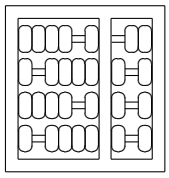

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

An Empirical Study on Design Diversity of Functionally Equivalent Web Services

Amanda S. Nascimento *Fernando Castor*
Cecília M.F Rubira *Rachel Burrows*

Technical Report - IC-12-18 - Relatório Técnico

June - 2012 - Junho

An Empirical Study on Design Diversity of Functionally Equivalent Web Services

Amanda S. Nascimento* Fernando Castor† Cecília M.F. Rubira‡
Rachel Burrows§

Abstract

A number of approaches leverage software fault tolerance techniques based on design diversity to tolerate software faults in service-oriented applications. These solutions moderate the communication between clients and functionally equivalent services, i.e., variant services. The use of design diversity depends on the assumption that variants rarely fail on the same input case. Nevertheless, it is unclear whether variant services are actually diverse and fail on disjoint subsets of the input space. In a previous work, we proposed an experimental setup to assess design diversity of variant services that realize a requirements specification. In this work, we utilize the proposed experimental setup to assess the design diversity of a number of third-party Web services adhering to seven different requirements specifications. In this paper, we describe in detail the main findings and lessons learned from this empirical study. Firstly, we investigate whether variant services present difference in their outputs and failure behaviours to look for evidence that variants are provided by different design and implementations. Secondly, we investigate the effectiveness of service diversity for tolerating faults. The results suggest that there is diversity in the implementation of variant services. However, in some cases, this diversity might not be sufficient to improve system reliability. Our findings provide an important knowledge basis for engineering effective fault-tolerant service applications.

1 Introduction

The design diversity approaches have specifically been developed to tolerate design faults in software arising out of wrong specifications and incorrect coding [1]. Design diversity is the provision of functionally equivalent software components, called variants, through different design and implementations [2]. The philosophy behind design diversity is to increase the probability that variants fail on disjoint subsets of the input space, when they do fail [2, 3]. In software fault-tolerant architectures based on design diversity, called FT-architectures, a task is executed by several variants. The results of these executions are provided to an

*Institute of Computing,University of Campinas,Campinas, SP, Brazil
†Informatics Center, Federal University of Pernambuco, Recife, PE, Brazil
‡Institute of Computing,University of Campinas,Campinas, SP, Brazil
§School of Computing, Lancaster Univeristy, Lancaster, UK

adjudicator, which operates upon them to determine which one to output as the presumably correct result [2]. We focus on voters as adjudicators [2]. Voters might fail, when coincident failures occur. A *coincident failure* is said to have occurred if variants fail on the same input case [2, 4]. Hence, FT-architectures might not support an improvement in reliability over a single software component [2, 5].

Nowadays, society is dependent on systems based on Service-Oriented Architecture (SOA) for its basic day-to-day functioning [6]. The cost and consequences of these systems failing can range from mildly annoying to catastrophic, with the possibility of great financial losses [6, 7, 8]. In order to achieve high levels of reliability, for applications comprising of autonomous services, usually controlled by third parties, it is necessary to tolerate design faults [2, 7]. When considering open and dynamic environments like the Web, a number of functionally equivalent services (*i.e.* variant services) exist to achieve a particular task [6, 8, 9, 10]. Due to the low cost of reusing variant services, several diversity-based solutions exist in the context of SOA. These solutions operate in the communication between clients and variant services. The latter are structured in fault-tolerant composite services [7, 8, 11], called *FT-compositions*. From the clients' viewpoint, the FT-composition works as a single, reliable service.

A considerable number of empirical studies aim to assess the effectiveness of diversity-based fault-tolerant applications. In these studies, the development teams, the adopted platform to implement variants and their source code are well known [5, 12, 13, 14]. However, in service-oriented computing, there is often a separation in time and space between service development and system integration [15]. On one hand, developers of services often do not have full knowledge of the different contexts in which the services will be used [15]. On the other hand, system integrators usually have no access to the internal design, source code and full specification of these services (*i.e.* services are black boxes) [15]. In addition, although there are services that aim to solve similar problems, the specifications of these problems are often diverse, developed by different organizations, with different members exhibiting different competences and understanding of the problem, and targeting different platforms. As a consequence of these factors, it is difficult to extrapolate the results of previous studies about the effectiveness of design diversity to the context of SOAs.

In our previous work, we proposed a set of directives to organize the preparation and execution of an experiment to assess, given a requirements specification, to what extent its variant services are able to tolerate software faults [16]. In this paper, we utilize the directives proposed in our previous work and execute the experimental procedures on a number of third-party, stateless, read-only Web Services adhering to seven different requirements specifications. This empirical study aims to provide more insight on the effectiveness of design diversity in service-oriented applications and its implications. Firstly, it is well known that variant services should be diverse to be usable for tolerating software faults [2, 3]. Moreover, the probabilities of coincident failures are decreased when variant exhibit diversity at the level of design, implementation and also in terms of failure behaviour [17]. Since services are black boxes, it is difficult to accurately quantify how different they are. Hence, we focused on looking for *evidence* on whether they are diverse from clients' viewpoint. We checked whether variant services presented significant differences in their outputs and failure behaviours. If variants have distinguishable observed behaviour, it suggests that they are

in fact provided by different design and implementations [16]. Secondly, even when variants present design diversity, it is recommended to actually assess how often variants fail on the same input case [17]. As previously suggested [16], by analysing the frequency of coincident failures, we estimated by how much the use of FT-compositions improved reliability over single services.

We found out that services implementing the seven requirements specifications are actually diverse at a 0.05 significance level. However, for three of the specifications, coincident failures of variant services are frequent enough that using the most reliable single service in isolation yields the best results. That is, the benefits of diversity-based solutions applied to SOAs are not straightforward. Data and observations regarding our studies are available at our study webpage [18]. Furthermore, we emphasize that empirical studies are needed from an Software Engineering (SE) perspective because they enable the development of scientific knowledge about how useful different SE solutions are, thus allowing for informed and well-grounded decision [19].

2 Background

In this section, we present an overview of service-oriented architecture and design diversity.

2.1 Service-Oriented Architecture

Many systems are being implemented following the Service-Oriented Architecture (SOA) approach in order to achieve higher levels of interoperability [6, 20]. SOA is described as a component-based model which interrelates different functional units (or services) by means of well-defined interfaces that should be neutral, platform- and language-independent [6]. Services running over heterogeneous systems may then interact and be used as building blocks for new applications [6]. A composite service, the basis for the construction of applications in the SOA world, can be regarded as a combination of activities invoked in a predefined order and executed as a whole [6].

SOA is most frequently found in Web service applications. *Web Services* often rely on XML-based standards such as SOAP (originally defined as *Simple Object Access Protocol*) and WSDL (*Web Services Description Language*) to exchange information with other applications over the Internet [21]. Web services can be read-only, which means that, given a request, these services provide access to data that may be read but not changed or deleted [16]. They can also be classified as stateless or stateful. Stateless services support no mechanism within themselves to handle state across requests [21]. Stateful services keep state information across requests [21].

2.2 Reliability

Reliability is defined as *'the ability of a system or component to perform its required functions under stated conditions for a specified period of time'* [22]. According to Lyu [3], when the execution time is not readily available, approximations such as the number of test cases executed may be used. In this sense, the successful execution rate (i.e. relative

frequency of successes) can be adopted to estimate the reliability of a system. The successful execution rate $q_{rate}(s)$ of a system s is *the estimated probability* that a request is correctly responded within the maximum expected time frame. The value of the success rate is computed from data of past invocations using the expression proposed by Zeng et. al. [23] $q_{rate}(s) = N_c(s)/k(s)$, where $N_c(s)$ is the number of successful results provided by a system s within the maximum expected time frame, and $K(s)$ is the total number of invocations of the system.

Moreover, by means of successful execution rate, we are able to estimate the difference in reliability of a FT-architecture and single non-fault-tolerant variants in order to find out evidence on whether the first supports a reliability improvement over the latter. A positive difference in reliability indicates an increase in reliability [3], that is, the FT-architecture tolerated faults of its variants, which rarely fail on the same input cases [2]. On the other hand, a negative difference indicates a reliability decrease [3]. The introduction of design diversity might lead to occurrence of coincident failures, which might defeat most voters [2, 3, 5].

2.3 Diversity-based Architectures

With software fault tolerance, we want to prevent failures by tolerating faults whose occurrences are known when errors are detected [2]. One of the means to protect against software faults and thus enhance software reliability is to use software fault tolerance techniques based on design diversity [2]. The use of design diversity depends on the assumption that variant components rarely fail on the same input case because this makes failures of variants detectable [2, 3].

In software fault-tolerant architectures based on design diversity, input is distributed to the variants, or versions, which then execute their operations. Potentially, several alternate results can be produced, from which a single correct or acceptable result must be derived [2]. The mechanism responsible for this task is called an *adjudicator*, or *decider*. There are two main types of adjudicators: voters and acceptance tests (ATs). When voters are employed, all the variants are executed and the voter compares all the produced results. With ATs, only one variant is executed at a time. The AT is responsible for checking whether the produced result is correct. In case it is not, another variant is executed until a correct result is obtained, if possible. In general, ATs are more difficult to construct in practice because they are strongly application-dependent and it is not always possible to determine a criterion to judge variant results [2], which further motivates our work. We focus on voters as adjudicators.

2.3.1 Voter Adjudicators

In particular, we are interested in three-variant voting systems since this is the minimum number of variants that allows a service composition to tolerate faults from one of its services. In a three-variant system, three kind of results are possible: *(i) correct result*- executions of a majority of the variants in the triplet result in identical or similar correct results from which a single correct result is adjudicated [13]; *(ii) failure*- a majority of

variants fail on the same input case resulting in similar or identical incorrect results (*i.e.* coincident failures) therefore the voter returns either an incorrect result as the presumably *'correct one'*, or no output [2, 13]; *(iii) failure exception*- three results that are not within an acceptable range (*i.e.* they are not similar) are returned by execution of variants, therefore, a voter is not able to decide whether results were successful or not [13].

3 Study Setting and Execution

In this section, we present study setting and execution, including, our hypotheses, target requirements specifications, variants and inputs; and how we collected the data. The data obtained is also described by means of set and mathematical notations to provide clear and concise understanding. Most of these notations were previously proposed in our experimental setup [16].

3.1 Research Questions and Hypotheses of the Study

To conduct our investigation, we analysed variant services adhering to seven different requirements specifications. This strategy has benefits compared with single-specification analyses: results will be more representative. This work aims to answer the following research questions.

RQ$_0$: *Do functionally equivalent services (*i.e. *variant services) present diversity in their design and implementations?*

To empirically investigate research question RQ_0, for each specification, we hypothesize the following:

Null hypotheses ($H_{01...07}$): There is no difference in observed outputs with respect to variant services.

Null hypotheses ($H_{11...17}$): There is no difference in observed frequency (or proportion) of failures with respect to variant services.

Although hypotheses are assigned to each requirements specification, we do not focus on understanding the real impact of each specification individually, only of the results. Furthermore, given any individual requirements specification, if we cannot reject at least one of its related hypotheses, within the scope of this work, we will report that there is not enough *evidence* to either confirm or deny the existence of diversity in the implementation of its variant services.

RQ$_1$: *Does the use of a FT-compositions, built with variant services and voters, support an improvement in reliability when compared to single services?*

3.2 Target Requirements Specifications and Variants

We selected seven requirements specifications that are performed by a population of third-party, stateless, read-only SOAP/WSDL-based Web services. We will refer to these as services, for simplicity. The selection of these services required cost-free variant availability of services and the possibility of comparing variant results. Table 1 briefly describes the requirements specifications. Variant services adhering to these specifications were selected

at online services repositories [25, 26]. For all variants, we identified their corresponding descriptions, specified in *WSDL* documents. Based on these descriptions, we created Web service clients by adopting the *Java API for XML Web Services (JAX-WS)*, a Java programming language API for creating and invoking Web services [21]. For each one of the variant services, the Java client is responsible for invoking its operation for all related input cases, thus, obtaining the output values.

Table 1: Target Requirements Specifications.

Requirements Specification (Functionality)
(1) *Email Validations* validates email addresses for client applications.
(2) *Credit Card Validations* validates credit card numbers.
(3) *Distance By ZIP Codes* finds the distance between any two ZIP Codes.
(4) *Currency Trading* returns 'up-to-date' currency rates.
(5) *Temperature Conversion* converts temperature from Kelvin to Celsius.
(6) *Weather Forecast* provides weather forecast for cities.
(7) *ZIP code geocoding* converts ZIP codes into geographic coordinates.

Definition 1. *Let* $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$ *be the set of all analysed requirements specifications.*

Definition 2. *For each requirements specification* $r \in R$*, let* $V(r)$ *be the set of variant services in* r*, such as* $V(r) = \{v_1, v_2, v_3\}$ *[16].*

The $V(r)$ set contains exactly three elements, the minimum number of variants employed by diversity-based techniques that leverage voters [2] (*Section 2.3.1*).

3.3 Target Input Cases

For each requirements specification $r \in R$, we select input cases at random from the input space [17]. The same inputs were supplied to each variant $v \in V(r)$ in order to improve the precision of the experiment [16, 19, 27]. The input cases were generated as follows:
(1) Email Validations: We choose three popular domains (*i.e.* Yahoo, Gmail and Hotmail) and two unpopular domains (*i.e.* ic.unicamp.br, ige.unicamp.br). Each one of these mail servers specifies its restrictions on login specifications. Based on these restrictions, we randomly generate 1149 mail addresses as inputs to our experiment. We generated valid email addresses including existing ones and not-valid addresses.
(2) Credit Card Validations: We randomly generated 1046 credit card numbers. The valid credit card numbers were generated conforming to the Luhn formula (MOD 10 check) [28]. *MOD 10 check* is a simple checksum formula used to validate a variety of identification numbers, including credit card numbers [28]. The adopted valid credit card types were *MasterCard, VISA 16 digit, VISA 13 digit, American Express, Discover, Diners Club, enRoute, JCB 15 digit, JCB 16 digit* and *Voyager*. Invalid credit card numbers were randomly generated and they did not conform to the Luhn formula.

(3) Distance By Zip Codes: We generated 1021 input pairs comprising of an origin Zip code and a destination one: $< Starting\ ZIP\ code, Ending\ ZIP\ code >$. Both codes were randomly selected from a database that contains 13137 Zip Codes within the United States [29].
(4) Currency Trading: The following currencies compose the input set used to get 'up-to-date' currency exchange rates: AUD *(Australian Dollars)*, CAD *(Canadian Dollars)*, CHF *(Swiss Francs)*, DKK *(Danish Kroner)*, EUR *(Euros)*, HKD *(Hong Kong Dollars)*, JPY *(Japanese Yen)*, NOK *(Norwegian Kroner)*, NZD *(New Zealand Dollars)*, SEK *(Swedish Krona)*, SGD *(Singapore Dollars)*, TWD *(Taiwan Dollars)*, USD *(US Dollars)* and ZAR *(South African Rand)*.
(5) Temperature Conversion: We randomly generated 1049 real numbers as input case including values below absolute zero, the lowest possible temperature [30].
(6) Weather Forecast: We randomly selected 926 Zip Codes from the database that contains 13137 US Zip Codes [29].
(7) US Zip code geocoding: We randomly selected 1200 Zip codes from a database with 13137 US Zip Codes [29].

Definition 3. *For each requirements specification $r \in R$, let $X(r)$ bet the set of input cases in r [16].*

3.4 Execution of Variant Services

The investigation was based on the analysis of the output space. For each requirements specification $r \in R$, we distributed the input cases $x \in X(r)$ to the variants $v \in V(r)$, which then execute their operations. We adopted the $exe(v, x)$ function that returns the output value resulting from the execution of a variant v under input x [16]. Afterwards, we identified sets of outputs, as follows.

Definition 4. *For each $r \in R$ and its variants $v \in V(r)$, let $O(r, v)$ be the set of outputs in r and v. Such as, $O(r, v) = \left\{(v, x, exe(v, x))|x \in X(r)\right\}$, that is, it is the set that associates the v variant service, inputs $x \in X(r)$ and the corresponding outputs $exe(v, x)$ [16].*

It should be noticed that the whole investigation was made based on the comparison of third elements of identified triples. The first two elements of the triples are identifiers adopted to guarantee that there is no duplicate element, which is not allowed (or is ignored) in sets [16].

3.5 Target Gold Versions and Acceptance Criteria

For each requirements specification $r \in R$, given its variant services $v \in V(r)$ and its inputs $x \in X(r)$, we identified whether a variant fails or not under certain input. Usually, in this type of experiment, the experiment administrator employs a 'gold' version as a means to identify failures [5, 13]. It is used by comparing the output of the variant service, for a given input case, to the output that is known to be correct, the one returned by the gold version [5, 13]. For each particular set of requirements $r \in R$, we either wrote or employed

a third-party gold version $g(r)$. In a complementary way, for some variant services, we adopted acceptance criteria, which is used to evaluate whether a variant result is acceptable compared to the one returned by the gold version, as follows.

Gold Versions: For the (1) Email validations, we implemented a program based on the restrictions on login specifications defined by the adopted mail servers. For (2) Credit card validations, we implemented a routine based on the *Luhn* algorithm. This algorithm is described in ISO/IEC 7812-1[31]. For the (3) Distance by ZIP codes, we implemented an algorithm based on Google API [32]. The gold version for (4) Currency trading is based on the values provided by *CNN Money* [33]. For the (5) Temperature conversion, our implementation was based on the relationship between Kelvin and Celsius scales [30]. For (6) Weather forecast, we implemented a routine based on information for weather forecast provided by *Weather.com*. The (7) Geocoder gold version is based on the Google API [32].

Acceptance Criteria The *(3) Distances by ZIP codes*, *(4) Currency Trading*, *(5) Temperature Conversions* and *(6) Code Geocoding* are realized by services that use floating-point arithmetic (FPA). The use of FPA in general computing produces a result that is accurate only within a certain range [2]. The use of design diversity, especially if FPA is used, can also produce individual variant results that differ within a certain range [2]. In this way, Pullum defines a tolerance as a variance allowed by a decision algorithm to decide whether results were successful [2]. Consequently, for the services whose outputs use FPA, we adopted acceptance criteria to check whether variant services 'disagree'or not with gold versions. We specify tolerance ranges based on measures of dispersion under all variant results and the gold version result, thus, reducing subjectiveness of the specified criterion.

3.6 Identification of Failing Inputs

We identified input cases under which at least one variant fails. This was performed in order to analyse and understand the failure behaviour of variant services [16]. To identify failing inputs, we adopted the performance function $p(v, x)$, defined by Littlewood and Miller [17]. This function indicates whether a variant v, when executed under the input x, returns either a correct result or a failure within the maximum expected time frame TF. The performance function $p(v, x)$ is defined as follows [17]:

$$p(v,x) = \begin{cases} 1 & \text{if } v \text{ does } \textbf{not} \text{ fail on } x \text{ within } TF \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Therefore, for a specification $r \in R$, $p(v, x) = 1$, where $v \in V(r)$ and $x \in X(r)$, iff the output of a variant v on input x is correct or acceptable compared with the output of the $g(r)$ gold version on input x. For instance, the maximum expected time frame (TF) was of 5 minutes.

The $FX(r)$ set of failing inputs, is defined as follows.

Definition 5. *For each requirements specification $r \in R$, let $FX(r) \subseteq X(r)$ be the set of failing inputs in r, such as, $FX(r) = \{x | x \in X(r) \ \wedge (\exists v \in V(r) \mid p(v, x) = 0)\}$ [16].*

We emphasize that for all analysed requirements specifications $|FX(r)| \geq 1$. That is, for each specification $r \in R$, given its variant services belonging to $V(r)$ and its inputs from the $X(r)$ set, at least one variant fails on at least one input case.

3.7 Variant Performances under Failing Input Cases

For each input case under which a variant fails, if any other variant does not fail on the same input case, there is evidence that *(i)* a diversity-based technique might tolerate that fault [34]; *(ii)* variant services fail on disjoint subsets of the input space [2]; *(iii)* variant services have distinguishable frequency of failures [5]. In order to look for such evidence, we analysed the performance of each variant under failing input cases [16]. We observed the outputs produced by all the variants under the inputs from the $FX(r)$ set and determined which variants failed under those inputs.

Definition 6. *For each requirements specification $r \in R$ and its variants $v \in V(r)$, let $PF(r, v)$ be the set of variant performances under failing inputs. Such as, $PF(r, v) = \Big\{(v, x, p(v, x))|x \in FX(r)\Big\}$, i.e., it is the set that associates the v variant service, failing inputs $x \in FX(r)$ and the related variant performance $p(v, x)$ [16].*

In a complementary way, for each requirements specification $r \in R$, we define:

Definition 7. *Let $FS(r) = PF(r, v_1) \cup PF(r, v_2) \cup PF(r, v_3)$ be the set of all variant performances under failing inputs in specification r. That is, the $FS(r)$ associates all variants services belonging to $V(r)$, failing input cases from $FX(r)$ and variant performances under such failing inputs. The $FS(r)$ set is called as failure scenario, for simplicity.*

3.8 Estimated Reliability of the Architectural Solutions

To measure the reliability of single services and of FT-compositions, we adopted, respectively, the reliability estimators $Rel\_Est_{NFT\_S_{rv}}$ and $Rel\_Est_{FT\_SOA_r}$, which were proposed in our experimental setup [16]. These estimators were defined by means of the successful execution rate (*Section 2.2)*. To improve the precision of the obtained measures, all the reliability estimators are based on the analysis of failing inputs, which belong to the $FX(r)$ set [16].

Estimator for Single Services

The $Rel\_Est_{NFT\_S_{rv}}$ estimator for NFT-services is defined as follows [16]:

$$Rel\_Est_{NFT\_S_{rv}} = \frac{\sum_{x \in FX(r)} p(v, x)}{|FX(r)|} \tag{2}$$

Where, $|FX(r)| \geq 1$, as already mentioned, and $0 \leq Rel\_Est_{NFT\_S_{rv}} \leq 1$.

If $Rel\_Est_{NFT\_S_{rv}} = 1$, it implies that the analysed variant service returned correct results for all failing input cases from FX_r. If $Rel\_Est_{NFT\_S_{rv}} = 0$, the analysed variant service failed on all failing inputs [16].

Estimators for a FT-Composition

The reliability estimator for a FT-composition $Rel\_Est_{FT\_SOA_r}$, is based on the general definitions of voters (Section 2.3.1) [16]. Specifically, we are interested in cases where a majority of the variants were successful and a voter is able to adjudicate a correct result [2, 3]. We assume that the voter is perfect. Previous studies evaluating design diversity as a technique to improve overall system reliability have made the same assumption [2, 5, 13]. We adopted the following reliability estimator $Rel\_Est_{FT\_SOA_r}$ for a FT-composition [16]:

$$Rel\_Est_{FT\_SOA_r} = \frac{\sum_{x \in FX(r)} geq\Big((p(v_1, x) + p(v_2, x) + p(v_3, x)), 2\Big)}{|FX(r)|} \quad (3)$$

where, *geq* is an operator that returns 1 if the first argument is greater than or equal to the second one and 0 otherwise, $|FX(r)| \geq 1$, and $0 \leq Rel\_Est_{FT\_SOA_r} \leq 1$. If $Rel\_Est_{FT\_SOA_r} = 1$, it implies that voters were always able to adjudicate a single correct result from all variant results, where variants were executed under input cases belonging to the $FX(r)$ set. Hence, voters were able to tolerate all faults whose activation has led to failure of variants. If $Rel\_Est_{FT\_SOA_r} = 0$, voters were not able to tolerate any faults whose activation has led to coincident failures [16].

4 Study Results and Discussion

Once all data has been collected, we analysed them to provide an in-depth analysis of design diversity in service-oriented applications. In this section, we present and discusses the results and their implications.

4.1 Investigating Research Question RQ_0

For each requirements specification $r \in R$ (*Def. 1*), the set of its input cases in $X(r)$ *(Def. 3)* is subjected to different treatments, that is, it is processed by variant services from the $V(r)$ set *(Def. 2)* [16]. We investigated whether different treatments have distinguishable effect on the observed behaviour from client's viewpoint, in particular, on the output values and frequency of failures [16]. In this way, regarding research question RQ_0, we found the probabilities $p_{01,...,07}$ and $p_{11,...,17}$ of rejecting, respectively, the null hypotheses $H_{01,...,07}$ and $H_{11,...,17}$ at the significance level α, where $\alpha = 0.05$. That is, given all probabilities $p_{01,...,07,11...,17}$, if a probability was less than the significance level α, we rejected its related null hypothesis with 95% confidence. To find out all probabilities, which are estimated based on observed data, we adopted R, a language and environment for statistical computing and graphics language [35]. Details on the calculations and our R working environment are available at our study webpage [18].

4.1.1 Comparison of resulting outputs

For each requirements specification $r \in R$, given its variant services $v \in V(r)$, we checked whether its set of outputs ($O(r, v)$ - *Def. 4*) are significantly different among themselves. We test the null hypotheses $H_{01,...,07}$ at the significance level α ($\alpha = 0.05$). On one hand, for *(1) Email Validations* and *(2) Credit Card Validations*, the output values returned by their variants are categorical data. On the other hand, for *(3) Distances By Zip Codes, (4) Currency Trading, (5) Temperature Conversions, (6) Weather Forecasts* and *(7) ZIP code geocoding*, the output values are data in at least an ordinal scale. Therefore, we adopted different statistical tests for testing $H_{01,02}$ and $H_{03...07}$.

To test the null hypotheses H_{01} and H_{02}, we adopted the *Cochran Q test*. The *Cochran Q test* provides a method for testing whether three or more matched sets of frequencies or proportions differ significantly among themselves [27]. The *Cochran Q Test* is particularly applicable when categorical data are dichotomized (*e.g.* the value of each sample is either '0' or '1') [27]. We emphasize this test is suitable for out study because *(i)* the data are from $|V(r)| > 2$ related groups; *(ii)* the data are dichotomized as *valid* and *not-valid*; *(iii)* we want to examine whether the frequency of both 'valid' and 'not-valid' as outputs are the same for all variant services when they are executed under the same sequence of inputs [16, 27]. To test the null hypotheses $H_{03...07}$, we adopted the *Friedman two-way analysis of variance by ranks*. The *Friedman analysis* tests statistically whether k samples have been drawn from identical populations [27]. This analysis is suitable for our study because *(i)* we want to find out whether variant services are from the same population or not, that is, have similar output values when executed under the same input cases; *(ii)* the analysed data are in at least an ordinal scale [16, 27].

According to the results, the probabilities p_{01}, p_{02}, p_{03}, p_{04}, p_{05}, p_{06} and p_{07} were less than the significance level ($\alpha = 0.05$). Regarding research question RQ_0, we have enough evidence to reject the hypotheses $H_{01...07}$. Hence, for each requirements specification, outputs provided by its variant services differed significantly at a 0.05 significance level [27]. These results suggest that diversity is applied to variant services, otherwise, they would have had similar or identical outputs values when executed under the same sequence of inputs. We refer to Siegel and Castellan Jr. [27] for further details on both *Cochran Q test* and *Friedman analysis* procedures.

4.1.2 Comparison of frequencies of failures

Table 2 lists the absolute size of the set of input cases ($X(r)$ - *Def. 3*) and the set of failing input cases ($FX(r)$ - *Def. 5*) for all analysed requirements specifications. For *(1) Email Validation*, for example, we have identified 454 input cases under which at least one variant failed, out of a total of 1449 input cases.

For each requirements specification $r \in R$, given its variants $v \in V(r)$, we checked whether its set of variant performances ($PF(r, v)$ - *Def. 5*) are significantly different among themselves. We test the null hypotheses $H_{11...17}$ at the significance level α ($\alpha = 0.05$). We adopted the *Cochran Q test* which is suitable for this study because *(i)* the data are from ($|V(r)| > 2$) related groups; *(ii)* the data are dichotomized as '1 '(*i.e.* success) and

Table 2: Frequency of Failures

Requirements Specification	$\|X_r\|$	$\|FX_r\|$
(1) Email validations	1449	454
(2) Credit Card Validations	1046	262
(3) Distance By ZIP Codes	1021	238
(4) Currency Trading	15	1
(5) Temperature Conversion	1049	765
(6) Weather Forecast	925	224
(7) ZIP code geocoding	1200	210

'0'(*i.e.* failure); and *(iii)* we want to examine whether the frequency of both correct results and failures are the same for all variant services, which are executed under the same input cases [16]. According to the results the probabilities p_{11}, p_{12}, p_{13}, p_{15}, p_{16} and p_{17} were less than the significance level (α). Therefore, the results provide us evidence to reject the null hypotheses $H_{11,12,13,15,16,17}$. Hence, for all requirements specifications, except for *(4) Currency Trading*, we conclude that frequencies of failures are dependent on their variant services with 95% confidence [27]. That is, variant services have a distinguishable effect on the distribution of failures. For *(4) Currency Trading*, there is only one failing input, thus we did not adopt the *Cochran Q test* to either accept or reject H_{14}.

For each specification $r \in R$, we graphically illustrate the relative frequency of both correct results and failures of each individual variant $v \in V(r)$ by means of $|R|$ bar charts. These charts are represented in Figure 1. Failures and correct results are represented, respectively, under label *Failure* and *Success*. Variant identifiers (*i.e.* $v1$, $v2$, $v3$) and *Total* represent the analysed categories. Overall success rate, under label *'Total'*, give us the total frequency of failures and successes presented into a failure scenario (the $FS(r)$ set, *Def. 7*). For each particular specification, differences regarding the observed behaviour of its variants and the grand total might suggest that the proportion of failures is dependent on the variant services. Otherwise, proportions of outputs of a particular kind would be similar or identical for all bars in the related chart [36]. In Figure 1, we can notice that differences among frequencies of failures are less pronounced for variant services belonging to *(5) Temperature Conversions* (Figure 1(5)). For other requirements specifications, at least two of their variants have marked differences on their proportions of failures (*e.g.* Figure 1(1); Figure 1(2); Figure1(7)). Moreover, Figure 1(4) suggests that we can reject the hypothesis H_{14}, otherwise its variants should present the same proportion of failures and correct results. All these observations reinforce our conclusions that variant services are diverse.

4.2 Investigating Research Question RQ_1

To investigate research question RQ_1, we examined which variants fail coincidentally and which ones return similar (or identical) correct results coincidentally (*Section 2.3*). Figure 2

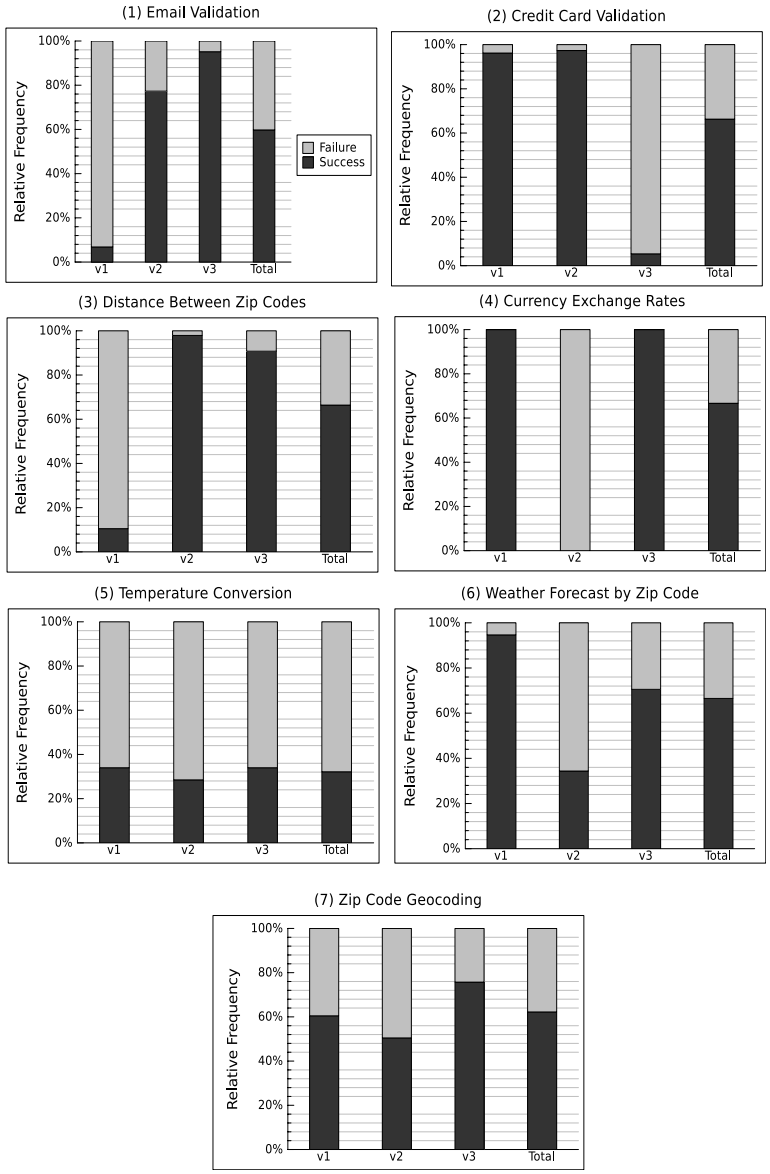

Figure 1: Joint frequency distribution of failures and successes in failure scenarios

presents the relative frequency of both coincident failures and similar correct results in each failure scenario (*i.e.* $PF(r, v)$, *Def. 7*) from a specification $r \in R$. Coincident failures are represented under the label *'Failure'*, while similar correct results are represented under label *'Success'*. Since we analysed three variant services, we represented all possible coincident failures, that is: $v1v2$ ($v1$ and $v2$ fail, $v3$ does not fail), $v2v3$ ($v2$ and $v3$ fail,$v1$ does not fail), $v1v3$ ($v1$ and $v3$ fail, $v2$ does not fail) and $v1v2v3$ (all variants fail). In a similar way, we represented similar correct results, that is, $v1v2$ ($v1$ and $v2$ are successful, $v3$ fails), $v2v3$ ($v2$ and $v3$ are successful, $v1$ fails) and $v1v3$ ($v1$ and $v3$ are successful, $v2$ fails). The

case where $v1$, $v2$ and $v3$ are correct does not belong to the set of failure scenarios, which is composed by variant performances under failing input cases. Moreover, we represented, under label *'Total'*, the total proportion of coincident failures and coincident similar results into a failure scenario.

By analysing Figure 2, it is possible to notice that frequencies of coincident failures are not the same under different matched subsets of variant services. Otherwise, for each particular requirements specification, in its related chart, proportions of coincident failures would be similar or identical for all five bars [36]. For example, regarding coincident failures, in Figure 2(1), the variants $v1$ and $v2$ fail coincidentally in approximately 20% of the input cases which compose the failure scenario, while $v1$ and $v3$ do not fail on the same input case. Related to similar correct results, e.g., in Figure 2(3), the variants $v1$ and $v3$ return similar correct results coincidentally in approximately 4% of the input cases, while the variants $v2$ and $v3$ are coincidentally successful in approximately 88% of the input cases which belong to the failure scenario.

Moreover, we can notice that behaviours in terms of both coincident failures and similar correct results differ among failure scenarios belonging to different requirements specifications. For example, in some circumstances, failure scenarios are composed predominantly by coincident failures, *e.g.* Figure 2(5), while other scenarios are mainly composed by similar correct results, *e.g.* Figure 2(2) and Figure 2(3). Moreover, for some requirement specifications, similar or identical correct results represent approximately 100% of failures scenarios (*e.g.* Figure 2(2); Figure 2(3) and Figure 2(4)). It is rare for all three variants to fail coincidentally. However, related to the *(5) Temperature Conversions*, in Figure 2(5), we can notice that all variants fail in some 36% of input cases. All these observations suggest that frequency of coincident failures is dependent on both the requirements specifications and their variant services.

On the basis of the data represented in Figures 1 and 2, for each requirements specification $r \in R$, we estimated the reliability of: *(i)* each single variant service $v \in V(r)$; and *(ii)* the FT-composition that leverages voters. These estimations were measured by adopting, respectively, the reliability estimators $Rel\_Est_{NFT\_S_{rv}}$ *(Eq. 2)* and $Rel\_Est_{FT\_SOA_r}$ *(Eq. 3)*. Among individual variant services, we selected the one that exhibits the greatest reliability, according to our estimates. Figure 3 summarized the obtained values for such estimators. For example, for *(1) Email Validations*, the reliability estimated for the most reliable single service is about 0.95, while the one estimated for the FT-Composition is about 0.80.

Based on the values represented in Figure 3, we estimated the overall differences in reliability achieved by adopting different architectural solutions by calculating ($Rel\_Est_{FT\_SOA_r} - Rel\_Est_{NFT\_S_{rv}}$), for each specification $r \in R$. The obtained values, expressed as a percentage, are summarized in Table 3. A positive difference in reliability indicates an increase in reliability [3], that is, the FT-composition tolerated faults of its variants, which rarely fail on the same input cases [2]. On the other hand, a negative difference indicates a reliability decrease [3]. The introduction of design diversity might lead to the occurrence of coincident failures, which might defeat most voters [2, 3, 5].

In Figure 3 we can observe that, for three specifications (*i.e.* Figure 3 (1); Figure 3 (4); Figure 3 (5)) the reliability estimation of fault-tolerant composite service is equal to or

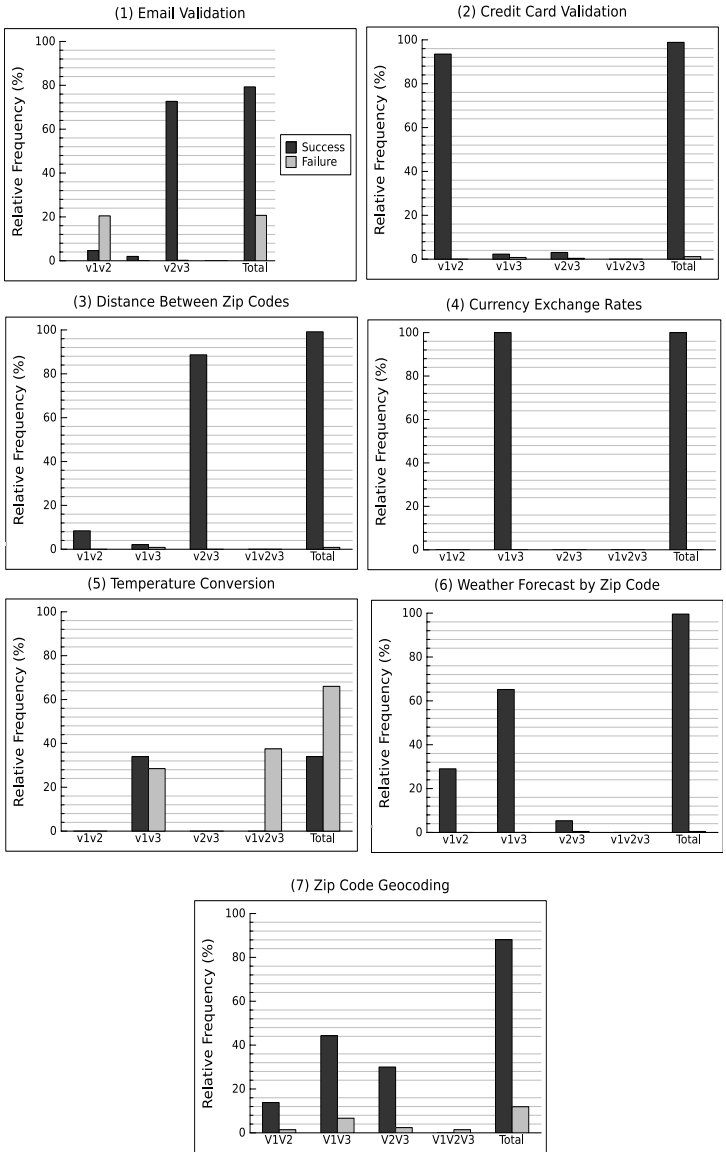

Figure 2: Relative frequency of coincident failures and similar correct results in failure scenarios FS_r, with $r \in R$.

less than the one achieved by a single service (*i.e.* the overall reliability either remains the same or decreases). Such fact is confirmed across the second column of Table 3. Therefore, regarding research question RQ_1, we cannot be confident that service diversity is always *efficient* to tolerate software faults.

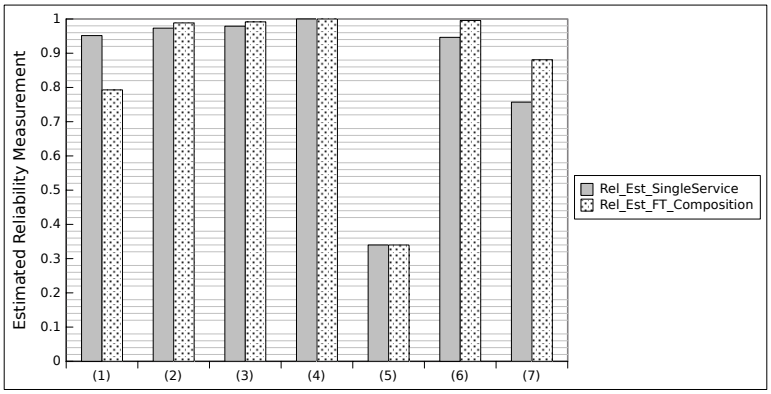

Figure 3: Reliability estimations for the different architectural solutions

Table 3: Estimation of the overall difference in reliability

Requirements Specification	**Percentage of reliability improvement**
(1) Email validations	*(-15.90)*
(2) Credit Card Validations	1.50
(3) Distance By ZIP Codes	1.30
(4) Currency Trading	0.00
(5) Temperature Conversions	0.00
(6) Weather Forecasts	4.90
(7) ZIP code geocoding	12.30

4.3 Lessons Learned:

We investigated design diversity of functionally equivalent services when facing software design faults. First, diversity-based fault tolerance techniques require some form of diversity among variants [2]. For all analysed requirements specifications, we found out that output samples returned by the execution of different variant services differ significantly among themselves. That is, the output values are dependent on the variant services. Therefore, we have sufficient evidence to conclude that variant services in fact seem to be diverse and *usable* for software fault tolerance. Second, design diversity aims to make variants as diverse as possible in order to minimize identical design faults and implementation mistakes. However, for some requirements specifications, the number of input cases under which most or all variants failed coincidentally was high enough that using the most reliable service in isolation yields the best results. Therefore, this study has shown that diverse designs in SOAs do not always result in increased overall service reliability.

For all analysed requirements specifications, the results also suggest that the frequency of failures depends on the variant service. Therefore, since there is difference in observed proportion of failures with respect to variant services, we can also conclude that the fre-

quency of coincident failures seems to be dependent on adopted variant services. That is, different combination of variant services, which are structured in FT-compositions, might imply in different measures of enhanced reliability. As we already mentioned, in the context of SOA, several services exist to achieve a particular task. Hence, in order to try to achieve higher measures of reliability by adopting diversity-fault tolerance techniques, we should first observe effects of combining different variant services.

Furthermore, care needs to be taken by software developers when selecting variant services based solely on diversity of their failure rates. In fact, the FT-composition that exhibited the highest increase in reliability was composed of variants that had only $\approx 30\%$ difference in the failure-rates between the most faulty and least faulty variant. For five of the requirements specifications, variants with higher variance in failure-rate actually displayed a lower improvement in reliability. For this reason, additional indicators must be utilized to assess how effective service diversity is to tolerate software faults, such as taking into consideration the individual failure rate of each variant. By applying a failure-rate threshold that a variant service must satisfy may help software designers pinpoint variants that are likely to negatively contribute to the overall service reliability. Once potentially problematic variants have been isolated, appropriate action can be taken such as eliminating the variant from the FT-composition or, alternatively, by increasing the total number of variants to the design so that the higher coincident failure-rates have less influence on the overall FT-composition. These findings are in line with those already published in the literature. In studies on the effectiveness of voting algorithms, it was shown that voters have a high probability of selecting the correct result value when the reliability estimated for each variant is greater than 0.5. However, when the probability of variant failure exceeds 0.5 then voters performed poorly [2, 37].

5 Study Limitations

In this section we discuss the limitations of our study based on the categories of validity threats presented by Wohlin et al. [19]. For each category, we identified the possible threats to validity and, whenever it is applicable, the measures we took to reduce the risks.

Internal Validity: One threat to internal validity we identified is guaranteeing that gold versions result in correct results under all input cases. Regarding identification of failures, the gold version actually just provides another version to check against [13]. It is, of course, possible that failures common to all of the versions, including the gold one, were not detected [3, 5, 13]. This is an unavoidable consequence of this type of experiment as pointed out in the related literature [3, 5, 13]. Hence, both variant services and analysed FT-compositions might produce results that are more or less reliable than the measured ones. To mitigate this risk, as part of the experiment, the gold version has been subjected to several test cases.

Construct Validity: This work does not address at all the aspects related to voter implementation problems (*e.g.* synchronization of the variants, delays due to communication between the end-user servers and the various remote servers, maintainability issues of fault-tolerant compositions). That is, the adoption of voters might affect other constructs

negatively. Since we do not observe these unintended side effects of voters, we identify one more threat to the construct validity: the restricted generalizability across constructs, as suggested by Wohlin et al. [19]. However, it should be noticed that the study of side effects of voters is outside the scope of this paper. We performed an investigation, specifically, on design diversity of variant services and we assume that one is able to develop a perfect voter, as already mentioned.

External Validity: We identified one threat to external validity. The variant services may not be representative of industrial practice since all of them are based on simple functionality. Regarding such risk, since we were looking for evidence on whether variant services are able to face software faults, the complexity of service functionality would have no negative effect on our final conclusions because more complex systems have larger design spaces. Therefore, there are more opportunities for the introduction of problems that have different causes [2, 5]. Furthermore, while the empirical analysis of design diversity was a hotly-debated topic in the mid-1980s and early 1990s [2], no studies so far have been carried out in the context of Web services and this study represents a step stone in this direction.

Conclusion Validity: We identified three threats to conclusion validity: *(i)* the number of requirements specifications *i.e.*, sample size; *(ii)* the homogeneity of input cases; and *(iii)* the time-out setting for the request. Risk *(i)* cannot be completely avoided due to the lack of requirements specifications implemented by cost-free, functionally equivalent SOAP/WSDL-based Web Services. Moreover, existing empirical studies on effectiveness of design diversity for fault tolerance are based on the analysis of only one requirements specification, which is implemented by several variant components [5, 38, 34]. Therefore, seven requirements functionalities seem to be sufficient to derive preliminary conclusions about the general design diversity of services. Regarding risk *(ii)*, according to Littlewood and Miller [17], in order to assess whether variants fail independently, it is necessary to guarantee that input cases are also independent, *i.e.*, heterogeneous. In this way, to mitigate risk *(ii)*, inputs from the input space were chosen at random for each requirements specification. Related to the risk *(iii)*, it is well known that services might take a variable amount of time to respond requests due to the dynamic and unpredictable nature of communication links. Consequently, some of the failures might be observed because services do not respond within the expected time frame. To mitigate this risk, we specified a high value of the time-out setting for requests (*i.e.* 5 minutes).

6 Related Work

A number of approaches [7, 8, 11, 39] operate in the communication between a service's clients and functionally equivalent services in order to tolerate software faults. In general, these solutions support different types of adjudicators (*e.g.* voters and acceptance test) and schemes to execute the variant services (e.g. sequential and parallel schemes). Nevertheless, for these existing diversity-based solutions, there is an underlying assumption that variant services can always be efficiently employed by means of diversity-based techniques. Nevertheless, findings from our studies indicate that variant services might not be able to tolerate software faults.

Knight and Leveson [5] and Eckhardt et al. [38], describe an experiment to investigate whether it is valuable to use N-version programming, a design-diversity technique based on voters, to achieve high levels of reliability. Their experiments are based on the analysis of the failure behaviour of several variants of a program. All variants were developed and validated according to a common specification using independent programming teams [5, 38]. These authors conclude that N-version programming must be used with care because the number of input cases under which most or all variants failed coincidentally was more than expected [5, 38].

Gashi et al. [34] studied design diversity as a means for tolerating design faults of four popular off-the-shelf SQL servers. Their study is based on an analysis of the bug reports available for the SQL servers. They conclude that design diversity is effective in this category of products since none of identified bugs affected more than two products. Findings from these empirical studies [5, 34, 38] reveal different conclusions regarding the effectiveness of software that relies on design diversity to tolerate faults. This in turn reinforces the necessity for a thorough assessment of the reliability of FT-compositions built with variant services. To the best of our knowledge, there is no studies assessing how effective is service diversity for tolerating faults. Our findings suggest opportunities for greater progress related to the design and implementation of fault-tolerant service-oriented applications.

In a previous work [16], we proposed an experimental setup that encompasses a set of directives to organize the preparation and execution of the experiment to investigate, given a requirements specification, (i) whether its variant services are diverse; and (ii) whether the reliability of a FT-composition that leverages voters is an improvement over one that uses a single service [16]. To exemplify and evaluate the experimental setup, it was employed to assess diversity of variant services adhering to two different requirements specifications, which were analysed individually [16]. In this work, we utilize directives of the proposed experimental setup to conduct our study. However, we focus on providing more insight of effectiveness of design diversity in service-oriented applications. We investigated whether variant services adhering to seven different requirements specifications are able to tolerate software faults and we discuss in detail our findings and lessons learned from this study. That is, this paper is a more elaborated case study to the already proposed method.

7 Concluding Remarks

We presented the results of a novel study, in the context of SOAs, to investigate whether functionally equivalent services (i.e. variant services) are able to tolerate software faults. We analysed 21 third-parties, cost-free, ready-only, stateless SOAP/WSDL-based Web Services adhering to 7 different requirements specifications. Since services are black boxes, it is a challenge issue to quantity how different they are. Due of this, we focused on looking for evidence on whether they are diverse from clients' viewpoint. We analysed whether variant services presented difference in their outputs and frequency of failures, thus indicating the presence of diversity among their design and implementations. Moreover, we investigated whether fault-tolerant composite services, called FT-composition, which structures variant services and leverages voters as adjudicators, are more reliable than single services.

We concluded that services seem to be in fact diverse. However, in some cases, coincident failures of two or more services might be frequent enough that using FT-compositions do not support an improvement in reliability compared to single services. We also demonstrated that the frequency of failures is dependent on individual variants. Consequently, the frequencies of coincident failures seem to be dependent on the set of selected variants. Therefore, variant services should be combined with care because there are differences in choosing different triplets for the same application.

We emphasize that our findings are similar to those already published in the literature. Existing work also reveals threats to the effectiveness of software that relies on design diversity to tolerate software faults. However, unlike other studies, we have investigated software components that are black boxes and independently developed by different organizations. We have conducted only one iteration of the experiments. In future work, we intend to reiterate the experiment by several times in order to analyse availability of variant services. We also intend to investigate whether our findings also apply to REST (*i.e.* Representational State Transfer) services.

Acknowledgment

This research was sponsored by UOL (www.uol.com.br), through its UOL Bolsa Pesquisa program, process number 20120217172801. Fernando is supported by CNPq (306619/2011-3 and 475157/2010-9), FACEPE (APQ-0395-1.03/10), and by INES (CNPq 573964/2008-4 and FACEPE APQ-1037-1.03/08). Cecília is supported by CNPq (305331/2009-4) and FAPESP (2010/00628-1).

References

[1] K. S. Trivedi, M. Grottke, and E. Andrade, "Software fault mitigation and availability assurance techniques," *International Journal of Systems Assurance Engineering and Management*, vol. 1, no. 4, pp. 340 – 350, 2010.

[2] L. L. Pullum, *Software fault tolerance techniques and implementation.* Norwood, MA, USA: Artech House, Inc., 2001.

[3] M. R. Lyu, Ed., *Handbook of software reliability engineering.* Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

[4] D. E. J. Eckhardt and L. D. Lee, "A theoretical basis for the analysis of multiversion software subject to coincident errors," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 12, pp. 1511 – 1517, Dec. 1985.

[5] J. C. Knight and N. G. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Transactions on Software Engineering*, vol. 12, no. 1, pp. 692–702, Jul. 1986.

[6] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38 – 45, Nov. 2007.

[7] A. S. Nascimento, C. M. F. Rubira, and J. Lee, "An spl approach for adaptive fault tolerance in soa," in *Proceedings of the 15th International Software Product Line Conference*, vol. 2, no. 15, pp. 1 – 8, Aug. 2011.

[8] Z. Zheng and M. R. Lyu, "An adaptive qos-aware fault tolerance strategy for web services," *Empirical Software Engineering*, vol. 15, no. 4, pp. 323 – 345, Aug. 2010.

[9] A. Papageorgiou, T. Krop, S. Ahlfeld, S. Schulte, J. Eckert, and R. Steinmetz, "Enhancing availability through dynamic monitoring and management in a self-adaptive soa platform," *International Journal on Advances in Software*, vol. 3, no. 3&4, pp. 434–446, Feb. 2011.

[10] S. Colucci, T. D. Noia, E. D. Sciascio, F. M. Donini, M. Mongiello, G. Piscitelli, and G. Rossi, "An agency for semantic-based automatic discovery of web services," in *AIAI*, pp. 315–328, 2004.

[11] E. M. Gonçalves and C. M. F. Rubira, "Archmeds: An infrastructure for dependable service-oriented architectures," in *Proceedings of the 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pp. 371 –378, Mar. 2010.

[12] J. L. Gersting, R. L. Nist, D. B. Roberts, and R. L. Van Valkenburg, "A comparison of voting algorithms for n-version programming," in *Proceedings of the Twenty-Fourth Annual Hawaii International Conference*, vol. 2, pp. 253 –262, Jan. 1991.

[13] T. J. Shimeall and N. G. Leveson, "An empirical comparison of software fault tolerance and fault elimination," in *Proceedings of the Second Workshop on Software Testing Verification and Analysis*, vol. 17, no. 2, pp. 180–187, Jul. 1998.

[14] N. G. Leveson, S. S. Cha, J. C. Knight, and T. J. Shimeall, "The use of self checks and voting in software error detection: an empirical study," *IEEE Transactions on Software Engineering*, vol. 16, no. 4, pp. 432 –443, Apr. 1990.

[15] F. Castor, P. A. C. Guerra, V. A. Pagano, and C. M. F. Rubira, "A systematic approach for structuring exception handling in robust component-based software," *Journal of the Brazilian Computer Society*, vol. 10, no. 3, pp. 5 – 19, Apr. 2005.

[16] A. S. Nascimento, F. Castor, C. M. F. Rubira, and R. Burrows, "An experimental setup to assess design diversity of functionally equivalent services," in *Proceedings of 16th International Conference on Evaluation and Assessment in Software Engineering (TO APPEAR)*, 2012.

[17] B. Littlewood and D. R. Miller, "Conceptual modeling of coincident failures in multiversion software," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, pp. 1596 –1614, Dec. 1989.

[18] "Variant Services - Study Webpage." last access: May. 2012. [Online]. Available: `https://sites.google.com/site/variantservices2s11/home`

[19] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[20] V. Casola, E. Mancini, N. Mazzocca, M. Rak, and U. Villano, "Building autonomic and secure service oriented architectures with mawes," in *Autonomic and Trusted Computing*, ser. Lecture Notes in Computer Science, vol. 4610, pp. 82–93, 2007.

[21] "JAX-WS Reference implementation," last access: Dec. 2011. [Online]. Available: `https://jax-ws.dev.java.net/`

[22] A. Geraci, *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. Piscataway, NJ, USA: IEEE Press, 1991.

[23] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311 – 327, May. 2004.

[24] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11 – 33, Jan.-Mar. 2004.

[25] "Seekda's Web Services portal ," last access: Dec. 2011. [Online]. Available: `http://webservices.seekda.com/`

[26] "ProgrammableWeb Repository," last access: Dec. 2011. [Online]. Available: `http://www.programmableweb.com/apis/directory/`

[27] S. Siegel and N. Castellan, *Nonparametric statistics for the behavioral sciences*, 2nd ed. New York, USA: McGraw–Hill, Inc., 1988.

[28] H. P. Luhn, "A statistical approach to mechanized encoding and searching of literary information," *IBM Journal of Research and Development*, vol. 1, no. 4, pp. 309 – 317, Oct. 1957.

[29] "US Zip Codes," last access: Dec. 2011. [Online]. Available: `http://www.census.gov/tiger/tms/gazetteer/zips.txt`

[30] F. A. Bettelheim, W. H. Brown, and M. K. Campbell, *Introduction to general, organic and biochemistry*. Thomson Brooks/Cole, 2007.

[31] "ISO/IEC 7812-1:2006: Identification cards," last access: Dec. 2011. [Online]. Available: `http://www.iso.org/iso/isocatalogue/catalogueetc/catalogue/detail.htm?csnumber=39698`

[32] "Google API for Geocoding," last access: Dec. 2011. [Online]. Available: `http://code.google.com/apis/maps/documentation/geocoding/`

[33] "CNN Money," last access: Dec. 2011. [Online]. Available: `http://money.cnn.com/data/currencies/`

[34] I. Gashi, P. Popov, and L. Strigini, "Fault tolerance via diversity for off-the-shelf products: A study with sql database servers," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, pp. 280 –294, Oct.-Dec. 2007.

[35] "The R Project for Statistical Computing," last access: Dec. 2011. [Online]. Available: `http://www.r-project.org/`

[36] D. S. Moore, *The Basic Practice of Statistics with Cdrom*, 2nd ed. New York, NY, USA: W. H. Freeman & Co., 1999.

[37] D. Blough and G. Sullivan, "A comparison of voting strategies for fault-tolerant distributed systems," in *Proceedings of the Ninth Symposium on Reliable Distributed Systems*, pp. 136 –145, Oct. 1990.

[38] D. Eckhardt, A. Caglayan, J. Knight, L. Lee, D. McAllister, M. Vouk, and J. Kelly, "An experimental evaluation of software redundancy as a strategy for improving reliability," *IEEE Transactions on Software Engineering*, vol. 17, no. 7, pp. 692 –702, Jul 1991.

[39] Y. Chen, "Ws-mediator for improving dependability of service composition," Ph.D. dissertation, Newcastle University, Newcastle upon Tyne, United Kingdom, 2008.