Submission date: 08/10/2020
Resubmission date: 17/01/2021
Resubmission date: 21/03/2021
Camera ready submission date: 13/04/2021

1st round notification: 07/12/2020
2nd round notification: 22/02/2021
3rd round notification: 05/04/2021
Available online: 14/05/2021
Publication date: 14/05/2021

Section: regular article

Comparison and Choice of Computational Architectures Based on Cost-Value Approach

Elivaldo Lozer Fracalossi Ribeiro[1,2], Daniela Barreiro Claro[2], Rita Suzana Pitangueira Maciel[3]

[1]Centre for Environmental Science, Federal University of Southern Bahia, BR-367, Km 10, 45.810-000, Porto Seguro, Bahia, Brazil

[2]FORMAS Research Group, Computer Science Department (DCC), IME, Federal University of Bahia, s/n Adhemar de Barros Ave., Ondina, 40.170-110, Salvador, Bahia, Brazil

[3]MATiSsE Research Group, Federal University of Bahia, s/n Adhemar de Barros Ave., Ondina, 40.170-110, Salvador, Bahia, Brazil

elivaldolozer@ufsb.edu.br, {dclaro, rita.suzana}@ufba.br

Abstract. *Software engineers make use of several computational architectures (CA) to host an application, such as desktop, web, and cloud computing architectures. As the requirements vary according to the desired CA, developers may face two problems: determining which requirement better fulfills a CA and determining which CA fulfills a given set of requirements. This paper presents a new approach based on the Cost-Value Approach (CVA). We have slightly modified the CVA method (SCVA method) and also developed a new method for choosing the most appropriate CA (MMACA method). Our results provide a set of requirements ordered by priority for each CA. Finally, we discuss the current and most appropriate CA for a real project solution.*

Keywords. *Requirements engineering; Requirements elicitation; Non-functional requirements; Computational architectures; Cost-value approach*

1. Introduction

Requirements describe information about features, services, functionalities, and restrictions of a software system [Bourque and Fairley 2014, van Lamsweerde 2009]. In the software industry, a requirement is defined as a formal definition of a system function, a high-level service description, or a system restriction [van Lamsweerde 2009]. How to meet stakeholders' needs and how to manage information during a software project are fundamental issues in the area [Olsson et al. 2019]. In recent years, requirements have become increasingly dynamic and heterogeneous due to the expansion of the web and social media [Zampoglou et al. 2016].

Requirements can be classified into functional and non-functional. Functional requirements are functions that a system might perform. For instance, in the academic

Cite as: Ribeiro, E. L. F., Claro, D. B. & Maciel, R. S. P. (2021). Comparison and Choice of Computational Architectures Based on Cost-Value Approach. iSys: Revista Brasileira de Sistemas de Informação (Brazilian Journal of Information Systems), 14(1), 56-88. DOI: 10.5753/isys.2021.968

domain, a secretary can list all undergraduate students, and a lecturer can book a laboratory. Non-functional requirements define behavioral conditions, constraints, or quality attributes for software. For instance, non-functional requirements can be the availability of a system, i.e., 99% available on the web, and database queries should take less than three seconds to perform [Bourque and Fairley 2014, van Lamsweerde 2009]. A successful information system is the one in which all software requirements are fulfilled [Belout and Gauvreau 2004, Kaur and Aggrawal 2013]. However, this is not a trivial task.

Whereas software components fulfill almost all functional requirements, some non-functional requirements are dependent on the computational architecture (CA) that hosts the information system. CAs are models for organizing physical and logical resources, providing a standard of development process [Patterson and Hennessy 2013]. Examples of CAs are desktop, web, cloud, fog, and mobile. Given this diversity, the choice of the most appropriate CA may be complex because it depends on a software requirement, such as availability, scalability, access mode, among others [Bessa et al. 2016].

Although similar, computational architectures and software architectures are different concepts. Computational architectures refer to the infrastructure (e.g., operating system, network platform, data storage, among others) in which the system or application is hosted. Software architectures concern system design decisions, software elements, and how these elements relate to each other [Taylor et al. 2010]. A computational architecture can host software with different software architectures because a computational architecture may be more suitable for a given software architecture standard.

The requirements engineering (RE) process comprises a set of activities to discover, analyze, validate, and manage a system requirement. Authors in [van Lamsweerde 2009] report that there is no single RE process suitable for all organizations. Moreover, they state that a generic process consists of: deciding whether the system is useful (feasibility study); discovering, ranking, and negotiating requirements (elicitation and analysis step); organizing the requirements to be implemented (specification step); and, checking whether the requirements fulfill the customer's desires (validation step). RE is one of the most important activities in information system development [Xu et al. 2012].

Requirements elicitation and analysis is a step which manages the requirements from the discovery to the implementation. It includes classification, prioritization, negotiation, and specification. Prioritization and negotiation activities are concerned with prioritizing requirements and resolving requirements conflicts through negotiation [van Lamsweerde 2009].

Considering the steps of *requirements analysis and elicitation*, the hierarchy of characteristics is not an exclusive process of software engineering [van Lamsweerde 2009]. For example, when buying a smartphone, a purchaser considers the most important features before buying the desired model. As each model (of smartphone) has a set of characteristics, the purchaser (end-user) has to identify the product that best meets his/her requirements. The choice is not trivial because it is based on a set of features, not just one. If the decision were based solely on the camera's resolution,

choosing the highest resolution model would be enough. However, the choice is made more difficult when the price, camera resolution, screen size, internal storage, and battery life all have to be taken into consideration.

Such problems frequently occur in modern information systems development as the number of requirements is often high. The requirements prioritization step solves conflicts among project stakeholders. The most important requirement for a customer is usually the cost to implement immediately. Additionally, software is developed incrementally, and prioritization techniques follow such increments [Ruhe 2005, Pegoraro and Paula 2017, dos Santos et al. 2016, Riegel and Doerr 2015].

Some techniques used to rank the requirements and to choose the most appropriate CA for an information system include the cost-value approach (CVA) [Karlsson and Ryan 1997], the analytic hierarchy process (AHP), numerical assignment, the 100-dollar test, the top-ten requirements, binary search tree, and bubble sort [van Lamsweerde 2009, Karlsson et al. 1998]. Among these techniques, AHP assists in complex decision-making, and it has been developed to analyze variables that are difficult to compare [Saaty 1980]. While AHP provides a hierarchical classification of requirements (e.g., ranking the cost in descending order), CVA implements a two-dimension approach to display the requirements' value against their cost and value[Karlsson and Ryan 1997]. The CVA process consists of applying the AHP method twice: one from a cost perspective and the other from the value perspective. In this paper, we state that 'cost' is the effort required to fulfill a requirement, and 'value' is the relevance of a requirement for a project [Karlsson and Ryan 1997]. In other words, 'cost' represents the effort exerted by the developer in working time to develop a given requirement, and 'value' represents the ease offered by a CA to fulfill a given requirement.

Given a software development project, our goals are: (i) to determine which non-functional requirements should receive more (or less) priority in each CA based on our slightly modified CVA method (SCVA method), and (ii) to determine which CA better fulfills a given set of requirements based on our method for choosing the most appropriate CA (MMACA method). The most appropriate CA is determined by our MMACA method based on a set of non-functional requirements. While several studies about CA focus from the perspective of management [Kazman et al. 1996, Christensen 2009, Huang et al. 2013], our focus is from the developer perspective.

Our approach compares desktop, web, and cloud architectures (i) to list the essential requirements and (ii) to determine the most appropriate CA given a set of requirements. We choose these three architectures to compare consolidated CAs (desktop and web) with a new CA (cloud). Briefly, a desktop CA runs software locally on a computer device; a web CA runs software over the Internet from a remote server as though local software; a cloud CA runs software as a service with rapid allocation/release and minimal effort from the service provider [Beaty et al. 2009, Voda 2014, Mell and Grance 2011].

Our SCVA method is applied to determine which requirement is better fulfilled by a CA, regardless of the project. Given a set of requirements as input, our method determines which requirements are relevant for desktop, web, and cloud CAs. Our SCVA method applies the CVA method, which, in turn, applies the AHP method twice (for cost

and value). We propose a generic set of requirements and therefore our method groups these requirements into two ways: stacked column charts and cost-value diagrams. The first analysis compares the aspects (cost and value) individually for each CA, and the second analysis combines both aspects to describe each requirement's priority based on both cost and value aspects. We call our method slightly modified because we apply the CVA method by development stages (planning, development, test, and operation), and we present the results into two ways (stacked column charts and cost-value diagram). In contrast, the CVA method [Karlsson and Ryan 1997] prioritizes requirements without distinguishing the development stage, and it presents a single analysis: a cost-value diagram.

Our MMACA method determines the most appropriate CA that best fulfills the information system requirements. Given a set of requirements as input, the method determines which CA better fulfills the input. Our MMACA method applies the SCVA method, calculates the ratio between cost and value for input requirements, and estimates the most appropriate CA (desktop, web, or cloud). The MMACA method considers the generic set of requirements built by the SCVA method.

Both methods require an initial set of requirements as input. Since there is no consensus in the literature, we built a set of requirements to evaluate our methods. We generated a set of 30 requirements based on bibliographical research, according to [Kitchenham and Charters 2007]. As our study focuses on the developer's viewpoint, our initial set of requirements was validated by a group of developers, generating a set of 25 requirements. This set is the input of the SCVA method, and any subset of this set is the input for the MMACA method. This subset may vary because it represents the most important requirements for an information system.

We performed an exploratory study to verify the feasibility of SCVA and MMACA. For the SCVA method, the developers evaluated the set requirements twice (once for cost and the other for value) and the SCVA method hierarchized the requirements by each CA.

For the MMACA method, we performed two evaluations. In the first evaluation, ten developers chose a subset of the requirements, and the method indicated the most appropriate CA. In the second evaluation, four developers from a real project created a single subset of the set of requirements, and the method indicated the most appropriate CA for this real information system.

Our results show the effectiveness and efficiency of our methods. Both SCVA and MMACA make potential contributions to the state of the art since they present novel methods for prioritizing requirements and choosing a CA.

The remaining of this paper is organized as follows: Section 2 presents an overview of CAs and requirements prioritization; Section 3 describes our related works; Section 4 presents our set of requirements and our methods (SCVA and MMACA); Section 5 describes validation our approach; Section 6 discusses our results; Section 7 presents some threats to validity; and finally, Section 8 presents our conclusions and future works.

2. Background

This section presents some background research on the topics supporting this work, starting with the definitions and details comparing the three CAs. Next, we examine requirements prioritization, focusing on the two techniques used: Analytic Hierarchy Process (AHP) and Cost-Value Approach (CVA).

2.1. Computational Architectures (CA)

Computational architectures (CA) are ways of structuring and organizing physical and logical devices. Architectures may differ in terms of memory management, data storage, processing capacity, and infrastructure [Patterson and Hennessy 2013].

In recent decades, application model sharing has evolved. Also known as a multi-user model, desktop architecture is user-focused because it hosts applications on a particular computer for each user. In this architecture, applications are limited by local hardware. Since applications are installed locally, the addition of non-functional requirements may require upgrading local hardware [Beaty et al. 2009, Wang et al. 2009]. Although not recent, the choice for desktop architecture is justified for some reasons. First, the cost varies depending on the CA and requirements so there is no guarantee that applications for one particular architecture will always be the cheapest [Khajeh-Hosseini et al. 2010]. Second, companies with desktop applications would incur a high cost to migrate their applications to another architecture. For this reason, legacy systems collaborate with the use of this architecture [Khajeh-Hosseini et al. 2010, Maenhaut et al. 2016]. Third, migrating desktop applications to another architecture can cause organizational changes and make small business operations impossible [Khajeh-Hosseini et al. 2010, Armbrust et al. 2009]. Fourth, some web and cloud computing challenges (e.g., security, elasticity, and privacy) can be resolved on desktop applications more quickly [Khajeh-Hosseini et al. 2010, Armbrust et al. 2009].

With the Internet and the need to increase availability, web architecture has emerged which focuses on applications. The model is no longer multi-user (one computer for several users), but it becomes multi-application (one user starts to use several applications) in a client-server framework. The user accesses the application through client software (e.g., a browser) and part of the required storage and processing is made available by the application server [Wang et al. 2009, Voda 2014].

More recently, cloud computing has enabled on-demand access to a set of resources (e.g., networks, servers, storage, applications, and services), with rapid allocation/release, and with minimal effort from the service provider [Mell and Grance 2011]. This model can be understood as an advance of the web model [Erdogmus 2009, Pallis 2010]. The main features of cloud computing are: (i) the user can access new resources according to their needs; (ii) resources can be accessed from any device; (iii) similar resources are grouped; (iv) resources can be allocated and released automatically; and (v) payment is based on usage [Mell and Grance 2011].

Each CA implements a set of requirements to provide a structure to host applications. For example, while desktop architecture provides requirements for access to client resources and integration with local hardware, web architecture provides portability and

hardware consumption. On the other hand, cloud computing architecture provides availability and scalability. Because the requirements vary according to each CA, hierarchizing requirements considering multiple architectures is required.

2.2. Requirements Prioritization

Requirements can be prioritized according to one or more aspects. An aspect here is a project property used as a focus for comparison (e.g., value, cost, time, and risk). Coming back to the smartphone scenario, it would be like choosing a phone based on the best camera resolution. However, considering the resolution of the camera and the size of the screen, the resource that requires the most development effort (cost) may not be the most important (value).

Briefly, a requirement prioritization technique ranks a set's requirements according to some criteria. This prioritization can be done on the (i) ordinal, (ii) ratio, or (iii) absolute scale. Considering the cost aspect and the ordinal scale, requirements are analyzed one by one, and they are classified by cost. In addition to classifying, the ratio scale measures how expensive one requirement is compared to another (scale generally in percentage). Finally, the absolute scale hierarchy is performed based on an absolute value (number of hours to implement each requirement, for example) [Achimugu et al. 2014]. A complete list of techniques is available in [Achimugu et al. 2014]. Some prioritization techniques are described as follows.

Numerical Assignment is a technique that classifies the requirements into different groups (usually three or five). Each group represents a level in the analyzed scale, e.g., "critical, standard, and optional". A disadvantage of this technique is that prioritization considers only the group. Thus, requirements in a group have the same priority [Garg and Singhal 2017].

In the *100-dollar Test* technique, each participant receives a certain amount of imaginary points (usually 100, considered as money or time) to allocate to the requirements. The result of each requirement will be between zero (0% important) and one (100% important). The problem of ranking many requirements can be mitigated with several points multiples of 100 [Santos et al. 2016, Ahuja et al. 2016].

In *Top-ten* requirements technique, given the set of all requirements, each stakeholder chooses their top ten, without assigning an order of priority to the requirements chosen. This technique is appropriate when there are several stakeholders involved, and they have equal importance [Achimugu et al. 2014].

The *Binary Search Tree* technique employs the structure of a binary tree and consists of the following: (i) initially, one of the requirements is positioned as the root node; then, (ii) each of the rest of the requirements is allocated in a tree, comparing each new requirement with the requirements already presented in the tree; finally, (iii) a tree with prioritized and balanced requirements is obtained [Bhandari and Sehgal 2014].

Bubble Sort organizes the requirements in a hierarchical array, and they are compared two-by-two. Thus the highest requirement (the most relevant, for example) is taken to the top (or highest position) of the array [Bhandari and Sehgal 2014].

The following subsections explain in detail both the requirements prioritization techniques used in our approach: AHP and CVA.

2.2.1. Analytic Hierarchy Process

AHP process can be explained in steps [Karlsson et al. 1998, Garg and Singhal 2017, Bhandari and Sehgal 2014, Dorado et al. 2014, Chung et al. 2006, Agarwal et al. 2013, Chaudhary et al. 2016, Saaty 1987]:

1. define the goal, variables, and comparison aspect
2. set up n requirements in an n x n matrix (each requirement will be one row and one column)
3. compare all requirements two-by-two according to a quantitative specific scale (presented in Table 1, detailed by [Saaty 1987])
4. calculate the eigenvalues of the matrix with the average of the columns based on the calculated eigenvalues, each requirement receives a criterion value, and
5. finally, the consistency rate (CR) is calculated and, if inconsistent (this is, if CR $>$ 10% [Saaty 1987]), then repeat steps 3 to 5.

Table 1. Quantitative specific scale for comparison of two requirements (i and j) in AHP process

Comparison value	Interpretation
1	i and j are of equal value
3	i has a moderately higher value than j
5	i has a strongly higher value than j
7	i has a very strongly higher value than j
9	i has an extremely higher value than j
2, 4, 6, and 8	Intermediates and optional

Concerning this kind of two-by-two comparisons (step 3), we can observe two important issues. First, if a ratio r between requirement i and j has value x (that is, if $r(i, j) = x$), then the same ratio r between j and i has value $\frac{1}{x}$ (that is, then $r(j, i) = \frac{1}{x}$). Second, the ratio r between a requirement i with itself always has value 1 (that is, $r(i, i) = 1$).

Step 6 aims to eliminate inconsistencies in evaluations. For instance, if $r(i, j) = 3$ (the ratio r between i and j is 3) and $r(j, k) = 5$ (the ratio r between j and k is 5), it makes no sense to describe $r(i, k) = \frac{1}{3}$ (the ratio r between i and k is $\frac{1}{3}$). For this, the consistency index (CI) is given by equation:

$$CI = \frac{\lambda_{MAX} - n}{n - 1} \tag{1}$$

where λ_{MAX} is the largest eigenvalue of the judgment matrix and n is the number of evaluated criteria (requirements). Lastly, CR is given by equation:

$$CR = \frac{CI}{RI} \tag{2}$$

where CI is the consistency index (Eq. 1) and RI is the ratio index. The value of RI is fixed for each value of n (presented in Table 2, detailed by [Saaty 1987, Agarwal et al. 2013, Karlsson and Ryan 1997]). Thus, the evaluation is consistent if CR $\leq 10\%$ [Saaty 1987].

Table 2. Ratio Index (RI) for different values of n, where n is the quantity of variables and RI is the denominator of Eq. 2

n	3	4	5	6	7	8	9	10
RI	0.58	0.89	1.12	1.24	1.32	1.41	1.45	1.49

2.2.2. Cost-Value Approach

The AHP method must be applied twice to gather the CVA method, considering both value and cost criteria. Then, a two-dimension graph is generated with the cost on the x-axis and the value on the y-axis, both as percentages [Karlsson and Ryan 1997].

Once the graph is plotted, the requirements are classified with high, medium, or low priority based on each requirement's value and cost. The graph is divided equally into three distinct areas: (i) requirements with a high priority, i.e., high ratio between value and cost (a value/cost ratio greater than 2); (ii) requirements with a medium priority, i.e., medium ratio between value and cost (a value/cost ratio between 0.5 and 2); and (iii) requirements with a low priority, i.e., with a low ratio between value and cost (a value/cost ratio lower 0.5) [Karlsson and Ryan 1997].

The strengths of the CVA technique are: (i) it considers two aspects (value and cost), (ii) it groups the requirements by priority levels, (iii) it has a systematized process, and (iv) it allows the comparison of criteria of different categories.

Among the alternatives for requirements prioritization, to the best of our knowledge, CVA is a good option for our problem since the technique relates two distinct aspects in a single analysis, and it groups requirements by three priority levels.

3. Related Works

This section presents related research regarding our problem. We can divide these into two categories: (i) AHP and CVA use experiences and (ii) a CA selection. We describe some related works on both topics, emphasizing the similarities to and differences from our approach.

In addition to the works discussed in the following sections, there are some other related works in the literature. Cost-Benefit Analysis Method (CBAM) [Kazman et al. 2002] and Economics-driven Software Architecture [Thurimella and Padmaja 2014] are examples of approaches that analyze the economic impact of architecture decisions. Value-based Software Engineering (VBSE) [Boehm and Huang 2013] and VALUE framework [Mendes et al. 2018] address a broader view of value in software-related decisions. Unlike our approach, all four works consider one aspect only (cost or value). Value-based approaches (e.g., VBSE and VALUE framework) consider value as personal attitudes and beliefs that may influence

requirements. In this paper, we consider value as the effort required to code a given requirement.

3.1. Both AHP and CVA use experiences

Dorado et al. [Dorado et al. 2014] proposed a method to choose software for engineering simulations, considering educational criteria such as interface, learning effectiveness, portability, help support, and cost. Chiam et al. [Chiam et al. 2013] developed an approach for selecting quality attributes in software. The authors present a method that considers risk management and process integration with the AHP method. An industry case study is performed to validate their method. Unlike our approach, authors in [Dorado et al. 2014, Agarwal et al. 2013, Chaudhary et al. 2016, Chiam et al. 2013] apply the AHP method to a single value not relating two criteria: value and cost.

CVA was originally proposed by Karlsson and Ryan [Karlsson and Ryan 1997], and they prioritized the requirements of two case studies: *the radio access network project* (with 14 generic requirements), and *the performance management traffic recording project* (with 11 generic requirements). The goal of the first case study was to identify and specify requirements for a mobile system. The second case study is a software system for recording and analyzing traffic. Chung, Hough, and Ojoko-Adams [Chung et al. 2006] prioritized software security requirements using the CVA method. Authors in [Karlsson and Ryan 1997, Chung et al. 2006] apply CVA to relate the value and cost of requirements in particular projects. However, they do not evaluate their results to compare different CA. Although the original purpose of the CVA method is to compare requirements [Karlsson and Ryan 1997], to the best of our knowledge, this is the first work to apply CVA to compare and select CAs.

The work most similar to ours is summarized by [Galster et al. 2010]. Galster, Eberlein, and Moussavi [Galster et al. 2010] propose a method for systematic selection of architecture styles (called SYSAS). SYSAS selects an architectural style (e.g., Pipe-and-filter, Client-server) based on desired properties of the system and properties of architectural elements (e.g., persistence, migratability, synchronicity, efficiency, scalability). This work compares the results of SYSAS with case studies judged by experts. Unlike our approach, which applies the CVA method with two aspects (cost and value), the authors in [Galster et al. 2010] conducted their study on a single aspect of importance (through the AHP method). Moreover, our work ranks the set of requirements by computational architecture.

3.2. Selection of CA

Christensen [Christensen 2009] proposes an architecture for future mobile applications based on cloud computing, web services, smart mobile devices, and context. In their study, smart mobile devices use sensors to get the application context, making applications smarter. These applications can be enhanced with the advantages of cloud computing to exceed the traditional capabilities of mobile devices. In the same context, Huang, Xing, and Wu [Huang et al. 2013] argue that mobile devices have become one of the major service providers. The authors propose a new model of mobile cloud computing with the focus on the user. In this approach, devices, clouds, and servers are highly interactive.

In contrast to our paper, the authors in [Christensen 2009, Huang et al. 2013] propose improvements to a particular architecture.

The software architecture analysis method (SAAM) is a method proposed by Kazman, Abowd, and Clements [Kazman et al. 1996] that facilitates the choice of CA based on scenarios, brief narratives of expected or anticipated system uses. SAAM creates scenarios (brief narratives about system uses) to provide a view of how the analyzed system satisfies quality attributes in various usage contexts. The method is composed of five activities: (i) description of the candidate architecture, (ii) creation of scenarios, (iii) evaluation of each scenario (in isolation), (iv) evaluation of the interaction between the scenarios, and (v) determining the weight of each scenario and its interactions. These activities follow a defined flow, and it can be applied for early problem detection and enable a better understanding of the architecture. For example, in the case study presented in the paper, the authors found limitations in obtaining specific requirements such as portability. Although this work lists conflicts between requirements and architectures, the proposed method is manual, and it does not classify requirements according to priority.

4. Our SCVA and MMACA methods for CA

The problem of choosing a CA is directly proportional to the number of requirements. This means that increasing the number of requirements increases the complexity of making a decision, and increases the complexity of determining the requirements with high priority.

Based on a literature search for non-functional requirements, our approach presents a set of requirements that guide the developers' choice of a CA for a given application. This approach comprises of (i) a set of requirements, grouped into four software development stages, (ii) a slightly modified CVA method (SCVA) to analyze priority requirements by CA, and (iii) a method for choosing the most appropriate CA (MMACA) based on a subset of the requirements. Examples of users of the SCVA method are software architects and senior developers, while examples of users of the MMACA method are other developers.

Fig. 1 summarizes the design of our approach. We emphasize that SCVA (step 2) has a set of requirements as input (step 1) and three subsets of this set of requirements (subsets with the highest priority requirements in desktop, web, and cloud CA: one subset for each CA) as output. The MMACA (step 4) has a set of requirements such as input (step 3) and CAs most appropriate for these requirements (in order of suggestion: first, second, and third) as output. Step 3 groups the requirements into low, medium, and high priority by CA. This step is the SCVA method output and MMACA method input.

In this section, we present the set of requirements considered, and both the SCVA and MMACA methods in detail.

4.1. The Set of Requirements

This section details the set of requirements necessary to perform both SCVA and MMASA methods. Initially, we present the adopted strategy to search and select requirements. Next, we evaluate the set of requirements. Finally, we present the final set of requirements and the methodology overview.

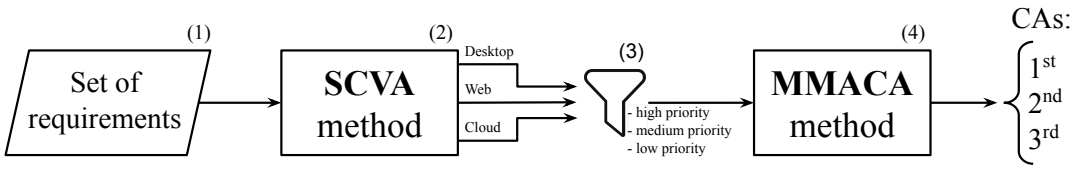

Figure 1. Design of our approach: SCVA has a set of requirements as input and the highest priority requirements by CA as outputs; MMACA has a set of requirements such as input and CAs most appropriate for these requirements as output.

4.1.1. Requirement search

The set of requirements was constructed based on the guideline proposed by [Kitchenham and Charters 2007]. We conducted a literature search for papers that present non-functional requirements. This literature search was conducted by two researchers with moderate experience in requirements engineering. The search method was as follows:

1. a search string was constructed to cover the largest amount of related works. After tuning, the field string was *"software engineering" AND (requirements OR features) AND architecture AND (desktop OR web OR cloud)*
2. the string was applied in four databases: Science Direct, IEEE Xplore, ACM Digital Library, and Springer
3. finally, papers were evaluated, and three exclusion criteria were applied: (i) papers not written in English, (ii) papers not published in a journal, a conference, or a workshop, and (iii) papers unrelated to CA, computational architecture, or development.

The final set of papers was [Kazman et al. 1996, Huang et al. 2013, Beaty et al. 2009, Garg and Singhal 2017, Chieu et al. 2009, Cunsolo et al. 2009, Chien et al. 2003, Rewatkar and Lanjewar 2010]. Appendix A presents the titles, authors, and references of the papers. For each paper, we identified the non-functional requirements, a total of 30 items. We identified the requirements from the literature with the help of two developers. Both developers have a master's degree and more than 5 years of experience in the field.

4.1.2. Requirement evaluate and filter

Six developers evaluated the set of 30 requirements. The participating developers were asked to answer the characterization questionnaire regarding the *professional experience time* (in years), *self-evaluation* (in Likert scale), and *experience in requirements engineering* (in Likert scale). Four developers have considerable experience (in Likert scale) with requirements engineering and two developers have moderate experience (in Likert scale). Table 3 summarizes the other answers reported by each developer interviewed. Since cloud computing is a recent paradigm, developers for this CA have less professional experience compared to other CAs.

Table 3. Profile of the six developers who evaluated the 30 requirements

	Desktop	Web	Cloud
Professional experience time (in years)			
< 1	1	0	3
$1-5$	2	1	3
$6-10$	0	3	0
$11-15$	1	2	0
> 15	2	0	0
Self-evaluation (in Likert scale)			
None (novice)	0	0	3
Little	2	0	1
Moderate	1	3	0
High	0	1	2
Very high (expert)	3	2	0

Each developer reported whether or not each requirement was significant for a comparative study among CAs. Thus, each requirement received a binary evaluation y-n, indicating that y developers answered 'yes' (yes, the requirement is relevant to the study), and n developers answered 'no' (no, the requirement is not relevant to the study).

Given each developer's opinion, we applied the Kappa Coefficient [Fleiss et al. 1971] to determine the level of agreement between responses and then to determine a set of relevant requirements.

Considering the analysis made by [Cyr and Francis 1992], our Kappa Coefficient for the set of 30 requirements according to the developers' opinions was k = 0.784. Results between 0.61 and 0.80 mean substantial agreement [Cyr and Francis 1992].

To increase the level of agreement, we removed the requirements with the highest degrees of discrepancy: 4-2 and 3-3 evaluations (remaining only 6-0 and 5-1). As a consequence, five requirements were removed: *Application integration with local hardware* (y-n evaluation: 3-3), *Access to customer resources* (y-n evaluation: 4-2), *Development environment* (y-n evaluation: 4-2), *Debug* (y-n evaluation: 4-2), and *Cold Backup* (y-n evaluation: 3-3). Our Kappa Coefficient for the 25 remaining requirements was k = 0.853. Kappa Coefficient greater than 0.81 indicates almost perfect agreement [Cyr and Francis 1992].

4.1.3. The final set of requirements

With 25 requirements and high agreement, we generated a single set with the most significant requirements. Although y-n evaluation was carried out regardless of CA, each requirement has advantages and disadvantages depending on the architecture. For instance, portability may have a higher cost in desktop rather than web architecture, or the scalability test can be more appropriate in a cloud architecture. Table 4 presents the 25 requirements.

Table 4. Requirements used in both methods and their respective stages of development as well as the y-n type evaluation

id	Requirement	Development stage	evaluation y-n
1	Security	Planning	6-0
2	Hardware consumption	Planning	5-1
3	Scalability	Planning	6-0
4	Difficulties in architecture	Planning	5-1
5	Session	Planning	6-0
6	Availability	Planning	6-0
7	Database	Planning	6-0
8	Portability	Development	6-0
9	User interface	Development	6-0
10	Development productivity	Development	5-1
11	Multiplatform development	Development	6-0
12	Resource elasticity	Development	6-0
13	Persistence	Development	5-1
14	Collaborative development	Development	5-1
15	Concurrency	Development	6-0
16	Classic tests	Test	6-0
17	End user test	Test	5-1
18	Scalability test	Test	6-0
19	Device sync test	Test	5-1
20	Update and maintenance	Operation	6-0
21	Deploy	Operation	6-0
22	Backup	Operation	5-1
23	Incompatibilities	Operation	5-1
24	Scalability of processing	Operation	5-1
25	Control of unexpected events	Operation	5-1

The set of requirements was organized into four developments stages according to lifecycle [van Lamsweerde 2009]: (i) *planning* consists of the first two steps (requirements analysis and definition, and software design) and it contains the requirements that must be considered before coding; (ii) *development* is the stage of implementation (codification) of a software; (iii) *test* is responsible for verifying the correctness of a design requirement; and (iv) *operation* involves the requirements derived from the use and maintenance of a system (Table 4). For each requirement in the set of requirements, we summarize the development stage and binary evaluation y-n.

Both SCVA and MMACA methods employ the stable artifact present in Table 4. While the SCVA method groups the requirements by CA and priority, the MMACA method lets the stakeholders select the set of the most important requirements for their projects and then the MMACA method determines the set of the most appropriate CA for those requirements. A description of each requirement is available at `https://github.com/FORMAS/detFSR`.

Requirements are grouped into four development steps to reduce the number of comparisons. Since n requirements need $\frac{(n*(n-1))}{2}$ comparisons [Karlsson and Ryan 1997], 25 requirements need 300 comparisons. The division by stages of development allows comparisons by each group (Table 4 shows the number of requirements for each stage of development). Thus, the comparisons are reduced from 300 to 70: 21 for planning, 28 for development, 6 for testing, and 15 for the operation stage.

Furthermore, the division emphasizes the development stage of requirements with higher priority and allows the methods to focus on the current stage of the application. For instance, developers of a given application may need to know the requirements with high priority for the operation and test stages, but not the planning stage. On the other hand, the operation stage may not be important for an application until the planning stage.

After that, we obtain the set of requirements: a set of 25 requirements, grouped by development stage, and with a high degree of agreement among the developers interviewed. The set of requirements is applied in different ways depending on the proposed method.

4.1.4. Overview

Fig. 2 summarizes the design applied to obtain the set of requirements. Initially, we conducted a literature review of papers about non-functional requirements for projects in desktop, web or cloud architectures (step 1). We selected eight papers, and we extracted 30 different non-functional requirements (step 2). Next, we invited six developers to evaluate the set of 30 requirements (step 3) and the Kappa Coefficient result was k = 0.784 (step 4). Results between 0.61 and 0.80 mean substantial agreement [Cyr and Francis 1992]. Since almost perfect agreement requires values greater than 0.81, we performed a new round. In the new round, five requirements evaluated as 4-2 and 3-3 were removed (step 5), and the new Kappa Coefficient result of k = 0.853 (step 6). These steps resulted in a set of 25 requirements (step 7). The developers participated only in the process of judging the requirements (step 3).

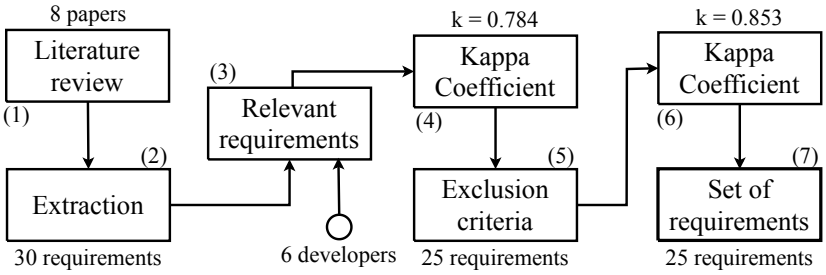

Figure 2. Overview of the steps performed to obtain the set of requirements. Six developers participated in the evaluation (step 3).

4.2. A Slightly Modified CVA (SCVA) for CA selection

Our Slightly Modified CVA (SCVA) for CA selection is introduced in two parts: description and SCVA use example.

4.2.1. Description of SCVA

SCVA determines which requirements have the highest priority in desktop, web, and cloud architectures. For this: (i) we separate the set of requirements according to the development stage, (ii) we apply the CVA method for each development stage, and finally, (iii) we list the highest priority requirement for each CA. The SCVA method must run once for each CA.

Fig. 3 illustrates the design of our SCVA method: CVA is applied individually for each development stage (CVA_1, CVA_2, CVA_3, and CVA_4), and individual results are presented in two analyses (R_1 and R_2). The first analysis (R_1) compares the aspects individually for each CA, i.e., given one requirement and one aspect, we can emphasize in which CA this requirement is more (or less) costly (or valuable). On the other hand, in R_2 the requirements are grouped by CA, i.e., given one requirement and one CA, we can emphasize the priority of this requirement (high, medium, or low) and what its development stage is. The cost and value of each requirement are available in Appendix B.

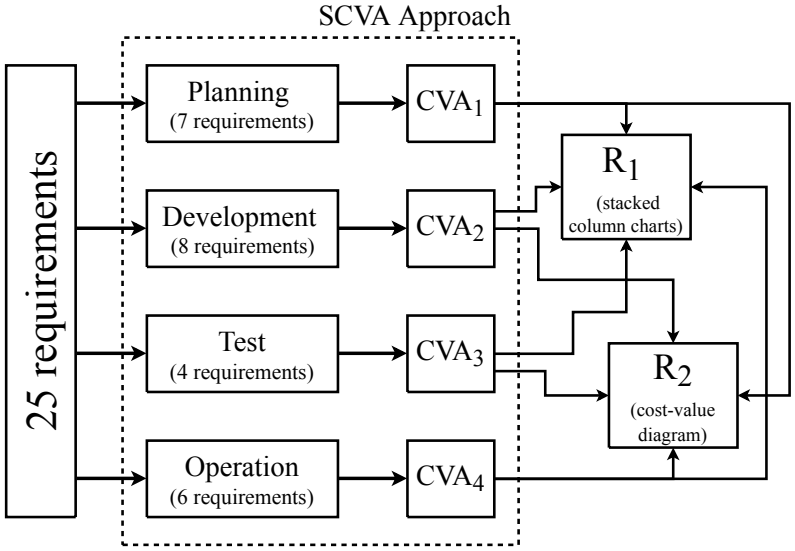

Figure 3. Design of our SCVA method: requirements are grouped into development stages, the CVA method (AHP twice) is applied for each step (CVA_1, CVA_2, CVA_3, and CVA_4), and the results are grouped into two analyses (R_1 and R_2).

While our SCVA applies a particular set of requirements in this work, it can be performed with any other set of requirements. For this, it is only necessary to follow the steps described in the next section.

4.2.2. SCVA use example

Considering that SCVA applies CVA, which in turn applies AHP twice, we show part of the requirements hierarchy process below. The example considers requirements of the operation stage (Table 4) of a project in cloud computing architecture. In this section we present the analysis for the cost aspect, and then, we perform the analysis considering the value aspect, as required by the CVA method.

Table 5 presents the $n \times n$ matrix after two-by-two comparison by developers for the cost aspect, operation stage, and cloud architecture.

Table 5. An example of two-by-two evaluation for the cost of operating requirements (id 20 to 25) of cloud applications

	id 20	id 21	id 22	id 23	id 24	id 25
Update and maintenance (id 20)	1	1	3	1/3	1	1/3
Deploy (id 21)	1	1	1	1/3	1/3	1/3
Backup (id 22)	1/3	1	1	1/5	1/5	1/5
Incompatibilities (id 23)	3	3	5	1	1	1/3
Scalability of processing (id 24)	1	3	5	1	1	1
Control of unexpected events (id 25)	3	3	5	3	1	1

Requirements are ranked according to the AHP method. Table 6 presents the classification for the comparison in Table 5. The "rank" column indicates the degree of the requirements, and the "order" column indicates the position of the requirements in the set. In this case, the order is inversely proportional to the cost. For instance, *Control of unexpected events* is the most costly requirement, while *Backup* is the cheapest (considering the CA cloud).

Table 6. An example of hierarchical requirements based on cost

Requirement	Rank	Order (Cost)
Update and maintenance	11.7%	4
Deploy	7.9%	5
Backup	5.0%	6
Incompatibilities	21.8%	2
Scalability of processing	21.5%	3
Control of unexpected events	32.1%	1

As the consistency ratio (CR) in this example is $5.2\% \leq 10\%$, the evaluation is consistent.

Steps (including the calculation of CR) were performed using the online tool BPMSG AHP Online System[1] and Excel worksheets.

[1] `http://bpmsg.com/academic/ahp_calc.php`

4.3. Method for Choosing the Most Appropriate CA (MMACA)

Our Method for Choosing the Most Appropriate CA (MMACA) is introduced in two parts: description and the MMACA use example.

4.3.1. Description of MMACA

Choosing the most appropriate CA is a task that involves many factors because the best CA depends on the business model and application features [Patterson and Hennessy 2013, Bessa et al. 2016]. Generally, this choice is made informally according to business rules. The choice based on a set of requirements may be complex or at least leaves room for discussion. Therefore, we present a method to assist in choosing the most appropriate CA for a given set of requirements. The MMACA method applies w requirements of the set of requirements present (Table 4), where $1 \leq w \leq 25$.

Among the requirements in the set (Table 4), let us suppose that stakeholders determine which requirements are the most important for a project, creating a subset with w requirements. Based on the values of our comparisons, this subset can be used to suggest the most appropriate CA (an example is described in Table 7). For this analysis, our method for choosing the most appropriate CA (MMACA) follows these steps:

1. among the requirements in the set (Table 4), stakeholders determine the most important requirements, creating one subset with w requirements
2. for each requirement in the subset, the ratio between value (in %) and cost (in %) is calculated considering CAs individually (according to [Karlsson and Ryan 1997])
3. after estimating value divided by cost, a geometric mean of ratios for each CA is calculated
4. finally, CA with the highest geometric mean is the most appropriate.

Stakeholders may have different opinions about the same project. This means that given the same project, if two stakeholders apply the MMACA method, both can use different inputs.

4.3.2. MMACA use example

Since requirements value and cost are available for different CAs, the decision of which one to use can be taken more securely. For instance, assume that requirements with ids 2, 9, 14, 21, and 22 (according to Table 4) are important for a project. In this case, the next step is to calculate the relationship between value and cost of each important requirement for each of the three CAs. Finally, we must calculate the geometric mean for each CA. Thus, the mean of the highest value represents the most appropriate CA.

Based on the method presented and the data presented in Table 7, the best choice for input above (2, 9, 14, 21, and 22) is web architecture, and cloud architecture as the second option. In contrast, assuming input with 4, 6, 8, 12, 16, 18, 21, and 24, the best choice is cloud architecture, and desktop the second choice.

Table 7. Example of architecture choice based on an input with ids 2, 9, 14, 21, and 22. Each row (except the last one) represents the ratio between value and cost for a particular requirement. Columns group the ratios by each CA. The last row presents the geometric mean of the ratios for each CA

id	Requirement	Ratio between value and cost for		
		Desktop	Web	Cloud
2	Hardware consumption	0.81	0.77	7.67
9	User interface	0.69	0.99	0.27
14	Collaborative development	0.69	1.67	1.97
21	Deploy	0.69	1.57	1.56
22	Backup	12.52	9.00	0.70
	Geometric mean	1.27	**1.78**	1.35

Both methods can be applied jointly or independently: stakeholders may apply SCVA only, MMACA only, or both SCVA and MMACA, depending on the current stage of the project. For instance, the SCVA method may determine the most priority requirements of projects that are in the early planning stage. On the other hand, the MMACA method may determine the most appropriate CA of projects that are in the final coding stage (or implementation stage). When it is necessary to apply both SCVA and MMACA, one possibility is to consider the output of the SCVA method as the input of the MMACA method.

5. Validation of SCVA and MMACA approaches

We performed the SCVA method with the set of requirements (Table 4) and MMACA method with some inputs to evaluate our approaches. To illustrate the SCVA and MMACA methods, (i) we analyze the requirements and classify the requirements in low, medium, and high priority, and (ii) we classify the most appropriate CA for ten generic projects and one real project (middleware for DaaS and SaaS [Ribeiro et al. 2019]).

Before the two validation rounds, we collected professional data about desktop, web, and cloud architectures. Initial steps were (i) the construction of the design of the experiment and (ii) filling a form. Design is an essential artifact in software engineering experiments, and the form aims to collect developers' professional experience in years and self-evaluation.

Stakeholders must follow the sequence of activities proposed by our methods. These activities generate input and output artifacts during their execution. In order to evaluate the effectiveness and efficiency of our methods, we are interested in two research questions:

RQ1. *Can developers apply our methods and get the outputs?* This research question explores the effectiveness of our approach by assessing whether developers can apply our methods according to their expertise.

RQ2. *Do the developers agree with the outputs?* This research question explores the efficiency of our approach by assessing whether developers agree with the results

obtained by our approach. In other words, we are interested in whether our results match the requirements of the evaluated projects.

Both RQ1 and RQ2 are important because our method would be unfeasible if stakeholders could not carry out the activities and generate the artifacts correctly. Therefore, RQ1 and RQ2 are related because RQ1 assesses the feasibility of carrying out the method and RQ2 assesses whether the results agree.

5.1. SCVA method validation

Fig. 4 illustrates the SCVA method validation. After two initial steps, the 30 requirements initially collected were judged and evaluated by six developers. They judged the relevance of each requirement generally, i.e., without considering a particular project. We created a unique set with the most important requirements to apply AHP to the SCVA method with the answers.

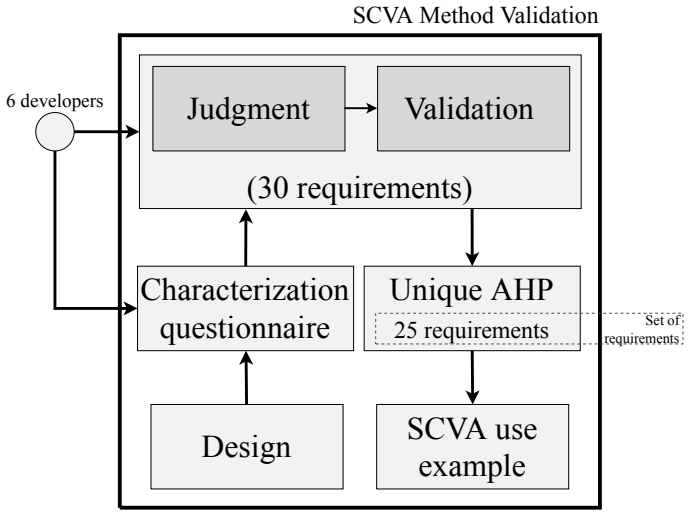

Figure 4. Details of the SCVA method validation. Six developers participated in this validation.

After applying the SCVA method, the requirements grouped by aspect reveal the CA in which the requirements have more (or less) value or cost. Stacked column charts provide a macro view of data. In our design, shown in Fig. 3, we call this analysis of R_1. Fig. 5 describes the requirements considering the value aspect, and Fig. 6 illustrates the cost view. For example, according to Fig. 5, requirement 16 is more relevant for desktop architecture than for cloud architecture, and requirement 19 is more relevant for web architecture than desktop architecture. In contrast, as Fig. 6 shows, requirement 1 has a higher cost in desktop than web architecture, and requirement 25 has a higher cost in cloud architecture than desktop architecture.

Fig. 7, 8, and 9 present the cost-value diagram for each CA, showing the requirements with the highest priorities. Both diagrams are useful for the composition of the requirements as they are grouped according to the ratio between value and cost values. We present this analysis in our design of Fig. 3 as R_2.

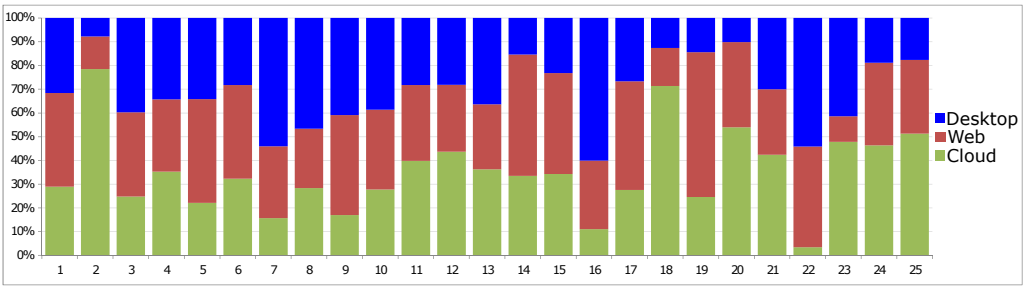

Figure 5. Comparison of the value (y-axis) of requirements (x-axis) for desktop, web, and cloud. This figure depicts value of each requirement in an architecture. Requirements are represented by id, according to Table 4.

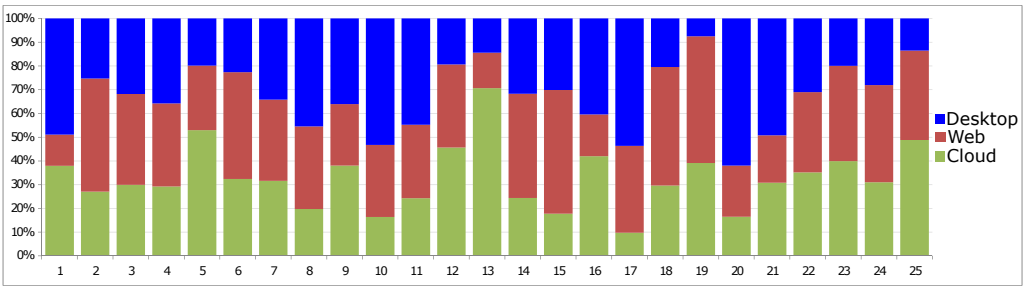

Figure 6. Comparison of the cost (y-axis) of requirements (x-axis) for desktop, web, and cloud. This figure depicts cost of each requirement in an architecture. Requirements are represented by id, according to Table 4.

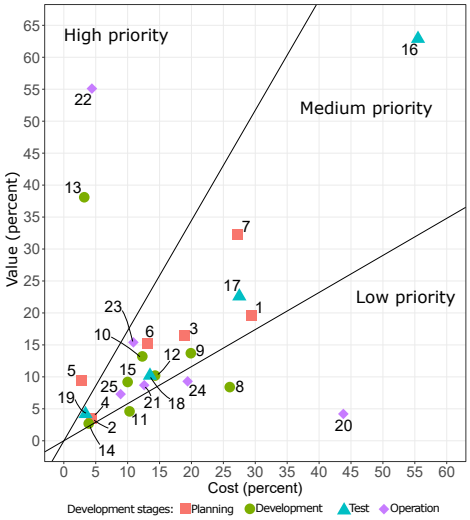

Figure 7. CVA with prioritized requirements in the desktop architecture (represented by id, according to Table 4).

Some open issues are important to discuss: (i) requirement 13 has high priority in both CAs, (ii) Requirements 5, 13, and 22 have high priority in desktop and web architectures, as well as 13 and 17 has high priority in web and cloud architectures, (iii) requirements 3, 4, 6, 7, 8, 9, 11, 12, 16, 19, 21, 23, 24, and 25 are not present in the set of

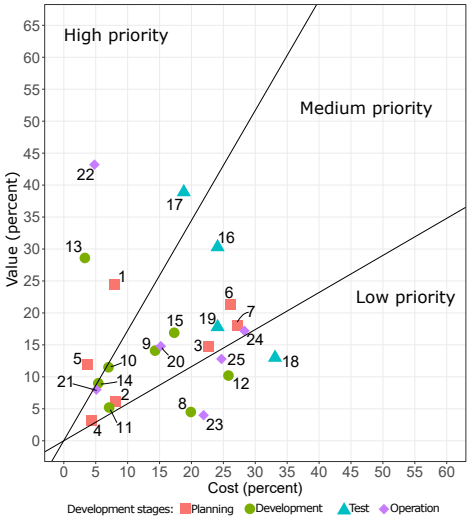

Figure 8. CVA with prioritized requirements in the web architecture (represented by id, according to Table 4).

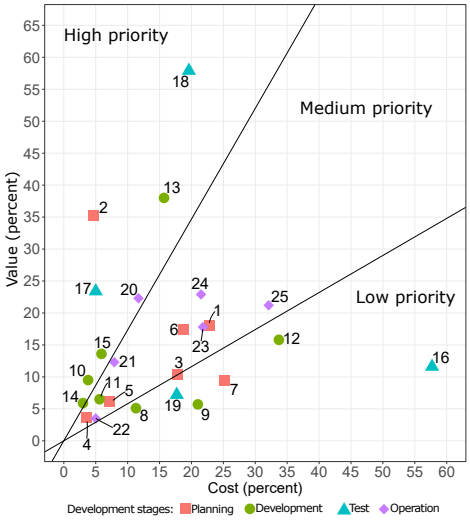

Figure 9. CVA with prioritized requirements in the cloud architecture (represented by id, according to Table 4).

high priority for no CA, (iv) only desktop architecture does not contain requirements of the test step as a high priority, and (v) all CAs have at least one planning, development, and operation requirement which is high priority.

Concerning the low priority requirements, (i) all CAs have requirement 8, (ii) desktop architecture does not contain any planning and test step requirements, (iii) web architecture does not contain any planning step requirements, and (iv) cloud architecture does not contain any operation step requirements.

These steps enabled the evaluation of RQ1. The developers applied our SCVA method to rank the requirements (Table 4) according to priority. This evaluation suggests the effectiveness of SCVA method since the developers were able to apply our approach

based on their expertise. Given that evaluating the expertise is subjective, we assume that the developers' opinion for the SCVA method is optimal.

The output of the SCVA method is the input of the MMACA method. Although running the SCVA method is laborious (i.e., AHP eight times), this method needs to be performed only once for each set of requirements. On the other hand, the MMACA method can be performed several times for each SCVA method's output. We emphasize that the benefits outweigh the cost of applying our approach.

5.2. MMACA method validation

We validate the MMACA method in two ways: 10 generic projects and one real project. First, we judge projects from 10 developers. For this, each developer was invited to indicate the relevant requirements of their respective projects. In this evaluation, we were not interested in the type or context of the project. Our interest was only in the relevant requirements of the projects, the suggested CA by MMACA method, and the actual CA. After performing the MMACA method for each project, we evaluated whether CA suggested by the MMACA method corresponded to the actual CA of each project. In the second evaluation, we evaluated a real project: Middleware for DaaS and SaaS [Ribeiro et al. 2019]. In this evaluation, we invited four developers of this project to nominate the middleware' relevant requirements. After that, we performed the MMACA method with the obtained requirements and evaluated whether the suggested CA corresponds to the actual CA of the middleware. We choose all projects and developers based on convenience sampling [Wohlin et al. 2012]. The following subsections describe both evaluations.

5.2.1. Generic Projects

Fig. 10 illustrates the MMACA method validation in the first evaluation. After designing construction and characterization questionnaire, we performed the MMACA method on ten generic projects.

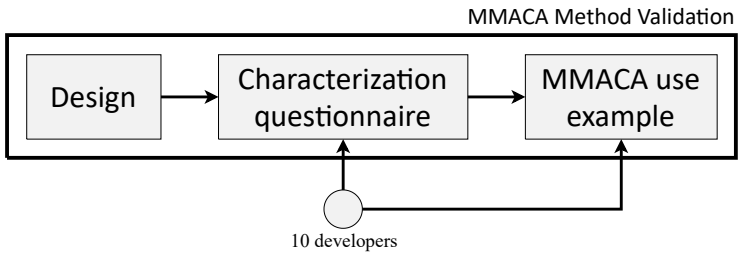

Figure 10. Flowchart of MMACA method validation for generic projects. Ten developers participated in this validation.

To evaluate the MMACA method, we executed the proposed algorithm with ten developers. Each developer chose a subset with the most relevant requirements (among the set of requirements) considering a generic project. In this case, this subset was

composed of requirements that developers considered essential for their project. Table 8 presents the results obtained by the MMACA method for each of the ten evaluated projects.

Table 8. Projects submitted to the MMACA method

	Most relevant requirements	Most Appropriate CA		
		First	Second	Third
1	1, 3, 5-11, 13, 14, 16-20, 22, 24, 25	Web	Desktop	Cloud
2	1, 3, 6, 8, 9, 15-17, 20, 24	Web	Cloud	Desktop
3	1, 2, 5-11, 13, 14, 16, 17, 20-22, 25	Web	Cloud	Desktop
4	1, 3, 6-12, 14-18, 20, 22, 24, 25	Web	Cloud	Desktop
5	1, 3, 6-10, 14, 16, 17, 19, 20, 23, 25	Cloud	Web	Desktop
6	1, 2, 3, 7, 9, 10-14, 16, 17, 20-25	Web	Cloud	Desktop
7	1, 3, 6, 7, 9, 11, 13, 15-18, 20-22	Web	Cloud	Desktop
8	1, 3, 6, 7, 9-11, 13, 16-21, 24	Web	Cloud	Desktop
9	1-3, 6-11, 13, 14, 16, 17, 20, 23, 25	Cloud	Web	Desktop
10	1, 6-9, 11, 13, 16-19, 24, 25	Web	Desktop	Cloud

After executing the MMACA method, each developer evaluated the result for their project. In 80% of the simulations, the method indicated the project's current architecture as the first option. In 90% of the projects, the developer stated that the order determined by the MMACA method conforms to the evaluated project's requirements. These data demonstrate the effectiveness of our method.

These steps enabled the evaluation of RQ1 and RQ2. The developers applied our MMACA method to rank CA that best fulfills a set with the most relevant requirements, and 90% of the developers agreed with the results. By doing this, we evaluated the effectiveness and efficiency of our method.

5.2.2. Real Project

Middleware for DaaS and SaaS (MIDAS) provides interoperability between cloud services such as Software as a Service (SaaS), Data as a Service (DaaS), and Database as a Service (DBaaS) [Ribeiro et al. 2019]. MIDAS mediates communication between SaaS and DaaS/DBaaS transparently. The current version of MIDAS is hosted on Heroku (https://www.heroku.com), an open cloud that provides enough storage space and a complete platform for the project. We select MIDAS to evaluate our MMASA method since we have access to the middleware source code and developers, and this allows us to evaluate features using an existing framework. Fig. 11 illustrates the use example of the MMACA method in the middleware MIDAS.

In step (1), each of the four developers pointed out the requirements of set (Table 4) that MIDAS should implement. At the end of this first step, each requirement was associated with an evaluation y-n, indicating how many developers voted 'yes' (yes, MIDAS must implement such a requirement) and 'no' (no, MIDAS should not implement such a

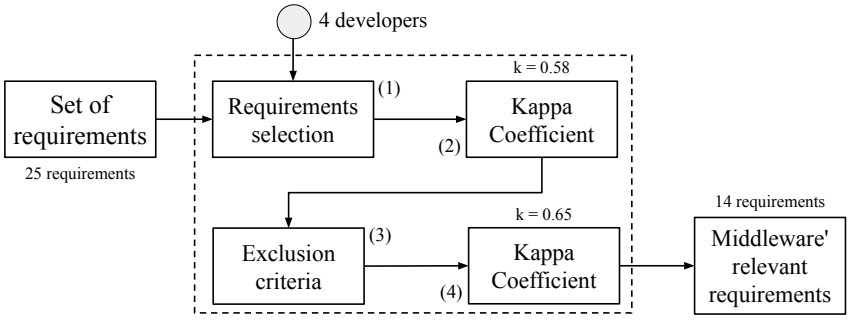

Figure 11. Flowchart of MMACA method validation in a real project: middleware MIDAS. Four developers participated in this validation.

requirement). For instance, the requirement *availability* (id 6) was evaluated 3-1 (3 votes 'yes' and 1 vote 'no'), while the requirement *security* (id 1) was evaluated 1-3 (1 vote 'yes' and 3 votes 'no'). Table 9 displays the compiled evaluations of the four developers for the current 25 requirements of the set of requirements.

Table 9. Evaluations (in the yes-no format) of 25 requirements of the set for MIDAS

id	y-n	id	y-n	id	y-n	id	y-n	id	y-n
1	1-3	6	3-1	11	4-0	16	3-1	21	3-1
2	3-1	7	3-1	12	2-2	17	2-2	22	1-3
3	3-1	8	2-2	13	1-3	18	4-0	23	4-0
4	1-3	9	1-3	14	3-1	19	2-2	24	4-0
5	0-4	10	3-1	15	3-1	20	4-0	25	2-2

In step (2), the level of agreement among the four developers was estimated based on the Kappa coefficient [Fleiss et al. 1971]. The result indicated a moderate agreement (k = 0.58).

In step (3), to increase the level of agreement of the collected opinions, the evaluations without an absolute majority (that is, evaluations of type 2-2: 2 votes 'yes' and 2 votes 'no') were removed. The requirements *Portability*, *Resource elasticity*, *End user test*, *Device sync test*, and *Control of unexpected events* (ids 8, 12, 17, 19, and 25, respectively) were removed, and the level of agreement was again evaluated as strong (k = 0.65) in step (4).

The four steps enabled the construction of a set of 14 requirements that MIDAS should hypothetically implement (if not already implemented). This set contains requirements with 4-0 and 3-1 evaluations. We eliminated requirements with 0-4 and 1-3 evaluations since most developers voted 'no', and the requirements with 2-2 evaluations (step 3). The 14 requirements considered were the following: *Hardware consumption*, *Scalability*, *Availability*, *Database*, *Development productivity*, *Multiplatform development*, *Collaborative development*, *Concurrency*, *Classic tests*, *Scalability test*, *Update and maintenance*, *Deploy*, *Incompatibilities*, and *Scalability of processing* (ids 2, 3, 6, 7, 10, 11, 14-16, 18, 20, 21, 23, and 24, respectively).

Finally, the MMACA method was performed based on 14 requirements important for MIDAS and the cloud was suggested as the most appropriate CA for the middleware, and web as the second option.

Since MIDAS is hosted in the Heroku cloud, this evaluation enabled the evaluation of RQ1 and RQ2. Developers applied our MMACA method to rank CA that best fulfills a real project (middleware MIDAS). Our result indicated cloud computing architecture as the first option for MIDAS, exactly the current CA. In this way, we evaluated the effectiveness and efficiency of our method.

6. Discussion

Information systems are increasingly complex, and this complexity is also directly proportional to the number of stakeholders and functional and non-functional requirements that should be fulfilled in a software project. While some requirements are codified into the development stage, others will be provided by CA. Choosing the most appropriate CA for a set of requirements is still a challenge because different architectures offer a distinct set of requirements and ways for this fulfillment. This choice is usually direct assigned to developers, not end-users.

Both SCVA and MMACA methods aim to systematize RE activities to choose the most appropriate CA for a set of requirements. While our SCVA method supports developers based on the literature (i.e., domain's stable knowledge), our MMACA method focuses on application-specific demands. Software engineers can also use our methods to evolve its artifacts by adding new requirements and adapt to requirements previously analyzed.

Our proposal intends to group consolidated (i.e., literature) and stakeholders' specific knowledge to point out the most appropriate CA. Several works aim to present and discuss different CA requirements fulfillment. Our proposal is related to the existing literature in two ways. First, requirements listed by our related works (e.g., [Kazman et al. 1996]) can compose our SCVA method. For this purpose, new requirements need to be compared with previous others (as shown in Fig. 3) from the three architectures' perspective, and so generating a new set of requirements, new stacked column charts (R_1 in Fig. 3), and new cost-value diagrams (R_2 in Fig. 3). Second, CA listed by other works (e.g., [Christensen 2009] and [Huang et al. 2013]) can also be part of our approach. In that case, new CAs need to be included in the requirements comparison activity (CVA method step in Fig. 3), generating new new stacked column charts (R_1 in Fig. 3) and new cost-value diagrams (R_2 in Fig. 3). The new requirements or CAs addition implies the need to perform our methods in an incremental approach since the MMACA method is performed based on the SCVA method results.

The approach generalization is related to replication our and new studies as the generalization of one architecture's knowledge depends on a generic set of requirements to cover a CA's characteristics in detail. We can gradually make the MMACA method more accurate by applying the SCVA method several times based on a more refined set of requirements. In summary, our methodology can be generalized by (i) expanding the set of requirements used in both methods (Table 4) and (ii) reproducing the methods

until stabilizing the knowledge about the applicability of a given CA for the same set of requirements. Our experiments are available at FORMAS Research Group website[2] and may be replicated in other studies. We hope that sharing this methodology version provides future contributions and improvements.

7. Threats of Validity

Our approach is subject to limitations. According to [Wohlin et al. 2012], threats to the validity of a study contribute to the trustworthiness of the results. In this section, we present the threats identified during this research.

Conclusion validity addresses the ability to obtain a correct conclusion from the relationship between the treatment and the outcome of the experiment [Wohlin et al. 2012]. In this paper, so that developers' experience did not influence the results, we consulted developers with varied and random experience, without any grouping. Furthermore, the number of developers interviewed could be considered low.

Internal validity addresses any factor that may affect the study's independent variables without the researchers' knowledge [Wohlin et al. 2012]. In this paper: (i) as different developers could interpret the same requirement differently, we give a brief description of each requirement, (ii) since there is an initial complexity to the AHP method, the comparisons were collected through interviews, (iii) in order to homogenize the selection of the developers, all the interviews were conducted on a voluntary basis, and (iv) in order for a developer's response not to influence another developer's response, the interviews were carried out individually.

Construct validity addresses factors (e.g., design or social) which make it difficult to generalize the results [Wohlin et al. 2012]. In this paper, to avoid social threats, developers were not evaluated based on performance. Additionally, the MMACA method assumes that stakeholders are aware of the non-functional requirements of their projects.

External validity addresses conditions that limit the generalization of the results of our experiment to industrial practice [Wohlin et al. 2012]. In this paper, in order to facilitate generalization, only developers with some experience participated in the experiment.

8. Conclusion

This paper analyzes a set of requirements to compare desktop, web, and cloud architectures. Moreover, we investigate the effects of choosing the architecture that best fulfills a set of requirements. First, requirements were classified based on the AHP method. These requirements were then grouped into priority levels based on the CVA method. High priority requirements simplify the comparison of CAs. Similarly, the most appropriate CA analysis helps decisions where requirements are particularly important for an information system.

Based on the questions RQ1 and RQ2, our results suggest the effectiveness and efficiency of the approach. Our research found that the CVA method is useful for prioritizing requirements, choosing a CA according to an information system's requirements. As

[2] `http://formas.ufba.br/page/downloads`

contributions, we compare different CAs based on a set of non-functional requirements (according to the SCVA method), and we present a strategy to facilitate the choice of a CA based on the potential requirements of a project (according to the MMACA method). As expected, we emphasize that the most appropriate CA depends on the requirements, available resources, and stakeholders' priorities.

The major limitation of this study is that the CVA method does not consider the dependencies among requirements. For instance, a low priority requirement may be required for the implementation of a high priority requirement.

Although we are aware of the limitations, validation answered the research questions positively. We intend to refine our strategy to provide better results. Additionally, we are working on (i) adding fog computing architecture [Bonomi et al. 2012] to the analysis of a new paradigm that emerges as an extension of cloud computing, and (ii) applying our MMACA method in more projects to generalize our results.

Acknowledgement

Elivaldo Lozer Fracalossi Ribeiro would like to thank FAPESB (Foundation for Research Support of the State of Bahia) for financial support (BOL0631/2016).

References

[Achimugu et al. 2014] Achimugu, P., Selamat, A., Ibrahim, R., and Mahrin, M. N. (2014). A systematic literature review of software requirements prioritization research. *Information and Software Technology*, 56(6):568–585.

[Agarwal et al. 2013] Agarwal, E., Aharwal, R., Garg, R. D., and Garg, P. K. (2013). Delineation of groundwater potential zone: An AHP/ANP approach. *Journal of Earth System Science*, 122(3):887–898.

[Ahuja et al. 2016] Ahuja, H., Sujata, and Purohit, G. N. (2016). Understanding requirement prioritization techniques. In *5th International Conference on Computing, Communication and Automation (ICCCA)*, pages 257–262, Greater Noida, India. IEEE.

[Armbrust et al. 2009] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical report.

[Beaty et al. 2009] Beaty, K., Kochut, A., and Shaikh, H. (2009). Desktop to cloud transformation planning. In *23rd IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–8, Rome, Italy. IEEE.

[Belout and Gauvreau 2004] Belout, A. and Gauvreau, C. (2004). Factors influencing project success: the impact of human resource management. *International Journal of Project Management*, 22(1):1 – 11.

[Bessa et al. 2016] Bessa, S., Valente, M. T., and Terra, R. (2016). Modular Specification of Architectural Constraints. In *10th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, pages 31–40, Maringá, PR, BR. SBC.

[Bhandari and Sehgal 2014] Bhandari, P. and Sehgal, R. (2014). An evaluation of methods to prioritize requirements. *International Journal of Computer Science and Mobile Computing*, 3(4):1336–1341.

[Boehm and Huang 2013] Boehm, B. and Huang, L. (2013). Value-Based Software Engineering: Reinventing "Earned Value" Monitoring and Control . *ACM Software Engineering Notes*, 28(2):1–7.

[Bonomi et al. 2012] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog Computing and Its Role in the Internet of Things. In *1st Workshop on Mobile Cloud Computing (MCC)*, pages 13–16, New York, NY, USA. ACM.

[Bourque and Fairley 2014] Bourque, P. and Fairley, R. E. (2014). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society.

[Chaudhary et al. 2016] Chaudhary, P., Chhetri, S. K., Joshi, K. M., Shrestha, B. M., and Kayastha, P. (2016). Application of an Analytic Hierarchy Process (AHP) in the GIS interface for suitable fire site selection: A case study from Kathmandu Metropolitan City, Nepal. *Socio-Economic Planning Sciences*, 53(1):60–71.

[Chiam et al. 2013] Chiam, Y. K., Staples, M., Ye, X., and Zhu, L. (2013). Applying a Selection Method to Choose Quality Attribute Techniques. *Inf. Softw. Technol.*, 55(8):1419–1436.

[Chien et al. 2003] Chien, A., Calder, B., Elbert, S., and Bhatia, K. (2003). Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5):597–610.

[Chieu et al. 2009] Chieu, T. C., Mohindra, A., Karve, A. A., and Segal, A. (2009). Dynamic scaling of web applications in a virtualized cloud computing environment. In *6th International Conference on E-Business Engineering (ICEBE)*, pages 281–286, Macau, China. IEEE.

[Christensen 2009] Christensen, J. H. (2009). Using RESTful web-services and cloud computing to create next generation mobile applications. In *24th International Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 627–634, Portland, Oregon, USA. ACM.

[Chung et al. 2006] Chung, L., Hung, F., Hough, E., and Ojoko-Adams, D. (2006). Security Quality Requirements Engineering (SQUARE): Case Study Phase III (CMU/SE-2006-SR-003). Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

[Cunsolo et al. 2009] Cunsolo, V. D., Distefano, S., Puliafito, A., and Scarpa, M. (2009). Volunteer computing and desktop cloud: The Cloud@Home paradigm. In *8th International Symposium on Network Computing and Applications (NCA)*, pages 134–139, Cambridge, MA, USA. IEEE.

[Cyr and Francis 1992] Cyr, L. and Francis, K. (1992). Measures of clinical agreement for nominal and categorical data: The kappa coefficient. *Computers in Biology and Medicine*, 22(4):239 – 246.

[Dorado et al. 2014] Dorado, R., Gómez-Moreno, A., Torres-Jiménez, E., and López-Alba, E. (2014). An AHP application to select software for engineering education. *Computer Applications in Engineering Education*, 22(2):200–208.

[dos Santos et al. 2016] dos Santos, J. R. F., Albuquerque, A. B., and Pinheiro, P. R. (2016). Requirements Prioritization in Market-Driven Software: A Survey Based on Large Numbers of Stakeholders and Requirements. In *10th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 67–72, Lisbon, PT. IEEE.

[Erdogmus 2009] Erdogmus, H. (2009). Cloud computing: Does nirvana hide behind the nebula? *IEEE Software*, 26(2):4–6.

[Fleiss et al. 1971] Fleiss, J. et al. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382.

[Galster et al. 2010] Galster, M., Eberlein, A., and Moussavi, M. (2010). Systematic selection of software architecture styles. *IET Software*, 4(5):349–360.

[Garg and Singhal 2017] Garg, U. and Singhal, A. (2017). Software requirement prioritization based on non-functional requirements. In *7th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pages 793–797, Noida, India. IEEE.

[Huang et al. 2013] Huang, D., Xing, T., and Wu, H. (2013). Mobile cloud computing service models: A user-centric approach. *IEEE Network*, 27(5):6–11.

[Karlsson and Ryan 1997] Karlsson, J. and Ryan, K. (1997). A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74.

[Karlsson et al. 1998] Karlsson, J., Wohlin, C., and Regnell, B. (1998). An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, 39(14):939–947.

[Kaur and Aggrawal 2013] Kaur, B. P. and Aggrawal, H. (2013). Exploration of Success Factors of Information System. *International Journal of Computer Science Issues*, 10(2):226–235.

[Kazman et al. 1996] Kazman, R., Abowd, G., Bass, L., and Clements, P. (1996). Scenario-based analysis of software architecture. *IEEE software*, 13(6):47–55.

[Kazman et al. 2002] Kazman, R., Asundi, J., and Klein, M. (2002). Making architecture design decisions: An economic approach. Technical Report CMU/SEI-2002-TR-035, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[Khajeh-Hosseini et al. 2010] Khajeh-Hosseini, A., Sommerville, I., and Sriram, I. (2010). Research challenges for enterprise cloud computing. *CoRR*, 1(18):1–11.

[Kitchenham and Charters 2007] Kitchenham, B. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical report, Keele University and Durham University Joint Report.

[Maenhaut et al. 2016] Maenhaut, P.-J., Moens, H., Ongenae, V., and Turck, F. D. (2016). Migrating legacy software to the cloud: Approach and verification by means of two medical software use cases. *Softw. Pract. Exper.*, 46(1):31–54.

[Mell and Grance 2011] Mell, P. M. and Grance, T. (2011). SP 800-145. The NIST Definition of Cloud Computing. Technical report, MISSING.

[Mendes et al. 2018] Mendes, E., Marín, P. R., Freitas, V., Baker, S., and Atoui, M. A. (2018). Towards improving decision making and estimating the value of decisions in value-based software engineering: the VALUE framework. *Softw. Qual. J.*, 26(2):607–656.

[Olsson et al. 2019] Olsson, T., Wnuk, K., and Gorschek, T. (2019). An empirical study on decision making for quality requirements. *Journal of Systems and Software*, 149(1):217 – 233.

[Pallis 2010] Pallis, G. (2010). Cloud computing: The new frontier of internet computing. *IEEE Internet Computing*, 14(5):70–73.

[Patterson and Hennessy 2013] Patterson, D. A. and Hennessy, J. L. (2013). *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., 3 edition.

[Pegoraro and Paula 2017] Pegoraro, C. and Paula, I. C. A.-s. d. (2017). Requirements processing for building design: a systematic review . *Production*, 27(1):1–18.

[Rewatkar and Lanjewar 2010] Rewatkar, L. R. and Lanjewar, U. L. (2010). Implementation of cloud computing on web application. *International Journal of Computer Applications*, 2(8):28–32.

[Ribeiro et al. 2019] Ribeiro, E. L. F., Vieira, M. A., Claro, D. B., and Silva, N. (2019). Interoperability between saas and data layers: Enhancing the midas middleware. In Muñoz, V., Ferguson, D., Helfert, M., and Pahl, C., editors, *Cloud Computing and Services Science (CLOSER 2018)*, chapter 6, pages 102–125. Springer.

[Riegel and Doerr 2015] Riegel, N. and Doerr, J. (2015). A Systematic Literature Review of Requirements Prioritization Criteria. In *21st Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 300–317, Cham. Springer International Publishing.

[Ruhe 2005] Ruhe, G. (2005). *Software Release Planning*, chapter 13, pages 365–393.

[Saaty 1987] Saaty, R. W. (1987). The analytic hierarchy process: what it is and how it is used. *Mathematical Modelling*, 9(3):161–176.

[Saaty 1980] Saaty, T. L. (1980). *The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation*. McGraw-Hill.

[Santos et al. 2016] Santos, R., Albuquerque, A., and Pinheiro, P. R. (2016). Towards the Applied Hybrid Model in Requirements Prioritization. *Procedia Computer Science*, 91(1):909–918.

[Taylor et al. 2010] Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2010). *Software Architecture - Foundations, Theory, and Practice.* Wiley.

[Thurimella and Padmaja 2014] Thurimella, A. K. and Padmaja, T. M. (2014). Economic models and value-based approaches for product line architectures. In Mistrik, I., Bahsoon, R., Kazman, R., and Zhang, Y., editors, *Economics-Driven Software Architecture*, pages 11–36. Morgan Kaufmann, Boston.

[van Lamsweerde 2009] van Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley Publishing.

[Voda 2014] Voda, I. (2014). Migrating Existing PHP Web Applications to the Cloud. *Informatica Economica*, 18(4):62–72.

[Wang et al. 2009] Wang, H. J., Moshchuk, A., and Bush, A. (2009). Convergence of Desktop and Web Applications on a Multi-service OS. In *4th Conference on Hot Topics in Security (USENIX)*, pages 1–6, Berkeley, CA, USA. USENIX Association.

[Wohlin et al. 2012] Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. (2012). *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

[Xu et al. 2012] Xu, X., Zhang, B., and Lin, J. (2012). Management information system requirements analysis model based on the agile development. In *1st International Conference on Control Engineering and Communication Technology (ICCECT)*, pages 986–990, Liaoning, China. IEEE.

[Zampoglou et al. 2016] Zampoglou, M., Papadopoulos, S., Kompatsiaris, Y., Bouwmeester, R., and Spangenberg, J. (2016). Web and social media image forensics for news professionals. In Heravi, B. R. and Zubiaga, A., editors, *Social Media in the Newsroom, Papers from the 2016 ICWSM Workshop*, volume WS-16-19 of *AAAI Workshops*, Cologne, Germany. AAAI Press.

Appendix A Papers retrieved

Table 10. Papers retrieved from the literature review

Title	Authors	Reference
Scenario-based analysis of software architecture	Kazman, R., Abowd, G., Bass, L., and Clements, P.	[Kazman et al. 1996]
Mobile cloud computing service models: A user-centric approach	Huang, D., Xing, T., and Wu, H.	[Huang et al. 2013]
Desktop to cloud transformation planning	Beaty, K., Kochut, A., and Shaikh, H.	[Beaty et al. 2009]
Software requirement prioritization based on non-functional requirements	Garg, U. and Singhal, A.	[Garg and Singhal 2017]
Dynamic scaling of web applications in a virtualized cloud computing environment	Chieu, T. C., Mohindra, A., Karve, A. A., and Segal, A.	[Chieu et al. 2009]
Volunteer computing and desktop cloud: The Cloud@Home paradigm	Cunsolo, V. D., Distefano, S., Puliafito, A., and Scarpa, M.	[Cunsolo et al. 2009]
Entropia: architecture and performance of an enterprise desktop grid system	Chien, A., Calder, B., Elbert, S., and Bhatia, K.	[Chien et al. 2003]
Implementation of cloud computing on web application	Rewatkar, L. R. and Lanjewar, U. L.	[Rewatkar and Lanjewar 2010]

Appendix B Requirements cost and value

Table 11. Cost (%), value (%) and ratio (cost/value) of each requirement by CA.

	Desktop			Web			Cloud		
id	cost	value	ratio	cost	value	ratio	cost	value	ratio
1	19.6	29.4	0.66	24.5	7.9	3.10	18.0	22.8	0.78
2	3.5	4.3	0.81	6.2	8.1	0.76	35.3	4.6	7.67
3	16.5	18.9	0.87	14.8	22.7	0.65	10.3	17.8	0.57
4	3.5	4.4	0.79	3.1	4.3	0.72	3.6	3.6	1.00
5	9.4	2.7	3.48	12.0	3.7	3.24	6.1	7.2	0.84
6	15.2	13.1	1.16	21.3	26.1	0.81	17.4	18.8	0.92
7	32.3	27.2	1.18	18.1	27.2	0.66	9.4	25.2	0.37
8	8.4	26.0	0.32	4.5	19.9	0.22	5.1	11.3	0.45
9	13.7	19.9	0.68	14.1	14.3	0.98	5.7	21.0	0.27
10	13.2	12.3	1.07	11.5	7.0	1.64	9.5	3.8	2.50
11	4.6	10.3	0.44	5.2	7.1	0.73	6.5	5.6	1.16
12	10.2	14.3	0.71	10.2	25.8	0.39	15.8	33.7	0.46
13	38.1	3.2	11.90	28.6	3.3	8.66	38.0	15.7	2.42
14	2.7	3.9	0.69	9.0	5.4	1.66	5.9	3.0	1.96
15	9.2	10.0	0.92	16.9	17.3	0.97	13.6	5.9	2.30
16	62.9	55.5	1.13	30.3	24.1	1.25	11.6	57.7	0.20
17	22.6	27.5	0.82	38.9	18.8	2.06	23.4	5.0	4.68
18	10.2	13.5	0.75	13.0	33.1	0.39	57.9	19.6	2.95
19	4.2	3.4	1.23	17.8	24.1	0.73	7.2	17.7	0.40
20	4.2	43.8	0.09	14.8	15.2	0.97	22.3	11.7	1.90
21	8.7	12.6	0.69	8.0	5.1	1.56	12.3	7.9	1.55
22	55.1	4.4	12.52	43.2	4.8	9.00	3.5	5.0	0.70
23	15.4	10.9	1.41	4.0	21.9	0.18	17.8	21.8	0.81
24	9.3	19.4	0.47	17.2	28.3	0.60	22.9	21.5	1.06
25	7.3	8.9	0.82	12.8	24.7	0.51	21.2	32.1	0.66