JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. 2015; **27**:674–699
Published online 21 July 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/smr.1734

An optimization-based tool to support the cost-effective production of software architecture documentation

Matias Nicoletti[1,2,*,†], Silvia Schiaffino[1,2] and J. Andres Diaz-Pace[1,2]

[1]*ISISTAN Research Institute, CONICET-UNICEN, Tandil, Argentina*
[2]*National Scientific and Technical Research Council (CONICET), Buenos Aires, Argentina*

ABSTRACT

Some of the challenges faced by most software projects are tight budget constraints and schedules, which often make managers and developers prioritize the delivery of a functional product over other engineering activities, such as software documentation. In particular, having little or low-quality documentation of the software architecture of a system can have negative consequences for the project, as the architecture is the main container of the key design decisions to fulfill the stakeholders' goals. To further complicate this situation, generating and maintaining architectural documentation is a non-trivial and time-consuming activity. In this context, we present a tool approach that aims at (i) assisting the documentation writer in their tasks and (ii) ensuring a cost-effective documentation process by means of optimization techniques. Our tool, called *SADHelper*, follows the principle of producing reader-oriented documentation, in order to focus the available, and often limited, resources on generating just enough documentation that satisfies the stakeholders' concerns. The approach was evaluated in two experiments with users of software architecture documents, with encouraging results. These results show evidence that our tool can be useful to reduce the documentation costs and even improve the documentation quality, as perceived by their stakeholders. Copyright © 2015 John Wiley & Sons, Ltd.

Received 11 July 2014; Revised 3 May 2015; Accepted 10 June 2015

KEY WORDS: software architecture; architecture documentation; multi-objective optimization; stakeholders; tool support; wikis

1. INTRODUCTION

The software architecture of a computing system is the set of structures needed to reason about that system, which comprises software elements, relations among them, and properties of both [1]. Design decisions are essential when developing an architecture solution, as they record the design rationale behind this solution. Both structures and decisions are usually captured in the so-called software architecture document (SAD). Architectural documentation is a key artifact to achieve success in a software project. Even the best architectural solution could fail its purpose if it is not properly documented, distributed, and understood by the system stakeholders [2].

Previous studies [3, 4] have analyzed the consequences of developing software projects with poor or little architectural documentation. Although architecture documentation is essential to support the development, maintenance, and evolution of software systems [5], the production of adequate documentation still faces many issues in practice. For instance, in iterative and incremental development (IID) projects [6], in which budget constraints and tight schedules are common, documentation activities might be considered as low-priority to favor the development of

*Correspondence to: Matias Nicoletti, CONICET-UNICEN, ISISTAN.
†E-mail: matiasnicoletti@gmail.com

user-visible (functional) features. This choice is detrimental of the SAD, which often becomes a slim document and gets quickly out-of-date.

Assuming a context of limited resources to deal with documentation tasks, we argue that the production of architectural documentation should be planned. We seek for a cost-effective documentation plan to deliver incremental SAD versions that contain good enough contents. Because architectural documentation tends to be more valuable for both stakeholders and the project, to the extent that it fulfills the key stakeholders' information needs, the documentation writer (or documenter)[‡] should follow a *reader-oriented documentation approach*, in which the actual SAD contents are steered by the stakeholders' concerns (e.g., interests or preferences on specific architectural views). This is a recommended practice in current standards for software architecture documentation [7, 8].

Determining a cost-effective plan for a given SAD is not straightforward, especially in scenarios involving medium-to-large SADs and various stakeholders. The problem for the documenter lies in choosing the SAD sections to work on, and their required level of detail, which can be a time-consuming and error-prone activity when performed manually. Given the combinatorial characteristics of this problem, we introduce an optimization-based tool, called *SADHelper*, which works as an assistant to the documenter for the generation of useful and low-cost versions of SADs.

Our approach is based on two main principles. First, in order to produce reader-oriented documentation, stakeholders' interests with respect to architectural documentation are assessed prior to the delivery of each SAD version. Along this line, our approach uses the Views & Beyond (V&B) documentation method [2, 9]. Second, the creation of plans for updating a SAD is modeled with a bi-objective optimization formulation, derived from the multiple-choice knapsack problem (MCKP) [10, 11]. To this end, *SADHelper* uses a heuristic algorithm to solve SAD problem instances with small computational times.

Our tool approach was evaluated in two experiments involving users working with SADs within a simulated environment. The main goal of our evaluation was to assess the optimization aspect of *SADHelper*, in order to check whether its advice reduces the effort to produce SAD versions with (at least) the same quality level than traditional documentation (i.e., produced without assistance). Although preliminary, the results from these experiments have been encouraging, showing that the assistant can effectively help to reduce documentation costs and lead to SAD contents being well-aligned with the stakeholders' concerns.

The rest of this article is organized as follows. Section 2 presents an overview of the *SADHelper* tool and describes its main features. Section 3 discusses the concept of reader-oriented documentation and explains how stakeholders' interests are modeled with Views and Beyond. Section 4 presents a formulation for the problem of computing a SAD documentation plan and describes a number of optimization algorithms to approach it. Section 5 describes the evaluation of our approach. Section 6 discusses related work. Finally, Section 7 gives the conclusions and outlines future work.

2. *SADHELPER*: AN ARCHITECTURAL DOCUMENTATION ASSISTANT

Let us suppose the following scenario. We have a community of stakeholders working on an IID project about a critical software system. Since the project is under a fixed-price contract and the budget is limited, the project manager has decided to prioritize the delivery of a functional product over the production of extensive architectural documentation. Nonetheless, in this sort of project, documenting the software architecture is essential to support the development (and evolution) of a high-quality system. In this scenario, the documenter should follow a cost-effective documentation process, making a balance between spending the least possible effort to produce a SAD and enabling the system stakeholders to obtain the most of it. In IID projects, the documentation can be delivered in incremental versions (across development iterations) [12, 13]. Then, the documenter must determine a backlog with a number of documentation tasks as well as a schedule for them, in such a way a fixed (small) documentation effort is allocated to each iteration. Examples of such

[‡]This role is usually played by members of the architecture team.

documentation tasks might be the creation of a new SAD section, the modification of an existing section in order to provide more details, among others. We refer to this backlog or schedule as a *SAD update plan*.

To assist the documenter in the SAD documentation process, we introduce *SADHelper* as a Web application that facilitates the management of documentation tasks and keeps track of the stakeholders' interests on the current SAD version. The main objective of *SADHelper* is to make recommendations to the documenter for the generation of reader-oriented SAD versions while considering effort constraints. Figure 1 presents a conceptual schema of *SADHelper*. Stakeholders consume architectural knowledge from a Wiki-based SAD. Stakeholders' concerns about the SAD contents are captured in a matrix that links stakeholders with SAD sections (Wiki documents) in terms of levels of interest. When a new version of a SAD needs to be delivered, *SADHelper* computes candidate SAD update plans according to both the actual state of completion of the SAD sections and the matrix of stakeholders' interests. Our assistant shows these plans to the documenter, who then can pick a plan and apply any of its tasks on the current SAD. Note that the execution of the tasks suggested by the tool is not mandatory. The documenter is expected to make the final documentation decisions for the SAD based on factors such as expertise, domain knowledge, or past experiences. This documentation process is cyclical, and several SAD versions with corresponding update plans might be generated as the development project moves forward.

2.1. Wiki-based architectural documentation

SADHelper assumes that SADs are documented using the V&B method [9] and Wikis [14]. Then, a SAD is structured around a main template plus templates for predefined architectural views. A view presents an aspect or viewpoint of the system (e.g., static aspects, runtime aspects, allocation of software elements to hardware, etc.). Each document (or Wiki article) represents a section of the template, such as a Deployment view or a Context diagram explaining the background of the architecture. The decision to use Wikis as the container of SADs was based on our own experiences, as well as on successful experiences reported by the software architecture community [15–17].

An example of a Wiki-based SAD for a system called 'Adventure Builder' is shown in Figure 2. The Wiki home page is the SAD main document containing links to the different SAD sections with the architectural views. In this example, some sections have already been documented with varying levels of detail. For instance, there is a Deployment view that lacks details, but presents an overview of how software components are mapped to hardware. There is also a Module view of a sub-system, which is detailed and describes the implementation units of that sub-system as well as their usage relations.

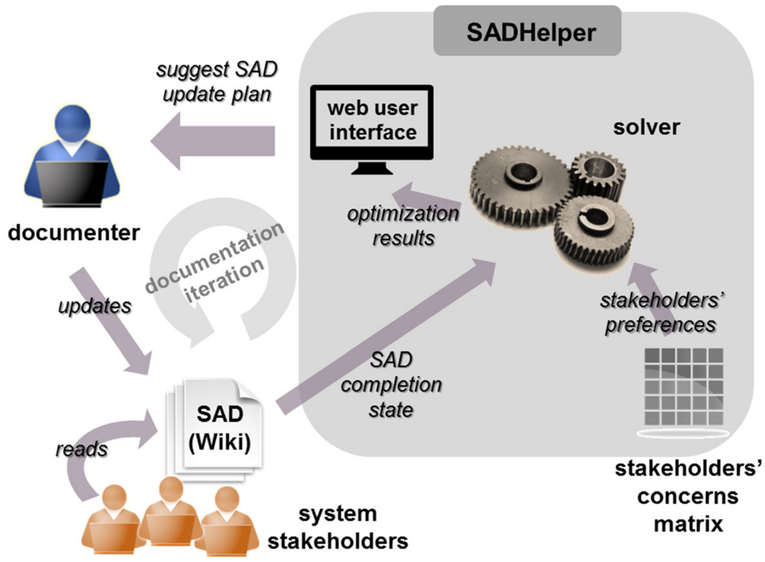
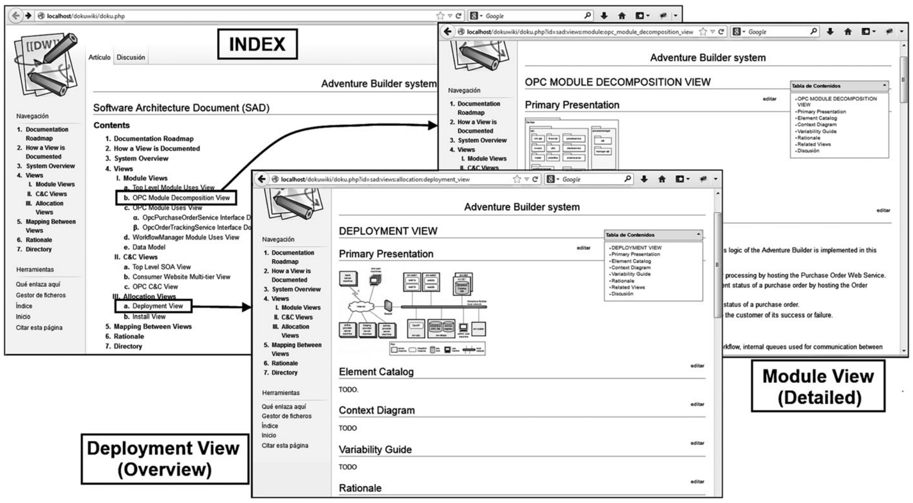


Figure 1. Overview of the working context of *SADHelper*. SAD, software architecture document.



localhost/dokuwiki/doku.php

Adventure Builder system

Software Architecture Document (SAD)

Contents

1. Documentation Roadmap
2. How a View is Documented
3. System Overview
4. Views
   I. Module Views
      a. Top Level Module Uses View
      b. OPC Module Decomposition View
      c. OPC Module Uses View
         α. OpcPurchaseOrderService Interface D
         β. OpcOrderTrackingService Interface Do
      d. WorkflowManager Module Uses View
      e. Data Model
   II. C&C Views
      a. Top Level SOA View
      b. Consumer Website Multi-tier View
      c. OPC C&C View
   III. Allocation Views
      a. Deployment View
      b. Install View
5. Mapping Between Views
6. Rationale
7. Directory

Navegación

1. Documentation Roadmap
2. How a View is Documented
3. System Overview
4. Views
   I. Module Views
   II. C&C Views
   III. Allocation Views
5. Mapping Between Views
6. Rationale
7. Directory

Herramientas

Qué enlaza aquí
Gestor de ficheros
Índice
Inicio
Citar esta página

localhost/dokuwiki/doku.php?id=sad:views:module:opc_module_decomposition_view

Adventure Builder system

OPC MODULE DECOMPOSITION VIEW

Primary Presentation

editar

Tabla de Contenidos
- OPC MODULE DECOMPOSITION VIEW
- Primary Presentation
- Element Catalog
- Context Diagram
- Variability Guide
- Rationale
- Related Views
- Discusión

s logic of the Adventure Builder is implemented in this

processing by hosting the Purchase Order Web Service.
nt status of a purchase order by hosting the Order

status of a purchase order.
the customer of its success or failure.

rkflow, internal queues used for communication between

localhost/dokuwiki/doku.php?id=sad:views:allocation:deployment_view

Adventure Builder system

DEPLOYMENT VIEW

Primary Presentation

editar

Tabla de Contenidos
- DEPLOYMENT VIEW
- Primary Presentation
- Element Catalog
- Context Diagram
- Variability Guide
- Rationale
- Related Views
- Discusión

Element Catalog

TODO.

Context Diagram

TODO

Variability Guide

TODO

Rationale

Figure 2. Example of a Software Architecture Document version hosted in a Wiki.

2.2. *Dealing with stakeholders' information needs*

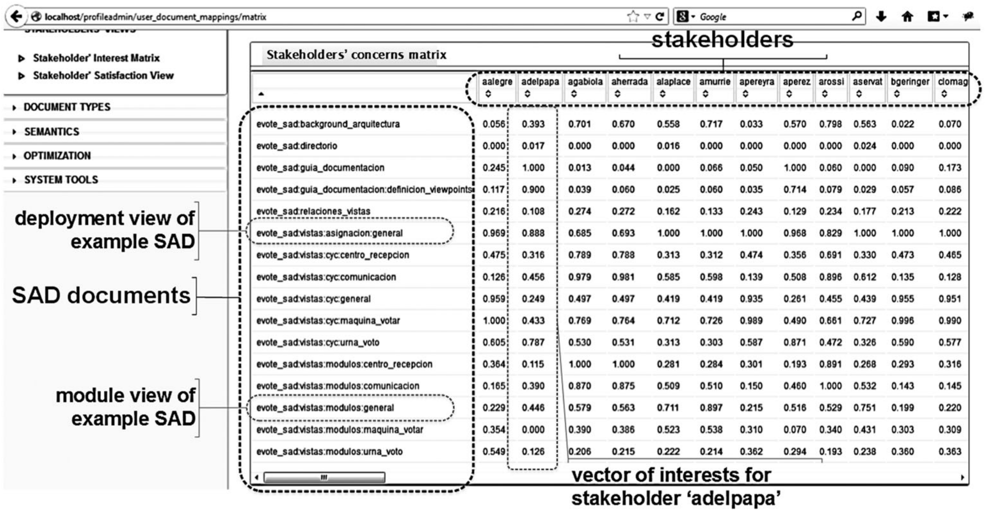
Producing reader-oriented documentation is key to performing a cost-effective documentation process. It is argued that documentation should be written from a readers' perspective, rather than from a writers' perspective [2]. The SAD is not an exception to such a rule. Along this line, stakeholders' interests should be assessed before producing the SAD, so as to ensure it addresses the actual stakeholders' information needs. In fact, both current standards on software architecture [8] and research works [18] have recognized the importance of considering stakeholders' concerns in the documentation process.

The quality of reader-oriented documentation is strongly linked to the satisfaction of the key (high-priority) stakeholders. The satisfaction of a given stakeholder can be seen as directly proportional to the degree of coverage of her interests (by the current SAD version) [2, 8, 19]. *SADHelper* models stakeholders' concerns with a matrix that stores the level of interest of the current SAD section (columns) for a given stakeholder (row), as suggested in the V&B approach. We argue that the information of this matrix is useful to steer a reader-oriented documentation process. We refer to this model as the *stakeholders*' *concerns matrix* (SCM). More details on how this matrix is populated can be found in Section 3.

Figures 3 and 4 show snapshots of *SADHelper* features related to the capture of stakeholders' interests. Figure 3 shows how the SCM is managed within the tool. For instance, the documenter can adjust the level of interest of specific stakeholders in the architectural views of Figure 2. To this end, a [0,1] scale is used, in which 1 means full interest. Figure 4 depicts a view of the stakeholders' levels of satisfaction for the current completion state of the SAD. The documenter can drill down the satisfaction value for a specific stakeholder and get a detailed description of the satisfaction at the document level. Thus, the documenter can detect which documents are obstacles to the satisfaction of a key stakeholder, and thus perform the necessary corrective actions (i.e., documentation tasks).

2.3. *Cost-effective documentation*

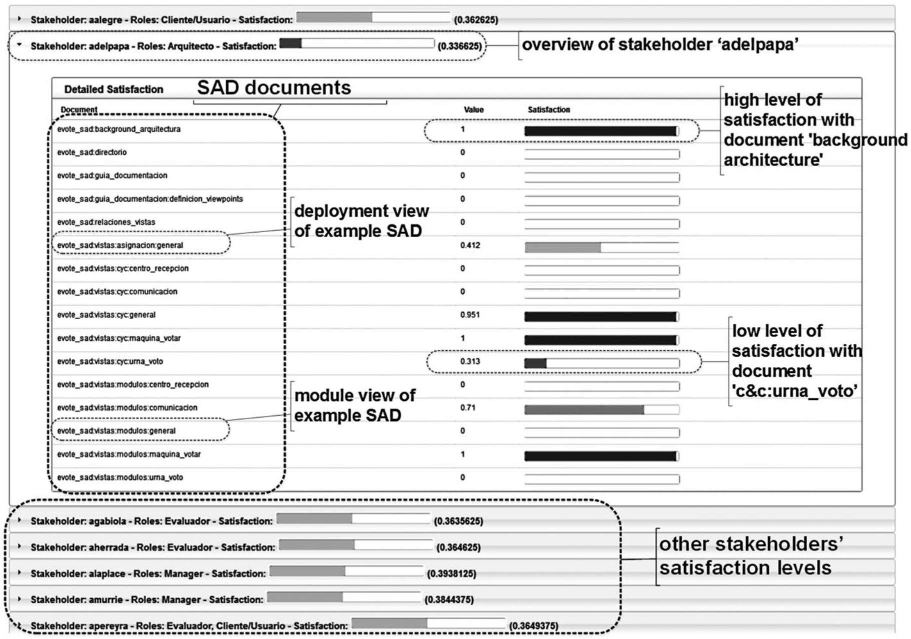
Once stakeholders' concerns have been assessed, a SAD update plan can be computed. Although this activity might be performed manually by the documenter in small settings (i.e., small-sized SADs and few stakeholders), it becomes a combinatorial problem often with a high time complexity in larger settings. The problem is similar to the well-known Next Release Problem (NRP) [20, 21]. In addition, this computation is complicated by other factors, such as the following: efforts allocated to the tasks (e.g., person-hours), priorities of the stakeholders, or conflicting optimization objectives (e.g., minimizing production costs and maximizing product quality), among others [2, 22].


localhost/profileadmin/user_document_mappings/matrix
Google
Stakeholder' Interest Matrix
Stakeholder' Satisfaction View
DOCUMENT TYPES
SEMANTICS
OPTIMIZATION
SYSTEM TOOLS
deployment view of example SAD
SAD documents
module view of example SAD
stakeholders
vector of interests for stakeholder 'adelpapa'

Stakeholders' concerns matrix

	aalegre	adelpapa	agabiola	aherrada	alaplace	amurrie	apereyra	aperez	arossi	aservat	bgeringer	clomag
evote_sad:background_arquitectura	0.056	0.393	0.701	0.670	0.558	0.717	0.033	0.570	0.798	0.563	0.022	0.070
evote_sad:directorio	0.000	0.017	0.000	0.000	0.016	0.000	0.000	0.000	0.000	0.024	0.000	0.000
evote_sad:guia_documentacion	0.245	1.000	0.013	0.044	0.000	0.066	0.050	1.000	0.060	0.000	0.090	0.173
evote_sad:guia_documentacion:definicion_viewpoints	0.117	0.900	0.039	0.060	0.025	0.060	0.035	0.714	0.079	0.029	0.057	0.086
evote_sad:relaciones_vistas	0.216	0.108	0.274	0.272	0.162	0.133	0.243	0.129	0.234	0.177	0.213	0.222
evote_sad:vistas:asignacion:general	0.969	0.888	0.685	0.693	1.000	1.000	1.000	0.968	0.829	1.000	1.000	1.000
evote_sad:vistas:cyc:centro_recepcion	0.475	0.316	0.789	0.788	0.313	0.312	0.474	0.356	0.691	0.330	0.473	0.465
evote_sad:vistas:cyc:comunicacion	0.126	0.456	0.979	0.981	0.585	0.598	0.139	0.508	0.896	0.612	0.135	0.128
evote_sad:vistas:cyc:general	0.959	0.249	0.497	0.497	0.419	0.419	0.935	0.261	0.455	0.439	0.955	0.951
evote_sad:vistas:cyc:maquina_votar	1.000	0.433	0.769	0.764	0.712	0.726	0.989	0.490	0.661	0.727	0.996	0.990
evote_sad:vistas:cyc:urna_voto	0.605	0.787	0.530	0.531	0.313	0.303	0.587	0.871	0.472	0.326	0.590	0.577
evote_sad:vistas:modulos:centro_recepcion	0.364	0.115	1.000	1.000	0.281	0.284	0.301	0.193	0.891	0.268	0.293	0.316
evote_sad:vistas:modulos:comunicacion	0.165	0.390	0.870	0.875	0.509	0.510	0.150	0.460	1.000	0.532	0.143	0.145
evote_sad:vistas:modulos:general	0.229	0.446	0.579	0.563	0.711	0.897	0.215	0.516	0.529	0.751	0.199	0.220
evote_sad:vistas:modulos:maquina_votar	0.354	0.000	0.390	0.386	0.523	0.538	0.310	0.070	0.340	0.431	0.303	0.309
evote_sad:vistas:modulos:urna_voto	0.549	0.126	0.206	0.215	0.222	0.214	0.362	0.294	0.193	0.238	0.360	0.363

Figure 3. *SADHelper*: administration of the stakeholders' concerns matrix. SAD, software achitecture document.


overview of stakeholder 'adelpapa'
SAD documents
high level of satisfaction with document 'background architecture'
deployment view of example SAD
low level of satisfaction with document 'c&c:urna_voto'
module view of example SAD
other stakeholders' satisfaction levels

Stakeholder: aalegre - Roles: Cliente/Usuario - Satisfaction: (0.362625)

Stakeholder: adelpapa - Roles: Arquitecto - Satisfaction: (0.336625)

Detailed Satisfaction

Document	Value	Satisfaction
evote_sad:background_arquitectura	1	
evote_sad:directorio	0	
evote_sad:guia_documentacion	0	
evote_sad:guia_documentacion:definicion_viewpoints	0	
evote_sad:relaciones_vistas	0	
evote_sad:vistas:asignacion:general	0.412	
evote_sad:vistas:cyc:centro_recepcion	0	
evote_sad:vistas:cyc:comunicacion	0	
evote_sad:vistas:cyc:general	0.951	
evote_sad:vistas:cyc:maquina_votar	1	
evote_sad:vistas:cyc:urna_voto	0.313	
evote_sad:vistas:modulos:centro_recepcion	0	
evote_sad:vistas:modulos:comunicacion	0.71	
evote_sad:vistas:modulos:general	0	
evote_sad:vistas:modulos:maquina_votar	1	
evote_sad:vistas:modulos:urna_voto	0	

Stakeholder: agabiola - Roles: Evaluador - Satisfaction: (0.3635625)

Stakeholder: aherrada - Roles: Evaluador - Satisfaction: (0.364625)

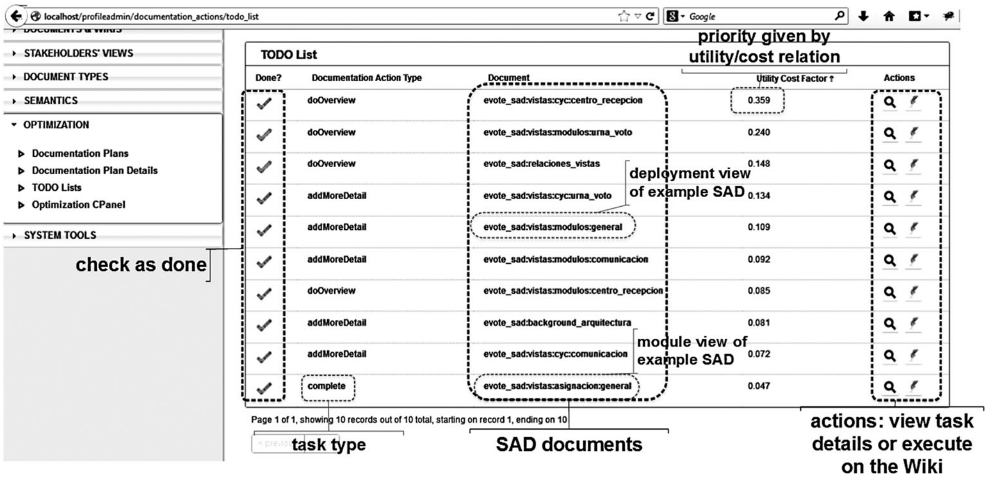
Stakeholder: alaplace - Roles: Manager - Satisfaction: (0.3938125)

Stakeholder: amurrie - Roles: Manager - Satisfaction: (0.3844375)

Stakeholder: apereyra - Roles: Evaluador, Cliente/Usuario - Satisfaction: (0.3649375)

Figure 4. SADHelper: stakeholders' satisfaction view. SAD, software achitecture document.

Given the complexity of computing a near-optimal SAD update plan, *SADHelper* assists the documenter in such an activity by focusing on the aspects of cost and benefit of the documentation tasks to be included in an update plan. The assistant generates update plan(s) as a backlog (or to-do-list) of documentation tasks, as shown in Figure 5. For instance, if we consider the Deployment view of the SAD of Figure 2, which has an overview level of detail, a candidate documentation task could be to add more detail to this document. When the documenter selects one of the suggested tasks, the assistant opens the target Wiki document in edit mode and offers writing guidelines to perform the task (e.g., expected content of the section, how to reach the required detail level). Additionally, the documenter can visualize the effect of the task on the stakeholders' satisfaction and, thereby, understand the reason why that task was suggested. More details on how this computation is implemented can be found in Section 4.


localhost/profileadmin/documentation_actions/todo_list
Google
STAKEHOLDERS' VIEWS
DOCUMENT TYPES
SEMANTICS
OPTIMIZATION
Documentation Plans
Documentation Plan Details
TODO Lists
Optimization CPanel
SYSTEM TOOLS
check as done
priority given by utility/cost relation
deployment view of example SAD
module view of example SAD
task type
SAD documents
actions: view task details or execute on the Wiki

TODO List

Done?	Documentation Action Type	Document	Utility Cost Factor ↑	Actions
✓	doOverview	evote_sad:vistas:cyc:centro_recepcion	0.359	
✓	doOverview	evote_sad:vistas:modulos:urna_voto	0.240	
✓	doOverview	evote_sad:relaciones_vistas	0.148	
✓	addMoreDetail	evote_sad:vistas:cyc:urna_voto	0.134	
✓	addMoreDetail	evote_sad:vistas:modulos:general	0.109	
✓	addMoreDetail	evote_sad:vistas:modulos:comunicacion	0.092	
✓	doOverview	evote_sad:vistas:modulos:centro_recepcion	0.085	
✓	addMoreDetail	evote_sad:background_arquitectura	0.081	
✓	addMoreDetail	evote_sad:vistas:cyc:comunicacion	0.072	
✓	complete	evote_sad:vistas:asignacion:general	0.047	

Page 1 of 1, showing 10 records out of 10 total, starting on record 1, ending on 10

Figure 5. SADHelper: backlog of documentation tasks (update plan). SAD, software achitecture document.

3. MODELING STAKEHOLDERS' INTERESTS WITH VIEWS & BEYOND

Reader-oriented documentation requires prior knowledge of stakeholders' interests. In our approach, this knowledge is captured in an SCM, which is derived from the V&B documentation method [2, 9]. Like other current documentation methods, V&B is view-centric. The basic V&B principle is that documenting a software architecture solution involves documenting the relevant architectural views, and then documenting the information that applies to more than one view, which is called information beyond views. The latter might include the following: context information of the system, specifications of system goals, descriptions of system stakeholders, or relations between views, among other sections.

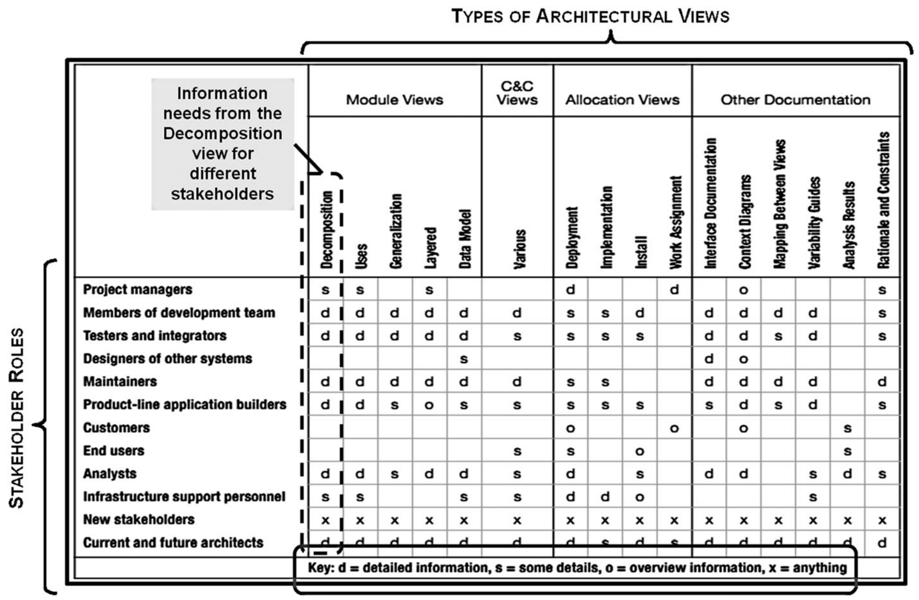
Views & Beyond helps the architect identify and record necessary architectural information during development, by providing a general SAD template and specific templates for architectural views.[§] The choice of the relevant views for the SAD depends on its anticipated usage by the stakeholders. V&B characterizes several types of stakeholders regarding their use of architectural views, as depicted by the matrix of Figure 6. A cell of the matrix indicates the information of view X (e.g., decomposition, deployment) needed by stakeholder role Y (e.g., developer, maintainer, manager). Each column can be seen as stakeholder preferences on the contents of a SAD view.

In a particular project, defining the 'right' matrix is a crucial activity for the organization of the SAD and its contents. In spite of proposing a generic model of information needs, which might apply to several projects, the matrix must be customized 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, etc.). In practice, the contents of the matrix are typically refined via interviews with stakeholders during the initial stages of the project [2]. For instance, in small software projects, there might be less different roles than the ones considered by the generic model (e.g., only managers, developers, and the client), or the number of stakeholders could be small enough so as to consider individual preferences instead of grouping them by their roles.

An important consideration for using V&B in our approach is the need to relate the detail levels of SAD sections to concrete documentation contents, so that the documenter can figure out the 'right' amount of documentation for fulfilling a set of stakeholders' needs. In Section 4.1, we discuss a mapping between increasing levels of detail (also referred to as document states) and sub-sections of the V&B templates. For example, according to our mapping, the architectural views of Figure 2 were classified as overview and detailed, for the Deployment and Module views, respectively.

Note that the V&B matrix (Figure 6) and the SCM of *SADHelper* (Figure 2) are slightly different. First, the V&B matrix shows types of stakeholders (roles) and architectural viewtypes, while the SCM

[§]V&B templates: http://www.sei.cmu.edu/downloads/sad/SAD_template_05Feb2006.dot


TYPES OF ARCHITECTURAL VIEWS
STAKEHOLDER ROLES
Information needs from the Decomposition view for different stakeholders

	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		o				s
Members of development team	d	d	d	d	d	d	s	s	d		d	d	d	d		s
Testers and integrators	d	d	d	d	d	s	s	s	s		d	d	s	d		s
Designers of other systems					s						d	o				
Maintainers	d	d	d	d	d	d	s	s			d	d	d	d		d
Product-line application builders	d	d	s	o	s	s	s	s	s		s	d	s	d		s
Customers							o			o		o			s	
End users						s	s		o						s	
Analysts	d	d	s	d	d	s	d		s		d	d		s	d	s
Infrastructure support personnel	s	s			s	s	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

Figure 6. Views & Beyond characterization of stakeholders' preferences on architectural views [2].

models individual stakeholders (i.e., persons playing stakeholder roles of V&B) and instances of architectural views (as prescribed by the V&B viewtypes). Second, the cells of the V&B matrix deal with detail level using an ordinal scale, while those of SCM refer to interest degree using a numerical scale. Therefore, to translate from the V&B matrix to the SCM (*SADHelper*), we simply assign a numeric value to each ordinal value of V&B, namely: anything $\rightarrow 0$, overview $\rightarrow 13$, more detail $\rightarrow 23$, and detailed $\rightarrow 1$. Assuming that the SCM of Figure 3 was built out of the original V&B matrix for the SAD of Figure 2, we could say that developers, testers, and maintainers might be mostly satisfied, whereas managers, who are likely to require more details from allocation views, might not.

An important drawback of existing documentation methods is that they provide few or no guidelines to the documenter for creating the documentation package [2, 23–25]. Although V&B offers a set of basic steps and rules to organize and develop the SAD, they are generic and do not necessarily ensure a cost-effective documentation process. Creating a good SAD can still be a time-consuming activity that strongly depends upon the documenter's skills and does not guarantee stakeholders' satisfaction [3]. This is the reason why we envision *SADHelper* as a supporting tool to apply the V&B method in IID projects.

4. COMPUTATION OF UPDATE PLANS AS AN OPTIMIZATION PROBLEM

In our context, an update plan is a set of documentation tasks that must be performed to deliver a new SAD version. This plan is expected to optimize the cost-benefit relation of that SAD version. The benefit is a function of the satisfaction of the stakeholders that will use the SAD, while the cost comes from the amount of effort (i.e., human resources) needed to carry out the tasks of the plan. These costs might be estimated by the documenter, a project manager, or by mutual agreement between them. For instance, the documenter could provide ballpark estimates (often, in collaboration with the manager or experienced architects), based on the following: historical effort information, importance of certain views, or number of design decisions involved, among others [2, 25].

Specifically, we say that a stakeholder is satisfied if the (current) SAD covers her information needs, which are, in fact, the set of SAD sections/views relevant to that stakeholder [8]. The priorities of the stakeholders in the project also influence the computation of satisfaction [19, 26]. Thus, the computation must not only maximize the overall satisfaction of the stakeholders but also target those stakeholders with the highest priority. In other words, the quality of a SAD is directly related to satisfaction of the key (high-priority) stakeholders [2, 8]. We refer to this quality measure as the

SAD utility, which can be seen as weighted satisfaction metric, in which the weights are related to the importance of the stakeholders.

Certainly, there is a trade-off between maximizing the utility of a SAD version and minimizing the cost of producing it. To begin with, we could set a fixed production cost (corresponding to the allowed documentation expense for one development iteration) and let a solver find SAD solutions with high utility that do not violate the cost threshold. This perspective would lead to a constrained mono-objective optimization, as in the original NRP formulation [20]. In practice, we might end up exploring SAD solutions for different cost thresholds, that is, solving different mono-objective optimization problems. This exploration process is natural in a cost-benefit analysis, because it is difficult for a decision maker to determine a priori (i) how much benefit can be sacrificed in order to considerably reduce cost or (ii) how much cost increase she can tolerate in exchange for a perceived high benefit. In such cases, the production cost is no longer a constraint but an objective, and our problem becomes a bi-objective optimization [21, 27, 28].

In this context, instead of a single SAD solution, we let a solver find a set of non-dominated solutions, which are solutions that dominate the others but do not dominate themselves in the Pareto sense. This set of solutions is known as *the Pareto front*. By analyzing this set, the decision maker can get insights on the trade-offs produced by cost and benefit and select the most convenient solution (i.e., a SAD update plan) depending on the context. We believe that the bi-objective aspect is a desirable feature in *SADHelper*, because it enables the documenter to make her documentation choices considering factors such as the following: the priority of the objectives at the moment of the decision, or the feasibility of executing the resulting update plans, among others.

4.1. Formulation of the optimization problem

We assume an IID software project in which the documenter releases several versions of the SAD, which is built incrementally. Conceptually, the SAD is composed of a set of n individual documents (e.g., Wiki pages), and each document generally corresponds to a SAD section according to the V&B templates. Let $SAD_t = \{D_1, \ldots, D_n\}$ 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 $(1 \le k \le n)$ is the detail level of document D_k at time t. We assume that a document d_k can be in one of four possible states [2, 9] so as to keep its evolution manageable, as shown in Equation (1).

$$d_k = \begin{cases} \textit{empty} & (\text{e.g.,}\ \ 0) \\ \textit{overview} & (\text{e.g.,}\ \ 13) \\ \textit{some\_detail} & (\text{e.g.,}\ \ 23) \\ \textit{detailed} & (\text{e.g.,}\ \ 1) \end{cases} \quad (1 \le k \le n) \tag{1}$$

Given a partially documented SAD_t, the documenter must select an update plan in order to produce a SAD_{t+1} with additional contents. Initially, we define a list L with all the possible tasks applicable to SAD_t. That is, $L = \{a_{11}, a_{21}, \ldots, a_{12} \ldots, a_{ln}\}$ where a_{jk} $(1 \le j \le l, 1 \le k \le n)$ is one feasible task (out of l possible tasks) for document D_k that leads to a state change $d_k d_k^{'}$. The computation of the update plan is constrained by a fixed-cost iteration, and the most valuable tasks from L must be selected. This entails a combinatorial optimization known as the 0–1 knapsack problem (0-1KP) [29].

In particular, we deal with a variant called the multiple-choice knapsack problem (MCKP) [10, 11]. The MCKP is stated as follows: given n classes $[D_1, \ldots, D_n]$ of a total of l items to pack in some knapsack of capacity C, each item $j \in D_k$ has a benefit b_{jk} and a cost c_{jk} $(1 \le j \le l, 1 \le k \le n)$, and the problem is to choose one item from each class such that the benefit sum is maximized without exceeding a total cost of C. The classes $[D_1, \ldots, D_n]$ are mutually disjoint. In our case, these classes become the SAD documents, while the items correspond to the tasks available per section (note that the same indexes were previously used in order to highlight the similarities).

Let us consider a set of atomic tasks $AT = \{\textit{doNothing}, \textit{doOverview}, \textit{addMoreDetail}, \textit{complete}\}$. An atomic task, if applicable, increases the detail of a SAD section. This effort is quantified as the cost of the task. Furthermore, we map these tasks to the parts of the view template of V&B, as depicted in Figure 7. Task *doOverview* starts with an *empty* section and asks (the documenter) to complete the

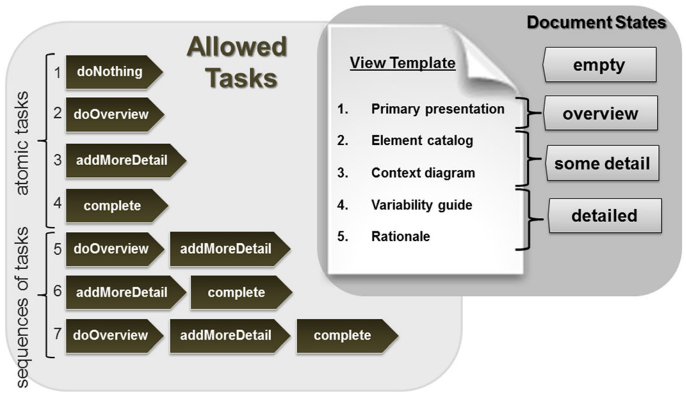


Figure 7. Examples of documentation tasks, based on the Views & Beyond view template.

'Primary presentation' part of the view. Task *addMoreDetail* suggests completing both the 'Element catalog' and 'Context diagram' parts of a section (having already an *overview* state). Task *complete* recommends filling out the two pending parts: 'Variability guide' and 'Rationale'. At last, task *doNothing* models the null action, which has a cost equal to 0 and leaves a document unmodified.

In our model of tasks and document states, we consider that a document has a given state of completion if certain sub-sections of it are completed. Nonetheless, this assumption does not mean that the remaining parts should not be empty. For example, the design rationale (in the 'Rational' sub-section) is normally written in parallel with other sub-sections, but having this sub-section completed is only required for the *detailed* level of the containing document. Also, notice that the proposed model can be extended with other kinds of tasks, or support other templates with different sub-sections. For instance, we might include a task that modifies an already-documented section because of an architecture design change. Regarding the four-state range and its mapping to the sections of a SAD document, it is inspired on the V&B matrix and follows recommended architecture documentation practices [2], but it should not be interpreted as strict.

The applicable tasks depend on the actual state of the section (document), because we do not model 'undo' tasks. Atomic tasks can be arranged in sequences. For example, if a section is in *overview*, it can only go to states *someDetail* or *detailed* by means of the task sequences $<addMoreDetail>$ or $<addMoreDetail, complete>$, respectively. Along this line, the candidate tasks for a section D_k (items of a class in MCKP) are all the allowed task sequences derivable from AT that follow the dependency chain $doOverview \rightarrow addMoreDetail \rightarrow complete$. These dependencies model the meaningful order in which atomic tasks should be performed on the V&B view template. With four atomic tasks (and no undo), we have a total of seven allowed tasks, including the *doNothing* task. Figure 7 shows a list of these tasks (left side). Note that the document states (right side) are dependent on each other, and so are the tasks in the context of the same document. However, tasks are independent when considering separate documents (i.e., the execution of a task on document X does not affect the applicable tasks for the other documents).

When it comes to the benefit of tasks (or task sequences) in L, we compute it indirectly on the basis of the stakeholders' interests over the SAD sections (or architectural views). Let $S = \{S_1, \ldots, S_m\}$ be a set of stakeholders, each S_i $(1 \leq i \leq m)$ with a priority p_i in the range $[0, 1]$, where 0 is the lowest priority and 1 is the highest one. For every document, D_k $(1 \leq k \leq n)$, a stakeholder S_i $(1 \leq i \leq m)$ has a value of interest (or preference) denoted by $interest(S_i, D_k) = v_{ik}$. In Section 3, we detailed how these values of interest are calculated.

The satisfaction of stakeholder S_i with respect to a document D_k with state d_k is given by a function, $satisfaction_{ik}(d_k, v_{ik}) \rightarrow [0, 1]$ $(1 \leq i \leq m, 1 \leq k \leq n)$, where the values for both d_k and v_{ik} are in the range $[0, 1]$. This function models the 'happiness' of the stakeholder S_i with the actual detail of the document, D_k, compared with her preferred level of detail (i.e., v_{ik}). Based on the V&B method [2, 9] and our experience with architectural documentation projects, the $satisfaction_{ik}(d_k, v_{ik})$ function can be approximated as described in Equation (2). A graphical description of the function is shown in Figure 8. Nonetheless, our definition of the satisfaction function is not final and our model admits alternative formulations.

$$satisfaction_{i,k}\left(d_k, v_{i,k}\right) = \begin{cases} 1 & if\,|d_k - v_{ik}| \le \varepsilon \\ 0 & if\, d_k - v_{ik} < -\varepsilon \\ 1 - d_k - v_{ik} & if\, d_k - v_{ik} > \varepsilon \end{cases} \tag{2}$$

The proposed function has a parameter ε, which represents the ‘allowed difference’ between the two detail levels. If the completion level d_k is higher than required (which might cause an information overload in the stakeholder), then there is a penalty proportional to the distance between v_{ik} and d_k. This function gets its maximum value when the completion level d_k is equal to the interest v_{ik} (or within the ε range). Based on the $satisfaction_{ik}(d_k, v_{ik})$ function, we compute the utility of a document state as given by Equation (3).

$$document\_utility_k(d_k) = \frac{\sum_{j=1}^{m} satisfaction_{jk}\left(d_k, v_{jk}\right).p_j}{\sum_{i=1}^{m} p_i} \tag{3}$$

We see $document\_utility_k(d_k)$ as a proxy for the benefit of task a_{jk} $(1 \le j \le l, 1 \le k \le n)$ that makes document D_k reach state d_k. Thus, each feasible task a_{jk} in L is assigned to a $benefit(a_{jk})$ $= document\_utility(d_k)$, for some document change. In the 0–1 formulation, the tasks a_{jk} only take values 0 or 1, depending on whether they are included in the update plan. Remember that a task can be either atomic or a sequence of atomic tasks. In the former case, $cost(a_{jk})$ is a constant. In the latter case, $cost(a_{jk})$ is the sum of the costs of the constituent tasks.

To complete the problem formulation, the optimization objectives must be determined. Based on the cost-benefit trade-offs involved in SAD generation, we define a bi-objective problem in which cost is minimized and (simultaneously) benefit is maximized, as expressed by Equations (4), (5), and (6). Note that this formulation can be quickly turned into a mono-objective one by transforming either Equation (5) or (4) into a constraint with a predefined (cost or utility) constant [30]. As we previously discussed, we prefer here a multi-objective formulation because it supports the exploration of alternative SAD solutions in the cost-benefit space. Furthermore, because both the SAD utility and the total cost are approximate measures, near-optimal solutions will be obtained in the Pareto front.

$$maximize: \quad sad\_utility(SAD_t) = \sum_{k=1}^{n} \sum_{j \in D_k} benefit\left(a_{jk}\right).a_{jk} \tag{4}$$

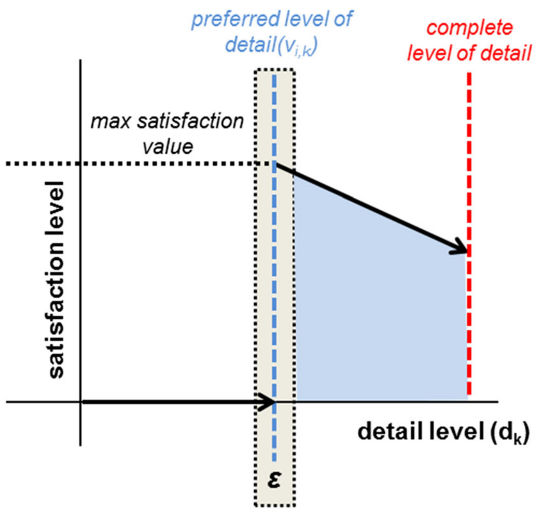


Figure 8. Graphical description of the satisfaction function.

$$minimize: \quad total\_cost(SAD_t) = \sum_{k=1}^{n} \sum_{j \in D_k} cost(a_{jk}).a_{jk} \tag{5}$$

$$subject\ to: \quad \sum_{j \in D_k} a_{jk} = 1 \tag{6}$$

4.2. Optimization techniques for the problem

Techniques to solve optimization problems can be classified as follows: (i) *exact algorithms*, which return optimal solutions but might require a high computational time depending on the size of the problem instance and (ii) *approximation or non-exact algorithms*, which produce near-optimal solutions with lower computational times. In this work, we propose a heuristic approach (non-exact) for our problem, which is based on a Greedy technique commonly employed in knapsack problems [10].

The heuristic, which is described in Algorithm 1, works in two stages. First, the applicable tasks are sorted in a decreasing order of utility-cost ratio (lines 3–4). Second, we pick as many tasks as possible beginning from the top of the ordered list until the knapsack is filled up (lines 7–15). Before selecting a task, it is checked if the execution of that task increases the global utility of the resulting SAD (lines 11–13). The idea is to 'add to the knapsack' those items (tasks) that give the most benefit per unit of cost and also ensure a benefit increment. The algorithm was adapted for bi-objective optimization by iterating over all possible costs, defining each one as a

Algorithm 1 Greedy algorithm to maximize benefit subject to a cost constraint (mono-objective).

```
1  INPUT: $A$, $DS$, $C_{max}$ //all tasks, SAD document states, and max cost
2  OUTPUT: $UP$ //final update plan
3  $CA \leftarrow$ filter-actions($A$, $DS$) //filter out actions already executed
4  sort-by-cost-benefit-ratio($CA$)
5  $B_{max} \leftarrow \varnothing$
6  $C_{current} \leftarrow \varnothing$
7  FOREACH $A_i$ IN $CA$ DO $UP_{sim} \leftarrow$ simulate-action-execution($A_i$, $DS$) //simulate the execution of
   the action
8        $B_{sim} \leftarrow$ compute-benefit ($UP_{sim}$)
9        $C_{sim} \leftarrow$ compute-cost ($UP_{sim}$)
10       IF ($B_{max} < B_{sim}$ AND $C_{sim} < C_{max}$) THEN //improves benefit and meets cost constraint
11           $UP \leftarrow$ add($UP$, $A_i$)
12           $B_{max} \leftarrow B_{sim}$
13       ENDIF
14  ENDFOREACH
15  RETURN $UP$
```

Algorithm 2 Main loop of the bi-objective Greedy algorithm.

```
1  INPUT: $A$, $DS$, $C_{limit}$ //all tasks, SAD document states, and cost limit
2  OUTPUT: $PS$ // pareto set
3  $PS \leftarrow \varnothing$
4   $C_{max} \leftarrow \varnothing$
5  WHILE ($C_{max} < C_{limit}$) DO
6        $UP \leftarrow$ compute-update-plan ($A$, $DS$, $C_{max}$) // call mono-objective greedy
7        $PS \leftarrow$ add ($PS$, $UP$) // dominance criterion applied
8   $C_{max} \leftarrow C_{max} + 1$
9  ENDWHILE
10 RETURN $PS$
```

constraint and then executing the heuristic as a benefit maximization problem. The main loop of the Greedy is given in Algorithm 2.

Our problem might be solved with other techniques as well. For instance, a well-known exact technique is Backtracking. A backtracking algorithm should progressively generate and evaluate all valid task vectors a_{jk} that derive from SAD_t. Although backtracking can be used as a baseline for other exact algorithms, it has trouble to deal with medium-to-large instances of the problem (e.g., SADs with $N \geq 15$, based on tests with synthetic data). Another exact technique, which often has a better performance than Backtracking, is SAT solvers such as the the SAT4J library [31]. The Pseudo-boolean (PB) solver of SAT4J can easily handle 0–1KP instances, but only in the form of mono-objective problems. Nonetheless, the same adaptation used in our Greedy heuristic for the bi-objective case can be applied to the PB solver. This schema might add some overhead, but still capitalizes on the good performance of SAT4J.

Non-dominated Sorting Genetic Algorithm II (NSGA-II) is a metaheuristic that has been successfully applied to optimization problems in the context of Search-Based Software Engineering [32–34]. NSGA-II uses an evolutionary process based on operators such as selection, genetic crossover, and genetic mutation, to evolve an initial population of task vectors a_{jk} through several generations until a given cut-condition is reached (e.g., a quality threshold, or a computing time-out, among others).

The adoption of a particular technique depends on the problem size (i.e., number of documents, possible states for each document, and number of stakeholders). Given the similarities of our root problem with the NRP, for whom it has already been shown, which is NP-hard [20, 35], we deemed adequate to apply non-exact techniques in the current prototype of *SADHelper*. Anyway, computational results of different techniques are discussed in Section 5.

5. EVALUATION

The main goal was to assess the usefulness of *SADHelper* in terms of SAD production costs (for the documenter) and quality of the resulting documentation (for its stakeholders). The evaluation was divided into two experiments, each one addressing a different perspective. In the first experiment, we took a theoretical perspective and performed a what-if analysis of several SADs generated with *SADHelper* against normal SADs. The second experiment had a practical perspective: we employed *SADHelper* to produce a sequence of SAD versions, and later used the stakeholders' feedback to assess their actual satisfaction with the SADs.

A secondary goal of the evaluation was to compare the performance of non-exact and exact algorithms when solving our optimization problem. In particular, we studied an exact algorithm based on the PB solver (SAT4J) and three non-exact algorithms, namely, Greedy, NSGA-II [32], and a simple Random Search [27]. The exact algorithm was aimed at providing a baseline of solutions, the non-exact algorithms. The Random Search option served as a 'sanity check', because it is known to perform worst than the other non-exact techniques. In this context, the main research questions of this work were the following:

- RQ1. Are SADs produced with the assistance of *SADHelper* better than SADs produced with no assistance? The goodness criterion is that 'SAD A is better than SAD B' if A requires less cost but gives equal or higher utility than B.
- RQ2. What optimization technique exhibits the best performance for our bi-objective problem? The performance criterion refers to both execution time and SAD utility.

Section 5.1 describes the test subjects used for the experiments. Section 5.2 explains the selection of non-dominated solutions generated with the algorithms. Sections 5.3 and 5.4 report on experiment 1 and experiment 2, respectively. Section 5.5 presents a general analysis of the results, and finally, Section 5.6 discusses the main threats to validity.

5.1. Test subjects

A total of 87 persons (i.e., SAD users) participated in both experiments. Almost 88% of the subjects were males, whereas the remaining 12% were females. Their ages ranged from 21 to 32 years old.

These users were undergraduate/graduate students of UNICEN University (Tandil, Argentina), and many of them were practitioners working on local software companies. The experiments were conducted at the end of two software architecture courses (degree in Systems Engineering and Master in Computer Science) of UNICEN University. Undergraduate students were in the last year of their bachelor degree, while the graduate students were pursuing a Master degree. At the beginning of the experiments, the subjects had learned and reinforced the key concepts on software architecture (including software architecture documentation and the V&B method). The students were asked to carry out different assignments that allowed us to evaluate the *SADHelper* approach, but these assignments did not compromise the students' learning process [36, 37]. As another motivation, previous studies have shown how students could benefit from empirical studies on Software Engineering [38].

5.2. *Selection of non-dominated solutions*

When a bi-objective optimization algorithm is used, it normally returns a set of non-dominated solutions. In our experiments, these solutions are not guaranteed to lie in the Pareto front, because our measures for the two objectives are not exact but rather approximations. The usage of heuristic techniques, such as our Greedy algorithm, also contributes to this situation. In case the decision maker is not available, generative methods such as linear scalarization [27, 28] can be applied to obtain one single solution to the optimization problem. Certainly, we do expect the documenter to be the decision maker in *SADHelper*, and thus, she will be responsible for choosing one of the non-dominated solutions based on her (subjective) judgment. For instance, she might decide to assign different weights to the cost and utility objectives.

In the evaluation procedure, we followed a strategy that prioritized the utility of solutions over their cost, in order to keep similar levels of quality in the SAD versions produced with *SADHelper*. Furthermore, we discarded those solutions with higher production costs than the normal SAD versions, in order to make a fair comparative analysis. Figure 9 shows a graphical example of how solutions were selected during the experiments. In this example, the optimization algorithm returned five non-dominated solutions. If we assume a cost threshold of 70 (i.e., producing a normal SAD version required 70 units of efforts), our strategy would consider only the set of solutions whose cost is below the threshold, and then would pick the solution with the highest utility from that set. As a result, the selected solution has a utility of 0.58 and a cost of 54 units of effort.

5.3. *Experiment 1: what-if analysis*

This experiment was intended to exercise the documentation generation capabilities of *SADHelper*. To do so, we produced a number of synthetic SAD versions (by means of update plans suggested by our

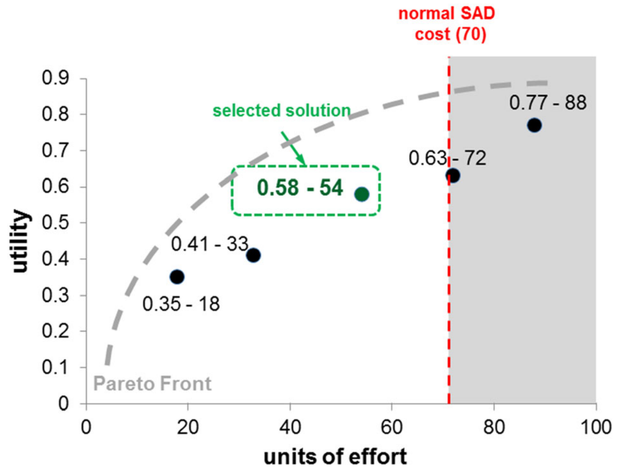


Figure 9. Example of selection of a single solution from the Pareto near-optimal set. SAD, software achitecture document.

optimization tool) and compare them against SAD versions generated in a traditional manner (i.e., with no assistance). For the synthetic SADs, we employed alternative optimization techniques inside *SADHelper*. Specifically, we worked with five types of SADs, namely, SADs optimized with Greedy, SADs optimized with NSGA-II, SADs optimized with Random Search, SADs optimized with SAT4J, and normal SADs. To compare the different SADs, the following metrics were used:

- *Cost*. Represents the units of effort necessary to produce a given SAD version (Equation (5)). We used the number of words (e.g., of a document) as a proxy to estimate units of effort.
- *Utility*. Represents the quality of a given SAD, and it is a function of the satisfaction and priority of the stakeholders (Equation (4)).
- *Satisfaction*. Represents the level of 'happiness' of a single stakeholder. It helps to explain potential differences in SAD quality (Equation (2)).

To carry out experiment 1, we asked a group of 77 test subjects to produce a Wiki-based SAD for a given system and release it in incremental versions. The V&B method was recommended as the guide to deliver the documentation. *SADHelper* was not used by the design teams. Afterwards, our goal was to perform a post-mortem analysis of the SADs, looking for potential improvements (cost, quality, satisfaction) that could have been made if the teams had worked with *SADHelper*. To this end, we kept records of the evolution of the SAD versions over time.

5.3.1. Experimental setup and procedure. The 77 subjects were arranged into 11 teams of seven members each. Each team simulated a small software project tasked with the production of architectural documentation. Each member played a distinctive stakeholder role: three members took the architect role, whereas the other four members were divided into one manager, one evaluator, and two clients. The stakeholder priorities were assigned as follows: $p_{manager} = 10$, $p_{architect} = 6$, $p_{client1} = 6$, $p_{evaluator} = 2$, $p_{client2} = 2$, in decreasing order of importance. The SCM[¶] for these roles was configured with baseline information from the V&B approach [2] (recall Figure 6).

The architects (of each team) were asked to (i) design an architecture for a model problem called the Clemson Transit Assistance System[‖] (CTAS) and (ii) use V&B to produce a SAD that should (progressively) satisfy all the stakeholders of the team. A separate Wiki (DokuWiki[**]) was set for the SADs of each team. We predefined a 9-week schedule for the teams, in which both the design and documentation work had to be performed in three iterations of 3 weeks, with two (partial) SAD versions for the first two iterations and a final release at the end (with 3–6 work hours per day). The managers, evaluators, and clients periodically assessed the architecture and its documentation and gave feedback to the architects regarding their interests. We refer to these documentation versions as *normal* or *traditional SADs*, because their production strategy was only based on the architects' criteria (and not steered by optimization techniques).

Once all the teams were finished, we obtained three normal SAD versions (or slices) per team. These slices came with different completion percentages: 10%–25% after the first iteration (slice 1), 40%–60% after the second iteration (slice 2), and 75%–85% after the third iteration (final SAD). We did not considered the final SADs 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 ≈200 words). The same metric was also used for the cost of producing a SAD document.

The costs and utilities of the normal SAD slices were established as baselines for assessing their counterparts with *SADHelper*. Next, we generated a set of synthetic SADs with *SADHelper*. We refer to those SAD versions as *optimized SADs*. Basically, we executed the optimization tool on the SAD corresponding to the slice 1 of every team. For these optimizations, we also set a cost threshold equal to the units of effort necessary to reach the completion percentage of slice 2. The same procedure was repeated for the transition between slice 2 and the final SAD, for every team. The transition from the initial SAD (slice 0) to slice 1 was not considered in our study, because it

[¶]The stakeholders' concerns matrix, as well as other resources used in the evaluation procedure, can be found at http://mnicoletti.sites.exa.unicen.edu.ar/jsep2014

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

[**]https://www.dokuwiki.org/dokuwiki

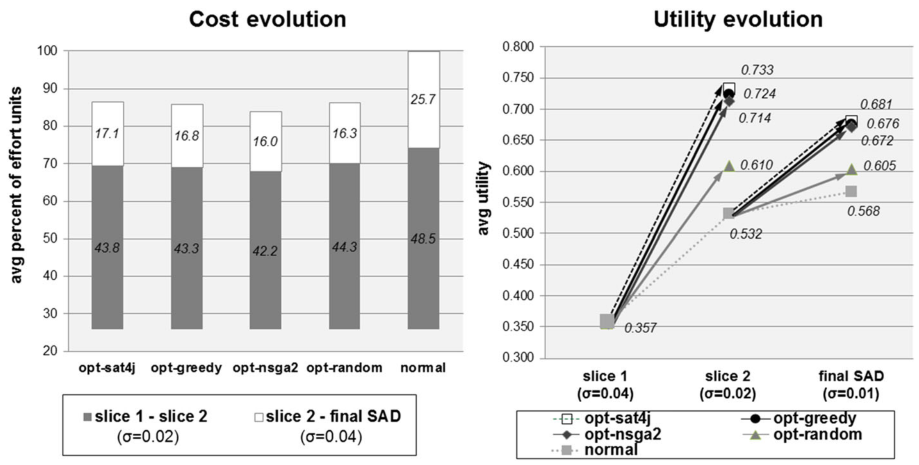


Figure 10. Optimized versus normal software achitecture documents (SADs) – variations in cost and overall utility (experiment 1).

makes little sense from a practical standpoint. In the first SAD version, the architect normally sketches the architecture solution (e.g., main architectural patterns, design tactics for quality attributes), and therefore, such a version is for internal usage rather than for the consumption of the whole set of stakeholders. For this reason, *SADHelper* always departs from an initial SAD and seeks for stakeholder-oriented improvements from there on. The completion levels of the initial SADs in our experiments ranged from 10%–15%.

In the aforementioned computations, parameter ε of Equation (2) was set to $\varepsilon = 0.1$, which is a 10% of 'allowed difference' between the document state and the level of interest. As regards the optimization parameters for NSGA-II and Random Search,[††] we used $initialpopulation : 100$, $maxiterations : 30000$ (for both), $selection : binarytournament$, $p_{crossover} : 0.9$, and $p_{mutation} : 0.001$ (only NSGA-II). Because these techniques are non-deterministic, the utility values reported are the average of 30 independent executions [39], with a global average standard deviation of 0.01 for NSGA-II and 0.02 for Random Search.

5.3.2. Experimental results. The charts of Figure 10 show the evolution of the average values (for the 11 teams) of cost and utility for each kind of SADs, namely, opt-greedy SADs (optimized with Greedy), opt-nsga2 SADs (optimized with NSGA-II), opt-random SADs (optimized with Random Search), opt-sat4j SADs (optimized with SAT4J solver), and normal SADs. The values of these charts were computed by following the procedure in Section 5.3.1. The Utility Evolution chart (right side) should be interpreted vis-a-vis with the Cost Evolution chart (left side), because the former depicts the utility values achieved with optimization techniques at each transition, while the latter shows the required cost for the same transitions (subject to a cost threshold equal to the cost of the normal SAD transitions).

The Cost Evolution chart plots the average percentages of units of effort for different slices. For instance, in the final SAD, opt-greedy SADs required $\approx 17\,\%$ of the total units of effort, whereas the normal counterparts required $\approx 26\,\%$. We noticed that, in general, the optimized SADs involved lower costs for generating each version, with a cost reduction of $10\,\% - 35\,\%$ (per slice). Because of how we estimated costs, a reduction in costs means that the optimization produced smaller SADs. This would suggest that the normal SADs tended to have 'unnecessary' contents, given the current stakeholders' needs. In this sense, we believe that cost reductions are more significant in the second transition because the detection of unnecessary contents gets easier when a SAD has an advanced level of completion.

An interesting observation is that optimized SADs achieved significantly higher levels of utility when compared with the normal counterparts. As it can be seen in the Utility Evolution chart, we observed a utility increment of $17\,\% - 28\,\%$ for opt-sat4j SADs, $16\,\% - 27\,\%$ for opt-greedy SADs,

[††]We relied on the implementations of NSGA-II and Random Search from the MOEA Framework (http://www.moeaframework.org/).

Table I. Statistical analysis for software architecture document (SAD) utility hypotheses (first transition).

Sample 1	Sample 2	p-value	Accepted H	Interpretation
opt-sat4j	opt-greedy	0.49	H_0	*opt-sat4j SADs = opt-greedy SADs*
opt-sat4j	opt-nsga	0.18	H_0	*opt-sat4j SADs = opt-nsga SADs*
opt-sat4j	opt-random	≈ 0	H_1	*opt-sat4j SADs > opt-random SADs*
opt-sat4j	normal	≈ 0	H_1	*opt-sat4j SADs > normal SADs*
opt-greedy	opt-nsga	0.49	H_0	*opt-greedy SADs = opt-nsga SADs*
opt-greedy	opt-random	≈ 0	H_1	*opt-greedy SADs > opt-random SADs*
opt-greedy	normal	≈ 0	H_1	*opt-greedy SADs > normal SADs*
opt-nsga	opt-random	≈ 0	H_1	*opt-nsga SADs > opt-random SADs*
opt-nsga	normal	≈ 0	H_1	*opt-nsga SADs > normal SADs*
opt-random	normal	0.001	H_1	*opt-random SADs > normal SADs*

Table II. Statistical analysis for software architecture document (SAD) utility hypotheses (second transition).

Sample 1	Sample 2	p-value	Accepted H	Interpretation
opt-sat4j	opt-greedy	0.59	H_0	*opt-sat4j SADs = opt-greedy SADs*
opt-sat4j	opt-nsga	0.16	H_0	*opt-sat4j SADs = opt-nsga SADs*
opt-sat4j	opt-random	≈ 0	H_1	*opt-sat4j SADs > opt-random SADs*
opt-sat4j	normal	≈ 0	H_1	*opt-sat4j SADs > normal SADs*
opt-greedy	opt-nsga	0.316	H_0	*opt-greedy SADs = opt-nsga SADs*
opt-greedy	opt-random	0.004	H_1	*opt-greedy SADs > opt-random SADs*
opt-greedy	normal	≈ 0	H_1	*opt-greedy SADs > normal SADs*
opt-nsga	opt-random	0.02	H_1	*opt-nsga SADs > opt-random SADs*
opt-nsga	normal	≈ 0	H_1	*opt-nsga SADs > normal SADs*
opt-random	normal	0.09	H_0	*opt-random SADs = normal SADs*

16 % − 25 % for opt-nsga2 SADs, and 6 % − 13 % for opt-random SADs. In particular, we noticed that higher utility values were reached in the first transition, in contrast with the second one. A possible explanation for this decrease is that the optimization of SADs with high levels of completion is not considering good-quality solutions that might involve 'undo' actions (with respect to the current SAD), as this kind of actions is not modeled in our current framework.

The results of the utility assessment indicate that normal SADs can always be improved by using optimization techniques. In particular, non-exact techniques performed reasonably well when compared with the exact technique (SAT4J). We did not perceived substantial differences in the performance (in terms of SAD utility/cost) of Greedy and NSGA-II. On the contrary, Random Search had an inferior performance, as it sometimes generated invalid solutions.

To corroborate our observations, we applied a statistical analysis based on the two sample Student's *t*-test. This parametric test serves to compare the means of two normal distributions (normality was verified with Shapiro–Wilk test and homoscedasticity with F-test, both with a confidence level of 95%). In our case, a sample is a collection of 11 utility values for a particular type of SADs, and we tested five different samples against each other.

We stated both the null hypothesis H_0: *the mean of utility values for a given distribution is equal to the mean of utility values for other distribution*, and the alternative hypothesis H_1: *the mean of utility values for a given distribution is higher or lesser than the mean of utility values for other distribution*. We then tested this hypothesis for each pair of samples with a significance level of 0.05. The results of this analysis are summarized in Tables I and II. For the transition from slice 1 to 2, we observe that all the optimized SADs were significantly better that the normal SAD, given that we reject H_0 with a critical p-value lower than 0.05. The same situation is observed in the transition from slice 2 to final SAD, except for opt-random SADs, which do not presented significant differences with the utility values of normal SADs ($p > 0.05$). Thus, the use of Random Search does not always ensure a higher utility. It is also important to note that there is not a significant difference between the performance of SAT4J, Greedy, and NSGA-II.

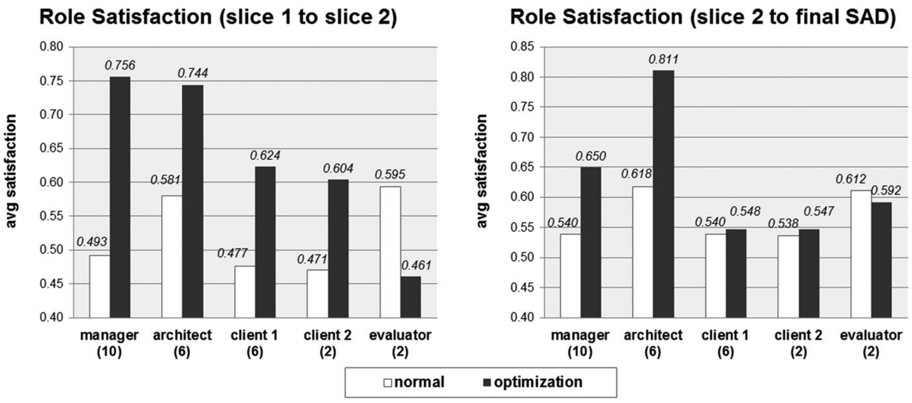


Figure 11. Opt-greedy versus normal software achitecture documents (SADs) – satisfaction of individual stakeholders.

As regards execution times,[‡‡] we obtained the following average times (in milliseconds): 110 ± 17 for opt-greedy, 462 ± 15 for opt-random, 693 ± 83 for opt-sat4j, and 1187 ± 60 for opt-nsga2. The main factor affecting the problem size was the number of documents of the SADs. The SAD sizes ranged from 13–18 documents. We noted that for SADs with a number of documents higher than 15, the time complexity grew exponentially when using basic exact techniques like backtracking. The computational complexity of SAT4J, in turn, was reasonably low for that size range, and began to grow for instances with more than 25 documents. We also noted that execution times were higher when the SADs had low levels of completion. It seems that the optimization problem gets 'easier' as the completion percentage of the SAD grows, given that the solution space becomes smaller.

The results earlier contribute to answering our two research questions. First, as regards RQ1, results show that it is possible to produce smaller (less costly) versions of SADs that not only maintain their utility but also could reach higher utility. We also observed that the level of completion of a SAD conditions the utility improvements attainable in successive versions. In general, initial SAD versions (~25% of completion) have the potential of producing higher utility, because the optimization algorithm has more 'degrees of freedom' during the search. This is partially due to our model of documentation tasks, which make it difficult to find improvements in more complete SADs. For example, a key stakeholder might want an *overview* of a section that is already *detailed*, but the section contents cannot be 'downsized' with our current tasks. As regards RQ2, we determined that SAT4J, Greedy, and NSGA-II have a similar performance, and Random Search performed worse (as it was expected). Our proposed Greedy algorithm solved the problem instances in low computing times with a small loss in solution quality.

To understand the advantages of optimized SADs over normal ones, we decomposed the global SAD utility into the individual stakeholders' satisfaction values, as it is shown in Figure 11 (a description of these metrics can be found in Section 5.3). Recall that this sort of information can be visualized in *SADHelper* through the stakeholders' satisfaction view (Figure 4). In both transitions, we noticed that all the stakeholders were evenly satisfied by the normal SADs. On the contrary, the opt-greedy SADs clearly favored high-priority stakeholders. Similar trends were observed for other optimization techniques, namely, NSGA-II and SAT4J. For instance, in the first transition (left chart), we have a bias toward managers and less care about evaluators, while the second transition (right chart) is mainly focused on managers and architects.

Although the update plans suggested by the tool were guided by the high-priority stakeholders, the plans could have tasks that still targeted less important stakeholders. This is the case of client 2 in the first transition, who improved her satisfaction with the second SAD version even when she was a low-priority stakeholder. In the second transition (right chart), the satisfaction improvements were minor and only noticeable for the main stakeholders (manager and architect). In this particular case, managers, who have the highest priority, seemed to receive less care than architects, who still have a

[‡‡]All experiments were executed on a PC (Windows OS) with an Intel Core i3-2310 processor and 8 GB RAM memory

high priority. We believe this effect was due to the precedent completion states of SAD documents of slice 2, which do not let managers be more satisfied beyond a certain threshold.

5.4. Experiment 2: users' feedback

In contrast to experiment 1, this experiment aimed at testing *SADHelper* from a practical perspective, that is, working with SAD users. To do so, a group of 10 graduate students acted as SAD readers in a simulated architectural documentation scenario, steered by *SADHelper* assistance, and we then requested the subjects to provide feedback about their actual satisfaction with the SADs. Experiment 2 also contributes to answering RQ1. The same metrics of experiment 1 were collected (cost, utility, satisfaction). Additionally, we introduced a new metric called *support*, which measures the degree of coverage of stakeholders' information needs by a given SAD based on the stakeholders' feedback. We hypothesized that support should have a direct correlation with utility.

5.4.1. Experimental setup and procedure. The materials[§§] used in this experiment were as follows: (i) a SAD based on V&B that described an architecture for CTAS (the same model problem of experiment 1) and (ii) questionnaires for evaluating the SAD. The SAD for CTAS contained 24 pages (sections) and 22 architectural diagrams, which is representative of real-life SADs. Specifically, there was one Wiki page per architectural view, and one Wiki page per additional section (i.e., documentation beyond views) of the V&B template.[¶¶]

The SAD contents were of acceptable quality, but still had some inconsistencies, omissions, and opportunities for improvement. The questionnaires were sets of questions about the CTAS architecture of CTAS, which were designed to discover these problems from the perspective of different types of stakeholders. A questionnaire included (i) a set of quality-attribute scenarios to be evaluated with an ATAM-like (acronym for Architecture Tradeoff Analysis Method) design review [1] and (ii) a set of stakeholder-specific questions (also ATAM-like) referring to the quality of the architectural descriptions [40]. That is, we provided a questionnaire per stakeholder role that attended her interests on the SAD contents, as derived from the V&B matrix. The idea is that the stakeholders playing a given role can be guided by the corresponding questionnaire (e.g., as a check-list mechanism) in their assessment of the SAD. This decision allowed us to know a priori the relevant contents for each subject. In fact, we constructed the SCM (used later for the optimization phase) by analyzing these questionnaires.

The SAD for the CTAS architecture was divided into three versions (namely, slice 1, slice 2, and final SAD), using a similar criterion as in experiment 1 (i.e., completion percentages per slice) and by mutual agreement among the authors. These versions were referred to as normal SADs. In parallel, we generated an optimized SAD version from slice 1, and after that, a new optimized SAD based on the previous version. Note that this setting of optimized SADs differs from the one used in experiment 1. *SADHelper* was here configured with our Greedy algorithm. The 10 subjects worked individually: four subjects on normal SAD versions (control group) and six subjects optimized SAD versions. The stakeholder priorities were defined as follows: $p_{manager} = 10$, $p_{evaluator} = 8$, $p_{client} = 6$, $p_{architect} = 4$.

The documenters were ourselves, as we released the consecutive (and incremental) SADs to the subjects for assessment. We asked each subject to perform an ATAM-based evaluation of these SADs versions. The activity consisted of applying the questionnaire (to every version) and elaborating a final evaluation report. This report included a feedback section, in which subjects informed us about the support of every item of the questionnaire. The support for a given item could take three values: *unsupported*, *probably-supported*, and *supported*. Thus, support can be seen as a measure of the actual stakeholders' satisfaction levels.

5.4.2. Experimental results. The results of cost and utility for the six SAD versions are shown in Figure 12. These charts suggest that optimized SAD required a lower amount of effort than the normal SAD for producing a satisfying documentation. In particular, the reduction in cost was in the order of 33%. The optimized SADs also reached higher (theoretical) utility values. In fact, for the final SAD,

[§§]These materials can be found at http://mnicoletti.sites.exa.unicen.edu.ar/jsep2014
[¶¶]See examples of V&B templates at http://wiki.sei.cmu.edu/sad

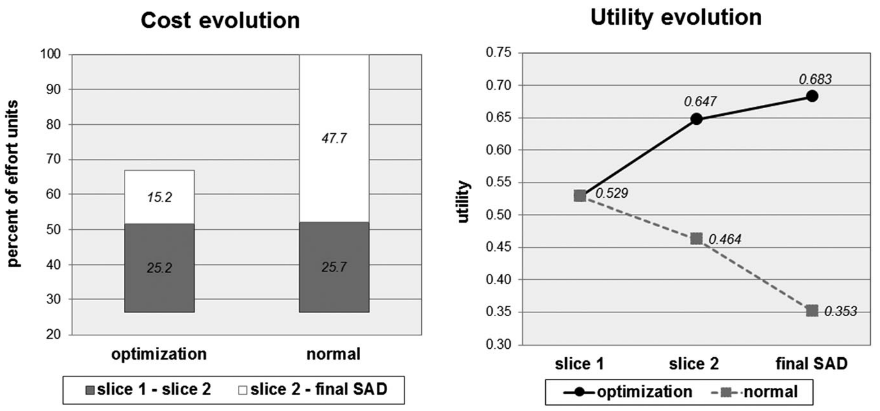


Figure 12. Optimized versus normal software achitecture documents (SADs) – variations in cost and utility (experiment 2).

the utility improvement was ≈ 93 %. We must note that the utility improvements in experiment 1 were generally lower than those of experiment 2. We believe this is due to the fact that in experiment 2 the SAD is fully generated using the *SADHelper* assistance, whereas in experiment 1 the SAD versions are optimized from pre-determined levels of completion that were generated without assistance.

From the results of experiments 1 and 2, we observe similar trends that demonstrate the superiority of optimized SADs. Thus, both experiments suggest that, with *SADHelper*, it is possible to produce a satisfying final SAD version by using only a fraction of the total units of effort (with a maximum of 23 for experiment 2). As the size of documentation grows, the utility of the SAD is diminished due to either unnecessary contents or conflicting stakeholders' interests. This effect was evident in the transition from slice 2 to the final SAD, for both normal and optimized SADs, although the effect was attenuated in the latter.

We also measured the effective improvements in terms of support. In particular, we computed how many items (of each questionnaire) a subject was able to respond with the documentation available in the SAD (slice) under consideration. Each item was weighted according to subject' feedback with the following rule: 0 for *unsupported*, 0.5 for *probably-supported*, and 1 for *supported*. The support for a given user is the average of the support values for every questionnaire item. Then, the support for a SAD slice x is computed as indicated in Equation (7), where m: number of questionnaire items, n: number of stakeholders, and $support_{i,j}$: is the support value for stakeholder i and item j.

$$avg\_support_{slicex} = \sum_{i=1}^{n} \left(priority_j . \sum_{j=1}^{m} support_{i,j} \right) \tag{7}$$

After performing the support assessment of all the SAD versions, we obtained the values described in Table III. These results indicate that the subjects working with the optimized SADs were able to solve, on average, just a few more items than the subjects working with the normal SADs. Although we observed that the average support values of optimized SADs are slightly higher than the support values of normal SADs, the differences are not (statistically) significant. It seems that improvements in the utility of SAD versions were not always perceived by test subject.

Table III. Average support values for different software architecture document (SAD) versions (experiment 2).

Type of SAD	Slice 1	Slice 2	Final SAD
Normal	0.22	0.38	0.59
Optimized	0.20	0.44	0.65

5.5. *General discussion*

Overall, the outcomes of the experiments were encouraging in the sense that corroborated that *SADHelper* can help the documenter to filter out 'unnecessary information' of architectural documentation. Examples of such unnecessary information include the following: (i) SAD sections (architectural views) that are of no interest to any key stakeholder or (ii) sections that, if present, might even reduce the stakeholders' satisfaction because of information overload. Information overload is a problem affecting a person who tries to deal with more information than she is capable of processing, in order to make decisions [41]. For instance, in the final SAD of experiment 2, our tool suggested that a given Module view should have an *overview* level of detail, since this view was only useful to low-priority stakeholders (i.e., client and architect). In this context, using optimization techniques to steer the SAD generation process seems to be a more cost-effective approach than documenting a SAD without assistance. Although the results of our study are still preliminary, they showed that savings in production costs can reach 33%, depending on the initial completion percentage of the SAD.

An important observation is that the optimized SADs not only required less effort to be produced but also had a utility level equal or higher than normal SADs. By analyzing individual values of satisfaction, we found two main factors that can explain this situation. First, the optimization algorithms took advantage of stakeholders' priorities to achieve high utility. In contrast, the normal SADs tended to satisfy all stakeholders alike or randomly. Second, the trade-offs between stakeholders' interests and levels of detail (of SAD sections) were not handled properly in the normal SADs. A balance among those trade-offs was generally sought in the optimized SADs, as modeled by Equation (2). These observations also shed light on the high complexity that producing satisfying documentation might entail when performed manually and in a multi-stakeholder context. This was one of the drivers behind the development of *SADHelper*.

Even though experiment 1 provided evidence that *SADHelper* could lead to SADs with higher utility than normal ones, we were not able to verify this trend in experiment 2. From the results of experiment 2, we observed that optimized SADs required less effort and gave at least the same utility than the normal counterparts, but they did not always give higher utility values. We noticed that there were some differences when measuring theoretical utility improvements versus practical improvements, as judged by the SAD users. The theoretical utility improvements were greater than those actually recorded in terms of support. The subjects reported that they were slightly more satisfied with the optimized SADs than with the normal SADs, although this perception could not be statistically checked on the data of experiment 2. Possible factors can explain the gap between (theoretical) utility and support. First, the CTAS documentation (although close to real SADs) could not have the volume or complexity necessary to produce dissatisfaction in the groups of stakeholders (e.g., problems of information overload). Second, the support measure is sensitive to the subjectivity of the participants' responses.

5.6. *Threats to validity*

We carefully designed the experiments in order to recreate real software development scenarios. However, there are still some threats to validity that should be considered [42].

5.6.1. Construct validity. There are some important considerations regarding construct validity. First, test subjects were not aware of the experimental objectives. Therefore, we minimized the chance that they had behaved in any particular way that might have biased the results. Second, even though results are encouraging as regards the performance of *SADHelper*, we must note that we used proxies to measure the actual utility and cost (effort) values. Future experiments must explore better proxies for these metrics (e.g., alternative satisfaction functions, amount of man-hours, level of difficulty of a given documentation activity, among others).

Related to the threats earlier, we should note that we expressed effort reductions to produce SAD contents in terms of units of effort, which were measured with a proxy based on the number of words per document. Nonetheless, the actual performance of a documenter when writing a given SAD section might vary from one case to another, regardless of whether she relies on the assistance of *SADHelper*. Performance variations can depend on different factors, such as the following: the documenters expertise, the complexity of the architecture being documented, whether the documenter already wrote other SAD sections, or the domain itself, among others [43].

5.6.2. Internal validity. Several artifacts used to carry out experiments were not part of the actual working context (e.g., artifacts, project tasks) of the subjects, so the values reported for support might be affected by this condition. The artifacts involved in this threat are as follows: (i) the model problem (CTAS), (ii) the corresponding SAD, and (ii) the questionnaires. Another threat is the possible bias caused by some subjects having prior knowledge of the software architectures (documented by the SADs) or having performed this kind of practical assignment before. We were careful when selecting the subjects to check that they had neither participated in similar activities nor worked with the SADs before.

We realize that the performance of *SADHelper* is affected by the adoption of a particular optimization technique, and particularly, if non-exact techniques such our Greedy approach are employed. To alleviate this threat, we performed a comparative analysis among commonly used techniques to tackle similar problems to ours. Our experimental results showed that optimal solutions from an exact optimization technique might not be required to improve the documentation process.

An additional threat could arise due to role assignment. Although the test subjects did not always play similar roles to those of their jobs, they had prior knowledge about the role descriptions and behavior based on either their academic or industrial experience. Also, having prior experience in a role does not necessarily bias the experiments, because roles were mainly used to assign an interest profile and a priority to each stakeholder. In the case of experiment 2, the questionnaires served as a guide to help the subjects in playing the assigned roles.

At last, some assumptions made in our model of documentations tasks and also in the optimization formulation certainly influenced our experimental results. Examples of such assumptions are as follows: the mapping between completion states and required sub-sections of a SAD document, the form of the satisfaction function, or the use of words to measure the cost of documentation tasks. These assumptions were either based on the authors' experience or taken from the literature, but they must be further validated.

5.6.3. External validity. Results of our study cannot be generalized to an industrial scenario, as we used students in an academic environment, which might differ from an industrial context with seasoned software practitioners. To mitigate this threat, we designed our experimental environment to be as realistic as possible. For instance, we ensure that the model problem and the architecture documentation were representative of real-life problems. Additionally, most students were actually software practitioners working for local industries with 1–3 years of experience.

Another decision to mitigate this threat was the fact that experiments were not designed as a 'quiet room'.[‖‖] If experiments had been based on a quiet room, our simulated development scenarios would have significantly differed from real scenarios, in which stakeholders perform several concurrent tasks and interact with each other. In our testing environment, the subjects were free to solve the assignments at the university or at their homes, even working in groups. We believe this latter scenario is closer to a real development environment.

5.6.4. Conclusion validity. In contrast with external validity, the use of graduate and undergraduate students instead of industry practitioners allowed us to obtain a good sample (a total of 87 subjects) to support the validity of the results. Indeed, it would have been harder to gather a similar number of industry professionals with the time availability required by the experiments. As a downside, the level of heterogeneity of the test groups was low, because most subjects shared similar levels of knowledge and technical background. This situation reduced threats to conclusion validity, but it trades off with external validity.

An additional aspect to be considered regarding conclusion validity is the reliability of measures. Measures that involve human judgment, such as support, are affected by this threat. This means there is a chance that experimental results might suffer variations if a different set of subjects is used to replicate the experiments.

[‖‖]A quiet room is a term referring to a place used for experimentation with human subjects in which there are no distractions that may bias the results of tests.

6. RELATED WORK

The related work is organized into two sub-sections, as follows: Section 6.1 describes works related to software architecture documentation, and Section 6.2 summarizes related work on search-based software engineering (SBSE) techniques applied to software architecture.

6.1. Software architecture documentation

In the last decades, there has been an increasing interest in approaches for capturing and communicating architectural knowledge among the stakeholders of a project. For instance, Farenhorst *et al.* [15] analyzed the requirements of a tool to capture and share architectural knowledge among architects. Based on these requirements, the authors created a Web portal called JIT AK Portal that helps architects to find contents relevant to their interests. However, no other types of stakeholders are considered, thus this approach fails to provide reader-oriented documentation in a broader sense. In JIT AK Portal, the documenter is responsible for providing a Web interface with structure and contents that match the architect's interests, and no guidelines are provided for this task. In contrast, our approach is focused on reducing the workload of the documenter and ensuring coverage of most stakeholders' concerns.

Another related tool is Knowledge Architect [44] developed by Jansen *et al.* The goals of this approach are to facilitate the access to a base of architectural knowledge and the search for specific issues in this knowledge base. In Knowledge Architect, documents are enriched with annotations from a formal model to support the retrieval of knowledge for multiple types of stakeholders. However, this approach is mainly focused on the readers' side, and requires an extra effort from documentation writers, which constitutes one of the limitation acknowledged by the authors (known as asymmetric benefits).

A relevant study about Wikis is that of Graaf *et al.* [17], in which semantic Wikis are discussed. In this research, the SAD is based on both a Wiki that supports semantic annotations and an ontology describing Software Architecture concepts. The authors argue that ontology-based SADs are more effective than traditional file-based approaches. Effective means here that a stakeholder is able to find more/better relevant information according to his/her interests. In a controlled experiment with a small group of software professionals, the authors show evidence that supports their hypothesis. Although the objectives of this research are different to ours, we consider this ontology-based approach as a form of reader-oriented documentation supported by Wikis. Similarly to Graaf *et al.*, other authors have reported experiences with Wikis applied to architecting tasks [14, 15]. Bachmann and Merson [14] discussed the role of a Wiki to record technical documentation in a collaborative setting, and specifically, how to use a Wiki with V&B documentation. In fact, the approaches of Farenhost *et al.* and Jansen *et al.* (described earlier) are also based on Web-based tools. These studies motivated the use of Wikis to host architectural documentation in our approach.

In addition to the different approaches for architectural knowledge sharing, several documentations methods have been proposed during the last decades [2, 23–25]. In general, these methods provide documentation templates to organize the contents of a SAD into a collection of architectural views (e.g., module views, allocation views) and complementary documentation. However, the methods do not often focus on the efficient execution of the documentation process, especially when the documentation corpus to be generated is large. Although the V&B method (Section 3) offers a set of basic steps and rules to organize and develop the SAD, they are still generic and do not always ensure a cost-effective documentation process.

Along this line, the work of Falessi *et al.* [45] is interesting, because it proposes a value-based approach to generate cost-effective documents in the context of Design Rationale Documentation (DRD). DRD is actually an important part of a SAD. The idea is to document just those information items (or documentation units) that are likely to be required depending on the category of the item (e.g., constraint, decision, issue, assumption) and the activity it aims to support (e.g., assess impact of design decisions, check inconsistencies, detect misunderstanding of requirements). The approach was validated through two controlled experiments involving post-graduate Software Engineering students. The results of this research indicate that it is possible to generate a customized DRD with a reduced amount of sections (and therefore, less effort) of the full documentation. These findings are aligned with our experimental results, and we believe that the application of a value-based approach in combination with optimization techniques can be an interesting extension to *SADHelper*.

6.2. SBSE techniques in software architecture

Optimization techniques have been widely used in several fields of Software Engineering, including Requirements Engineering, Software Design and Development, Software Maintenance and Evolution, or Project Planning, among others [34, 46, 47]. In the context of Requirements Engineering, Bagnall *et al.* [20] stated the NRP, which has several similarities with our SAD optimization problem. In the basic NRP formulation, a given software company has to determine a subset of requirements that should be included in the next release of a given product, in such a way that the overall satisfaction of the company customers gets maximized. Every customer is interested in a subset of features. Every requirement has an associated cost, and the total cost of the scheduled features must not exceed the available resources in each release iteration.

The NRP was later extended to a multi-objective formulation, known as multi-objective NRP (MONRP) [21, 48]. Instead of treating cost as a constraint, MONRP cost as a minimization objective. Therefore, MONRP admits several solutions, that is, a Pareto front. Experiments with synthetic data [21, 49] have shown that metaheuristics such us NSGA-II exhibited a good performance in finding a large part of the Pareto front, even for large problem instances. This is the reason why we considered this sort of techniques for *SADHelper*.

Although the increasing amount of SBSE work over the last years [34], the study of production strategies for software documentation, and in particular architectural documentation, has not yet been targeted by the SBSE community. However, SBSE techniques have been widely applied to other activities related with architectural design of software systems [50]. As software systems become more complex, architects must deal with a growing number of candidate design alternatives and decisions. SBSE can offer several advantages in this regard, because it provides tools to help architects in the exploration of the design space with reasonable time and effort. SBSE works in software architecture design have been mainly focused on Component-based Software Engineering (CBSE) and Software Product Lines (SPLs).

As regards CBSE, an interesting problem to apply optimization techniques is the substitution of software components during the system maintenance stage. When one or more components become obsolete, they must be replaced with one or more new components (e.g., COTS, Commercial-Off-The-Shelf) that have similar characteristics and functionality. Desnos *et al.* [46] have studied this combinatorial problem and evaluated the performance of several optimization techniques using synthetic data. Another CBSE problem that is suitable for SBSE techniques is the selection of components for the next version of a software system, which is a similar problem to NRP. In the component selection problem, a subset of components must be selected for implementation in order to maximize the benefit (e.g., client priority, impact on the system) without violating a cost constraint. Baker *et al.* [51] proposed the use of non-exact techniques such as greedy or simulated annealing to find near-optimal solutions to this problem, which was cast as a 0–1 knapsack problem.

More recently, the application of SBSE techniques to solve SPL problems has received some attention [52]. For instance, Colanzi *et al.* [53] used optimization techniques to assist the design of an SPL architecture. The objective was to find an SPL that maximizes several quality characteristics, such as extensibility, component cohesion, and level of modularity, among others. The problem was cast as a multi-objective optimization and solved with evolutionary algorithms such as NSGA-II.

7. CONCLUSIONS AND FUTURE WORK

In this work, we presented *SADHelper* as a supporting tool for the generation of SADs, which emphasizes a low-cost process and a high-quality product (i.e., the resulting SAD versions). The approach is based on the V&B approach. Stakeholders' interests are captured in the form of matrix that links those interests to different SAD sections and suggests specific levels of detail. In background, our tool relies on optimization algorithms to compute near-optimal update plans that are suggested to the documenter. In addition to the update plans, the documenter is assisted by *SADHelper* by means of several other features (e.g., visualization of current satisfaction of stakeholders, documentation actions to be performed on a Wiki-based SAD, among others).

Our approach was validated with two experiments that targeted the optimization aspect of *SADHelper*. These experiments provided evidence to support our hypothesis about effort reductions in the production of architectural documentation without reducing its utility for the stakeholders. The final goal of *SADHelper* is to guide a documenter to prefer smaller SADs that target key stakeholders, over extensive architectural documentation. The experiments also allowed us to determine suitable techniques to solve our optimization problem. In this sense, both the proposed Greedy algorithm or the NSGA-II algorithm shown promising performance results.

The development of *SADHelper*, although still a proof-of-concept, has several important implications for architecture documentation practices. It shows the feasibility of generating useful but just enough architectural documentation in software projects with tight schedules (even if such documentation is produced manually). It brings to the front information overload issues experienced by documentation readers, and furthermore, it shows strategies to mitigate those issues. In particular, we proposed a combination of optimization techniques and stakeholders' concerns, but other strategies could apply as well.

Nonetheless, our approach has still a number of drawbacks, most of them related to our optimization framework. For instance, our model does not consider non-additive actions that a documenter might perform on a SAD (e.g., updating a section due to system refactoring, or deleting a section). The use of four discrete document states, initially derived from the V&B matrix, might be limited for some document writing processes. We need to further explore extensions of our basic framework with other possible documentation actions, more document states, dependencies among sections, or fuzzy rules for the evolution of details and sections. The measures used for the objectives of cost and utility are still crude approximations, and better estimation proxies should be applied in real scenarios. For instance, the satisfaction function should be calibrated on the basis of stakeholders' activity on different SADs [54]. Another limitation is that SAD quality is currently assessed from an 'external' perspective (the satisfaction of the stakeholders); however, it is important to consider its 'internal' quality as well, particularly during the first SAD versions. For instance, internal quality is related to aspects of (i) consistency among the different sections of the SAD, (ii) traceability between key requirements and design decisions, or (iii) suitable architectural views to support those decisions.

As future work, we plan to

- Replicate experiment 2 with a different group of test subjects and on different SADs, so as to obtain more data and check our initial findings. In particular, we believe a larger study with our optimization framework can help us to investigate the correlation between SAD quality (e.g., utility) and stakeholders' satisfaction (e.g., support).
- Evaluate and compare the performance of NSGA-II with other state-of-the-art multi-objective algorithms to solve our optimization problem.
- Enhance the current prototype of *SADHelper* and integrate it with development tools (IDEs, CASE tools, or planning tools), so as to foster the adoption of our approach in real software projects.
- Study measures for quantifying the internal quality of a SAD, and possibly incorporate it as a third objective in our optimization formulation.
- Apply sensitivity analysis techniques to make our optimization formulation more robust (e.g., to variations in costs, stakeholders' priorities, or satisfaction functions, among others).
- Customize the SCM on the basis of individual stakeholder interests via user profiling techniques [55].

ACKNOWLEDGEMENTS

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. The authors would like to thank the participants of the experiments for their collaboration with this study.

REFERENCES

1. Bass L, Clements P, Kazman R. *Software Architecture in Practice*, 3rd edn. Addison-Wesley Professional: Boston, USA, 2012.
2. 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: Boston, USA, 2010.
3. Parnas DL. Precise documentation: The key to better software. *The Future of Software Engineering*, Nanz S (ed.). Springer: Berlin, Germany, 2010; 125–148.
4. Unphon H, Dittrich Y. Software architecture awareness in long-term software product evolution. *Journal of Systems and Software* 2010; **83**(11):2211–2226.
5. Cuesta CE, Navarro E, Perry DE, Roda C. Evolution styles: using architectural knowledge as an evolution driver. *Journal of Software: Evolution and Process* 2013; **25**(9):957–980.
6. Ruping A. *Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects*. John Wiley & Sons, 2003.
7. IEEE. IEEE Std 1471–2000: recommended practice for architectural description of software-intensive systems, 2000.
8. ISO/IEC/IEEE. Iso/iec/ieee 42010: systems and software engineering – architecture description, 2011.
9. Clements P, Bachmann F, Bass L, Garlan D, Ivers J, Little R, Nord R, Stafford J. A practical method for documenting software architectures. In *ICSE*, 2003.
10. Martello S, Toth P. *Knapsack Problems: Algorithms and Computer Implementations*, Wiley-Interscience Series in Discrete Mathematics and Optimization. J. Wiley & Sons: New York, USA, 1990.
11. Pisinger D. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research* 1995; **83**:394–410.
12. Hadar I, Sherman S, Hadar E, Harrison J. Less is more: architecture documentation for agile development. In *Cooperative and Human Aspects of Software Engineering* (*CHASE*), *2013 6th International Workshop on*, May 2013; 121–124.
13. Rost D, Naab M, Lima C, von Flach Garcia Chavez C. Software architecture documentation for developers: a survey. In *Proceedings of 7th ECSA*, Springer-Verlag: Berlin, Heidelberg, 2013; 72–88.
14. Bachmann F, Merson P. Experience using the web-based tool wiki for architecture documentation. Technical Note CMU/SEI-2005-TN-041, SEI, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2005.
15. Farenhorst R, Izaks R, Lago P, Vliet Hv. A just-in-time architectural knowledge sharing portal. *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture* (*WICSA 2008*), WICSA'08. IEEE Computer Society, Washington, DC, USA, 2008; 125–134.
16. Clerc V, de Vries E, Lago P. Using wikis to support architectural knowledge management in global software development. In *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge*, SHARK'10, ACM: New York, NY, USA, 2010; 37–43.
17. de Graaf KA, Tang A, Liang P, van Vliet Hans van Vliet H. Ontology-based software architecture documentation. In *Proceedings Joint Working Conference on Software Architecture & 6th European Conference on Software Architecture* (*WICSA/ECSA*), WICSA 2012, IEEE Computer Society, 2012; 315–319.
18. Lago P, Avgeriou P, Hilliard R. Guest editors' introduction: software architecture: framing stakeholders' concerns. *IEEE Software* 2010; **27**:20–24.
19. Power K. Stakeholder theory and agile software development. In *Proceedings of the 11th International Conference on Product Focused Software*, PROFES'10, ACM: New York, NY, USA, 2010; 97–98.
20. Bagnall A, Rayward-Smith V, Whittley I. The next release problem. *Information and Software Technology* 2001; **43** (14):883–890.
21. Zhang Y, Harman M, Mansouri SA. The multi-objective next release problem. In *Proceedings of the 9th GECCO*, ACM: New York, NY, USA, 2007; 1129–1137.
22. Keuler T, Knodel J, Naab M, Rost D. 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*, Aug 2012; 234–238.
23. Hofmeister C, Nord R, Soni D. *Applied Software Architecture*, 1st edn. Addison-Wesley Professional: Boston, USA, 2000.
24. Kruchten P. The 4 + 1 view model of architecture. *IEEE Software* 1995; **12**(6):42–50.
25. Rozanski N, Woods E. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives* (2nd edn). Addison-Wesley, 2011.
26. Mitchell RK, Agle BR, Wood DJ. Toward a theory of stakeholder identification and salience: Defining the principle of who and what really counts. *Academy of Management Review* 1997; **22**:853.
27. Branke J, Deb K, Miettinen K, Slowinski R. *Multiobjective Optimization: Interactive and Evolutionary Approaches*, LNCS sublibrary: theoretical computer science and general issues. Springer: Berlin, Germany, 2008.
28. Diwekar U. *Introduction to Applied Optimization*, 2nd edn. Springer Publishing Company, Incorporated: New York, USA, 2010.
29. Martello S, Pisinger D, Toth P. New trends in exact algorithms for the 0–1 knapsack problem. *European Journal of Operational Research* 2000; **123**(2):325–332.
30. Diaz-Pace J, Nicoletti M, Schiaffino S, Vidal S. Producing just enough documentation: the next sad version problem. *Search-Based Software Engineering*, volume 8636 of *Lecture Notes in Computer Science*, Le Goues C, Yoo S (eds.). Springer International Publishing: New York, USA, 2014; 46–60.

31. Berre DL, Parrain A. The sat4j library, release 2.2. *The Journal of Substance Abuse Treatment* 2010; **7**(2–3):59–56.
32. Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: nsga-ii. *Trans. Evol. Comp* 2002; **6**(2):182–197.
33. Coello Coello C. Recent trends in evolutionary multiobjective optimization. *Evolutionary Multiobjective Optimization*, Advanced Information and Knowledge Processing, Abraham A, Jain L, Goldberg R (eds.). Springer: London, 2005; 7–32.
34. Harman M, Mansouri SA, Zhang Y. Search-based software engineering: trends, techniques and applications. *ACM Computing Surveys* 2012; **45**(1):11:1–11:61.
35. Papadimitriou CH, Steiglitz K. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications: New York, USA, 1998.
36. Wang A, Arisholm E, Jaccheri L. Educational approach to an experiment in a software architecture course. In *Software Engineering Education Training, 2007. CSEET'07. 20th Conference on*, 2007; 291–300.
37. Carver J, Jaccheri L, Morasca S, Shull F. A checklist for integrating student empirical studies with research and teaching goals. *Empirical Software Engineering* 2010; **15**(1):35–59.
38. Staron M. Using experiments in software engineering as an auxiliary tool for teaching–a qualitative evaluation from the perspective of students' learning process. *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, 2007; 673–676.
39. Arcuri A, Briand L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE'11, ACM: New York, NY, USA, 2011; 1–10.
40. Nord RL, Clements PC, Emery DE, Hilliard R. Reviewing architecture documents using question sets. In *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA)*, IEEE, 2009; 325–328.
41. Maes P. Agents that reduce work and information overload. *Communications of the ACM* 1994; **37**:30–40.
42. Wohlin C, Runeson P, Host M, Ohlsson M, Regnell B. *Experimentation in Software Engineering* (vol. **978-3-642-29043-5**), Springer: New York, USA, 2012.
43. SÃ¡nchez-Rosado I, RodrÃguez-Soria P, MartÃn-Herrera B, Cuadrado-Gallego J, MartÃnez-HerrÃ¡iz J, GonzÃ¡lez A. Assessing the documentation development effort in software projects. *Software Process and Product Measurement*, volume 5891 of *Lecture Notes in Computer Science*, Abran A, Braungarten R, Dumke R, Cuadrado-Gallego J, Brunekreef J (eds.). Springer Berlin Heidelberg: Berlin, Germany, 2009; 337–346.
44. Jansen A, Avgeriou P, van der Ven JS. Enriching software architecture documentation. *Journal of Systems and Software* 2009; **82**(8):1232–1248. SI: Architectural Decisions and Rationale.
45. Falessi D, Briand LC, Cantone G, Capilla R, Kruchten P. The value of design rationale information. *ACM Transactions on Software Engineering and Methodology* 2013; **22**(3):21.
46. Desnos N, Huchard M, Tremblay G, Urtado C, Vauttier S. Search-based many-to-one component substitution. *Journal of Software Maintenance and Evolution: Research and Practice* 2008; **20**(5):321–344.
47. Keeffe MO, CinnÃ©ide M. Search-based refactoring: an empirical study. *Journal of Software Maintenance and Evolution: Research and Practice* 2008; **20**(5):345–364.
48. Durillo JJ, Zhang Y, Alba E, Nebro AJ. A study of the multi-objective next release problem. In *Proceedings of the 1st SSBSE*, IEEE Computer Society: Washington, DC, USA, 2009; 49–58.
49. Durillo JJ, Zhang Y, Alba E, Harman M, Nebro AJ. A study of the bi-objective next release problem. *Empirical Softw. Eng.* 2011; **16**(1):29–60.
50. Aleti A, Buhnova B, Grunske L, Koziolek A, Meedeniya I. Software architecture optimization methods: a systematic literature review. *IEEE Transactions on Software Engineering* 2013; **39**(5):658–683.
51. Baker P, Harman M, Steinhofel K, Skaliotis A. Search based approaches to component selection and prioritization for the next release problem. In *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*, Sept 2006; 176–185.
52. Harman M, Jia Y, Krinke J, Langdon WB, Petke J, Zhang Y. Search based software engineering for software product line engineering: a survey and directions for future work. In *18th International Software Product Line Conference, SPLC'14, Florence, Italy, September 15–19, 2014*, 2014; 5–18.
53. Colanzi TE, Vergilio SR, Gimenes IMS, Oizumi WN. A search-based approach for software product line design. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC'14, ACM: New York, NY, USA, 2014; 237–241.
54. Chajewska U, Koller D, Parr R. Making rational decisions using adaptive utility elicitation. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00)*, 2000; 363–369.
55. Nicoletti M, Diaz-Pace J, Schiaffino S, Tommasel A, Godoy D. Personalized architectural documentation based on stakeholders' information needs. Journal of Software Engineering Research and Development 2014; **2**(1):1–26.