Quantifying the Influence of Failure Repair/Mitigation Costs on Service-Based Systems

Vittorio Cortellessa
Università dell'Aquila, Italy
Dipartimento di Ingegneria e
Scienze dell'Informazione, e Matematica

Fabrizio Marinelli
Università Politecnica delle Marche, Italy
Dipartimento di Ingegneria dell'Informazione

Raffaela Mirandola
Politecnico di Milano, Italy
Dipartimento di Elettronica, Informazione e Bioingegneria

Pasqualina Potena
Università degli Studi di Bergamo, Italy
Dipartimento di Ingegneria

***Abstract*—The analysis of non-functional properties of Service-Based Systems (SBSs) is a complex task, mostly because it requires models that encompass the composition of service properties into architectural properties. For example, the reliability of a SBS is given by the composition of service and interconnection reliabilities. Although several approaches have been introduced in the last few years to address these issues, the tradeoff analysis among non-functional properties of software services has not yet been studied enough. The goal of this paper is to introduce a set of optimization models that allow quantifying the costs of service failure repair/mitigation actions aimed at keeping the whole SBS reliability over a certain threshold. On the basis of our previous work in this area, we first introduce an optimization model aimed at selecting either in-house built or provided services with the goal of minimizing the SBS cost while guaranteeing a certain level of reliability. Thereafter we strengthen the reliability constraints, and we build two different optimization models that aim to solve the same problem under new constraints, where one model starts from the solution obtained in the original model and tries to improve it, while the other one looks for an optimal solution in the whole search space. Finally, we introduce a fourth model, based on stochastic optimization, with the goal of rather searching for solutions that explicitly take into account the stochastic nature of the problem and search for new repair/mitigation actions cheaper than the ones identified by the other models. Each optimization model has been experimented on about 300 variations of a nominal model. The experimental results show the efficacy of our optimization models to quantify the costs of different failure repairing/mitigation actions in different contexts.**

I. Introduction

Service based systems (SBSs) are playing an increasingly important role in several application domains such as healthcare, manufacturing and aerospace. Besides, they provide the building blocks for some of today's most significant topics, such as Cloud computing or Internet of Things [1]. SBSs are often realized by composing network-accessible loosely-coupled services together with possible existing in-house developed services. Network-accessible services are owned (developed, deployed, maintained, and operated) by different stakeholders or providers. They are, therefore, considered by their users as black boxes. In order to select a service, its functional and non-functional properties have to be exposed in a specific registry and according to a semantics agreed upon by the parties. The selection is implemented through a query on the registry returning the set of services matching the user functional and non-functional requirements such as reliability, performance and cost. On the other hand, in-house developed services are owned by the users and present well known and measurable functional and non-functional properties.

In this paper we focus on reliability, which is highly relevant in these domains and need to be properly addressed at design-time, not just as an afterthought in the coding stage. However, guaranteeing software reliability can influence other quality attributes such as performance or cost and, in the worst case, improving the reliability could degrade these attributes. Consider, for example, a highly dynamic system where the set of discoverable services may change over time, because the availability and/or the cost of certain services may vary according to the users location or to the network connectivity. In these settings a more reliable or efficient service might become available and thus an appropriate service selection method may allow the improvement of the overall quality of service. A further example is a loss of scalability because the number of users concurrently accessing the system increases. In this case appropriate repair/mitigation actions should be adopted (such as deciding to switch to an in-house developed service that uses its own computation resource) to tackle the peaks in the workload.

Finding the best balance between different, possibly conflicting quality requirements that a system has to meet is an ambitious and challenging goal. As a first step towards this goal, this paper presents an approach for service selection taking into account costs and reliability requirements. In particular, we define a set of optimization models that allow quantifying the costs of service failure repair/mitigation actions aimed at keeping the whole SBS reliability over a given threshold. On the basis of our previous work in this area [2], we first introduce an optimization model aimed at selecting either in-house built or provided services with the goal of minimizing the SBS cost while guaranteeing a given level of reliability. Thereafter we strengthen the reliability constraints, and we build two different optimization models that aim to solve the same problem under new constraints. Then, we introduce a fourth model, based on stochastic optimization, with the goal

978-1-4799-2366-3/13/$31.00 ©2013 IEEE

of searching for solutions that explicitly take into account the stochastic nature of the problem and searching for new repair/mitigation actions cheaper than the ones identified by the other models. The proposed approach is then analyzed through a wide set of experiments that quantify the costs of different failure repairing/mitigation actions in different contexts.

The remainder of the paper is organized as follows. Section II summarizes the works existing in this area. In Section III we give a high level overview of our approach, while the set of optimization models are formalized in Section IV. A wide experimentation has been conducted considering the different models; the obtained results along with lessons learned are described in Section V. Section VI draws the conclusion and outlines future work in this research direction.

II. Related work

As outlined in [1], [3], in the last years the topic of definition and analysis of non-functional properties in SBSs has been studied in several communities and from different perspectives. In particular, a list of approaches dealing with software architecture optimization methods for the analysis of non-functional properties of SBSs can be found in [4]. Our work proposes an approach, based on the definition of optimization models, that allows the design of SBSs able to satisfy both costs and reliability constraints under the hypothesis that repair and mitigation actions can be undertaken to maintain the service's reliability over a given threshold. The application of the proposed approach can facilitate the software architects in the design reasoning process by improving their abilities to deliver a satisfactory design. Therefore, hereafter, we review works appearing in the literature dealing with (i) *analysis of non-functional properties for SBS systems*, and (ii) *design reasoning*.

a) Analysis of non-functional properties for SBSs: A basic problem to be solved when dealing with non-functional properties for SBSs is how to determine these attributes for a composite system, given the single properties of its component services. Some papers have focused on this specific issue [5], [6], while others deal with it as a step within the more general problem of quality based model-driven runtime adaptation of SBSs (see, for example, [1], [3] and references therein). Most of the papers focused on dynamic service selection, using different approaches such as genetic algorithms in [7], optimization models with integer programming [8], mixed integer programming [9], linear programming [1], and consider non-functional attributes like performance and reliability.

Another class of related papers deals with Web services availability and reliability assessment based on analytical models [10], [11], [12] and empirical studies [13]. The analytical models exploit different kinds of Markov processes to define availability/reliability models for a composite Web service. The empirical analyses consider both workloads and the reliability of Web servers, and propose to distinguish between inter-session and intra-session Web characteristics. More recently, some papers tackled the problem of composing a service-oriented system from publicly available Web services [14], taking into account different types of web service failures.

Reliability and costs together have been considered in different contexts, for example to provide guidelines in evaluating the effort spent to test the software [15], or to deal with the resource allocation during the test process in modular software systems [16], [17].

b) Design reasoning: The approach here presented can be used complementarily to other decision-making techniques to facilitate the overall design reasoning process. One of the most known approach that explicitly analyzes the impact of architectural decisions on system quality is the Architecture Tradeoff Analysis Method [18]. Other challenges related to the quality analysis are represented by different types of uncertainties that can be faced during decision-making process [19], [20]. Research efforts have also been spent in order to deal with parameters uncertainty, adopting for example a robust optimization approach [21], or a bayesian approach [22]. The closest work to our one is presented in [23], where stochastic programming is exploited to support the service composition under quality attributes tradeoffs. In particular, the service composition problem is formulated as a multi-objective stochastic model that simultaneously optimizes quality of service parameters (i.e., workflow duration, service invocation costs, availability, and reliability). Differently from the work in [23], we adopt here stochastic programming to search for repair/mitigation actions that guarantee pre-defined system reliability levels.

With respect to existing works, this paper proposes: (i) a service selection approach that trades off between cost and reliability, where (ii) the cost function is explicitly defined exploiting the COCOMO II cost model, and (iii) different optimization models taking into account the possibility that repair actions are introduced to keep the required service reliability level.

III. Approach Overview

As outlined in the introduction, the aim of our approach is to define SBSs composing network-accessible loosely-coupled services together with possibly existing in-house developed services. These systems should be able to satisfy both costs and reliability constraints under the hypothesis that repair and mitigation actions can be undertaken to maintain the service's reliability over a certain threshold. To this end, we define a service selection approach based on the definition of a set of optimization models whose goal is to minimize the overall application cost while guaranteeing the required level of reliability. The models are formally defined in Section IV, while a high level view of the proposed approach is sketched in Figure 1.

The input of our approach is the set of functional and non-functional requirements representing the goal/objective of the SBS to-be. Considering that the system can include in-house developed services, as a first step of our approach, we explicitly define a software development cost function model based on the COCOMO II cost model [24]. Then on the basis of our previous work in this area [2], we first introduce an optimization model, called *base model*, aimed at selecting either in-house built or provided services taking into account both cost and reliability constraints. Thereafter we strengthen the reliability constraints, and we build two different optimization models that aim to solve the same problem under new constraints, where one model starts from the solution

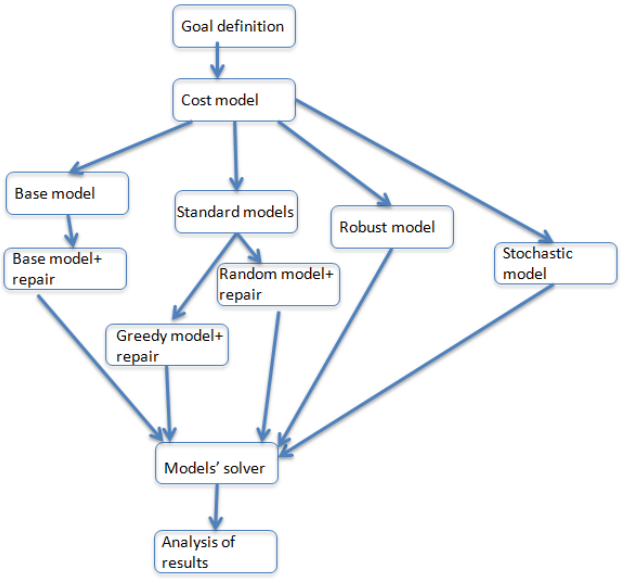


Fig. 1. High-level Approach Overview

obtained in the original model and tries to improve it, i.e. *base model with repair*, while the other one looks for an optimal solution in the whole search space, i.e. *robust model*. Then, we introduce a fourth model, based on stochastic optimization and called *stochastic model*, with the goal of rather searching for solutions that explicitly take into account the stochastic nature of the problem and search for new repair/mitigation actions cheaper than the ones identified by the other models. Finally, for the sake of comparison we also consider two reference models, i.e. *random model with repair* and *greedy model with repair*, under the strengthened reliability constraints defined for the repair model.

These models are then solved by using Couenne[1] solver, ver. 0.3.2 for Windows 32bit OS, with default settings and feasibility tolerance set to 10^{-12}. Couenne is an open source branch-and-bound algorithm for non-convex Mixed Integer Nonlinear Problems [25]. The obtained results are then analyzed to understand their strengths and weaknesses and to assess specific scenarios where some models may outperform the other ones.

IV. Approach Implementation: Optimization Models for Service Selection

In this section we formalize the set of optimization models that we have introduced in the approach overview of Section III. They all share the context that we briefly describe here below.

We assume that a service-based system made of n nominal services has to be assembled, and for each nominal service $S_i, (i = 1, \dots n)$ several alternative implementations are available, which can be split into: (i) an in-house service implementation, (ii) service implementations available for purchase by providers. We assume that all implementations of a service S_i are functionally equivalent, namely each alternative can be plugged within the system without needing any adaptation[2]. Besides, we assume that the system provides a number of k functionalities (or external services) to users, where each functionality can be described by an interaction scenario (i.e. a workflow) that involves a certain number of services S_i.

A. Base model

We start with an optimization model that stems from our previous work in the context of component-based software [2], where we have introduced a model to support build-or-buy decisions about software components while minimizing costs under reliability and delivery time constraints.

With respect to the original model, here: (i) we have plugged the problem in a service-oriented paradigm, where the build-or-buy decisions refer to services rather than components, (ii) we have refined the software development cost function (that in the original work was a linear function of the development time) with a COCOMO II cost function [24], and (iii) we have removed the delivery time constraint, for sake of focusing on reliability concerns.

The *base model* can be formulated as follows:

$$\min \sum_{i=1}^{n} (c_i y_i + \sum_{j=1}^{m} c_{ij} x_{ij}) \tag{1}$$

$$\prod_{i=1}^{n} e^{-f_i} \geq R \tag{2}$$

$$\forall i = 1 \dots n, \qquad y_i + \sum_{j=1}^{m} x_{ij} = 1 \tag{3}$$

$$\forall i = 1 \dots n, \qquad N_i \leq N^{max} y_i \tag{4}$$

We associate to each service S_i two types of 0-1 decision variables, that are: y_i and x_{ij}. They represent the alternative for a service to be in-house created (i.e. $y_i = 1$) or provided by one of the available m providers (i.e. $x_{ij} = 1$ for a $1 \leq j \leq m$). These variables are subject to the constraint (3) in order to implement a build-or-pay strategy[3].

The objective function (1) represents the cost of the whole service-based system, that has to be minimized, as induced by the choices made on all system services. We have introduced a COCOMO II-based definition for the development cost c_i of an in-house service. The original COCOMO II model [24] introduces a software cost function that depends on the size (i.e., the lines of code) and the type (i.e., simple, intermediate and complex) of software. These two attributes allows estimating the amount of effort, in terms of personmonths, needed to deliver the software.

We have adapted this model to our context, because we have introduced a variable N_i that represents the number of tests performed on a service before delivery. N_i appears both in the development cost function and in the reliability constraint,

[1] https://projects.coin-or.org/Couenne

[2] This assumption simplifies the model formulations, even though adaptation costs can be easily taken into account without heavily modifying the models.

[3] Note that if the i-th service has only $m_i < m$ providers available then its x_{ij}'s are preemptively fixed to 0 for $m_i < j \leq m$.

so as to take into account the tradeoff between the cost of testing and the required system reliability.

Hence, we introduce the following cost function for an in-house developed service:

$$c_i = cost_{pm} \cdot (((a \cdot size_i)^b \cdot (1 - testperc)) + pmt \cdot N_i) \quad (5)$$

The $(a \cdot size_i)^b$ factor is the COCOMO II model for the development personmonths of a service by $size_i$, where constants a and b depend on the software size and type. In our case this effort has been scaled by a factor $(1 - testperc)$ representing the percentage of development effort that is not spent in testing. Reliable estimates of the percentage $testperc$ of testing effort that depend on the software size and type can be found in [24].

The $pmt \cdot N_i$ term represents the effort spent in testing, as resulting from multiplying the number N_i of performed tests and the amount pmt of personmonths to perform a single test.

The whole development effort is finally multiplied by the cost of a personmonth $cost_{pm}$.

The reliability constraint (2) is typical for systems that may incur only in crash failures, i.e. failures that (immediately and irreversibly) compromise the behaviour of the whole system, where e^{-f_i} represents the reliability of service S_i [26], i.e., the probability of no failures occurring in a Poisson distribution with parameter f_i. The right hand side R represents the minimum reliability of the whole system, as specified in the system requirements.

We refer to the reliability on demand definition in [27], thus we define the average number of failures f_i of service S_i over a certain number of invocations as follows:

$$f_i = inv_i \cdot (\lambda_i y_i + \sum_{j=1}^{m} \mu_{ij} x_{ij}) \quad (6)$$

where λ_i is the average probability of failure of an in-house developed service, μ_{ij} is the same quantity for the j-th provided service, and inv_i is the average number of invocations of S_i across all considered interaction scenarios.

We assume that μ_{ij} is provided within the service datasheet or, in the worst case, it can be estimated on the basis of previous user experiences [14]. The parameters λ_i can be instead defined on the basis of any reliability growth model that has a closed-form expression; for this work we chose the one defined in [28] that links the failure probability and the number of tests as follows:

$$\lambda_i = 1 - \frac{1 - \pi_i}{(1 - \pi_i) + \pi_i (1 - \pi_i)^{(1-\pi_i)N_i}} \quad (7)$$

where π_i is the service testability [29], that is the probability that a single execution of a service fails on a test case chosen from a certain input distribution. The input distribution represents the operational profile that we assume for the service, as obtained from the operational profile of the whole system [30]. It can be estimated through random testing applied to the service [31].

Finally, the constraint (4), although not strictly necessary for the model formulation, has been introduced to improve the performance of the model solver, as it limits the search space to a maximum number N^{max} of tests.

B. Repair model

The *base model* is suited to contexts where only average values are of interest or available. In particular, the number of failures in (6) is based on the average number of invocations of S_i, therefore the reliability constraint (2) only guarantees that, averaged overall interaction scenarios, the system reliability meets the required threshold R. Hence, the choice of services resulting from the solution of the *base model* does not guarantee that the reliability constraint can be violated within a specific interaction scenario.

In this section we introduce the *repair model* that, starting from the solution of the *base model*, aims at modifying it under the constraint that the system reliability requirement must be satisfied within each interaction scenario. This model is formulated as follows:

$$\min \sum_{i=1}^{n} \Big(c_i (1 - \bar{y_i}) y_i + \sum_{j=1}^{m} c_{ij} (1 - \bar{x_{ij}}) x_{ij} \Big) \quad (8)$$

$$\forall l = 1 \dots k, \qquad \prod_{i=1}^{n} e^{-\hat{f_i^l}} \geq R \quad (9)$$

$$\forall i = 1 \dots n, \qquad y_i + \sum_{j=1}^{m} x_{ij} = 1 \quad (10)$$

$$\forall i = 1 \dots n, \qquad N_i \leq (N^{max} - N_i^{add}) y_i \quad (11)$$

The model considers the fact that the solution of the *base model*, here represented by the vectors $(\bar{y_i}, \bar{x_{ij}})$, led to incur in a certain cost. The goal of this model is to minimize the additional cost for "repairing" the original solution in order to satisfy the reliability constraints for all interaction scenarios.

The cost (8) will be zero if the same solution of the *base model* is chosen, whereas additional costs will appear if different solutions for certain services S_i have to be considered in order to reach the reliability threshold. The reliability constraint (2) of the *base model*, in fact, has been here expanded in a set of constraints (9) that force the required reliability R to be achieved in all k interaction scenarios. $\hat{f_i^l}$ here represents the number of failures of service S_i within the scenario l, and it is expressed as follows:

$$\hat{f_i^l} = inv_i^l \cdot (\hat{\lambda_i} y_i + \sum_{j=1}^{m} \mu_{ij} x_{ij}) \quad (12)$$

Here inv_i^l is the number of invocations of S_i within the l-th scenario, and $\hat{\lambda_i}$ represents the failure probability of the in-house developed service as resulting from repairing actions that basically consist in more testing, namely:

$$\hat{\lambda_i} = 1 - \frac{1 - \pi_i}{(1 - \pi_i) + \pi_i (1 - \pi_i)^{(1-\pi_i)(\bar{N_i}\bar{y_i} + N_i^{add})}} \quad (13)$$

where $\bar{N}_i$ was the original amount of testing allocated to the (possibly selected) in-house service S_i in the *base model* solution, and N_i^{add} represents the additional amount of testing that the solution of this model possibly suggests. Note that also the number N_i of test in (5) has consistently been replaced with $(\bar{N}_i + N_i^{add})$.

Finally, the constraint (11) has the same role as in the *base model* for limiting the search solution space.

C. Robust model

In this section we introduce the *robust model* that takes a more radical approach to the problem of single scenario reliability violations, in that it does not consider the solution provided by the *base model*, whereas it rather searches for new solutions that guarantee the satisfaction of all reliability constraints. The model is formulated as follows:

$$\min \sum_{i=1}^{n}(c_i y_i + \sum_{j=1}^{m} c_{ij} x_{ij}) \tag{14}$$

$$\forall l = 1 \dots k, \qquad \prod_{i=1}^{n} e^{-f_i^l} \geq R \tag{15}$$

$$\forall i = 1 \dots n, \qquad y_i + \sum_{j=1}^{m} x_{ij} = 1 \tag{16}$$

$$\forall i = 1 \dots n, \qquad N_i \leq N^{max} y_i \tag{17}$$

In other words, it is a *base model* where the reliability constraint has been expanded as in the *repair model*. The parameter f_i^l here represents the number of failures of service S_i within the scenario l without possibility of repair, and it is expressed as follows:

$$f_i^l = inv_i^l \cdot (\lambda_i y_i + \sum_{j=1}^{m} \mu_{ij} x_{ij}) \tag{18}$$

D. Stochastic model

Since we would expect that the solutions provided by the *robust model* are not so cheap, we introduce a final model aimed at providing cheaper solutions while satisfying the reliability constraints for all interaction scenarios. For this goal, we have adopted the stochastic optimization paradigm [32] that allows the explicitly embedding of probabilities within an optimization model. This paradigm allows us to separate the interaction scenarios also in the cost minimization function and, at the same time, to introduce repair actions based on service interaction patterns. In particular, we figure out the possibility of improving the system reliability by decreasing the number of invocations of a service within a scenario.

For each scenario l we define:

- p^l: probability of triggering the scenario, which comes from the user operational profile;
- δ_i^l: a new model variable that represents the number of invocations of S_i saved in the l-th scenario;
- C_i^l: unitary repairing cost for each failure of S_i saved in the l-th scenario.

The *stochastic model* is formulated as follows:

$$\min(\sum_{i=1}^{n} c_i y_i + \sum_{j=1}^{m} c_{ij} x_{ij}) + \sum_{l=1}^{k} p^l \sum_{i=1}^{n} (C_i^l \delta_i^l) \tag{19}$$

$$\prod_{i=1}^{n} e^{-\tilde{f}_i^l} \geq R \qquad \forall l = 1 \dots k \tag{20}$$

$$y_i + \sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i = 1 \dots n \tag{21}$$

$$\delta_i^l \leq inv_i^l \qquad \forall i = 1 \dots n, \forall l = 1 \dots k \tag{22}$$

$$\forall i = 1 \dots n, \;\; N_i \leq N^{max} y_i \tag{23}$$

The leftmost term of the cost function (19) is the same as for the *base model*. The rightmost term $\sum_{l=1}^{k} p^l \sum_{i=1}^{n} (C_i^l \delta_i^l)$ represents the repairing cost, that is the total cost for removing δ_i^l invocations from service i and scenario l, over all services and scenarios. Here the stochastic optimization paradigm allows introducing a weighted sum over all scenarios, where each scenario contribution to this term is weighted with its probability p^l to be triggered.

The reduction of the number of invocations obviously leads to decrease the average number of service failures as follows:

$$\tilde{f}_i^l = (inv_i^l - \delta_i^l)(\lambda_i y_i + \sum_{j=1}^{m} \mu_{ij} x_{ij}) \tag{24}$$

The remaining model constraints are obvious.

The model illustrated above is based on the possibility to modify the interaction patterns among services, namely it is based on repair actions that concern the system design. Several other options can be conceived for avoiding service failures, such as improving the reliability of involved services. Even though these actions are more difficult to be realized in practice (e.g. the reliability of provided services can only be improved by wrapping the service with fault-tolerant mechanisms), for sake of completeness we formulate here below an extended stochastic model that, however, will not be used for experimentation.

Foremost, for each service S_i, we define these additional parameters:

- δ_i: a new model variable that represents the amount of improvement of the in-house developed S_i failure rate;
- C_i: unitary repairing cost for the in-house developed S_i;
- δ_{ij}: a new model variable that represents the improvement of the failure rate of the provided S_i;
- C_{ij}: unitary repairing cost for the provided S_i.

The extended stochastic model can then be formulated as follows:

$$\min(\sum_{i=1}^{n} c_i y_i + \sum_{j=1}^{m} c_{ij} x_{ij}) +$$

$$+(\sum_{i=1}^{n} C_i \delta_i y_i + \sum_{j=1}^{m} C_{ij} \delta_{ij} x_{ij}) + \sum_{l=1}^{k} p^l \sum_{i=1}^{n} (C_i^l \delta_i^l) \quad (25)$$

$$\prod_{i=1}^{n} e^{-\hat{f}_i^l} \geq R \qquad \forall l = 1 \dots k \quad (26)$$

$$y_i + \sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i = 1 \dots n \quad (27)$$

$$\delta_i \leq \lambda_i, \delta_{ij} \leq \mu_{ij} \qquad \forall i = 1 \dots n, \forall j = 1 \dots m \quad (28)$$

$$\delta_i^l \leq inv_i^l \qquad \forall i = 1 \dots n, \forall l = 1 \dots k \quad (29)$$

where:

$$\hat{f}_i^l = (inv_i^l - \delta_i^l)((\lambda_i - \delta_i) y_i + \sum_{j=1}^{m} (\mu_{ij} - \delta_{ij}) x_{ij})$$

It is evident the role of new variables δ_i and δ_{ij} that work on the failure rates of available service alternatives.

E. A greedy approach

The above non-linear optimization models can be difficult to solve and sometime a heuristic solution can be preferred. Therefore we introduce in this section a greedy approach for the service selection that follows the same criteria of the optimization models (i.e. low cost and high reliability) but it is based on the following lightweight heuristic algorithm:

- For each service S_i
  - identify j such that $\frac{c_{ij}}{\mu_{ij}}$ is the minimum value;
  - carry out $\tilde{N}_i = \min\{N_i | \lambda_i \leq \mu_{ij}\}$;
  - if $c_i(\tilde{N}_i) > c_{ij}$
    then $x_{ij} = 1; N_i = 0$
    else $y_i = 1; N_i = \tilde{N}_i$;

In practice, among all possible alternatives for a service S_i, the one that achieves the minimum failure rate at the minimum cost is selected, including the in-house implemented one. In our experimentation, we compare the results of the optimization models with the ones obtained by this greedy approach.

F. Random service selection

Finally, as a bottom-line comparison, we introduce a random solution to our problem. It is obviously a nonsense for practical reason, but it allows quantifying the improvements introduced by all previously illustrated models with respect to a dummy solution that does not require any judgement. The random selection algorithm is as follows:

- For each service S_i
  - let ψ be a random number extracted by an uniform distribution between 0 and 1;
  - if $\psi \leq 0.5$ then
    - $y_i = 1$;
    - let ϕ be a random number extracted by an uniform distribution between 0 and N^{max};
    - $N_i = \phi$;
  - else
    - let θ be a random number extracted by an uniform distribution between 1 and m;
    - $j = \theta$;
    - $x_{ij} = 1$;

In practice, first it is randomly chosen whether in-house developing the service or purchasing it from a provider. In the former case the number of tests to perform is also randomly chosen within a fixed range.

V. Experimental results

In order to show the effectiveness of the proposed approach we here illustrate the results that we have obtained from a quite extensive experimentation. For sake of result robustness, we have generated 30 instances of each model starting from a nominal one and randomly modifying several parameters within prefixed ranges. Thereafter, we have solved each instance for 10 different values of the reliability threshold R between 0.95 and 0.995. Overall, we have solved 300 instances for each optimization model.

Nominal instance parameters.

The nominal instance has been set to $n = 5$ services interacting within $k = 4$ scenarios. Table I shows the nominal values that we have considered for the model parameters related to the available provided services. Three alternatives for each service have been devised and identified as S_{ij}. In addition to these values, we have set an unique repairing cost $C_i^l = 50$ ([4]).

Table II shows the nominal values of the main parameters related to in-house developed services. In addition to these values, we have set a development cost $c_i = 1500$ per personmonth.[5]

Random generation of model instances.

Starting from the nominal values of the parameters, we have generated 30 different instances (here also called *perturbed configurations*) by randomly changing the parameters. The perturbed configuration parameters have been varied within 10% of the nominal values, with the exception of: (i) the cost of repair C_i^l that has varied within 50% of its nominal value, (ii) the personmonths for test $\#pmt$ that has varied within 20% of its nominal value.

For each perturbed configuration, we have solved the four optimization models along with the greedy and random strategies, as presented in Section IV, for R that spans from 0.95 to 0.995 by steps of 0.005.

Model solutions.

In Figure 2 we report the obtained results, where each bar indicates the cost of a model averaged over its 30 perturbed configurations.

[4] All costs are expressed in euros.

[5] The nominal instances values are available at: http://cs.unibg.it/potena/FailureRepairMitigationCosts/NominalInstance.pdf

TABLE I. NOMINAL PARAMETERS OF PROVIDED SERVICES

	Provided services	Cost c_{ij}	Failure prob. μ_{ij}	Number of invocations for scenario inv_i^l
S_1	S_{11}	200	0.001	35 25 28 12
	S_{12}	220	0.00075	
	S_{13}	240	0.00065	
S_2	S_{21}	180	0.002	7 15 21 2
	S_{22}	195	0.0009	
	S_{23}	205	0.00063	
S_3	S_{31}	430	0.0013	9 32 21 11
	S_{32}	465	0.0009	
	S_{33}	480	0.0007	
S_4	S_{41}	520	0.001	8 44 29 5
	S_{42}	524	0.0006	
	S_{43}	539	0.0002	
S_5	S_{51}	700	0.0008	12 12 33 18
	S_{52}	720	0.0005	
	S_{53}	745	0.0001	

TABLE II. NOMINAL PARAMETERS FOR IN-HOUSE DEVELOPED INSTANCES

	Size (in kloc)	Type	Testability π_i
S_1	0.1	intermediate	0.001
S_2	0.7	simple	0.0005
S_3	0.5	complex	0.0007
S_4	1.5	intermediate	0.0008
S_5	2.0	simple	0.0009

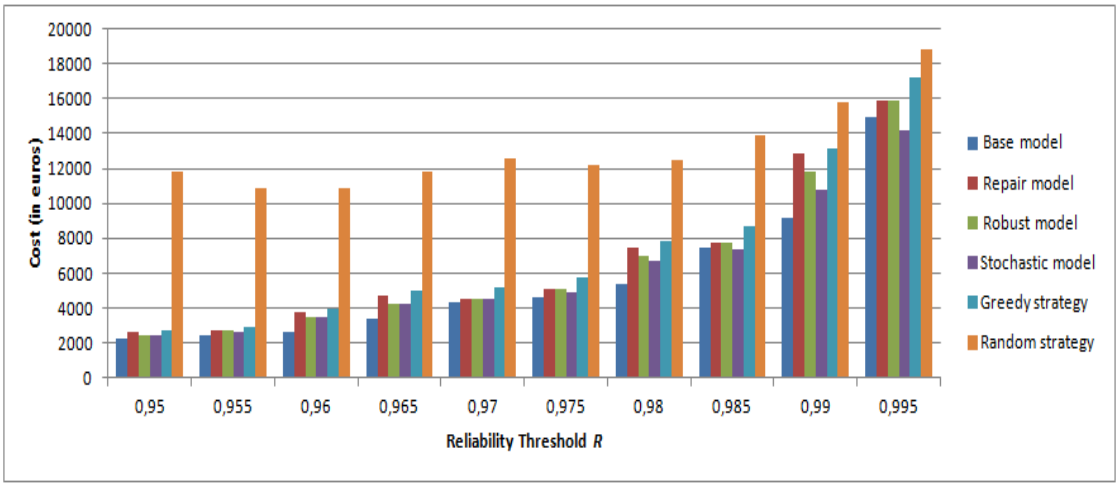


Fig. 2. System costs vs reliability thresholds.

As expected, the system cost is not decreasing (except few cases, mainly for the random strategy) while increasing the reliability threshold R. In particular, with the *base model*, the average cost is about 2285 euros for $R = 0.95$, whereas it raises to 4302 euros for $R = 0.97$.

The average costs of the *base model* are typically less than the average costs of the other models. However, the *base model* often does not provide solutions that are feasible for all scenarios, as in 97% of our cases the solutions are infeasible for at least one scenario.

On one hand, for the same value of R the average costs obtained with the *robust model* are always less than or equal to the ones obtained with the *repair model*. For example, with the *repair model*, the average cost is about 7437 euros for $R = 0.98$, whereas with the *robust model* it decreases around 6946 euros. On the other hand, for the same value of R the average costs obtained with the *stochastic model* are always less than or equal the ones obtained with the *robust model*. For example, with the *stochastic model*, for $R = 0.98$ the average costs decrease from 6946 euros about to 6752 euros.

The results highlight, in general, that the solutions of the optimization models and the greedy strategy do not show discrepancies in case of non-complex search space (i.e., for low reliability thresholds), in that the average costs of their solutions are only slightly different. On the other hand, the discrepancies become more evident as the tightness of the requirements increases. The results show that the choices of the *robust model* are better than those of the *repair model* and of the two heuristic strategies. Finally, the *stochastic model* convincingly outperforms all of them.

These assertions can be better evaluated by considering Figure 3, where a more extensive analysis of the models'

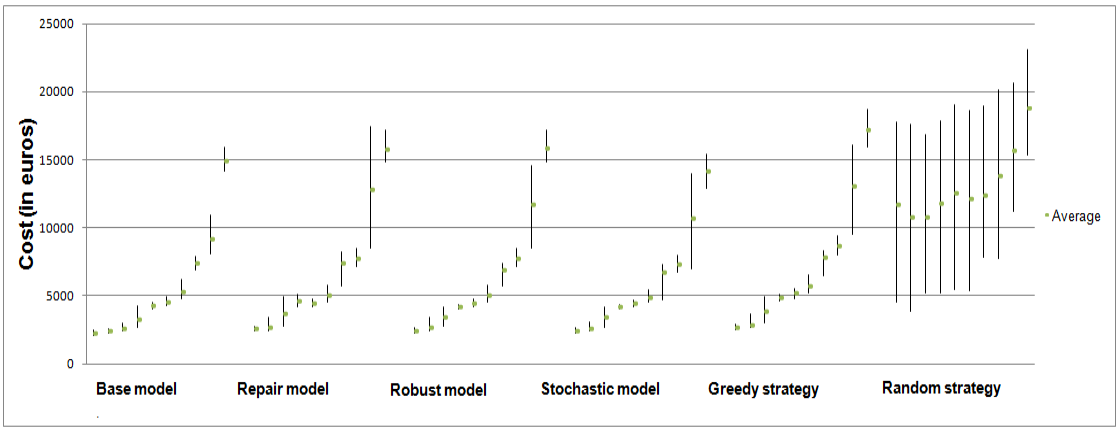


Fig. 3. A more extensive models' costs analysis vs reliability threshold

costs is performed. Each group of ten bars - corresponding to one model - refers to the model's results with 300 instances. In particular, each bar - corresponding to the model solution over the 30 perturbed configurations with a fixed value of the threshold R - reports the highest, lowest and average costs obtained.

As expected, all costs of a model (except for the random strategy) almost always increases while increasing the reliability threshold R.

By looking at the solution details, we observe for example that for a perturbed configuration with $R = 0.95$, the *base model* returns the following solution: $[S_{13}, S_{22}, S_{33}, S_{43}, S_{53}]$ with a cost equal to about 2228 euros.[6] The solution is infeasible for the second and third scenario. On one hand, the *repair model* improves this solution by suggesting for the first service the in-house developed instance with a number of tests equal to 1064, and the cost for "repairing" equal to about 569 euros. Thus, the cost increases to about 2797 euros. On the other hand, the *robust model* returns the solution: $[(S_{10}, 694), S_{23}, S_{33}, S_{43}, S_{53}]$. The cost is lower than the one of the *repair model* solution: it decreases to about 2512 euros. In fact, the *robust model* suggests to include service S_{23} instead of S_{22}, whereas the *repair model* suggests to keep S_{22} and to perform more testing on the in-house instance of S_1. The *stochastic model* provides essentially the same solution of the *robust model*. On the contrary, if we raise R to 0.985 the *robust model* provides the following solution: $[(S_{10}, 2707), (S_{20}, 2058), (S_{30}, 2990), S_{43}, S_{53}]$ with a cost equal to about 7620 euros. Instead, if we apply the *stochastic model* then the cost decreases to about 6710 euros. Such solutions differ because the *stochastic model* does not suggest an amount of testing for the in-house instances, but it improves the system reliability by decreasing the number of invocations of the services within the scenarios.

In order to compare the efficacy of the *repair model*, *robust model* and *stochastic model* and to quantify the costs of different failure repairing/mitigation actions, we have analyzed in Figure 4 the related cost gains.

[6]Each element of one solution is either a provided instance or an in-house instance. In the latter case, the name of the instance is paired with the number of test to perform on it. The in-house instance of service i is named S_{i0}.

Figure 4(a) shows the percentage gain in cost obtained for different reliability thresholds (on x-axis) by replacing the *repair model* with the *robust model* for the 30 perturbed configurations. Specifically, each bar corresponds to one reliability threshold value R, and it contains the highest, lowest and the average relative profits. For a perturbed configuration and a reliability threshold R, we have estimated the profit as follows: $(Cost_{REP}(s) - Cost_{ROB}(s'))/Cost_{REP}(s)$, where s and s' represent the solutions returned by the *repair model* and the *robust model*, respectively, and $Cost_{REP}(s)$ and $Cost_{ROB}(s')$ represents their costs.

These results confirm our hypothesis on the fact that the costs of the *robust model* are always less than or equal to the ones obtained with the *repair model*.

On one hand, the *repair model* starts from the solution obtained in the *base model* and tries to improve it (see Section IV-B). On the other hand, the *robust model* takes a more radical approach to the problem of single scenario reliability violations, in that it does not consider the solution provided by the *base model*, whereas it rather searches for new solutions that guarantees the satisfaction of all reliability constraints (see Section IV-C).

In a similar way, in Figure 4(b) we compare the results of the *robust model* and the *stochastic model*. The figure shows the percentage gain in cost obtained for different reliability thresholds (on x-axis) by replacing the *robust model* with the *stochastic model*. These results show how the solutions provided by the *robust model* could be improved in terms of costs: cheaper solutions may be found by using the stochastic optimization paradigm.

As expected, all cost gains increase while increasing the reliability threshold (R). For example, with R equals to 0.985, the average profit is about 0.0495, whereas if R increases to 0.99, then the average profit increases to about 0.118. This confirms the fact that, in general, the solutions of the *robust model* and the *stochastic model* do not show discrepancies in case of non-complex search space (i.e., for low reliability thresholds). On the contrary, the effectiveness of the *stochastic model* becomes more evident as the tightness of the requirements increases.

Moreover, the different behavior of the *stochastic model*

can also be observed by fixing a value on the x-axis and observing the values on two figures while increasing the threshold R. For example, with R equals to 0.99, the average profit of the *robust model* with respect to the *repair model* is about 0.0641, whereas the average profit of the *stochastic model* with respect to the *robust model* is about 0.118.

Figure 5 shows how the *robust model* improves the accuracy of the solutions of the *base model* in terms of system reliability.

We recall that the *base model* is suited to contexts where only average values are of interest. In particular, the model estimates the number of failures of the service i as a function of the average number of invocations (i.e., inv_i). Hence, the choice of services resulting from the solution of the *base model* does not guarantee that the reliability constraint can be satisfied within a specific interaction scenario. On the contrary, the *robust model* forces the required reliability R to be achieved in all interaction scenarios by estimating the number of failures of the service i as a function of the average number of invocations of the service within the scenarios (i.e., inv_i^l).

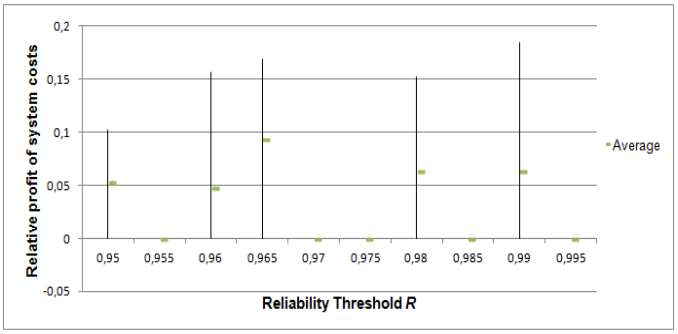
Figure 5 shows the results for the 30 perturbed configurations of the *robust model*. The x-axis represents the costs averaged over the 30 perturbed configurations for each value of the reliability threshold. In this case, for each solution of the *robust model* we have carried out by hand the system reliability as a function of the average number of service invocations.

Each bar corresponds to one reliability threshold value R, and it contains the highest, lowest and the average system reliability, obtained by considering the average service failures. System reliability (and the average costs) increases while increasing the reliability threshold R. For example, with the average costs about to 4504 euros - corresponding to R equal to 0.97 -, the average reliability is about to 0.973331, whereas if the costs increases about to 11805 euros - corresponding to R equal to 0.99, the average reliability increases about to 0.991327.

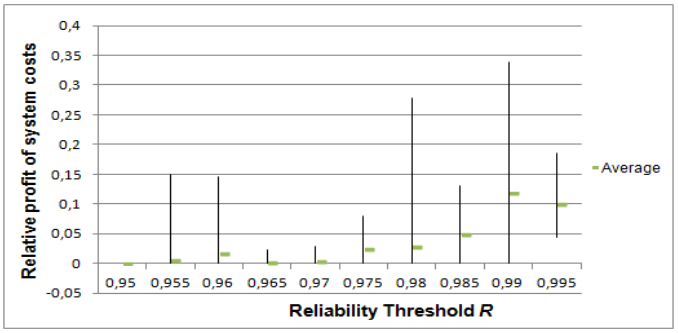
This result confirms that strengthening the reliability constraints is a key factor for a trustworthy prediction of the reliability of SBSs, and leads an optimization model to a more precise (and less pessimistic) estimation of the SBS reliability.

VI. Conclusions

In this paper we have presented an approach for the definition of SBSs obtained by a combination of both provided loosely-coupled services and in-house developed services, while satisfying costs and reliability requirements. Specifically, we have presented a service selection method based on the definition of a set of optimization models whose solution can give insights on the service composition that best fit the requirements considering an explicit cost model and the possibility to define repair actions to improve the system reliability. This approach can help software architects in the decision-making process of assembling architectures satisfying quality requirements. We have performed an extensive experimentation to show the efficacy of our optimization models to quantify the costs of different failure repairing/mitigation actions in different contexts.



(a) Relative Profit of the *robust model* with respect to the *repair model*



(b) Relative Profit of the *stochastic model* with respect to the *robust model*

Fig. 4. Comparing the relative profit of the Robust and Stochastic models

In order to validate the effectiveness of the models, we intend to experiment them on real world case studies by considering realistic model parameter values. Beside this, we also plan to experiment meta-heuristic techniques for searching wider solution spaces that would be induced by large size case studies.

As future work, we also plan to extend the concept of mitigation/repair actions to other options, such as the ones formulated in the extended stochastic model. Of course, these additional options should always be expressed in closed forms (or by using alternative techniques, like discrete event simulation [33]) to be embedded into optimization models.

Acknowledgement

This work has been partially supported by the IDEAS-ERC Project SMScom, by the Italian Ministry of Research within PRIN project "GenData 2020" (2010RTFWBH), and by the VISION European Research Council Starting Grant (ERC-240555).

References

[1] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola, "MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems," *IEEE Trans. Software Eng.*, vol. 38, no. 5, pp. 1138–1159, 2012.

[2] V. Cortellessa, F. Marinelli, and P. Potena, "An optimization framework for "build-or-buy" decisions in software architecture," *Computers & OR*, vol. 35, no. 10, pp. 3090–3106, 2008.

[3] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS Management and Optimization in Service-Based Systems," *IEEE Transactions on Software Engineering*, vol. 37, pp. 387–409, 2011.

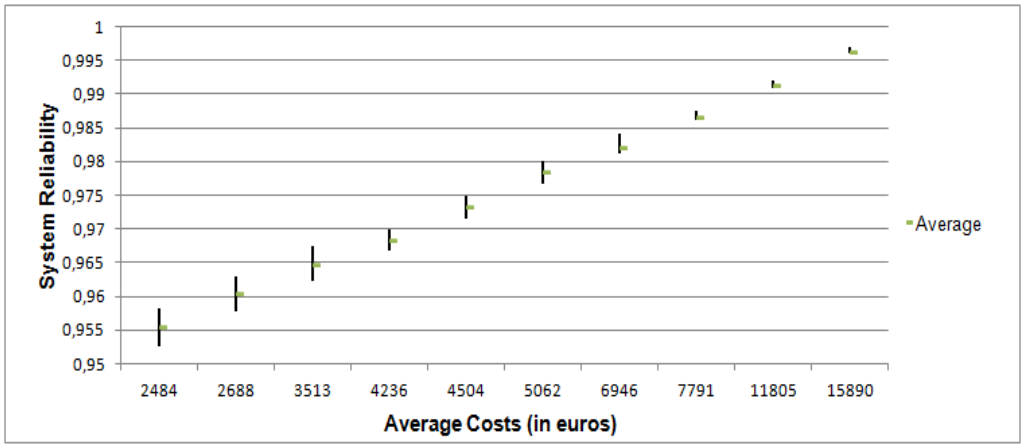


Fig. 5. Reliability for the *robust model*

[4] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software Architecture Optimization Methods: A Systematic Literature Review," *IEEE Trans. Software Eng.*, vol. 39, no. 5, pp. 658–683, 2013.

[5] J. Cardoso, "Complexity analysis of BPEL Web processes," *Software Process: Improvement and Practice*, vol. 12, no. 1, pp. 35–49, 2007.

[6] M. Marzolla and R. Mirandola, "Performance Prediction of Web Service Workflows," in *Proc. QoSA '07*, ser. LNCS, vol. 4880. Springer, 2007, pp. 127–144.

[7] G. Canfora, M. Di Penta, R. Esposito, and M. Villani, "A framework for QoS-aware binding and re-binding of composite web services," *J. Syst. Softw.*, vol. 81, no. 10, pp. 1754–1769, 2008.

[8] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnamam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, May 2004.

[9] D. Ardagna and R. Mirandola, "Per-flow Optimal Service Selection for Web Services Based Processes," *J. Syst. Softw.*, vol. 83, no. 8, 2010.

[10] S. Gokhale and J. Lu, "Performance and Availability Analysis of an E-Commerce Site," in *Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International*, vol. 1, sept. 2006, pp. 495 –502.

[11] N. Janevski and K. Goseva-Popstojanova, "Session Reliability of Web Systems Under Heavy-Tailed Workloads: An Approach based on Design and Analysis of Experiments," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, p. 1, 2013.

[12] N. Sato and K. Trivedi, "Accurate and efficient stochastic reliability analysis of composite services using their compact Markov reward model representations," in *Services Computing, 2007. SCC 2007. IEEE International Conference on*, july 2007, pp. 114 –121.

[13] K. Goseva-Popstojanova, A. D. Singh, S. Mazimdar, and F. Li, "Empirical Characterization of Session-Based Workload and Reliability for Web Servers," *Empirical Software Engineering*, vol. 11, no. 1, pp. 71–117, 2006.

[14] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 35–44.

[15] C.-Y. Huang, S.-Y. Kuo, and M. R. Lyu, "An Assessment of Testing-Effort Dependent Software Reliability Growth Models," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 198–211, 2007.

[16] O. Berman and M. Cutler, "Resource allocation during tests for optimally reliable software," *Computers & OR*, vol. 31, no. 11, pp. 1847–1865, 2004.

[17] C.-Y. Huang and J.-H. Lo, "Optimal resource allocation for cost and reliability of modular software systems in the testing phase," *Journal of Systems and Software*, vol. 79, no. 5, pp. 653 – 664, 2006.

[18] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and S. Carrière, "The Architecture Tradeoff Analysis Method," in *ICECCS*, 1998, pp. 68–78.

[19] N. Esfahani, E. Kouroshfar, and S. Malek, "Taming uncertainty in self-adaptive software," in *SIGSOFT FSE*, 2011, pp. 234–244.

[20] C. Ghezzi, L. Pinto, P. Spoletini, and G. Tamburelli, "Managing Non-functional Uncertainty via Model-Driven Adaptivity," in *Proc. of ICSE 2013*, 2013.

[21] I. Meedeniya, A. Aleti, and L. Grunske, "Architecture-driven reliability optimization with uncertain model parameters," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2340–2355, 2012.

[22] D. Doran, M. Tran, L. Fiondella, and S. S. Gokhale, "Architecture-based Reliability Analysis With Uncertain Parameters," in *SEKE*, 2011, pp. 629–634.

[23] W. Wiesemann, R. Hochreiter, and D. Kuhn, "A Stochastic Programming Approach for QoS-Aware Service Composition," in *CCGRID*, 2008, pp. 226–233.

[24] I. Sommerville, *Software engineering (9th ed.)*. Addison Wesley, 2010.

[25] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter, "Branching and bounds tightening techniques for non-convex MINLP," *Optimization Methods and Software*, vol. 24, no. 4-5, pp. 597–634, 2009.

[26] H.-W. Jung, C.-S. Chung, and K. O. Lee, "Selecting Optimal COTS Products Considering Cost and Failure Rate," in *Fast Abstracts of ISSRE*, 1999.

[27] K. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications, 2nd Edition*. Wiley-Interscience, 2001.

[28] A. Bertolino and L. Strigini, "On the Use of Testability Measures for Dependability Assessment," *IEEE Trans. Software Eng.*, vol. 22, no. 2, pp. 97–108, 1996.

[29] J. Voas and K. Miller, "Software testability: the new verification," *Software, IEEE*, vol. 12, no. 3, pp. 17–28, 1995.

[30] D. Hamlet, D. Mason, and D. Woit, "Theory of software reliability based on components," in *Proceedings of the 23rd International Conference on Software Engineering*, ser. ICSE '01, 2001, pp. 361–370.

[31] W. Abdelmoez, D. E. M. Nassar, M. Shereshevsky, N. Gradetsky, R. Gunnalan, H. H. Ammar, B. Yu, and A. Mili, "Error Propagation In Software Architectures," in *IEEE METRICS*, 2004, pp. 384–393.

[32] W. Klein Haneveld and M. van der Vlerk, "Stochastic integer programming: General models and algorithms," *Annals of Operations Research*, vol. 85, no. 0, pp. 39–57, 1999.

[33] I. D. Lins and E. L. Droguett, "Redundancy allocation problems considering systems with imperfect repairs using multi-objective genetic algorithms and discrete event simulation," *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 362 – 381, 2011.