Producing Just Enough Documentation: The Next SAD Version Problem

J. Andres Diaz-Pace, Matias Nicoletti, Silvia Schiaffino, and Santiago Vidal

ISISTAN Research Institute, CONICET-UNICEN, Campus Universitario, Paraje Arroyo Seco (B7001BBO) Tandil, Buenos Aires, Argentina
{adiaz,mnicolet,sschia,svidal}@exa.unicen.edu.ar

Abstract. Software architecture knowledge is an important asset in today's projects, as it serves to share the main design decisions among the project stakeholders. Architectural knowledge is commonly captured by the Software Architecture Document (SAD), an artifact that is useful but can also be costly to produce and maintain. In practice, the SAD often fails to fulfill its mission of addressing the stakeholders' information needs, due to factors such as: detailed or high-level contents that do not consider all stakeholders, outdated documentation, or documentation generated late in the lifecycle, among others. To alleviate this problem, we propose a documentation strategy that seeks to balance the stakeholders' interests in the SAD against the efforts of producing it. Our strategy is cast as an optimization problem called "the next SAD version problem" (NSVP) and several search-based techniques for it are discussed. A preliminary evaluation of our approach has shown its potential for exploring cost-benefit tradeoffs in documentation production.

Keywords: architecture documentation model, stakeholders, information needs, combinatorial optimization, search-based techniques.

1 Introduction

As software systems grow large and complex, the reliance on some form of documentation becomes a necessity in many projects [1]. Since producing documentation does not come without cost, software engineers must carefully consider how this process plays out in the development lifecycle (e.g., artifacts, techniques, tools), and furthermore, identify the goals of the project stakeholders. In particular, a useful model for describing the high-level structure of a system is the *software architecture* [2], which is the main domain explored in this work. The architecture is typically captured by the so-called *Software Architecture Document (or SAD)*, as an information repository that enables knowledge sharing among the architecture stakeholders [3]. The SAD is structured into sections that contain text and design diagrams, known as *architectural views*, which permit to reason about the architectural solution from different perspectives.

Documenting an architecture with multiple stakeholders poses challenges for the SAD. A first challenge is that the SAD contents target readers that might

C. Le Goues and S. Yoo (Eds.): SSBSE 2014, LNCS 8636, pp. 46–60, 2014.
© Springer International Publishing Switzerland 2014

have different backgrounds and information needs [4]. 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 SAD is loaded with development-oriented contents that only consider a few (internal) stakeholders. In practice, the documentation usefulness decreases as more information is added, because finding relevant information in a large set of documents becomes difficult [1]. A second challenge is the effort necessary for creating (and updating) the SAD, an expenditure that developers and managers do not wish to bear, mainly because of budget constraints, tight schedules, or pressures on developing user-visible features. As a result, the architecture knowledge ends up informally captured. Besides the stakeholders' dissatisfaction, the problem of ineffective documentation brings hidden costs such as: knowledge vaporization, re-work, and poor quality [5].

Recently, some works have investigated the practices and value of architecture documentation [6]. In this context, we argue that the SAD should be produced in incremental versions and concurrently with the design work. Thus, the main question becomes: how much documentation is good enough for the next SAD release? Answering this question involves a tradeoff between documenting those aspects being useful to the stakeholders and keeping the documentation efforts low. To deal with this tradeoff, we previously proposed [7] an optimization tool that, for a given SAD version, is able to assist the documenter in choosing a set of SAD updates that brings high value for the stakeholders. The tool is based on the Views & Beyond (V&B) method [8,3], which explicitly links the candidate architectural views for the SAD to the needs of its stakeholders. The optimization was treated as a knapsack problem that maximizes the stakeholders' utility without exceeding a cost constraint. Yet, considering that documentation is more a business decision than a technical one, we believe that alternative optimizations can be required, depending on cost-benefit concerns of the project.

In this work, we provide a general formulation of the SAD documentation strategy and its associated optimization problem(s), that we call the Next SAD Version Problem (NSVP), by analogy with the well-known Next Release Problem (NRP) [9,10]. As its main contribution, our proposal considers two variants for NSVP: a single-objective cost minimization and a bi-objective optimization (cost versus utility), in addition to the single-objective utility maximization of [7]. We also investigate different satisfaction functions for stakeholders. The experimental results, although preliminary, show that the NSVP optimization approach helps to explore alternative documentation strategies with reduced costs.

The article is organized as follows. Section 2 provides background about architecture documentation. Section 3 formally defines the NSVP as an optimization problem. Section 4 discusses exact and heuristic algorithms for NSVP. Section 5 reports on an empirical evaluation with a SAD case-study. Section 6 discusses related work. Finally, Section 7 gives the conclusions and future work.

2 Background

The software architecture is the set of structures needed to reason about a computing system, comprising software elements, their relations, and properties of both [2]. Design decisions are part of the architecture, as they record the rationale behind the architects' solution [11]. An example of decisions is the use of certain patterns, such as layers or client-server, to meet stakeholders' goals, such as modifiability or performance qualities, among others. Thus, the architecture acts as a blueprint in which the main stakeholders' concerns can be discussed. By *stakeholder* [12], 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). In order to share the architecture knowledge among the stakeholders, it must be adequately documented and communicated. The SAD is the usual "knowledge radiator" and can take a variety of formats, for instance: Word documents, UML diagrams, or Web pages in a Wiki [13,14].

The notion of *architectural views* is key in the organization of architectural documentation, and it is part of most current documentation methods [3]. A view presents an aspect or viewpoint of the system (e.g., static aspects, runtime aspects, allocation hardware, etc.). Typical views include: module views (the units of implementation and their dependencies), component-and-connector views (the elements having runtime presence and their interactions), or allocation views (the mappings of software elements to hardware). In addition, these views include text describing the design elements and decisions that pertain to the views. Therefore, we can see a SAD as a collection of documents with textual and graphical contents. Figure 1 shows a snapshot of a Wiki-based SAD[1], in which their documents (Wiki pages) adhere to the V&B templates[2].

In architecture-centric projects, the SAD usually emerges as a by-product of the architects' design work. The stakeholders (both internal and external ones) are the main SAD consumers. Moreover, a SAD is useful as long as its contents satisfy the *stakeholders' information needs*. A good strategy to ensure this goal is to deliver the SAD in incremental versions along with the (iterative & incremental) development of the architecture itself [15,1]. In the documentation process, the documenter must decide what should be added (or updated) in a given SAD version. She is expected to follow the well-known rule: "write the SAD contents from the reader's perspective rather than from writer's" [3], but also consider the so-called TAGRI principle[3]: "They [the stakeholders] Ain't Gonna Read It", which advocates for documenting only what reflects true needs. To realize these ideas, a model of stakeholders' interests regarding the architectural contents of the SAD is needed. For instance, we can have a matrix of S stakeholders (or stakeholder roles) and D SAD documents (or view types), in which a cell indicates that stakeholder S_i is interested in the information of document D_j.

[1] SEI example:
`https://wiki.sei.cmu.edu/sad/index.php/The_Adventure_Builder_SAD`
[2] V&B templates:
`http://www.sei.cmu.edu/downloads/sad/SAD_template_05Feb2006.dot`
[3] Scott Ambler's website: `http://www.agilemodeling.com/essays/tagri.htm`

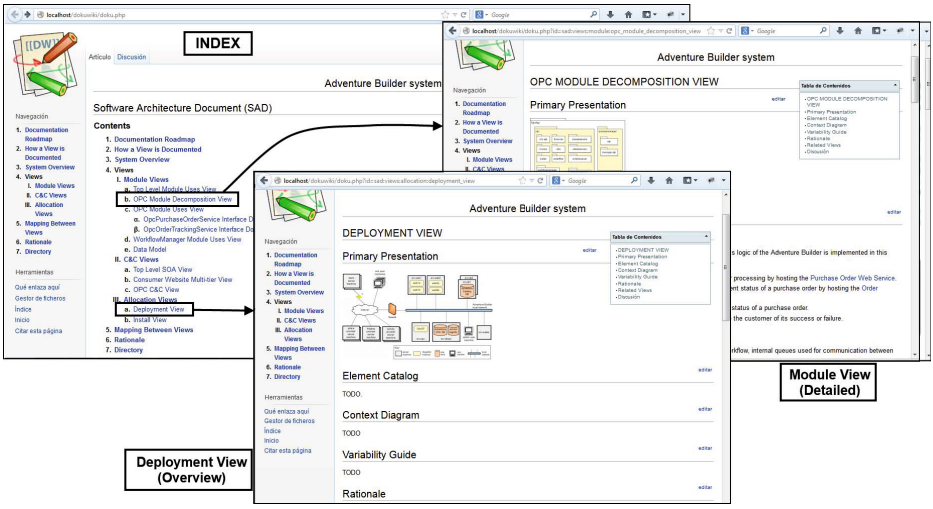

Fig. 1. Example of a Wiki-based SAD with architectural views of the V&B method

In fact, some documentation methods, and specifically V&B, provide guidance for generating the SAD contents according to such a model [8,16].

In an ideal situation, one would analyze the matrix and produce a SAD version addressing all stakeholders' interests. Unfortunately, this is seldom the case in practice, either because the documentation resources are scarce or because of conflicts between stakeholders' interests. Empirical studies [17] have shown that individual stakeholder's concerns are met by a SAD fraction (less than 25%), but for each stakeholder a different (sometimes overlapping) fraction is needed. Therefore, determining the "delta" of SAD contents that brings most utility (or benefit) to the stakeholders as a whole is not a straightforward decision.

3 The Next SAD Version Problem (NSVP)

We see the SAD as an artifact that contains n documents, each one associated to a predefined view template[4]. In this work, we assume the usage of the V&B views and templates, although other view-centric methods are equally possible. Let $SAD_t =< d_1^t, ..., d_n^t >$ be a SAD version at time t, in which each vector position corresponds to a document and 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 a document k, based on the sub-sections prescribed by the V&B templates. Figure 2 depicts how a view document of V&B can be filled with new contents over time. This should not be interpreted as a strict documentation progression for the documenter, but rather as a guideline based on the relative importance of the sub-sections

[4] Admittedly, other documents with no architectural views are usually part of a SAD (e.g., system context, main stakeholders, architectural drivers, or glossary). These documents are out of the current NSVP scope, but still considered in our evaluation.

for the view under consideration [8] (e.g., a documenter might begin adding rationale information, but it is recommended that she works first on the primary presentation of her solution and describes its main elements before providing a detailed solution rationale). In particular, we here assume 4 completion states $DS = \{empty, overview, someDetail, detailed\}$ for a document, so as to keep its evolution at a manageable level. In general, the mapping between discrete completion states and required view sections can be adjusted, depending on the chosen documentation method and templates.

Given a partially-documented SAD_t, let us consider an arbitrary next version $SAD_{t+1} =< d_1^{t+1}, ..., d_n^{t+1} >$ with $d_i^{t+1} \geq d_i^t$. We define an increment vector $\Delta =< x_1, ..., x_n >$ such that $d_i^t + x_i = d_i^{t+1}$. Note that changes in document states are currently assumed to be always additive (fixes to a sub-section are also allowed with a cost, but they are not expected to alter the current state). For example, a deployment view like in Figure 1 can be refined with information for the sections *Element Catalog* and *Context Diagram*, which implies a transition from *overview* to *someDetail*. Thus, Δ is actually a decision vector, because one can choose alternative increments (i.e., levels of detail) for the documents in order to fulfill objectives related to stakeholders' satisfaction and cost.

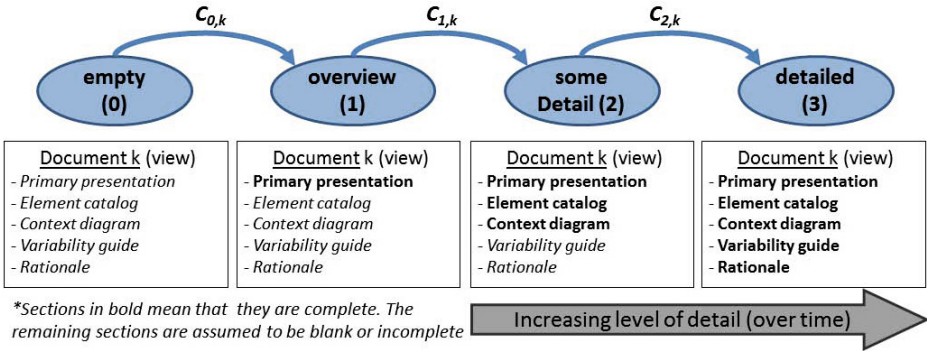

Fig. 2. Evolution of the contents of a document (based on *View* template of V&B)

The cost of a document state change $d_k \rightarrowtail d'_k$ is assumed to be a fixed quantity. Then, we have a cost vector $C_\Delta =< c_1, ..., c_n >$ with $c_k = cost(d_k, d'_k)$, and the total cost of an increment Δ, denoted by $Cost(\Delta)$, is the sum of the individual costs of changing document. If $d_k = d'_k$, a zero cost is assigned, unless fixes were applied to the document (without altering its completion level). We refer to $Cost(\Delta)$ as the *production cost* for the next SAD version.

The expected utility of an increment Δ is a function of the vectors SAD_t and SAD_{t+1}, but it also depends on the stakeholders' preferences on those two state vectors. Similarly to the cost formulation, we assume a benefit vector $B_\Delta =< b_1, ..., b_n >$ with $b_k = benefit(d_k, d'_k, satisfaction_k(S))$ in the range $[0, 1]$ (0 means no utility, and 1 means high utility). Given a set of m stakeholders $S = \{S_1, ..., S_m\}$, $satisfaction_k(S)$ captures the combined preferences of all stakeholders on the document state transition $d_k \rightarrowtail d'_k$. For example, stakeholder X might prefer a deployment view D in *overview*, while stakeholder Y

might instead prefer the same document in *detailed*. In other words, b_k is the "happiness" of the stakeholders group with an increased detail in document k, and $Benefit(\Delta)$ is computed as the sum of the utilities through all documents. $Benefit(\Delta)$ is a measure of the *stakeholders' utility* with the next SAD version.

The NSVP consists of *choosing an increment Δ for a SAD_t such that both $Cost(\Delta)$ and $Benefit(\Delta)$ are optimized*. Specifically, 3 variants are possible: (i) NSVP-B, in which $Benefit(\Delta)$ is maximized with $Cost(\Delta)$ lower than a threshold C; (ii) NSVP-C, in which $Cost(\Delta)$ is minimized with $Benefit(\Delta)$ above a threshold B; or (iii) NSVP-BC, in which $Benefit(\Delta)$ is maximized while $Cost(\Delta)$ is minimized. The first two variants are constrained, mono-objective optimizations, whereas the third variant is a bi-objective optimization. In the latter, we are interested in the *non-dominated solutions* of the benefit-cost space.

3.1 Determining Production Costs

The production cost of the next SAD version is $Cost(\Delta) = \sum_{k=1}^{n} cost(x_k) = \sum_{k=1}^{n} cost(d_k^t, d_k^{t+1})$. In our 4-state model the detail level of a document increases according to the sequence $empty\,(0) \rightarrow overview\,(1) \rightarrow someDetail\,(2) \rightarrow detailed\,(3)$. Thus, we assume an "atomic" cost associated to each transition in the sequence (see Figure 2). An atomic cost $c_{i,k}$ 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 is considered to have a 100% of completeness, its atomic costs can be $c_0 = c_1 = c_2 \simeq 333$. This estimation is crude, since costs are affected by the document type (e.g., architectural view, template used, typical amount of text required, or need of design diagrams). In practice, the final length of a SAD document can be unknown. Nonetheless, it is possible for the documenter to provide ballpark estimates (often, in collaboration with the manager or experienced architects), based on: historical effort information, importance of certain views, or number of design decisions involved, among others.

3.2 Assessing Stakeholders' Utility

The benefit of the next SAD is given by $Benefit(\Delta) = \sum_{k=1}^{n} benefit(x_k) = \sum_{k=1}^{n} benefit(d_k^t, d_k^{t+1}, satisfaction_k(S))$. Computing $benefit(d_k, d'_k, satisfaction_k(S))$ for a document k requires the specification of: i) the satisfaction information $satisfaction_k(S)$, and ii) a procedure to combine individual satisfactions into one single value. For every SAD document, a stakeholder can prefer any state in $DS = \{empty, overview, someDetail, detailed\}$. Note here

that *empty* is interpreted as the stakeholder being "not interested" in the document. This model of preferences is derived from the V&B method [8,3]. In order to translate preferences to satisfaction values, we introduce the notion of *satisfaction function*. We depart from the assumption that a stakeholder somehow knows her "perfect" level of detail for a document, based on her own information needs and the expected information to be conveyed by an architectural view. This knowledge is modeled by functions of the form $sf : DS \times DS \rightarrow [0, 1]$, which depend on both the actual and preferred completion states of a document and also on the view type. Based on our experience with architectural documentation projects, we propose three candidate functions that can be assigned to stakeholders, as described in Figure 3. Anyway, other satisfaction functions that take into account the semantics of document changes are also possible.

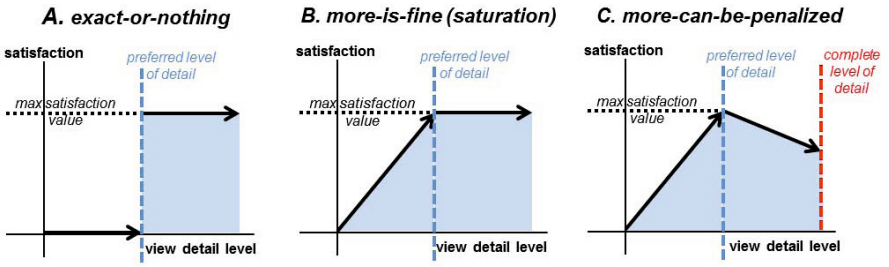

Fig. 3. Satisfaction functions for stakeholders' preferences on detail of documents

Document k (someDetail)	S_1	S_2	S_3	S_4	S_5
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. 4. Example of converting stakeholder preferences to satisfaction values

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 $pref_k$ and a document state d_k, that is, $pref_k \approx d_k \Rightarrow \mid pref_k - d_k \mid \leq \varepsilon$. Eliciting the right satisfaction function of a stakeholder is not trivial, and it is out of the scope of this work.

In the end, after applying this kind of satisfaction functions, we obtain a vector $satisfaction_k(S) =< s_k(S_1), ..., s(S_m) >$ with $s_k(S_i) \in [0, 1]$. Examples of satisfaction vectors computed with functions A, B, and C are shown in Figure 4. 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 based on a weighted average schema. 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. Along this line, the average utility of all stakeholders with document k is $benefit(d_k\,, d'_k,\, satisfaction_k(S)) = (\sum_{i=1}^{m} u_k(S_i) * p_i) / \sum_{i=1}^{m} p_i$.

4 Exact and Heuristic Algorithms for NSVP

To solve NSVP instances (i.e., find a "good" 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 this work, we explored 2 exact and 2 heuristic implementations, namely: i) Backtracking, ii) SAT4J [19], iii) Random Search, and iv) the NSGA-II algorithm [20]. The goal here was to assess their performance and optimality (of results) using synthetic data for SAD state vectors, costs, and stakeholders' preferences. The goodness (or score) of a solution is tied to the NSVP variant, namely: highest benefit (NSVP-B), lowest cost (NSVP-C), or non-dominated cost-benefit pairs (NSVP-BC). Figure 5 shows outputs of SAT4J, NSGA-II and RandomSearch for NSVP-BC. The points represent the Pareto front of pairs *(benefit, cost)*.

The 4 implementations are summarized next. First, *Backtracking* is a algorithm that progressively generates and evaluates all valid decision vectors Δ that derive from SAD_t. We used backtracking as a baseline for the SAT4J algorithm, knowing in advance that backtracking would have trouble with medium-to-large instances (e.g., SADs with $N \geq 20$). Second, *SAT4J* treats the document state representation as if it were a *0-1 Knapsack Problem* [7]. The tasks model the transitions between document states, like in Figure 2. With this representation, we took advantage of the state-of-the-art Pseudo-boolean (PB) solver of SAT4J [19]. The PB solver only deals with single-objective minimization subject to constraints (NSVP-C), but its adaptation to single-objective maximization

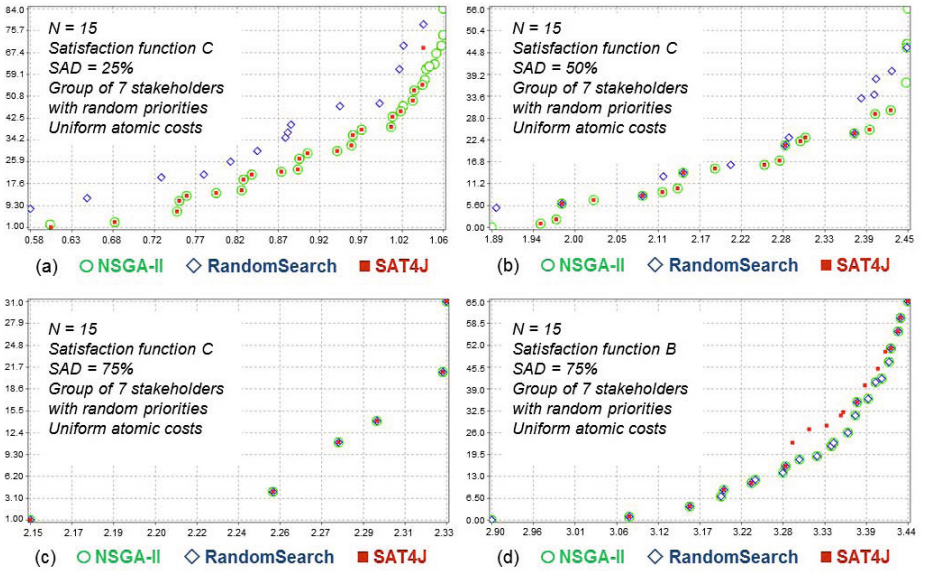

Fig. 5. Sample of Pareto fronts (cost versus benefit) using exact/heuristic algorithms

(NSVP-B) is straightforward. In NSVP-BC, we iterate over all possible costs, defining each one as a constraint and then invoking the solver with an NSVP-B instance. Non-dominated solutions are returned. This schema adds some overhead, but still capitalizes on the SAT4J performance.

Third, the *Random Search* implementation takes the current SAD_t and then randomly generates vectors Δ leading to SAD_{t+1}. It was built on top of an implementation provided by the MOEA framework[5], as a baseline for the NSGA-II algorithm. Despite the low execution times of Random Search for the 3 NSVP variants, the solutions are seldom optimal (see Figures 5a and 5b), and sometimes they even violate the cost (or benefit) constraints. Fourth, *NSGA-II* is a well-known genetic algorithm for multi-objective optimization [20]. Our implementation was based on the NSGA-II version of MOEA, and solves any of the NSVP variants. 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 Δ is evolved through several generations.

For the sake of brevity, we focus our analysis on the NSVP-BC formulation. From Figure 5 (and other Pareto fronts, not included here), we observed that the percentage of SAD completion affects the number of non-dominated solutions: the higher the percentage the fewer the solutions. Also, the differences in solution quality (cost and benefit) for the 3 algorithms get smaller, as the completion percentage increases. A possible explanation for this trend is that the

[5] `http://www.moeaframework.org/`

optimization problem gets "easier" for SAD percentages greater than 70% (because the potential solution space to search for is small). Nonetheless, there were differences between the solutions of NSGA-II and Random Search for NSVP-BC, particularly for large SADs ($N \geq 30$), as NSGA-II produced Pareto fronts closer to the reference front of SAT4J. In addition, the SAD completion influenced the costs attainable by the NSVP-BC solutions, with wider cost fluctuations in the Pareto sets for SAD instances with completion lower than 30%. Along this line, we can say that the completion range 25-50% offers the best opportunities for the documenter to find diverse cost-benefit solutions.

Another interesting observation is how the satisfaction functions modulate the Pareto solutions. The choice of function B (more-is-fine) leads to more solutions than the other functions (see Figures 5c and 5d), because function B is more likely to accommodate many stakeholder preferences, even if the stakeholders have different priorities. Function A (exact-or-nothing) showed the opposite behavior, with a very "strict" preference satisfaction that generates a sort of competition among stakeholders, which ultimately leads to few solution alternatives.

A performance analysis of the SAT4J and NSGA-II showed that their scalability varies, as expected, depending on the SAD sizes. For example, Figure 6 presents execution times of the 3 algorithms for a range of N (15-40) and incremental SAD completions (25%, 50%, and 75%). Overall, NSGA-II came out as an efficient alternative for the 3 NSVP variants. On one hand, NSGA-II showed bound execution times, with a slight increase at $N \simeq 30$, independently of the SAD completion. This behavior can be attributed to the evolutionary process of the genetic algorithm. On the other hand, SAT4J performed well for small SADs ($N \leq 30$), with a fast response for SADs above 50%. Its execution times started to degrade beyond $N = 30$ but only for incomplete SADs (completion around 25%). Although more experiments are needed, this finding suggests that SAT4J is very competitive for NSVP instances involving small-to-medium SADs and medium-to-high completion levels. For a general setting (or future, more complex NSVPs), we conclude that NSGA-II should be the solver of choice.

N □	SAT4J			NSGA-II			RandomSearch		
	SAD 25%	*SAD 50%*	*SAD 75%*	*SAD 25%*	*SAD 50%*	*SAD 75%*	*SAD 25%*	*SAD 50%*	*SAD 75%*
15	319	5	2	799	503	488	130	95	91
20	488	10	3	1184	704	636	175	125	128
25	2624	11	3	937	629	591	195	145	141
30	2995	23	3	990	693	677	203*	176*	169
35	**19089**	34	4	**1415**	909	743	302*	200*	183
40	**50938**	50	7	**1307**	881	835	283*	227*	218

Group of 7 stakeholders with random priorities - Uniform atomic costs
NSVP-B with C <= 50 (*) solutions with cost <=50 might (randomly) appear

Fig. 6. Performance (in ms.) of exact/heuristic algorithms for different SAD sizes

5 Case-Study

In addition to the experiments with synthetic data, we evaluated our approach on Wiki-based SADs being accessed by groups of (simulated) stakeholders. The main goal was to compare SAD increments produced by the SAT4J (or NSGA-II) algorithm against SAD versions generated in a usual manner (i.e., with no content optimization whatsoever). The objects of the study were incremental SAD versions, and we measured the cost of producing those increments as well as their benefit for predefined stakeholders. The subjects (stakeholders) were undergraduate students from a Software Architecture course taught at UNICEN University (Tandil, Argentina), with 2-4 years of developing experience. We organized these students into 11 groups of 7 members, each member playing a distinctive stakeholder role: 3 members took the architect role, whereas the other 4 members were divided into: 1 manager, 1 evaluator (responsible for design reviews) and 2 clients (e.g., representatives of external companies). Their priorities were as follows: $p_{manager} = 10$, $p_{architect} = 6$, $p_{client1} = 6$, $p_{evaluator} = 2$, $p_{client2} = 2$.

The architects (of each group) were asked to: i) design an architecture solution for a model problem called CTAS[6], and ii) use V&B to produce SADs (one per group) that should satisfy the concerns of the other roles of the group. The stakeholders' preferences were derived from the view-role matrix of [3]. Both the design and documentation work had to be done in 3 iterations of 3 weeks, with 2 (partial) SAD versions for the first 2 iterations and a final release after 9 weeks. The managers, clients, and evaluators periodically assessed the solution and documentation quality, and gave feedback to the architects. We refer to these documentation versions as *normal SADs*, because their production strategy was only based on architects' criteria (and not steered by optimization techniques).

Once all the groups were finished, we obtained 3 normal SAD versions (or slices) per group with the different completion percentages: 10-25% after the 1st iteration (slice 1), 40-60% after the 2nd iteration (slice 2), and 75-85% after the 3rd iteration (final SAD). The final SADs were not considered as 100% complete, because some sections were unfinished or lacked details. These percentages were estimated by counting the number of words and images per document (an image $\approx$ 200 words). The same metric was also used to estimate the cost of producing a SAD document, by considering a word as a unit of effort. We established the costs and utilities of the normal SAD slices as references for the optimization counterparts. Under a NSVP-BC formulation, we initially executed the optimization algorithms on slice 1 of every group (transition to slice 2). The same procedure was then repeated for slice 2 (transition to final SAD). We refer to the SAD versions produced by our algorithms as *optimized SADs*. Based on the Pareto fronts, we analyzed two situations: i) the best benefit reachable for each slice (and its corresponding cost), and ii) the cost for having the same benefit shown by the (next) normal slice.

[6] `http://people.cs.clemson.edu/~johnmc/courses/cpsc875/resources/Telematics.pdf`

Figure 7 shows the evolution of the average values of costs and benefits across the normal SAD versions, and the vectors represent *what-if* optimization analyzes (for the 3 satisfaction functions). The Utility Evolution chart (left side) compares utility values of normal SADs against optimized SADs. For instance, at slice 2, *normal-Fa* (normal SAD for function A) reached a utility of 5.52, whereas *opt-optimal-Fa* (best solution of optimized SAD for function A) reached a utility of 6. The Cost Evolution chart (right side) compares the costs of producing normal SADs against: i) optimized SADs with maximum benefit, and ii) optimized SADs with the same benefit as normal SADs. For instance, at slice 2, the cost of having a normal SAD was 98, while the costs of *opt-optimal-Fa* and *opt-same-utility-normal-Fa* were 103 and 78 respectively. In *opt-optimal-Fa* we had a slightly higher cost than that of normal SADs but the utility was better. In *opt-same-utility-normal-Fa,* the optimization helped to reach a utility of 5.52 (*normal-Fa* at slice 2 in Utility Evolution chart) but with a lower effort.

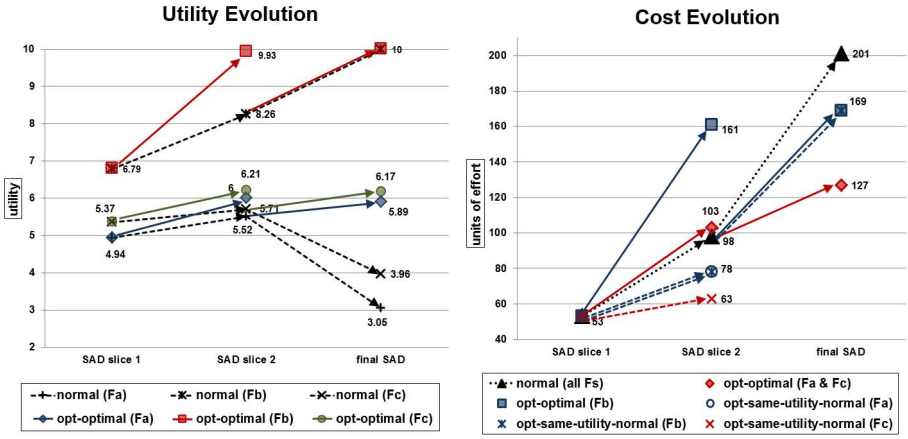

Fig. 7. Evolution of utility and cost over different SAD versions

From the results of Figure 7, we noticed that the optimized SADs achieved higher benefits for the 3 satisfactions functions, ranging between 9-20% of improvement for slice 2 and 55-93% for slice 3. Although, the costs to compute optimal solutions increased by 11%. However, for the same utility values of the normal SADs, we observed lower costs when applying optimization with savings of 44-78%. These savings mean that our algorithms produced smaller SADs with comparable utilities. Furthermore, it would suggest that the normal SADs had unnecessary contents for the stakeholder's needs.

We applied a Mann-Whitney-Wilcoxon (MWW) test to statistically validate results of this case-study. We tested the null hypothesis H_0 : *individual utility values of normal SADs are equal to those of optimized SADs,* for slice 2 and

the final SAD. With a significance level of 0.05, we verified a significant difference between utility values in favor of optimized SADs, except for function C at slice 2 and function B at the final SAD. Nevertheless, in those cases the average utility values of optimized SADs were still higher. Furthermore, in order to get insights on why optimized SADs outperformed normal ones, we manually inspected a sample of manual SADs in terms of satisfaction values. We realized that documenters who produced normal SADs tended to satisfy all stakeholders alike. On the contrary, the optimized SADs clearly favored high-priority stakeholders. These observations also support our conjecture about the complexity of producing satisfying documentation for stakeholders with competing interests.

At last, the evaluation has some validity threats. First, the results of our case-study cannot be generalized, as we used students in an academic environment, which might differ from an industrial context with seasoned practitioners. However, many students were actually working for software companies. Second, our effort unit for SAD production costs (amount of words of documents) is a simplification, and a better estimation proxy should be applied in real scenarios.

6 Related Work

In the last decade, several methods for software architecture documentation have emerged, namely: *Kruchten*'s *4+1 View Model*, *Siemens Four Views*, *Software Systems Architecture* and *SEI Views & Beyond* (V&B) [3,16]. A common aspect is the prescription of a SAD structure (i.e., templates) and the usage of views for different system viewpoints, which might be related to stakeholders' concerns. Nonetheless, these methods do not provide guidelines for creating the documentation package, except for the steps suggested by V&B [3]. V&B also proposes basic rules for relating stakeholder roles and views. The documenter is expected to apply these rules when determining the contents of a SAD release, but still this might be a complex and time-consuming task. This drawback motivated our NSVP work as a semi-automated aid to the documenter. However, our approach is not tied to V&B and can apply to other strategies, such as ACDM [15].

Optimization techniques have been used in several Software Engineering fields [21]. In particular, the Next Release Problem (NRP) is initially due to Bagnall et al. [9]. In the NRP formulation, a software company has to determine a subset of features that should be included in the next product release, in such a way the overall satisfaction of the company customers is maximized. Every customer is interested in a given subset of features and her satisfaction is proportional to the percentage of relevant features being included in the next release. Each feature is associated to a cost, and the total cost of the scheduled features must not exceed the available resources. The NRP was lately extended to a multi-objective formulation, known as MONRP (Multi-Objective Next Release Problem) [10,22], which treats the cost as a minimization objective. Along this line, MONRP has admits several solutions. Experiments with synthetic data [10,22] to solve (large) MONRP instances have shown that NSGA-II algorithms outperformed other ones with acceptable execution times. Nonetheless, the application of these ideas to other domains is still a topic of research.

7 Conclusions and Future Work

In this article, we have formalized the NSVP as an optimization problem for the production of software architecture documentation. A novel aspect of NSVP is that it seeks to balance the multiple stakeholders' interests in the SAD contents against the efforts of producing them. To do so, we characterize common stakeholder profiles and relate them to architectural views prescribed by SAD templates. The stakeholders' benefits were quantified by means of possible satisfaction functions. Our NSVP formulation admits 3 optimization variants. The project context should inform the selection of the most suitable variant. Two algorithms (SAT4J and NSGA-II) for efficiently solving the NSVP were discussed, although other algorithms are also possible. This kind of algorithms supports the documenter's tasks, assisting her to explore alternative solutions (e.g., the Pareto fronts). We actually envision a practical scenario where she can quickly identify "unnecessary" architectural information (with the consequent effort savings), and then prioritize relevant contents for the next SAD version.

A preliminary evaluation of the NSVP algorithms with both synthetic data and a case-study showed encouraging results. We observed a clear improvement in the quality of the optimized SAD versions, when compared to SAD counterparts generated in the usual manner. Regarding the satisfaction functions, we corroborated that, beyond a certain completion ($\approx$ 75%), the addition of more documentation reduces the SAD global utility. There is a "sweet spot" for applying optimized documentation when the SAD has a 25-40% of completion.

The current NSVP formulation still has shortcomings, mostly related to the assumptions of our model, such us stakeholders' interests on views, the completion actions for templates, the satisfaction functions, and the cost measures. These assumptions are either based on the authors' experience or taken from the literature, but they must be further validated (e.g., with user studies). We need to empirically investigate the correlations between the satisfaction computed by the functions and the actual stakeholders' satisfaction [6]. Regarding the SAD structure, some aspects ignored in today's model include: dependencies between SAD sections, or varied documentation actions (e.g., updating a section due to system refactoring, or deleting a section). These extensions to the NSVP pose a more complex optimization problem, and might emphasize the role of heuristic solvers. As for the V&B schema of stakeholder preferences, we will enhance it with user profiling techniques that incorporate personal dynamic interests, instead of using just predefined roles. Finally, we plan to consider stakeholders' concerns that might crosscut the SAD with the support of document processing/recognition techniques. For instance, interests on topics, such us system features or quality attributes, that affect more than one view or document.

Acknowledgments. This work was supported by PICT Project 2011-0366 (ANPCyT) and also by PIP Project 112-201101-00078 (CONICET) - Argentina.

References

1. Ruping, A.: Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects. John Wiley & Sons (2003)
2. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 3rd edn. Addison-Wesley Professional (2012)
3. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond, 2nd edn. Addison-Wesley Professional (2010)
4. ISO/IEC/IEEE: ISO/IEC/IEEE 42010: Systems and software engineering - architecture description. C/S2ESC. IEEE Computer Society (2011)
5. Parnas, D.L.: Precise documentation: The key to better software. In: Nanz, S. (ed.) The Future of Software Engineering, Springer, pp. 125–148. Springer (2010)
6. Falessi, D., Briand, L.C., Cantone, G., Capilla, R., Kruchten, P.: The value of design rationale information. ACM Trans. Softw. Eng. Methodol. 22(3), 21 (2013)
7. Diaz-Pace, J.A., Nicoletti, M., Schiaffino, S., Villavicencio, C., Sanchez, L.E.: A stakeholder-centric optimization strategy for architectural documentation. In: Cuzzocrea, A., Maabout, S. (eds.) MEDI 2013. LNCS, vol. 8216, pp. 104–117. Springer, Heidelberg (2013)
8. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: A practical method for documenting software architectures. In: Proceedings of the 25th ICSE (2003)
9. Bagnall, A., Rayward-Smith, V., Whittley, I.: The next release problem. Information and Software Technology 43(14), 883–890 (2001)
10. Zhang, Y., Harman, M., Mansouri, S.A.: The multi-objective next release problem. In: Proceedings of the 9th GECCO, pp. 1129–1137. ACM, New York (2007)
11. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: Proceedings Working Conf. on Software Architecture, pp. 109–120 (2005)
12. Mitchell, R.K., Agle, B.R., Wood, D.J.: Toward a theory of stakeholder identification and salience: Defining the principle of who and what really counts. Academy of Management Review 22, 853 (1997)
13. Farenhorst, R., van Vliet, H.: Experiences with a wiki to support architectural knowledge sharing. In: Proc. 3rd Wikis4SE, Portugal (2008)
14. Bachmann, F., Merson, P.: Experience using the web-based tool wiki for architecture documentation. Technical Report CMU/SEI-2005-TN-041, SEI, CMU (2005)
15. Lattanze, A.: Architecting Software Intensive Systems: A Practitioners Guide. Taylor & Francis (2008)
16. Rozanski, N., Woods, E.: Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, 2nd edn. Addison-Wesley (2011)
17. Koning, H., Vliet, H.V.: Real-life it architecture design reports and their relation to ieee std 1471 stakeholders and concerns. Auto. Soft. Eng. 13, 201–223 (2006)
18. Diwekar, U.: Introduction to Applied Optimization, 2nd edn. Springer Publishing Company, Incorporated (2010)
19. Berre, D.L., Parrain, A.: The SAT4J library, release 2.2. JSAT 7(2-3), 59–64 (2010)
20. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. Trans. Evol. Comp. 6(2), 182–197 (2002)
21. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Comput. Surv. 45(1), 1–11 (2012)
22. Durillo, J.J., Zhang, Y., Alba, E., Harman, M., Nebro, A.J.: A study of the bi-objective next release problem. Empirical Softw. Eng. 16(1), 29–60 (2011)