift 8	0.4 (0.03)	0.37 (0.02)	–	31.58	**27.55**	Large
Lift 9	0.43 (0.09)	0.38 (0.04)	–	31.69	**27.55**	Large

Table A.23: Evolution Scenario 4 (change presence condition - strengthening): MinePump

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
MP 0	0.6 (0.1)	0.54 (0.02)	–	36.66	36.66	–
MP 1	0.44 (0.03)	**0.4 (0.04)**	Large	35.70	**28.66**	Large
MP 2	0.44 (0.03)	**0.39 (0.03)**	Large	35.70	**28.66**	Large
MP 3	0.43 (0.04)	0.4 (0.04)	–	35.70	**28.66**	Large
MP 4	0.5 (0.1)	0.4 (0.05)	–	35.70	**28.66**	Large
MP 5	0.45 (0.03)	**0.39 (0.04)**	Large	35.70	**28.66**	Large
MP 6	0.44 (0.03)	0.4 (0.04)	–	35.70	**28.66**	Large
MP 7	0.45 (0.03)	**0.39 (0.03)**	Large	35.70	**28.66**	Large
MP 8	0.47 (0.07)	0.4 (0.04)	–	35.70	**28.66**	Large

Table A.24: Evolution Scenario 4 (change presence condition - strengthening): TankWar

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
TW 0	124.0 (2.0)	123.3 (0.8)	–	391.16	393.05	–
TW 1	124.1 (0.8)	**24.3 (0.2)**	Large	412.64	**275.23**	Large
TW 2	127.0 (1.0)	**19.8 (0.1)**	Large	497.26	**268.58**	Large
TW 3	127.0 (1.0)	**20.1 (0.3)**	Large	498.22	**268.69**	Large
TW 4	126.0 (1.0)	**19.47 (0.09)**	Large	473.81	**269.15**	Large
TW 5	125.0 (1.0)	**19.47 (0.1)**	Large	447.95	**269.29**	Large
TW 6	124.7 (1.0)	**19.5 (0.2)**	Large	432.12	**269.21**	Large
TW 7	124.0 (1.0)	**19.48 (0.08)**	Large	429.43	**269.02**	Large
TW 8	124.5 (0.7)	**19.5 (0.1)**	Large	431.95	**268.36**	Large
TW 9	123.8 (0.6)	**19.5 (0.1)**	Large	407.13	**268.86**	Large
TW 10	123.5 (0.8)	**19.5 (0.1)**	Large	408.65	**269.47**	Large
TW 11	123.9 (0.7)	**19.5 (0.1)**	Large	406.56	**268.65**	Large
TW 12	123.6 (0.8)	**19.46 (0.08)**	Large	409.19	**269.32**	Large
TW 13	123.3 (0.4)	**19.5 (0.1)**	Large	408.56	**269.02**	Large
TW 14	123.8 (0.6)	**19.5 (0.1)**	Large	409.45	**269.60**	Large
TW 15	123.9 (0.8)	**19.43 (0.09)**	Large	408.00	**267.26**	Large
TW 16	124.2 (0.7)	**19.46 (0.1)**	Large	407.91	**269.05**	Large
TW 17	123.6 (0.9)	**19.5 (0.1)**	Large	407.31	**269.21**	Large
TW 18	124.0 (1.0)	**19.6 (0.4)**	Large	408.11	**269.03**	Large
TW 19	123.6 (0.8)	**19.5 (0.2)**	Large	408.12	**268.92**	Large
TW 20	123.6 (0.9)	**19.5 (0.09)**	Large	408.49	**269.01**	Large

Table A.25: Evolution Scenario 4 (change presence condition - weakening): BSN

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
BSN 0	0.52 (0.09)	0.52 (0.07)	–	35.57	35.56	–
BSN 1	0.5 (0.1)	**0.41 (0.03)**	Medium	34.61	**28.68**	Large
BSN 2	0.47 (0.05)	0.42 (0.03)	–	34.61	**28.68**	Large
BSN 3	0.48 (0.09)	0.42 (0.03)	–	34.61	**28.68**	Large
BSN 4	0.46 (0.08)	0.41 (0.02)	–	34.62	**28.68**	Large
BSN 5	0.47 (0.06)	**0.42 (0.02)**	Medium	34.62	**28.68**	Large
BSN 6	0.47 (0.09)	0.41 (0.03)	–	34.52	**28.68**	Large
BSN 7	0.46 (0.03)	**0.42 (0.03)**	Large	34.61	**28.68**	Large
BSN 8	0.44 (0.03)	0.42 (0.03)	–	34.61	**28.68**	Large
BSN 9	0.45 (0.03)	0.41 (0.03)	–	34.62	**28.68**	Large
BSN 10	0.45 (0.02)	**0.42 (0.03)**	Medium	34.61	**28.68**	Large
BSN 11	0.45 (0.02)	**0.4 (0.03)**	Large	34.61	**28.68**	Large
BSN 12	0.46 (0.03)	0.42 (0.02)	–	34.62	**28.68**	Large

Table A.26: Evolution Scenario 4 (change presence condition - weakening): Email

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Email 0	0.52 (0.05)	0.52 (0.05)	–	34.61	34.61	–
Email 1	0.43 (0.04)	0.39 (0.03)	–	32.68	**27.54**	Large
Email 2	0.41 (0.02)	0.38 (0.04)	–	32.68	**27.54**	Large
Email 3	0.45 (0.09)	0.39 (0.04)	–	32.66	**27.54**	Large
Email 4	0.4 (0.1)	0.39 (0.03)	–	32.66	**27.54**	Large
Email 5	0.5 (0.1)	0.39 (0.03)	–	32.67	**27.54**	Large
Email 6	0.4 (0.02)	0.38 (0.03)	–	32.66	**27.54**	Large
Email 7	0.41 (0.02)	0.39 (0.03)	–	32.67	**27.54**	Large
Email 8	0.42 (0.02)	0.39 (0.03)	–	32.66	**27.54**	Large
Email 9	0.43 (0.04)	0.4 (0.03)	–	32.67	**27.54**	Large

Table A.27: Evolution Scenario 4 (change presence condition - weakening): InterCloud

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
IC 0	1.24 (0.09)	1.3 (0.2)	–	40.73	40.73	–
IC 1	1.2 (0.1)	**0.91 (0.05)**	Large	40.70	**18.43**	Large
IC 2	1.3 (0.2)	**0.92 (0.05)**	Large	40.69	**18.84**	Large
IC 3	1.2 (0.1)	**0.91 (0.05)**	Large	40.69	**18.83**	Large
IC 4	1.3 (0.2)	**0.9 (0.05)**	Large	40.69	**18.43**	Large
IC 5	1.2 (0.2)	**0.91 (0.05)**	Large	40.69	**19.04**	Large
IC 6	1.2 (0.1)	**0.91 (0.06)**	Large	40.69	**18.63**	Large
IC 7	1.2 (0.1)	**0.91 (0.05)**	Large	40.69	**18.43**	Large
IC 8	1.2 (0.2)	**0.91 (0.04)**	Large	40.69	**18.23**	Large
IC 9	1.2 (0.1)	**0.9 (0.04)**	Large	40.69	**19.24**	Large
IC 10	1.3 (0.2)	**0.91 (0.05)**	Large	40.69	**18.43**	Large
IC 11	1.3 (0.2)	**0.92 (0.05)**	Large	40.69	**18.63**	Large
IC 12	1.2 (0.1)	**0.91 (0.04)**	Large	40.69	**19.24**	Large
IC 13	1.3 (0.2)	**0.92 (0.05)**	Large	40.69	**18.83**	Large
IC 14	1.3 (0.1)	**0.92 (0.05)**	Large	40.69	**18.63**	Large
IC 15	1.2 (0.1)	**0.9 (0.04)**	Large	40.69	**18.63**	Large
IC 16	1.3 (0.2)	**0.9 (0.05)**	Large	40.69	**18.63**	Large
IC 17	1.2 (0.1)	**0.91 (0.05)**	Large	40.69	**18.43**	Large
IC 18	1.2 (0.2)	**0.92 (0.04)**	Large	40.69	**18.43**	Large
IC 19	1.2 (0.1)	**0.91 (0.03)**	Large	40.69	**18.43**	Large
IC 20	1.2 (0.1)	**0.92 (0.03)**	Large	40.69	**19.04**	Large

Table A.28: Evolution Scenario 4 (change presence condition - weakening): Lift

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
Lift 0	0.49 (0.03)	0.48 (0.02)	–	32.68	**32.68**	Medium
Lift 1	0.42 (0.06)	0.38 (0.03)	–	31.59	**27.55**	Large
Lift 2	0.41 (0.03)	0.38 (0.04)	–	31.59	**27.55**	Large
Lift 3	0.41 (0.03)	0.39 (0.04)	–	31.58	**27.55**	Large
Lift 4	0.42 (0.03)	0.39 (0.03)	–	31.58	**27.55**	Large
Lift 5	0.41 (0.02)	0.4 (0.03)	–	31.59	**27.55**	Large
Lift 6	0.41 (0.03)	0.39 (0.03)	–	31.59	**27.55**	Large
Lift 7	0.42 (0.02)	0.39 (0.03)	–	31.58	**27.55**	Large
Lift 8	0.41 (0.03)	0.39 (0.04)	–	31.58	**27.55**	Large
Lift 9	0.41 (0.03)	0.39 (0.03)	–	31.58	**27.55**	Large

Table A.29: Evolution Scenario 4 (change presence condition - weakening): MinePump

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
MP 0	0.6 (0.1)	0.54 (0.02)	–	36.66	36.66	–
MP 1	0.5 (0.1)	**0.4 (0.04)**	Large	35.70	**28.66**	Large
MP 2	0.5 (0.1)	**0.4 (0.04)**	Large	35.70	**28.66**	Large
MP 3	0.5 (0.1)	0.4 (0.04)	–	35.70	**28.66**	Large
MP 4	0.46 (0.03)	**0.4 (0.04)**	Large	35.70	**28.66**	Large
MP 5	0.46 (0.05)	0.42 (0.04)	–	35.70	**28.66**	Large
MP 6	0.5 (0.2)	0.41 (0.04)	–	35.70	**28.66**	Large
MP 7	0.46 (0.03)	**0.4 (0.03)**	Large	35.70	**28.66**	Large
MP 8	0.45 (0.03)	**0.4 (0.03)**	Large	35.70	**28.66**	Large

Table A.30: Evolution Scenario 4 (change presence condition - weakening): TankWar

Model	RO Time (s)	RE Time (s)	Effect Size	RO Memory (MB)	RE Memory (MB)	Effect Size
TW 0	124.0 (2.0)	123.3 (0.8)	–	391.16	393.05	–
TW 1	124.0 (0.6)	**24.3 (0.3)**	Large	411.55	**273.06**	Large
TW 2	124.3 (0.4)	**19.79 (0.07)**	Large	447.98	**269.71**	Large
TW 3	125.8 (0.9)	**20.0 (0.2)**	Large	470.47	**270.20**	Large
TW 4	125.4 (0.5)	**19.45 (0.07)**	Large	471.18	**269.47**	Large
TW 5	127.0 (0.7)	**19.43 (0.1)**	Large	514.00	**269.22**	Large
TW 6	127.0 (0.7)	**19.5 (0.1)**	Large	510.65	**268.80**	Large
TW 7	125.7 (0.6)	**19.6 (0.4)**	Large	486.77	**268.86**	Large
TW 8	125.5 (0.6)	**19.4 (0.09)**	Large	482.62	**269.90**	Large
TW 9	125.9 (0.6)	**19.5 (0.1)**	Large	486.28	**269.41**	Large
TW 10	126.0 (0.9)	**19.44 (0.08)**	Large	491.49	**267.71**	Large
TW 11	126.1 (0.9)	**19.5 (0.2)**	Large	491.27	**268.53**	Large
TW 12	126.0 (0.8)	**19.5 (0.2)**	Large	490.31	**269.22**	Large
TW 13	125.9 (0.7)	**19.5 (0.1)**	Large	490.84	**267.90**	Large
TW 14	125.8 (0.7)	**19.5 (0.2)**	Large	490.94	**268.79**	Large
TW 15	125.3 (0.7)	**19.5 (0.1)**	Large	467.06	**267.93**	Large
TW 16	125.0 (0.8)	**19.5 (0.1)**	Large	467.07	**269.23**	Large
TW 17	125.3 (0.7)	**19.4 (0.1)**	Large	468.36	**268.60**	Large
TW 18	125.0 (0.5)	**19.6 (0.3)**	Large	469.07	**269.75**	Large
TW 19	125.0 (1.0)	**19.5 (0.1)**	Large	468.64	**268.37**	Large
TW 20	124.9 (0.5)	**19.5 (0.1)**	Large	470.75	**268.10**	Large