



# Producing Just Enough Documentation: An Optimization Approach Applied to the Software Architecture Domain

J. Andrés Díaz-Pace<sup>1</sup> · Christian Villavicencio<sup>1</sup> · Silvia Schiaffino<sup>1</sup> ·  
Matías Nicoletti<sup>1</sup> · Hernán Vázquez<sup>1</sup>

Received: 31 August 2014 / Accepted: 17 September 2015 / Published online: 24 October 2015  
© Springer-Verlag Berlin Heidelberg 2015

**Abstract** The Software Architecture is an important asset in a software development process, which serves to share and discuss the main design concerns among the project stakeholders. The architecture must be properly documented (e.g., via a Wiki environment) to be effectively used by these stakeholders. However, the process of producing architecture documentation often fails to deliver contents that address the stakeholders' information needs. To address the problem, we argue for a knowledge management strategy in which: (i) architecture documentation is created incrementally; and (ii) its contents are driven by a model of stakeholder preferences. In this work, we present an information optimization approach applied to the architecture documentation domain, derived from an existing documentation method called Views & Beyond. To do so, we define the Next SAD Version Problem (NSVP) and then provide tool support to assist architects in producing cost-effective documentation. Based on prior work, we perform a sensitivity analysis of the optimization model and develop a robust formulation that takes into account uncertainty in the parameter estimations for NSVP

instances, thus improving the outcomes of our documentation assistant.

**Keywords** Architecture documentation model · Stakeholders · Information needs · Discrete optimization · Sensitivity analysis · Robustness

## 1 Introduction

As software systems grow large and complex, their reliance on some form of documentation becomes essential to avoid situations of knowledge vaporization or stakeholders' dissatisfaction, among others [40]. Furthermore, documentation serves to preserve knowledge within an organization. However, since producing technical documentation does not come without cost, both managers and software engineers must carefully consider the documentation process (e.g., artifacts, techniques, supporting tools) as an integral part of a development project, and furthermore, understand the information needs of the project stakeholders. In particular, a useful model for describing the high-level structure of a system (since early development stages) is the *software architecture* [5], which is the information domain explored in this work. The software architecture is also the container of the *main design decisions* for satisfying the *stakeholders' concerns* (e.g., performance, time-to-market, reliability, modifiability, cost-of-ownership, or usability, among others). The architecture is typically captured by the so-called *Software Architecture Document (or SAD)*. Conceptually, the SAD is an information repository that enables communication and knowledge sharing among the architecture stakeholders [14]. The SAD is commonly structured into sections that contain text and design diagrams—known as *architectural views*, which permit to

✉ J. Andrés Díaz-Pace  
andres.diazpace@isistan.unicen.edu.ar  
Christian Villavicencio  
christian.villavicencio@isistan.unicen.edu.ar  
Silvia Schiaffino  
silvia.schiaffino@isistan.unicen.edu.ar  
Matías Nicoletti  
matias.nicoletti@isistan.unicen.edu.ar  
Hernán Vázquez  
hernan.vazquez@isistan.unicen.edu.ar

<sup>1</sup> ISISTAN Research Institute, CONICET-UNICEN, Campus Universitario, Paraje Arroyo Seco (B7001BBO), Tandil, Argentina

understand and reason about the architectural solution from different perspectives.

Having a software system with multiple stakeholders poses challenges for the production of quality documentation. In the case of the SAD, a first challenge is that the architectural contents should target multiple readers, which might have different backgrounds and information needs [27]. For example, project managers are mainly interested in high-level module views and allocation views, whereas developers need extensive information about module views and behavioral views. Many times, the documenters<sup>1</sup> tend to load the SAD with development-oriented contents that only consider a few (internal) stakeholders. Studies [31, 47] have shown that individual stakeholder's concerns are addressed by a fraction (less than 25 %) of the SAD, but for each stakeholder a different (sometimes overlapping) SAD fraction is needed. A second challenge is the effort for creating and updating the SAD, an expenditure that developers and managers do not wish to bear. This can be due to budget constraints, tight schedules, or pressures on developing user-visible features.

In this context, we argue that the production of architecture documentation should be *planned*. This is a compromise between documenting those aspects that are useful to the stakeholders, and avoiding “too much” documentation, because the resources available are often limited. A *documentation strategy* to achieve this goal should: (i) produce architectural documents driven by the stakeholders' interests, and (ii) build the SAD in incremental versions concurrently with the design work. In this article, we propose a semi-automated approach called ProSAD for the generation of SAD contents, in which the documents are explicitly linked to the needs of its consumers (i.e., the stakeholders). To do so, we rely on the Views and Beyond (V and B) method [14], which provides a semantic model for understanding the relationships between stakeholders' interests and possible architectural views to be included in the SAD.

In previous work [38], the problem of choosing the most useful tasks for the next SAD version was cast as a discrete optimization problem, in which the objective was to maximize the SAD utility (or benefit) for the stakeholders without exceeding a cost constraint. In this work, a number of improvements to the basic approach are presented. First, we extend the optimization problem to include dependencies among the sections of the SAD. Second, a limitation of [38] was the assumption that the stakeholders had a fixed set of interests on the SAD contents. Thus, we here discuss a model for incorporating variations in the stakeholders' interests based on user profiling techniques [37]. Third, in [38], it was necessary to provide certain documentation costs and priorities of stakeholders as input parameters to the optimization model. In both cases, these parameters were based on

estimates or proxies (e.g., number of words of SAD sections), which were difficult to obtain due to the dynamic environment in which the software documentation process occurs. To deal with this situation, we perform a sensitivity analysis for identifying the most influential parameters in the optimization output. Based on this analysis, we then propose a robust formulation of NSVP, which naturally sacrifices optimality in the utility results in exchange for less variations due to uncertainty in parameter estimates.

The extensions proposed above are evaluated with a real-life example of a SAD. Although the proposed approach is exercised within the software architecture domain, we believe that it also applies to other content generation domains in which the contents to be delivered must be connected to the interests of their readers.

The rest of the article is organized into 6 sections. Section 2 presents the main concepts behind ProSAD, based on the principles of the V and B method. Section 3 covers the original formulation of the optimization problem called NSVP and explains its extension with dependencies. Section 4 presents a global sensitivity analysis, and then proposes a robust formulation of the optimization. Section 5 provides details about the generation of stakeholders profiles, which feed the SAD optimization performed by ProSAD. Section 6 discusses related work, and performs a comparative analysis of documentation approaches. Finally, Sect. 7 gives the conclusions and outlines future work.

## 2 On the Architecture Documentation Process

The software architecture of a computing system is the set of structures needed to reason about that system, which comprise software elements, relations among them, and properties of both [5]. Design decisions are an essential part of the architecture, because they record the rationale behind the solution developed by the architects [28]. Examples of decisions are the use of certain patterns, such as layers or client-server, to meet modifiability or performance qualities. The architecture serves as a blueprint in which the main concerns of the stakeholders can be discussed, at an abstraction level that is reasonably manageable, even for non-technical stakeholders. By *stakeholder* [34], we mean any person, group or organization that is interested in or affected by the architecture (e.g., managers, architects, developers, testers, end-users, contractors, auditors). To share the architecture knowledge among the stakeholders, it must be adequately documented and communicated. The Software Architecture Document (SAD) is the usual artifact for capturing this knowledge. The SAD should be clear in explaining: (i) how the functional requirements are fulfilled by the responsibilities assigned to the different software components, (ii) how the component structures satisfy the different quality

<sup>1</sup> This role is usually played by members of the architecture team.

attributes requirements, and (iii) how restrictions and business goals are met by the architectural solution. The SAD format can range from Word documents to UML diagrams within a CASE tool, or a collection of Web pages hosted in a Wiki. Over the last years, there has been an increasing interest in using Wikis to host the SAD and facilitating the access to the architectural knowledge [2, 22].

The architecture documentation is generally structured around the concept of *architectural views*, which represent the many structures that are present simultaneously in software systems. A view presents an aspect or viewpoint of the system (e.g., static aspects, runtime aspects, allocation of software elements to hardware). Typical examples of views are: module views (the units of implementation and their dependencies), component-and-connector views (the elements that have runtime presence and their pathways of interaction), or allocation views (the mappings of software elements to hardware). In addition, architectural views are accompanied by textual sections that describe the main design decisions related to the views. Therefore, the SAD consists of a collection of documents with textual and graphical contents, structured according to predefined templates. Figure 1 shows a snapshot of a Wiki-based SAD, with the main page (SAD index) and module and deployment views.

The stakeholders are important actors in the documentation process, as they are the main consumers of the SAD. Moreover, a SAD is useful as long as its contents satisfy the *stakeholders' information needs*. In a typical development, the architecture is the result of an iterative design process [5], in which the solution is designed (and assessed) incrementally until it is stable enough to proceed downstream with the implementation efforts. Since this process must be supported by appropriate documentation, the documentation work must go hand in hand with the design work. Along this line, we argue that the SAD should be delivered in incremental ver-

sions to the stakeholders. This premise is followed by our ProSAD approach, based on the theoretical underpinnings provided by the Views and Beyond method.

## 2.1 The Views and Beyond Method

Views and Beyond (V and B) [14] is an architecture documentation method developed by the Software Engineering Institute. Like other documentation methods, V and B is view-centric. Normally, several views are required to fully describe the architecture of a system. The basic V and B principle is that documenting a software architecture involves documenting the *relevant views*, and then documenting the information that applies to more than one view (e.g., relations between a module view and component-and-connector view, or mappings between architectural drivers and design decisions). Furthermore, V and B helps the architect identify and record the necessary architectural information during development, by providing a general SAD template and specific templates for views. In fact, the Wiki excerpt from Fig. 1 adheres to the V and B templates.

The choice of the relevant views for the SAD depends on its anticipated usage by the stakeholders. It is well known that documentation should be written from a readers' perspective rather than from a writers' perspective. In fact, it is a recommended practice of current standards for architectural documentation, such as the IEEE 1471-2000 [26] and the ISO/IEC/IEEE 42010 [27]. To produce reader-oriented documentation, stakeholders' interests with respect to architectural documentation are assessed prior to the delivery of each SAD version. To this end, V and B characterizes several types of stakeholders regarding their use of architectural views, as depicted by the matrix of Fig. 2. A cell of the matrix indicates the information of view X (e.g., decomposition, deployment) needed by stakeholder role Y (e.g., developer,

**Fig. 1** Example of a Wiki-based SAD for a system called AdventureBuilder

**Fig. 2** V and B characterization of stakeholder preferences on architectural views [14]

	TYPES OF ARCHITECTURAL VIEWS															
	Module Views	C&C Views	Allocation Views	Other Documentation												
	Decomposition	Uses	Generalization	Layered	Data Model	Various	Deployment	Implementation	Install	Work Assignment	Interface Documentation	Context Diagrams	Mapping Between Views	Variability Guides	Analysis Results	Rationale and Constraints
Project managers	s	s	s			d	d		d		o	d	d	d	s	s
Members of development team	d	d	d	d	d	d	s	s	d		d	d	d	d	s	s
Testers and integrators	d	d	d	d	d	s	s	s	s		d	d	s	d	s	s
Designers of other systems						s					d	o				
Maintainers	d	d	d	d	d	d	s	s			d	d	d	d	d	d
Product-line application builders	d	d	s	o	s	s	s	s	s		s	d	s	d	s	s
Customers							o			o	o	o	o		s	
End users						s	s		o						s	
Analysts	d	d	s	d	d	s	d	s			d	d	s	d	s	s
Infrastructure support personnel	s	s		s	s	d	d	d	o						s	
New stakeholders	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Current and future architects	d	d	d	d	d	d	d	s	d	s	d	d	d	d	d	d

Key: d = detailed information, s = some details, o = overview information, x = anything

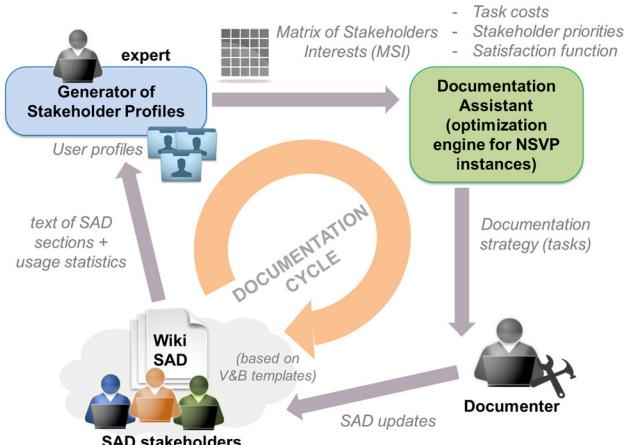
maintainer, manager). Each column can be seen as stakeholder preferences on the contents of a particular SAD view.

We should note that not every stakeholder requires the same level of detail of the architectural views (e.g., detailed information, some details, overview information, anything). Furthermore, not every stakeholder generally has the same relevance within a given project: some of them will naturally have a higher level of importance than others. This aspect needs to be considered when planning the SAD contents. Thus, V and B recommends to customize the matrix based on specific characteristics of the target project (e.g., quality attributes driving the system, project size, project criticality, community of stakeholders to be served by the SAD, role of SAD within the development process).

The emphasis of V and B on planning for the SAD contents makes it suitable to our work. Still, an adoption barrier is that practitioners often view V and B as a heavy-duty method, due to the amount of documentation (or bureaucracy) imposed on the documenter. We believe that a way of lowering this barrier is via “intelligent” tools that operationalize the prescriptions of documentation methods. The approach described next is an example of that position.

## 2.2 The ProSAD Approach

ProSAD (PeRsonalization and Optimization approach to assist the production of Software Architecture Documentation) is a documentation approach centered around the interactions between the software architecture (of a system) and its stakeholders. The main objectives are: (i) to capture the architectural interests (or concerns) of the stakeholders, (ii) to assist the documenter in the production of architec-



**Fig. 3** ProSAD: an architecture documentation driven by stakeholders' concerns

tural documentation (i.e., the SAD), and (iii) to provide tool support for the two previous goals. The general schema of ProSAD is shown in Fig. 3. The main actors are: the documenter (or SAD writer), the stakeholders (or SAD readers), and an expert that manages the generation of stakeholders' profiles. The approach follows an iterative and incremental cycle, which relies on a matrix called MSI (Matrix of Stakeholders Interests) inspired in that of Fig. 2. The information of the MSI can be derived from multiple sources, such as: literature guidance for predefined stakeholder types, or user profiling techniques applied to stakeholders [36].

In an ideal setting, the documenter could take the MSI, analyze it (along the guidelines of V and B), and then produce a version of the SAD in which its contents address all the stakeholders' concerns. Unfortunately, this is seldom the case

in practice, because the resources to invest in documentation are often scarce and because conflicts between the stakeholders' interests may arise. Therefore, for each incremental version of the SAD, the documenter must decide which delta of architectural contents brings most satisfaction (or utility) to the whole set of stakeholders. We refer to these decisions as a *documentation strategy* (or update plan). At the heart of ProSAD, we have an intelligent assistant that takes the stakeholders' profiles (from the MSI) and recommends a documentation strategy to the documenter. This strategy consists of set of documentation tasks for different SAD sections. The assistant is backed up by a discrete optimization engine, as detailed in Sect. 3. The interests of stakeholders can drift over time, depending on the type of project but also on the natural learning of human beings. To account for this situation, ProSAD relies on a profiling component, which monitors the activity of the stakeholders on the SAD and constructs user profiles that periodically update the MSI, as explained in Sect. 5.

The assistant is able to process the Wiki-based SAD along with the MSI and show a backlog of high-priority documentation tasks to the documenter. This backlog is implemented with a to-do-list metaphor. When a specific task is chosen, the tool opens the corresponding SAD section in the Wiki and provides writing guidelines to the documenter. Examples of these guidelines are: expected contents of the section, or a brief description of the task effect on the current stakeholders' concerns. The execution of the suggested tasks on the SAD is not mandatory. The documenter makes the final documentation decisions based on her architecture expertise, domain knowledge, or past experiences.

### 3 The Next SAD Version Problem

In [17], we defined the so-called Next SAD Version Problem (NSVP) as an optimization formulation that models the transition from one given SAD version to the next one. We assume that the increment between the current SAD and the next one is the result of applying what we call documentation tasks. A *documentation task* takes a SAD section, which typically contains one architectural view, and increases the level of detail (or completion) of that section. For example, a task can take an empty section and add contents to it (e.g., adding a "Primary presentation" according to the V and B view template). Another task can take an overview section and make it more detailed (e.g., including an "Element catalog" in V and B terminology). From the documenter's perspective, these tasks are units of work stored in some backlog, from which she can pick and apply the most relevant ones to the current SAD. In the context of a model of stakeholder preferences, like the one of Fig. 2, updating a SAD section through certain

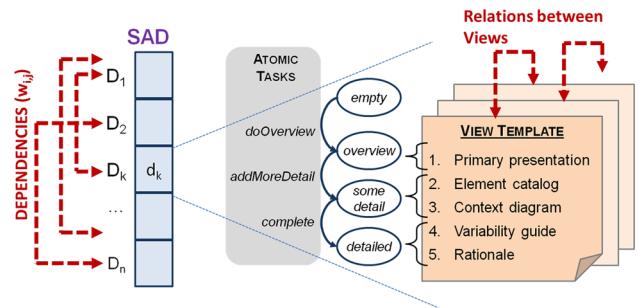
task(s) makes direct contributions to the stakeholders' needs. This is a key point of our approach.

In an ideal setting, the documenter would analyze the stakeholder profiles and then perform all the tasks necessary for fulfilling their preferences, which in turn would produce a new (satisfying) SAD. Unfortunately, this is seldom the case, because the choice of tasks must consider factors such as: documentation efforts allocated to the tasks (e.g., person-hours), priorities of the stakeholders, or the need of incremental SAD versions to support people's work, among others. Furthermore, dependencies between certain sections must be considered. For instance, if a section has a detailed level of a part of the system, V and B recommends to include also a section with an overview of the same part, so that the SAD readers can easily navigate from overview to detail. These kind of relations imply dependencies between the tasks, and were not covered in [17, 38]. Therefore, for each SAD increment, the documenter must decide which tasks from the backlog will change SAD sections that best satisfy the stakeholders' needs. In this article, we deal with the NSVP\_B variant for NSVP, which can be stated as follows:

$$\text{maximize } \text{Benefit}(\Delta) \quad (1)$$

$$\text{subject to } \text{Cost}(\Delta) \leq \text{MaxCost} \quad (2)$$

Let us consider the SAD as an artifact that contains  $N$  sections (Wiki pages), each one associated to a predefined view template (from V and B). Let  $\text{SAD}_t = \langle d_1^t, \dots, d_N^t \rangle$  be a SAD version at time  $t$ , in which each position of the vector corresponds to a section (or document). In this vector,  $d_k$  ( $1 \leq k \leq N$ ) is its level of detail at time  $t$ . We assume a discretization of the possible completion states of document  $k$ . In particular, a document can be in one of 4 possible states, namely  $\text{DS} = \{\text{empty}, \text{overview}, \text{someDetail}, \text{detailed}\}$ , as illustrated in Fig. 4. These states should not be interpreted as a strict order of documentation but rather as a guideline for the documenter, based on the relative importance of the sub-sections of the view template [13]. For instance, a documenter might begin adding "Rationale information", but it



**Fig. 4** Documentation tasks for a SAD section, based on the V and B view templates

is recommended that she works first on the “Primary presentation” of her solution, and then describes its main elements, before providing a rationale of the solution.

Given a partially documented SAD<sub>t</sub>, the documenter must select an update plan to produce a SAD<sub>t+1</sub> with additional contents. Let us consider an arbitrary next version SAD<sub>t+1</sub> =  $\langle d_1^{t+1}, \dots, d_N^{t+1} \rangle$ . We define an increment vector  $\Delta = \langle x_1, \dots, x_N \rangle$  such that  $x_i = d_i^{t+1} - d_i^t$  (with  $x_i \geq 0$ ). Note that we assume that increments are always additive, but not all sections  $d_i$  will be necessarily updated in the next version. Based on DS, we have a list with all possible tasks applicable to SAD<sub>t</sub>. That is,  $L = \{a_{11}, a_{21}, \dots, a_{N1}, \dots, a_{4N}\}$  where  $a_{jk}$  ( $1 \leq j \leq 4, 1 \leq k \leq N$ ) is a feasible task for document  $D_k$  that leads to a state change  $d_k \rightarrow d'_k$ . The feasible tasks (i.e., applicable) depend on the current state of the SAD sections. For instance, a view can be refined with information from the sections “Element catalog” and “Context diagram”, which implies a transition from overview to someDetail by means of addSomeDetail. Note that we do not model “undo” tasks.

Let AT = {doOverview, addMoreDetail, complete} be a set of atomic tasks. An atomic task, if applicable, increases the detail of a SAD section. This effort is quantified as the cost of the (atomic) task. Furthermore, we map these tasks to the parts of the view template provided by V and B, as depicted in Fig. 4. Task doOverview takes an empty section and asks (the documenter) to fill in the “Primary presentation” part of the view. Task addMoreDetail takes a section (having already an overview state) and tries to add information in the “Element catalog” and “Context diagram” parts. At last, task complete takes a section (having already some detail) and fills out the pending parts: “Variability guide” and “Rationale”. Note that this framework can be extended with other kinds of tasks, or support other templates with different parts. Atomic tasks can be arranged in sequences to derive composite tasks. For example, if a section is in overview, it can only go to states someDetail or detailed by means of the task sequences  $\langle \text{addMoreDetail} \rangle$  (atomic task) or  $\langle \text{addMoreDetail}, \text{complete} \rangle$  (composite task), respectively. Along this line, the candidate tasks for a section  $d_k$  are all the allowed task sequences derivable from AT that follow the state chain doOverview → addMoreDetail → complete. This chain models the meaningful order in which atomic tasks should be performed on the V and B view template. With 4 atomic tasks and no undo, we have a total of 6 possible tasks in L (not all of them are shown in Fig. 4, for the sake of clarity).

On one hand, the cost of a state change  $d_k \rightarrow d'_k$  in a SAD section is assumed to be a fixed quantity. Then, we have a cost vector  $C_\Delta = \langle c_1, \dots, c_N \rangle$  with  $c_k = \text{cost}(d_k, d'_k)$  and the total cost of an increment  $\Delta$ , denoted by Cost( $\Delta$ ), is the sum of the individual costs of changing each section (or

document). If  $d_k = d'_k$ , a zero cost is assigned. Cost( $\Delta$ ) represents the *production cost of the next SAD version*. The cost for making the transition  $d_k \rightarrow d'_k$  is dependent on the costs of the atomic tasks being applied to the document. We assume an “atomic” cost associated to each transition in the state sequence of Fig. 4. An atomic cost denotes the (documenter’s) effort of updating document  $k$  with current detail  $i$  to its next consecutive level  $i + 1$ . For a transition between not consecutive states, we use a “composite” cost equal to the sum of the atomic costs across the transition. Certainly, estimating the costs of writing SAD sections is a subjective activity. One proxy for estimating such costs is the number of words. For instance, if a document has 1000 words and has a 100 % of completeness, the atomic cost for the 3 tasks in AT can be  $c = \frac{1}{3}$ .

On the other hand, the benefit (or expected utility) of an increment  $\Delta$  is a function of the vectors SAD<sub>t</sub> and SAD<sub>t+1</sub>, but it also depends on the stakeholders’ preferences on the SAD contents. Similar to the cost formulation, we assume a benefit vector  $B_\Delta = \langle b_1, \dots, b_N \rangle$  with  $b_k = \text{benefit}(d_k, d'_k, \text{satisfaction}_k(S))$ , in the range [0, 1] (0 means no utility, and 1 means high utility). Given a set of  $M$  stakeholders  $S = \{S_1, \dots, S_M\}$ , satisfaction<sub>k</sub>(S) captures the combined preferences of all stakeholders on state change  $d_k \rightarrow d'_k$ . In other words,  $b_k$  is the “happiness” of the stakeholders (as a whole) with an increased detail in section  $k$  of the SAD. Then, Benefit( $\Delta$ ) is computed as the sum of the benefits through all the sections, and it gives a measure of the stakeholders’ *utility with the next SAD version*.

$$\begin{aligned} \text{Benefit}(\Delta) &= \sum_{k=1}^M b_k \\ &= \sum_{k=1}^M \text{benefit}(d_k, d'_k, \text{satisfaction}_k(S)) \end{aligned} \quad (3)$$

### 3.1 Dependencies Among SAD Sections

To model dependencies between SAD sections in NSVP\_B, we need to consider the different types of sections (i.e., architectural views) as well as their completion states. In experiments with SAD versions, if we consider that the required dependencies between sections are met in the next SAD version, we can have drops of up to 20 % in the (maximized) benefit, when compared to optimizations without considering those dependencies. In Fig. 4, the dotted lines exemplify dependencies between sections (or views), which correspond to dependencies between elements of the vector SAD<sub>t</sub> in the mathematical formulation. For the case of  $N$  documents each one in 4 possible states, let us consider  $N$  matrices of dimensions  $4 \times 4$ , and each cell of a matrix has a weight  $w_{i,j}^k = 1$  if the resulting SAD has the required

dependency  $d_k \rightarrow d'_p$  between documents  $k$  and  $p$  with states  $d_k \rightarrow d'_p$ , respectively. Otherwise  $w_{i,j}^k = 0$ , in case of no required dependencies between documents, or alternatively, if the documents involved in the dependency are empty. Along this line, we can add a second objective Dependencies( $\Delta$ ) grouping all the weights to in Eqs. 1 and 2. The closer Dependencies( $\Delta$ ) is to 1 (considering a normalization based on the number of documents of the SAD), the lesser the violations of dependencies in the resulting SAD. Thus, we have the following:

$$\text{maximize } \text{Dependencies}(\Delta) = \left( \sum_{k=1}^N \sum_{i=1, j=1}^4 w_{i,j}^k \right) / (4 * N) \quad (4)$$

We refer to this new formulation to as NSVP\_BD (“D” for Dependencies) and the problem becomes a bi-objective optimization. To solve NSVP instances (i.e., find an optimal SAD documentation strategy), discrete optimization techniques can be applied [18], based either on exact or heuristic algorithms. In our case, the number of SAD documents ( $N$ ) is the main contributor to problem size, affecting the choice between exact or heuristic algorithms. Real-life SAD sizes have typically 15–40 documents, depending on how critical the architecture is for the system (and hence, its documentation). In previous work [17], we explored an implementations based on SAT4J [6], and an heuristic implementation based on the NSGA-II algorithm [16] provided by the MOEA framework<sup>2</sup>. In this article, we work with solutions generated with NSGA-II, which is a well-known genetic algorithm for multi-objective optimization [16]. In short, NSGA-II uses an evolutionary process with operators such as selection, genetic crossover and genetic mutation, applied to the document state representation of NSVP. An initial population of vectors  $\Delta$  is evolved through several generations.

### 3.2 Types of Satisfaction Functions

When it comes to the utility associated to tasks (either atomic or composite ones) in  $L$ , we compute the utility indirectly on the basis of the stakeholders’ interests over the SAD sections. The benefit obtained from a particular section is given by  $b_k = \text{benefit}(d_k, d'_k, \text{satisfaction}_k(S))$ . More specifically, computing each benefit requires the specification of (i) a function  $\text{satisfaction}_k(S)$  and (ii) a procedure to aggregate individual satisfactions into a single value. For every SAD section, a stakeholder can prefer any state in  $DS = \{\text{empty}, \text{overview}, \text{someDetail}, \text{detailed}\}$ . These preferences actually come from the MSI of ProSAD. Note here that empty

is interpreted as the stakeholder being “not interested” in the document. For estimating  $\text{satisfaction}_k(S)$ , we depart from the assumption that a stakeholder knows the “perfect” level of detail required for a section, based on her own information needs and the expected information to be conveyed by the architectural view of the section. This knowledge is modeled by functions  $\text{satisfaction}_k(S) : DS \times DS \rightarrow [0, 1]$ , which depend on both the actual and preferred completion states of a document. Based on our experience with architectural documentation projects, we propose three candidate functions, as described in Fig. 5. Like in [38], note that other types of satisfaction functions are also possible.

Function A (exact-or-nothing) gives maximal satisfaction (1.0) when the current detail of the document matches exactly the stakeholder preference, and 0.0 satisfaction otherwise. Function B (more-is-fine) proportionally increases the satisfaction value as the current detail of the document gets closer to the stakeholder preference, and beyond that point the satisfaction gets the maximal value (1.0). This reflects the situation in which the stakeholder does not care having more detail than required. Function C (more-can-be-penalized) is a variant of Function B. It begins with a proportional increase until the document detail matches the stakeholder preference, but for higher detail than required the satisfaction value decreases slightly. This situation would happen when the stakeholder is overwhelmed by an excess of information. In all functions, we set  $\varepsilon = 0.1$  as the “allowed difference” for a matching between a preference and a document state  $d_k$ . Eliciting the right satisfaction function of a stakeholder is not trivial, and it is out of the scope of this work.

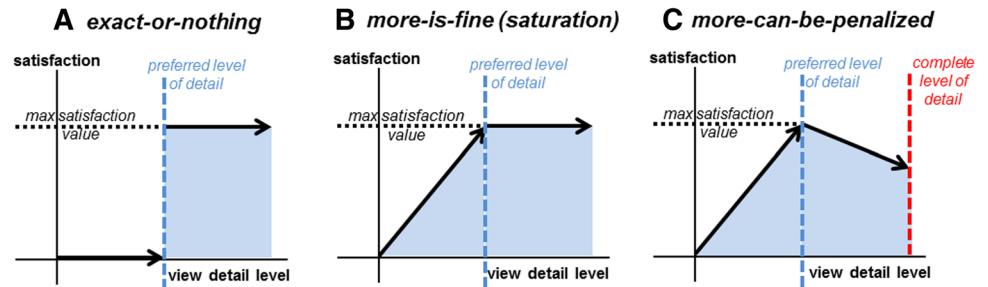
After applying the satisfaction functions above, we obtain a vector  $\text{satisfaction}_k(S) = \langle s_k(S_1), \dots, s_k(S_M) \rangle$  with  $s_k(S_i) \in [0, 1]$ . Examples of satisfaction vectors computed with functions A, B, and C for 5 stakeholders and a single document are shown in Fig. 6. Note that two (or more) stakeholders might have competing preferences on the same document, which cannot be solved by means of the satisfaction functions (except perhaps when Function B is used). This tradeoff situation means that selecting a detail level for a document might satisfy some stakeholders but might just partially satisfy others. In our model, the aggregation of the stakeholder satisfaction is computed with a weighted average based on stakeholders’ priorities. Specifically, we assign each stakeholder  $S_i$  a priority  $p_i$ , in the range [0, 10], where 0 is the lowest priority and 10 is the highest one. This priority can be defined by the role that the stakeholder plays in the project.

## 4 Towards a Robust Formulation of NSVP-BD

A limitation of the previous formulation is that both the costs (of atomic tasks) and the priorities of the stakeholders

<sup>2</sup> <http://moeaframework.org/>

**Fig. 5** Satisfaction functions based on stakeholders preferences over SAD sections



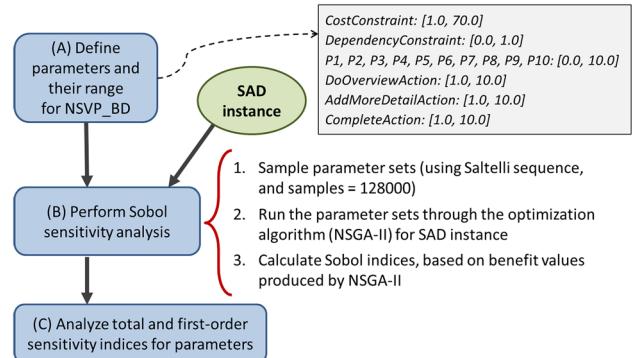
Document k (someDetail)	S <sub>1</sub>	S <sub>2</sub>	S <sub>3</sub>	S <sub>4</sub>	S <sub>5</sub>
Preference	overview	detailed	detailed	someDetail	empty
function A →	0.00	0.00	0.00	1.00	0.00
function B →	1.00	0.75	0.75	1.00	1.00
function C →	0.33	0.75	0.75	1.00	0.00

**Fig. 6** Example of converting stakeholder preferences to satisfaction values

(for the calculation of the aggregate satisfaction function) need to be known in advance. Although these parameters can be estimated by the documenter (or by an expert), the estimations are subject to uncertainties. Furthermore, when solving NSVP instances using optimization algorithms such SAT4J or NSGA-II, we observed that different choices for costs and priorities often led to variations in the (maximal) benefit of the SAD. For this reason, we were interested in applying robust optimization techniques to NSVP. Robust optimization is an operational research framework that identifies and quantifies uncertainty in optimization problems [7]. This framework admits that some aspects (or parameters) are uncertain, and builds up models which seek robust solutions, i.e., even in the presence of noisy input data, the models can produce solutions with good quality. However, this desired robustness generally comes with some loss in solution quality, which is usually known as the “price of robustness” [8]. As a prerequisite for such a robust formulation, it is necessary to know what parameters of the NSVP optimization have the greatest influence on the maximal benefit. One way of doing this study is via sensitivity analysis [43]. In particular, we used *global sensitivity analysis*, as all parameters of interest can be varied simultaneously over the entire parameter space.

#### 4.1 Global Sensitivity Analysis

A well-known method is Sobol’s [46], which applies variance decomposition techniques to provide a quantitative measure of the contribution of the inputs to the output variance. The decomposition of the output variance employs the principles of factorial design, but the analysis is not intended to identify the causes of the input variability. It just indicates to what extent each parameter will contribute to the model output. A nice feature of Sobol’s method is that no assumptions are



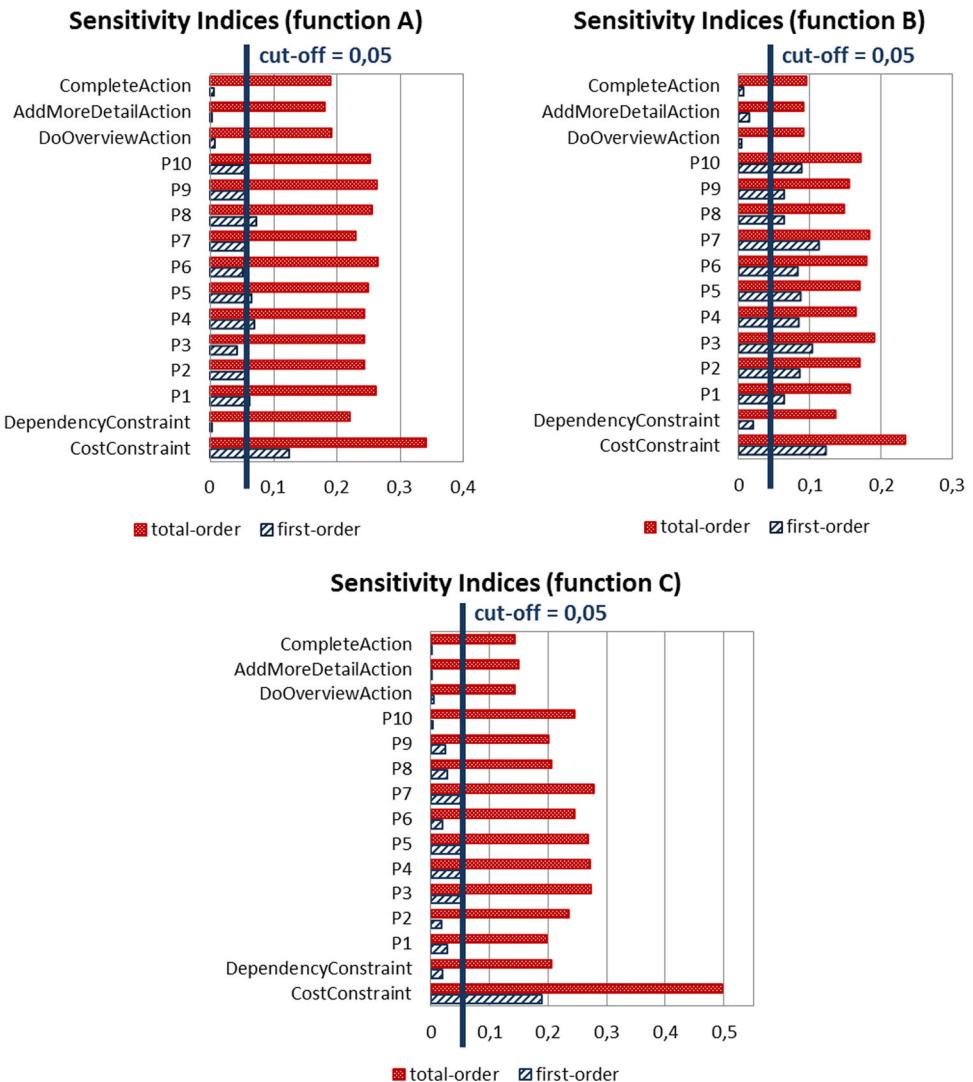
**Fig. 7** Steps of the Sobol sensitivity analysis applied to NSVP\_BD

made about the model inputs and output. The application of Sobol’s sensitivity analysis is summarized in Fig. 7. First, a sample of parameter values is generated. Second, the parameters are run through the model. In our case, the model is an optimization algorithm for NSVP. Third, the so-called Sobol indices are computed. There are two types of indices of relevancy to our work: total-order and first-order indices. First-order sensitivity indices reflect the “main effect”, and are used to measure the fractional contribution of a single parameter to the output variance. Total-order indices, in turn, take into account the main, second-order and also higher order effects of the parameters on the output variance. The interpretation of the analysis is that the higher the value of sensitivity indices, the more influential the respective parameters for the model. Usually, a threshold of 0.5 is used to distinguish important (i.e., most sensitive) parameters in complex models. In addition, the most sensitive parameters should have a narrow confidence intervals in the calculation, normally less than 10 % of the indices values. For our work, we employed the implementation of the Sobol sensitivity analysis provided by the MOEA framework.<sup>3</sup>

Since the Sobol analysis can only be performed on a single objective, we preferred to tackle the benefit objective, and cast the dependencies objective as a constraint. For the purposes of sensitivity analysis, we defined  $MinDependency \in [0, 1]$  so that Eq. 4 becomes  $Dependencies(\Delta) \geq Min$

<sup>3</sup> <https://waterprogramming.wordpress.com/2012/08/13/running-sobol-sensitivity-analysis-using-moeaframework/>.

**Fig. 8** Sobol sensitivity indices computed for the CTAS SAD



Dependency. Both MaxCost and MinDependencies were treated as parameters in the analysis (see *CostConstraint* and *DependencyConstraint* in Fig. 7, respectively).

Figure 8 shows the results of the sensitivity analysis performed on a real-life SAD, called Clemson Transit Assistance System<sup>4</sup> (CTAS). The study involved an input set of 16 parameters and the usage of the 3 satisfaction functions above. For simplicity, we assumed that each of 3 atomic tasks had the same cost for all the SAD documents. Initially, the CTAS SAD had 20 documents (many of them nearly empty) and a completion level of 14 %. For this SAD, we had a set of 10 stakeholders with varying priorities. In the analysis of first-order effects, we noticed that the *CostConstraint* per SAD increment (MaxCost in Eq. 2) is the most important parameter ( $\text{index} \geq 10\%$ ), and then a moderate influence of the priorities of stakeholders

( $P1$  to  $P10$ ). When it comes to the analysis of total-order effects, the *CostConstraint* turned to be the most important parameter as well ( $20\% \leq \text{index} \leq 50\%$ ). An interesting observation here was that the choice of the satisfaction function (either A, B or C) somehow affects the contribution of *CostConstraint*. The remaining parameters, including the costs of tasks (*CompleteAction*, *AddMoreDetailAction*, *DoOverviewAction*), showed a moderate influence from the perspective of total effects. Nonetheless, since their respective indices were always in the range 10–20 %, they must be considered as a source of variations in the benefit. We believe that the costs of tasks had a greater (and significant) index for total effects than for first-order effects because these costs are dependent on the maximum allowed cost (*CostConstraint*). In all the cases, we should also notice a marginal influence of *DependencyConstraint* on the output.

A first conclusion of the previous analysis is that the value of  $\text{MaxCost}$  chosen for a SAD increment affects the attainable benefit when maximizing  $\text{Benefit}(\Delta)$ . This relation makes

<sup>4</sup> <http://people.cs.clemson.edu/johnmc/courses/cpsc875/resources/Telematics>

sense if we recall that NSVP can be seen as a variant of the so-called knapsack problems. Briefly, the knapsack problem can be stated as follows: given a set of items, each with a weight and a value, one must determine the number of each item to include in a collection so that the total weight is less than a given limit and the total value is as large as possible. Like in NSVP, a very common problem is the 0–1 knapsack problem, which restricts the number of copies of each kind of item to either zero or one. However, a less obvious conclusion for NSVP is that, to reduce possible variations in  $\text{Benefit}(\Delta)$  due to adjustments of the (other) parameters of NSVP, we should try to control: (i) the priorities assigned to the stakeholders, and (ii) the costs of the documentation tasks. In the following, we extend the optimization model of Sect. 3 to incorporate different types of uncertainties.

## 4.2 Adding Robustness to Stakeholder Priorities and Costs

The first robustness strategy has to do with our measure of the stakeholders' satisfaction. This measure can be uncertain, as it depends on the satisfaction function chosen and also on the priorities assigned to the stakeholders. As discussed in [39], this type of uncertainty can be quantified in a discrete and probabilistic way, using the concept of *scenarios* [50]. In this context, a scenario comprises a set of values that represent the occurrence of certain events. In our case, these events have to do with varying priorities for the stakeholders in  $\text{satisfaction}_k(S)$ . More formally, let  $\text{Scenarios} = \{e_1, e_2, \dots, e_Q\}$  where each scenario is represented by  $e_i = \{p_1^{ei}, p_2^{ei}, \dots, p_M^{ei}\}$  with  $p_k$  being an arbitrary stakeholder priority in scenario  $e_i$ . Each  $e_i$  leads to an aggregate value  $\text{satisfaction}_k^e(S)$ . In the assignment of possible values to the scenarios, we need to consider the probability of those events actually taking place. For each scenario  $e_i$ , it is defined an occurrence probability  $r_{ei}$ , with  $\sum_{i=1}^Q r_{ei} = 1$ . Thus, we can re-write Eq. 3 and substitute the term  $\text{satisfaction}_k(S)$  by a set of scenarios:

$$\begin{aligned} & \text{Benefit}(\Delta) \\ &= \sum_{e=1}^Q \left( \sum_{k=1}^M \text{benefit}(d_k, d'_k, \text{satisfaction}_k^e(S)) \right) * r_e \end{aligned} \quad (5)$$

Note that Eq. 5 can be seen as a generalization of Eq. 3. If we consider a single scenario  $e$ , then we obtain Eq. 3.

The second robustness strategy refers to the documentation costs, which have a kind of uncertainty that is different from that of stakeholder satisfaction. Here, we quantify the uncertainty related to cost in a deterministic and continuous way, as done in [39]. In addition to the cost  $c_i$  of each (atomic) task, we consider a value  $c_i^\varepsilon$  that indicates the maximum expected cost variation. In other words, a given cost

can be in the range  $[c_i - c_i^\varepsilon, c_i + c_i^\varepsilon]$ . Along this line, we can write  $\text{Cost}(\Delta)$  as:

$$\text{Cost}(\Delta) = \sum_{k=1}^N c_i + \sum_{k=1}^N c_i^\varepsilon \quad (6)$$

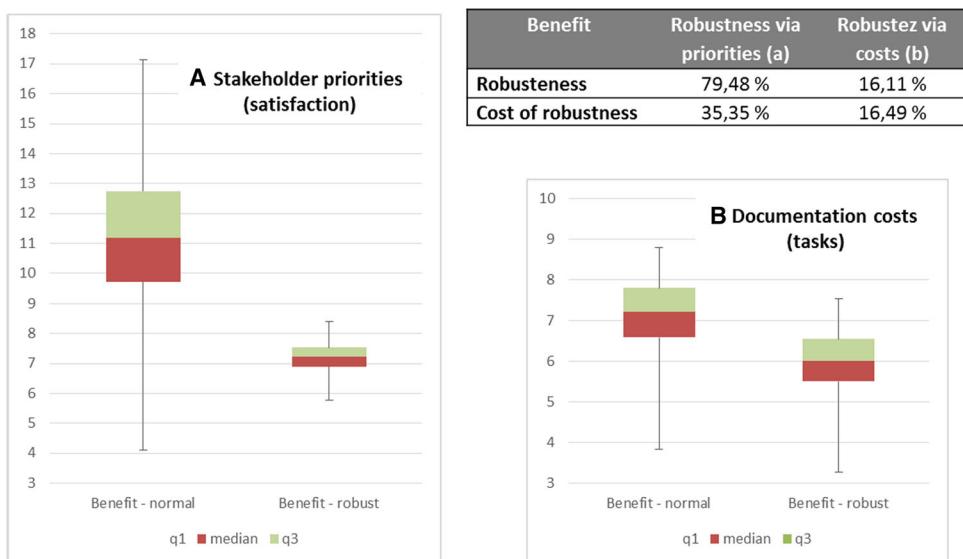
The cost calculation above guarantees that, even when all the costs reach their upper bounds  $c_i^\varepsilon$ , the cost restriction will be satisfied. This schema is the most conservative one, because all the costs have worst-case values. A more realistic schema is to consider only some worst-case costs. If  $\rho = c_i^\varepsilon/c_i$  is the percentage of cost variation, we can define a function  $\text{worstCases}(\delta, \rho)$  that sums up only the  $\delta$  worst cases for a given  $\rho$ . Equation 6 becomes:

$$\text{Cost}(\Delta) = \text{worstCases}(\delta, \beta) + \sum_{k=1}^N c_i^\varepsilon \quad (7)$$

Overall, the robust formulation of NSVP\_BD becomes a matter of substituting Eqs. 1 and 2 by Eqs. 5 and 7, respectively. Since the parameter *DependencyConstraint* did not seem to be influential in our sensitivity analysis, we did not consider a robustness strategy for the objective  $\text{Dependencies}(\Delta)$ .

Figure 9 shows the maximal benefit obtained in the robust and non-robust (normal) formulations of NSVP\_BD, for our two strategies: (i) stakeholder priorities, and (ii) documentation costs. We ran our NSGA-II algorithm for solving 200 samples of the CTAS SAD and then computed metrics such as: average benefit, average benefit variance, percentage of robustness (gain), and price of robustness (loss). The priorities and probabilities of the different scenarios were randomly generated. The percentage of robustness is the difference between the variance of the robust distribution ( $\sigma_R^2$ ) and that of the non-robust one ( $\sigma^2$ ). That is,  $\%robustness = \sigma^2 - \sigma_R^2 / \sigma^2 * 100$ . The “price of the robustness” is the difference between the average of the robust distribution ( $\mu_R$ ) and the non-robust one ( $\mu$ ), divided by the former. That is,  $\text{priceOfRobustness} = \mu - \mu_R / \mu * 100$ . In Fig. 9, the gains in robustness are shown graphically by the height of the rectangles (variance) and by the lower median values. We note that the first strategy (a) proved to be the most effective one, as it contributed to significantly reduce the variance in the benefit by sacrificing a 35 % of its quality. In the second strategy (b), the variance was slightly reduced when compared to the normal case, with a minimal cost of robustness. This analysis suggests that the treatment of the documentation costs (on their own) seems not very influential in the robustness results. Based on the differences between total-order and first-order effects for documentation costs in Fig. 8, we conjecture that this strategy (b) should be enhanced with other parameters.

**Fig. 9** Benefit obtained with and without robustness for the two strategies (MaxCost = 30)



$\rho$	$\delta$					
	0	0.1	0.3	0.5	0.7	1
0	34,11%					
0.1		34,15%	34,17%	34,11%	34,17%	34,18%
0.2			34,39%	35,02%	34,14%	35,02%
0.3				34,37%	36,07%	36,06%
0.4					35,07%	37,39%
0.5						35,56%

**Fig. 10** Variations in cost of robustness, depending on percentages of documentation costs (MaxCost = 30)

Figure 10 shows the variations in the cost of robustness (loss) due to variations in function  $worstCases(\delta, \rho)$ , being  $\delta$  the number of cost worst cases for the documentation tasks and  $\rho$  the percentage of cost variation in those tasks. We again ran the NSGA-II algorithm over 200 samples based on the CTAS SAD. In the table, we departed from an initial price of robustness of 34 % as an optimistic value, which progressively gets worse (gray cells in Fig. 10) as we move along the axes and both  $\delta$  and  $\rho$  are increased. Thus, in more realistic settings, the metrics show that the cost of robustness can go up to a 10 % of the initial value.

Despite more empirical work and involving different SADs is needed, we can say that a reasonable guideline to make NSVP\_BD more robust is to apply the strategy of scenarios for the stakeholders' priorities, and also tune the parameters  $\delta$  and  $\rho$  adequately, so as to obtain a good tradeoff between the percentage of robustness and its associated cost.

## 5 Learning Stakeholders' Preferences with Respect to Views

In our approach, we model the stakeholders' preferences with a matrix called MSI that stores the level of interest of the cur-

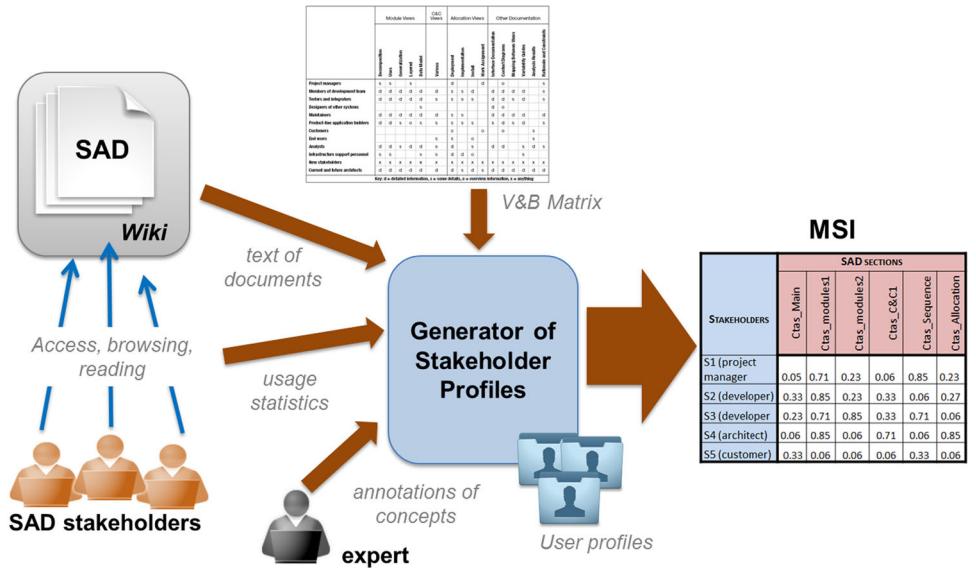
rent SAD section (columns) for a given stakeholder (rows). In a way, we can think of these interests of each stakeholder as its *user profile* [45]. Initially, the profiles for the different stakeholder types are derived from the V and B matrix of Fig. 2. However, these profiles do not remain static, as stakeholders normally change their interests (or preferences) on the views (SAD sections) over time. As shown in Fig. 3, ProSAD relies on a profiling component that monitors the access to the stakeholders to the Wiki-based SAD and generates user profiles. As stakeholders browse the SAD, their profiles are enriched with information coming from Wiki documents via Natural Language Processing (NLP) techniques [3] and implicit interest indicators [1, 12].

Basically, the NLP processing extracts the key tokens of the text by means of syntactic/semantic analyses of the Wiki pages. We first apply a process of token mining, and then complement the tokens with a process of concept mining. The NLP tasks for token mining are based on the OpenNLP<sup>5</sup> library and involve: (i) text parsing of the input text to remove custom annotations from the Wiki syntax as well as invalid characters; (ii) sentence detection, which splits the previous text into a set of sentences; (iii) identification of tokens (terms) in the sentences; (iv) stop-words removal for frequently used terms; and (v) stemming, in which the terms are reduced to their root form to improve the keyword matching. Porter's Stemming<sup>6</sup> algorithm is used for stemming. A set of concepts is finally associated with each token, based on a pre-defined dictionary of Software Architecture concepts. This dictionary was built by combining concepts from an existing ontology for software architectures [22] with concepts

<sup>5</sup> OpenNLP homepage: <http://opennlp.apache.org/>.

<sup>6</sup> Porter's Stemming homepage: <http://snowball.tartarus.org/>.

**Fig. 11** Pipeline for constructing stakeholder profiles and updating the MSI



described by the Software Engineering Institute bibliography [5].

The interest indicators refer to time spent on reading a page, number of visits, mouse scrolls, mouse clicks, and the ratio between scrolls and time (which represents the frequency of scrolls while reading of Web page), among others. With all this information, the profiling component is able to establish links between a given profile and the SAD sections that best match that profile, so as to update the MSI. A general schema of the working of this component is shown in Fig. 11. For more details of the profile construction techniques, please refer to [37].

The design consists of a semi-automated pipeline that: (i) generates user profiles, (ii) generates document representations, and (iii) computes matching relationships among users and SAD sections. We refer to these relationships as relevance links. The relevance links are computed on the basis of the similarity between the user profiles and the document representations. Both the user profiles and the document representations are term-based vectors. The pipeline computes a set of weighted links between users and sections, in which the weights indicate the relevance of the sections for each stakeholder.

We should note that the V and B matrix (Fig. 2) and the MSI matrix (Fig. 11) are slightly different in their contents. First, the V and B matrix shows types of stakeholders (roles) and architectural view types, while the MSI models individual stakeholders (i.e., persons playing stakeholder roles of V and B) and instances of architectural views (as prescribed by the V and B templates). Second, the cells of the V and B matrix deal with detail level using an ordinal scale (detailed information, overview information, some details, anything), while those of MSI refer to interest degree using a numerical scale ([0, 1]). For this reason, we need to map the informa-

tion provided by the relevance links to the MSI. To do so, we simply assign a numeric value to each ordinal value of V and B, namely: anything → 0, overview → 1/3, some details → 2/3, and detailed → 1. As a result, the MSI is ready to be consumed by the optimizing component (i.e., the documentation assistant).

## 6 Related Work

Several architecture documentation methods exist in the literature [14, 24, 32, 42]. Common to all these methods is the prescription of a SAD structure (i.e., templates) and the use of views for different system viewpoints. These viewpoints might be related to stakeholders' concerns. Nonetheless, the methods do not provide guidelines for creating the documentation package, except for the steps suggested by V and B. Thus, the documenter is responsible for determining how the SAD contents will be staged for delivery.

Lattanze [33] classifies architecture documentation strategies into horizontal and vertical ones, emphasizing that both require SAD planning as well as identification of stakeholders and their information needs. In a vertical strategy, custom documents are created for specific stakeholders. These documents might have duplication or require considerable efforts, but the SAD will certainly fulfill the stakeholders' needs. V and B falls in this category, although it helps documenters save some efforts, if the method is applied in a disciplined way. In a horizontal strategy, on the contrary, the documenter creates small, isolated documents and then tries to reuse them to form documentation suites for specific stakeholders. Usually, some writing has to be added to link the documents together. Thanks to the reuse, this strategy reduces the SAD production efforts and its maintenance. Currently, our assis-

tant implements a vertical strategy but it can accommodate a horizontal one with some modifications.

As it regards the profiling of documentation readers, Su [47] proposed an automated approach based on pieces of architectural information, called chunks. These chunks are the result of specific exploration paths followed by a user when reading a SAD. When a new user is about to navigate the SAD, a tool recommends her candidate sections by reusing previous (similar) exploration paths. This approach effectively assists readers to find information, but unlike our approach, it does not support documentation strategies.

Different authors have reported experiences with Wikis applied to architecting tasks [2]. The Wiki approach can be traced to previous efforts in adaptive systems for hypermedia based on user models [10]. Related to our conception, Bachmann and Merson [2] discussed the role of a Wiki to record technical documentation in a collaborative setting, and specifically, how to use a Wiki with V and B documentation. Farenhorst et al. built JIT AK Portal to capture architecture knowledge and share it among architects, but not among other stakeholders. Another related tool is Knowledge Architect [29], which supports the retrieval of knowledge for multiple stakeholder types, although only focused on the reader's side. Neither JIT AK Portal nor Knowledge Architect consider the process of producing architectural contents, which is the cornerstone of our tool. Graaf et al. [22] conducted an empirical study on Wiki-based SADs. The SAD is built on top of a semantic Wiki equipped with annotations, using an ontology of architecture concepts. We believe semantic Wikis are an interesting feature for boosting the "intelligence" of our documentation assistant.

We have performed a comparative analysis of the different approaches we have found in the literature according to different aspects we consider that an efficient and stakeholder-centric documentation approach should be met. These requirements are based on several factors, which include the opinion of experts in the field [14, 23, 40, 42], current standards on architectural documentation [26, 27] and the experience of several researchers (including ours) with industrial projects [4, 25, 41, 49]. In our context, efficient means that the documentation process should optimize the use of resources to achieve good quality documentation. In addition, a documentation approach should produce documentation that satisfies its readers and that provides real value for the project, while also it should help the readers to recover relevant contents from large knowledge databases. To this end, it is advisable to follow a process centered in the stakeholders' interests and information needs. In summary, the requirements for a documentation approach should be the following:

- *R#1: ensure the satisfaction of stakeholders' interests*  
High-quality documentation should be produced from

readers' perspective rather than from writers' perspective. A documentation approach should capture and model the stakeholders' interests to produce useful documentation. This complies with current standards for architectural documentation [27].

- *R#2: consider stakeholders' priorities* Given that not every stakeholder has the same level of importance within a software project, documentation efforts should be centered in satisfying the information needs of key stakeholders. This strategy allows to produce useful documentation with a limited amount of resources.
- *R#3: follow an efficient production process* An efficient documentation process is essential when software projects have tight agendas and limited resources for documentation activities [25]. To achieve an efficient process, it should be planned [30]. A documentation approach should provide a framework to plan the production of architectural documentation. In addition, since planning the process in an cost-effective manner is not a trivial task, the approach should provide support mechanisms. Ideally, a good documentation plan should provide a high level of coverage of stakeholders' interests with a reduced documentation effort.

The results of our analysis are summarized in Table 1, in which the approaches are compared according to the following features:

- *Goal (G)* The main goals of the approach, which could be: architecture knowledge management and recovery, efficient documentation production, produce high-value documentation.
- *Tool support (TS)* Whether the work includes a tool to facilitate its adoption.
- *Type of documentation (TD)* The type of documentation targeted by the work (e.g., architecture, rationale, requirements).
- *Target stakeholders (TS)* The type of stakeholders that are benefited from the research work.
- *Stakeholder-centric (SC)* Whether the approach considers the stakeholders' interests and produces reader-oriented documentation.
- *Stakeholder priorities (SP)* Whether the approach considers the stakeholders' priorities during the process.
- *Efficient process planning (EPP)* Whether the proposal provides some sort of assistance to plan the documentation process in an efficient manner.
- *Empirical evaluation (EE)* The proposal has been empirically evaluated to demonstrate its usefulness.

The features selected above were based on the authors' experience on the subject as well as on a study of the "main tokens" being mentioned in the sources from the literature. Table 1

**Table 1** Comparison of related works

References	G	TS?	TD	TS	SC?	SP?	EPP?	EE?
Clements et al. [13, 14]	Produce high-value documentation	No	Architecture	Multiple	Yes (matrix static model)	No	No (some basic guidelines)	No
Rozanski et al. [42]	Produce high-value documentation	No	Architecture	Multiple	Yes (no explicit model)	No	No (some basic guidelines)	No
Kruchten [32]	Produce high-value documentation	No	Architecture	Technical stakeholders	No	No	No	No
Hofmeister et al. [24]	Produce high-value documentation	No	Architecture	Technical stakeholders	No	No	No	No
Naeem et al. [35]	Produce high-value documentation	No	Architecture	Multiple	Yes (through the OSV)	No	No	Yes (preliminary)
Farenhorst et al. [20, 21]	AK management and recovery	Yes	Architecture	Architect	No	No	No	Yes (preliminary)
Jansen et al. [29]	AK management and recovery	Yes	Architecture	Multiple	No	No	No	Yes
Boer y van Viet [9]	AK management and recovery	No	Architecture	Auditor/evaluator	No	No	No	Yes
Su et al. [47, 48]	AK management and recovery	Yes	Architecture	Multiple	Yes (dynamic preference model)	No	N/a	No
Castro-Herrera et al. [11]	Support requirements elicitation	No	Requirements	Multiple	Yes (dynamic preference model)	No	N/a	Yes (preliminary)
Falessi et al. [19]	Efficient documentation production	No	Architecture (rationale)	Multiple	No	No	No	Yes
Hadar et al. [23]	Efficient documentation production	Yes	Architecture	Multiple	No	No	No	No
Savolainen et al. [44]	Efficient documentation production	No	Architecture	Multiple	Yes (static model)	No	No (some basic guidelines)	No

shows that, currently, there is not an integral, empirically evaluated approach that meets all the above requirements. All works lack some important features. On the one hand, there is a subset of works towards the stakeholder-centric recovery of architectural knowledge, but they do not employ the acquired information about preferences to produce high-value documentation or to perform an automated recovery process. On the other hand, there is another subset of works that focus on the reduction in the effort required to produce architectural documentation by identifying low-value contents. However, this sort of works do not provide usually software tools to facilitate its adoption and do not follow a stakeholder-centric strategy to select those contents, so the satisfaction of the key stakeholders cannot be guaranteed.

## 7 Conclusions and Future Work

In this work, we have proposed a robust and stakeholder-centric approach and tool support for optimizing the information contents of software architecture documents. In ProSAD, the strategy for managing the architecture knowledge is driven by the needs of the stakeholders interested in the SAD, but it also considers the document production efforts. The problem of generating a satisficing SAD is seen as a discrete optimization problem and can be solved either with exact or heuristic algorithm (e.g., NSGA-II). We have developed tool support for ProSAD in the form of a documentation assistant to the architect. In particular, we have leveraged on the concepts and guidelines of the V and B documentation model and mapped this model to a formulation called NSVP, which constitutes a novel aspect of the proposal.

In this article, we have improved the NSVP formulation in several ways, namely: (i) consideration of SAD dependencies, which turned out NSVP into a bi-objective optimization problem; (ii) sensitivity analysis of model parameters, which led to a robust formulation of NSVP; and (iii) integration of user profiles that capture the dynamics of the stakeholders with respect to the SAD. We have performed some preliminary evaluations on specific SAD instances with good results. This provides initial evidence that ProSAD is feasible and practical for architects.

The contributions of our approach are twofold. First, unnecessary information is not documented, with the consequent effort savings. There is a well-known relationship between the amount of software documentation and its usefulness. Beyond a certain point, the usefulness of documentation decreases when more information is added, because finding relevant information becomes more and more difficult as the overall amount of documentation increases. Second, the (stakeholder-centric) motivation for having architecture knowledge is reinforced (i.e., made it concrete via specific

tasks) in the development team. Nonetheless, more experiments with users and real-word case-studies are needed to confirm our partial results. The current robust framework can be improved regarding the usage of satisfaction functions. It seems possible to “reuse” the same idea of starting with a basic MSI and periodically update it via learning for the case of satisfaction functions. For instance, we could depart from a predefined satisfaction function and then make adjustments to it based on (learning from) stakeholders’ feedback. As for the integration of user profiling techniques in ProSAD, we argue that the component can have a dual purpose, supporting the writers and the readers of the SAD. On the reader’s side, a personalization tool could identify potentially relevant SAD sections for specific stakeholders, alleviating information overload problems [37]. As future work, we would like to study measures for quantifying the internal quality of a SAD, and possibly incorporate it as a third objective in our optimization formulation. Another line of work is to apply our approach to process a large information repository to extract small documentation suites for particular goals. This scenario would need “undo” tasks in our optimization framework.

Finally, as a long-term research goal, we want to investigate if our approach can be customized to other information domains. Along this line, we speculate that the production planning strategy can be applicable to other documentation artifacts or content-management systems, as long as some documentation structure is available and the information items can be linked to user profiles. Some interesting applications of user models to knowledge-aware Web Services have been reported in [15], and our approach could be adapted to that context.

**Acknowledgments** This work was partially supported by ANPCyT (Argentina) through PICT Project 2011 No. 0366, and also by CONICET (Argentina) through PIP Project No. 112-201101-00078.

## References

1. Al Halabi WS, Kubat M, Tapia M (2007) Time spent on a web page is sufficient to infer a user’s interest. In: IASTED European Conference on Proceedings of the IASTED European Conference: internet and multimedia systems and applications. ACTA Press, Anaheim, CA, USA, pp 41–46
2. Bachmann F, Merson P (2005) Experience using the web-based tool wiki for architecture documentation. Technical Note CMU/SEI-2005-TN-041, SEI, Carnegie Mellon University, Pittsburgh, Pennsylvania
3. Baeza-Yates R, Ribeiro-Neto B (2011) Modern information retrieval: the concepts and technology behind search, 2nd edn. Addison-Wesley Professional
4. Bass L, Kazman R, Ozkaya I (2011) Developing architectural documentation for the hadoop distributed file system. In: Hissam S, Russo B, de Mendonca Neto M, Kon F (eds) Open Source Systems: Grounding Research, IFIP Advances in Information and Communication Technology, vol 365, Springer Boston, pp 50–61

5. Bass L, Clements P, Kazman R (2012) Software Architecture in Practice, 3rd edn. Addison-Wesley Professional
6. Berre DL, Parrain A (2010) The sat4j library. release 2.2. JSAT 7(2–3):59–6
7. Bertsimas D, Sim M (2004) The price of robustness. Oper Res 52(1):35–53
8. Beyer HG, Sendhoff B (2007) Robust optimization—a comprehensive survey. Comput Methods Appl Mech Eng 196(33–34):3190–3218
9. de Boer RC, van Vliet H (2008) Architectural knowledge discovery with latent semantic analysis: constructing a reading guide for software product audits. J Syst Softw 81(9):1456–1469
10. Cannataro M, Cuzzocrea A, Mastrianni C, Ortale R, Pugliese A, universitÀ Della Calabria D (2002) Modeling adaptive hypermedia with an object-oriented approach and xml. In: In Proc. of Web-Dyn'02
11. Castro-Herrera C, Cleland-Huang J, Mobasher B (2009) Enhancing stakeholder profiles to improve recommendations in online requirements elicitation. In: Requirements Engineering Conference, 2009. RE '09. 17th IEEE International, pp 37–46
12. Claypool M, Le P, Wased M, Brown D (2001) Implicit interest indicators. In: Proceedings of the 6th international conference on Intelligent user interfaces, ACM, New York, NY, USA, IUI '01, pp 33–40
13. Clements P, Bachmann F, Bass L, Garlan D, Ivers J, Little R, Nord R, Stafford J (2003) A practical method for documenting software architectures. In: ICSE
14. Clements P, Bachmann F, Bass L, Garlan D, Ivers J, Little R, Merson P, Nord R, Stafford J (2010) Documenting software architectures: views and beyond, 2nd edn. Addison-Wesley Professional
15. Cuzzocrea A (2006) Combining multidimensional user models and knowledge representation and management techniques for making web services knowledge-aware. Web Intell Agent Sys 4(3):289–312
16. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. Trans Evol Comp 6(2):182–197
17. Diaz-Pace JA, Nicoletti M, Schiaffino S, Vidal S (2014) Producing just enough documentation: The next sad version problem. In: Le Goues C, Yoo S (eds) Search-Based Software Engineering, Lecture Notes in Computer Science, vol 8636, Springer International Publishing, pp 46–60
18. Diwekar U (2010) Introduction to applied optimization, 2nd edn. Springer Publishing Company, Incorporated
19. Falessi D, Briand LC, Cantone G, Capilla R, Kruchten P (2013) The value of design rationale information. ACM Trans Softw Eng Methodol 22(3):21
20. Farenhorst R, Lago P, van Vliet H (2007) Eagle: effective tool support for sharing architectural knowledge. Int J Coop Inf Syst 16(3/4):413–437
21. Farenhorst R, Izaks R, Lago P, Vliet Hv (2008) A just-in-time architectural knowledge sharing portal. In: Proceedings Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), IEEE Computer Society, Washington, DC, USA, WICSA '08, pp 125–134
22. de Graaf KA, Tang A, Liang P, van Vliet H (2012) Ontology-based software architecture documentation. In: Proceedings Joint Working Conf. on Software Architecture & European Conf. on Software Architecture (WICSA/ECSA), IEEE Computer Society, WICSA 2012, pp 315–319
23. Hadar I, Sherman S, Hadar E, Harrison J (2013) Less is more: Architecture documentation for agile development. In: Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on, pp 121–124
24. Hofmeister C, Nord R, Soni D (2000) Applied Software Architecture, 1st edn. Addison-Wesley Professional
25. Hoorn JF, Farenhorst R, Lago P, van Vliet H (2011) The lonesome architect. J Syst Softw 84(9):1424–1435
26. IEEE (2000) Ieee std 1471-2000: Recommended practice for architectural description of software-intensive systems
27. ISO/IEC/IEEE (2011) Iso/iec/ieee 42010: Systems and software engineering—architecture description
28. Jansen A, Bosch J (2005) Software architecture as a set of architectural design decisions. In: Proceedings Working Conf. on Software Architecture, IEEE Computer Society, pp 109–120
29. Jansen A, Avgeriou P, van der Ven JS (2009) Enriching software architecture documentation. J Syst Softw 82(8):1232–1248
30. Keuler T, Knodel J, Naab M, Rost D (2012) Architecture engagement purposes: Towards a framework for planning 'just enough'-architecting in software engineering. In: Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on, pp 234–238
31. Koning H, Vliet HV (2006) Real-life it architecture design reports and their relation to ieee std 1471 stakeholders and concerns. Autom Softw Eng 13:201–223
32. Kruchten P (1995) The 4+1 view model of architecture. IEEE Softw 12(6):42–50
33. Lattanzio A (2008) Architecting software intensive systems: a practitioners guide. Taylor & Francis
34. Mitchell RK, Agle BR, Wood DJ (1997) Toward a theory of stakeholder identification and salience: defining the principle of who and what really counts. Acad Manag Rev 22:853
35. Naeem S, Imtiaz S (2014) Architecture coverage: Validating optimum set of viewpoints. In: Proceedings of the 9th International Conference on Software Engineering Advances (ICSEA)
36. Nicoletti M, Diaz-Pace JA, Schiaffino S (2012) Towards software architecture documents matching stakeholders interests. In: Cipolla-Ficarra F (ed) Advances in New Technologies, Interactive Interfaces and Communicability, no. 7547 in LNCS, Springer Berlin Heidelberg, pp 176–185
37. Nicoletti M, Diaz-Pace J, Schiaffino S, Tommasel A, Godoy D (2014) Personalized architectural documentation based on stakeholders information needs. J Softw Eng Res Dev 2(1):9
38. Pace JAD, Nicoletti M, Schiaffino SN, Villavicencio C, Sanchez LE (2013) A stakeholder-centric optimization strategy for architectural documentation. In: Model and Data Engineering—Third International Conference, MEDI 2013, Amantea, Italy, September 25–27, 2013. Proceedings, pp 104–117
39. Paixão M, Souza J (2013) A scenario-based robust model for the next release problem. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, GECCO '13, pp 1469–1476
40. Parnas DL (2010) Precise documentation: The key to better software. In: Nanz S (ed) The Future of Software Engineering, Springer, pp 125–148
41. Rost D, Naab M, Lima C, von Chavez CFG (2013) Software architecture documentation for developers: a survey. In: Proceedings of 7th ECSA, Springer-Verlag, Berlin, Heidelberg, pp 72–88
42. Rozanski N, Woods E (2011) Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, 2nd edn. Addison-Wesley
43. Saltelli A, Tarantola S, Campolongo F, Ratto M (2004) Sensitivity analysis in practice: a guide to assessing scientific models. Halsted Press, New York
44. Savolainen J, Mannisto T (2010) Conflict-centric software architectural views: exposing trade-offs in quality requirements. IEEE Softw 27(6):33–37
45. Schiaffino S, Amaldi A (2009) Intelligent user profiling. In: Brammer M (ed) Artificial Intelligence: An International Perspective, Lecture Notes in Computer Science, vol 5640, Springer Berlin/Heidelberg, pp 193–216

46. Sobol IM (2001) Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Math Comput Simul* 55(1–3):271–280
47. Su MT (2010) Capturing exploration to improve software architecture documentation. In: Proceedings 4th European Conference on Software Architecture: Companion Volume, ACM, New York, NY, USA, ECSA '10, pp 17–21
48. Su MT, Hosking J, Grundy J (2011) Capturing architecture documentation navigation trails for content chunking and sharing. In: 2011 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp 256–259
49. Unphon H, Dittrich Y (2010) Software architecture awareness in long-term software product evolution. *J Syst Softw* 83(11):2211–2226
50. Yu G (1996) On the max-min 0–1 knapsack problem with robust optimization applications. *Oper Res* 44(2):407–415