A Value-Based Approach for Documenting Design Decisions Rationale: A Replicated Experiment

Davide Falessi
Univ. of Roma "Tor Vergata", DISP
Viale del Politecnico 1, 00133
Rome, Italy
+39 0672597942
falessi@ing.uniroma2.it

Rafael Capilla
Department of Computer Science,
University Rey Juan Carlos
c/ Tulipan s/n, 28933, Madrid. Spain
+34 91 488 81 19
rafael.capilla@urjc.es

Giovanni Cantone
Univ. of Roma "Tor Vergata", DISP
Viale del Politecnico 1, 00133
Rome, Italy
+39 0672597392
cantone@uniroma2.it

ABSTRACT
The explicit documentation of the rationale of design decisions is a practice generally encouraged but rarely implemented in industry because of a variety of inhibitors. Known methods for Design Decisions Rationale Documentation (DDRD) are aimed to maximize the benefits for practitioners who should utilize the DDRD by imposing the burden on the developers of documenting all the potentially useful information. In our view, the adoption of a tailored DDRD, consisting only of the required set of information, would mitigate the effects of DDRD inhibitors. This paper focuses on confirming empirically the feasibility of a value-based approach for documenting the rationale behind design decisions, and the importance of different DDRD information categories. In this context, this work describes a replicated experiment carried out at the University Rey Juan Carlos of Madrid (Spain) aimed to validate previous results from an analogous study conducted at the University of Roma Tor Vergata (Italy). Results confirm that the level of utility related to the same category of DDRD information significantly changes depending on its purpose.

Categories and Subject Descriptors
D.2.11 [**Software Architectures**]

General Terms
Measurement, Documentation, Design, Experimentation.

Keywords
Design rationale, design decisions, software architecture, documentation, empirical software engineering.

1. INTROUDCTION
During last four years, the software architecture research community has been focusing its attention on the importance of documenting the architectural design decisions that lead to a software architecture, whatever its kind might be.

SHARK'08, May 10–18, 2008, Leipzig, Germany.

As stated in [1], software architectures should be considered as the result of a set of design decisions, rather than a set of components and connectors. Hence, the importance of including design decisions as an add-on to the traditional software architecture documentation has been recently highlighted. Such idea was early recognized by Perry and Wolf [2] who mentioned the importance of the rationale in software architecture in the formula: Software Architecture = (Elements, Form, Rationale). Moreover, Kruchten et al. [3] modernized such idea, as they consider Architectural Knowledge = Design Decisions + Design. This Architectural Knowledge (AK) comprises different types of knowledge (e.g.: design patterns, styles, decisions), which is used during the construction of the software architecture.

To date, the most traditional architecting activity often neglected recording and documenting explicitly this AK because it has only focused on describing and documenting the architecture of a system from different points of view [4]. However, because design decisions can be needed by subsequent maintenance and evolution processes, one of the main benefits of having AK available is the chance of using it again. Therefore, convincing users to employ such AK might be based on empirical results able to demonstrate the practical utility of design rationale in the long term. In order to overcome some cultural and technical barriers that may inhibit the practical usage of DDRD, this paper aims to provide empirical confirmation that the level of utility related to the same category of DDRD information significantly changes depending on its purpose [5]. In particular, the present paper describes a controlled experiment carried out at University of Rey Juan Carlos Madrid as a replication of an analogous experimental study recently conducted at University of Roma Tor Vergata. The structure of the rest of the paper is as follows. Section 2 describes the background of our work. Section 3 describes the experiment we carried out based on the previous motivation and Section 4 discusses its main results. Section 5 summarizes the conclusions and lessons learned.

2. BACKGROUND

2.1 Introduction to DDRD
Because architectures have high costs for change and may erode during the evolution of the system [6], design decisions should be captured and documented to avoid knowledge vaporization. Hence, in order to prevent the erosion of software design and knowledge vaporization, we need to capture these decisions and their underlying reasons that led to any particular architecture. To achieve this goal, we first need to provide a way to represent the knowledge we want to capture. Several authors have proposed

some template list of attributes for representing and capturing this AK.

J. Tyree and A. Akerman [6] state that "most architectural design decisions and development processes don't document decisions explicitly". Architectures comprise different types of decisions with different types of granularity, and all this knowledge should be described and captured appropriately. Hence, they propose a template list of up 14 attributes for characterizing the decision we want to capture and document. Also, the relationships between decisions and other software artifacts should be described and documented explicitly in order to represent the choices selected and the alternatives for each choice.

Kruchten et al. [7] use an ontology to organize the different types of decisions, as well as an extensive list of attributes, some of them focused on documenting the evolution of such decisions.

Capilla et al. [8] describe a flexible approach in the form of mandatory and optional attributes for characterizing architectural design decisions that can be tailored to the specific needs of each particular organization. Therefore, software architects can decide the amount of information to store when describing their decisions and estimate how much effort would be needed for capturing the design rationale.

As a result, in order to manage decisions that remain implicit in the architect's mind, we need specific activities for capturing, managing, sharing, and documenting architectural design decisions concurrently with typical development and maintenance processes. Most of these activities can be based on uses cases, like those described in [9], and used, for instance, to enact incremental architecture review, get a rationale, and add a decision.

2.2 The importance of DDRD

Capturing the design rationale of a system is not trivial and requires certain effort in addition to typical architecture modeling tasks. As stated in [6] there are many claims about the problems caused of not recording design rationale. Because of the additional effort needed to capture this rationale, architects and business stakeholders many times do not fully understand the importance of recording this AK.

The survey reported in [10] stresses some findings about the perception of the importance of design rationale, its use and documentation. The empirical evidence from the results of this experiment shows that the users that participated in the case study [10] feel that design rationale is an important part of design that should be documented. Otherwise, they perceive a general lack of methodology and tool to support activities for capturing, using, and documenting the design decisions and its rationale. Technical and socio-technical factors influence in many cases the design decisions, but most of the respondents in the experiment perceived useful using design rationale.

L. Brathall et al. [11] presented a controlled experiment to evaluate the importance of DDRD when predicting change impact on software architecture evolution. Results show that DDRD clearly improves effectiveness and efficiency.

L. Karsenty [12] proposed an empirical evaluation concerning the utility of design rationale documents in maintenance of a nine months old software project.

M. Heindl and S. Biffl [13] reported a case study on a type of value-based requirements tracing, which aims to systematically support project managers in tailoring the traces, and is based on the following parameters: stakeholder value, requirements risk/volatility, and tracing costs. The main results of that case study are: (a) value-based requirements tracing takes around 35% of the effort required by full tracing; (b) more risky requirements need more detailed tracing.

D. Falessi et al. [14] studied the importance of DDRD with regard to effectiveness and efficiency of individual/team decision-making in presence of requirement changes. One of the expected outcomes was whether the availability of the documentation of the design decision rationale would improve the correctness of design decisions. In the study, conducted as a controlled experiment, fifty post-graduate Master students participated. As a result, the correctness of decisions improves when design decision rationale documentation is available, both for individual decision-making and team decision-making.

2.3 DDRD inhibitors

Despite of the usefulness of the DDRD in its different forms, its applicability for software engineering and more specifically for software architecture can be inhibited by the following factors [5].

Bad timing and delayed benefit. The period in which design decisions are taken is usually critical for the success of the software project. The project pressure is one of the main reasons for not documenting this DDRD. Our experience shows that suggesting people for documenting design decisions rationale in the right time is perceived useless.

Unpredictable information. The DDRD consumer and producer are often different persons. People who are in charge to evolve a system are often not the original designers, and an agreement should come up to decide which design rationale information would be relevant for the project. Agile approaches are needed to produce complete and valuable information that can be documented with moderate effort.

Overhead. Several DDRD techniques already exist; however they are usually focused on maximizing the consumer benefits rather than minimizing the producer effort. Consequently, people involved in the documentation and maintenance activities are supposed to spend a lot effort recording and documenting the rationale.

Unclear benefit. Decision-makers do not know many times for which purpose it is useful to learn the rationale of a design decision. Therefore, some previous training is particularly needed, as the users can perceive the usefulness of documenting the rationale.

Lack of motivation. This inhibitor is usually caused by the absence of clear or personal interest because people that developed the system for the first time don't act in the role of maintainers. Therefore they do not perceive the expected benefits to waste time recording the design rationale. Lee [15] already stressed the importance of this problem.

Designer attitude. The traditional architecting attitude neglects documenting the design decisions as these are the result of a mental activity of the architect, that users perceive hardly and expensive to be recorded.

Owner of the knowledge. The value of this intellectual capital is perceived by the creators as a property that should not be transferred or communicated to others.

Lack of maturity. Only few tools are currently available to support DDRD and therefore the field is to consider as rather immature.

Potential inconsistencies. DDRD implicitly represents the results of the design. If DDRD and the design documents are not well updated, potential inconsistencies in case of decision changes might occur.

These and other inhibitors may hamper capturing, using, and documenting the DDRD. In order to overcome such barriers, in next section we analyze the analyze the feasibility, of value based approach for DDRD, through a replicated experiment which constitutes a value-based approach for documenting design rationale.

2.4 Value-based software engineering

Nowadays, "much of current software engineering practice and research is done in a value-neutral setting, in which every requirement, use case, object, test case, and defect is equally important" [16]. The Standish Group CHAOS reports [17] figured out that value-oriented shortfalls, e.g., lack of user input, incomplete requirements, changing requirements, lack of resources, unrealistic expectations, unclear objectives, and unrealistic time frames, are the common causes of most software project failures. Consequently, "a resulting value-based software engineering agenda has emerged, with the objective of integrating value considerations into the full range of existing and emerging software engineering principles and practices, and of developing an overall framework in which they compatibly reinforce each other" [16]. In the present work we apply value-based software engineering principles on DDRD; we propose a value-based approach to DDRD (VB DDRD) which focuses on documenting only the set of required information based on the documentation purpose.

3. A REPLICATED EXPERIMENT TO CONFIRM THE FEASIBILITY OF A VALUE-BASED DDRD

Past DDRD methods aimed to maximize the DDRD consumer's benefit by forcing the DDRD producer to document all the potentially useful information. Such methods have been employed independently from the system requirements and business context: hence, we still have to consider them as value-neutral methods. However, because current (value-neutral) approaches includes DDRD inhibitors (see Section 2.3), the question is: What approach should we follow for mitigating the impact of those inhibitors? Our conjecture is that the answer is the injection of Value-Based software engineering principles on DDRD methods.

This work empirically analyze the feasibility of a Value-Based DDRD (VB DDRD) approach which includes certain aspects relevant for every software engineering practice, such as: *Where* (project context), *Who* (the beneficiary stakeholders), *When* (DDRD type of use), *Why* (process and product metrics) and *How* (DDRD required information). Hence, we try to take advantage of an *a priori* understanding of who will profit later on, from what set of information, and in which amount.

3.1 DDRD Use-cases

We can think of several usage scenarios for DDRD. A DDRD Use-Case (DDRD UC) occurs when one or more actors use DDRD in order to obtain a certain advantage while enacting a particular activity in a given project. A DDRD UC is characterized by the following attributes

- **ID**: An identifier of the scenario.
- **Actor**: Represent the people involved in the DDRD UC which may have different roles, such as DDRD producers and consumers.
- **Context**: System or industry characteristics where the usage scenario happens. Information about the environment is also includes.
- **Activity**: Are those tasks actors enact when they use DDRD.
- **Advantages**: Product or process metrics able to determine the benefit of the DDRD usage

The list of uses cases we employed in our experiment are the following:

- **UC1**: Detection of wrong knowledge on decisions. (e.g.: Which characteristics of the chosen decision are wrong?)
- **UC2**: Detection of wrong requirements understanding. (e.g.: Which characteristics of the problem to address have been wrongly understood?)
- **UC3**: Design checking (verification and evaluation). (e.g.: Do you approve the decision made?)
- **UC4**: Detection of conflicts among new requirements and old decisions. (e.g.: Is the old decision still valid for the new set of requirements?)
- **UC5**: Impact evaluation. (e.g.: Which is the impact of new requirements in the system?)

3.2 Key idea

In our approach we use the following terms:

- "useful information", is a kind of information that helps to a small or large extent the readers to understand the meaning of something;
- "required information" refers to that kind of information without which, independently from the effort of the readers, the meaning of something cannot be understood;
- "optional information" means information that is not required to understand something, but it can be useful.

Our key idea is that all the information included in a DDRD might be useful but sometimes some information are merely optional. Moreover, we expect that the amount of importance related to the information included in the DDRD depends on the DDRD UC. In other words, we expect that different DDRD UC(s) require different categories of DDRD information. In such case, the DDRD can be tailored based on the DDRD UC(s) to enact. In our view, the adoption of a tailored DDRD, consisting only of the required set of information, would mitigate the effects of DDRD inhibitors. The idea to consider a subset of relevant items for DDRD is similar to the use of mandatory and optional attributes

for design decisions described in [8]; it differs in the fact that the proposed VB-DDRD is focused on the DDRD UC to enact.

3.3 Experiment at the URJC

In the absence of any valid formal models for our VB DDRD, we are unable to compute pros and cons of our approach. Our decision is hence to proceed empirically, in particular to conduct a replicated experiment, with the goal of confirming the feasibility of our approach rather than comparing it with other ones. In next subsections we describe a replica of an experiment carried out at the University "Tor Vergata" of Rome (Italy) [5], this new experiment has been carried out at the University Rey Juan Carlos (URJC) of Madrid (Spain).

3.3.1 Research question

The goal of our study, in the sense given by Basili [18], is to analyze the DDRD for the purpose of evaluation with respect to the perceived utility from the point of view of the researcher in the context of post-graduate Master students of software engineering. The aims of this empirical study is both explorative and confirmative: i) to discover the perceived levels of importance related to each category of DDRD information for enacting different DDRD use cases, ii) to confirm past empirical results [5] that the level of importance related to the same category of DDRD information is different for different DDRD UC(s). The latter is the key principle of our proposed VB DDRD, without which our VB DDRD would not work.

3.3.2 Hypotheses

In order to investigate that the levels of importance related to DDRD categories are different with respect to different DDRD UC(s), we derive the following null hypotheses. When enacting DDRD UC(s) with ID(s) *i* and *j*, there is no significant difference (H_{0ij}) (resp. there is significant difference, H_{1ij}) between the levels of perceived importance related to the category of DDRD information *x* (H_{x}). Notice that *i* and *j* can be any DDRD UC(s).

We highlight that the assumption that the levels of importance related to the same category of DDRD information are different for different DDRD UC(s) is valid only if the above null hypothesis is rejected for one or more combinations of *i* and *j*. In particular, each specific combination of *i* and *j* in which the null hypothesis is rejected, identifies a specific pattern in the perceived importance level of a category of DDRD information.

3.3.3 Variables

The experiment takes into account five DDRD UC(s) that represent five levels of this independent variable (i.e. treatments). As dependent variables, we used the utility related to each specific category of DDRD information, as perceived by the subjects, for enacting a specific DDRD UC. Participating subjects expressed quantitative measures of the utility for a specific category of DDRD by a 3-point ordinal scale (useless, optional, or required). We used the template list of attributes of DDRD proposed by Tyree and Akerman in [6]. We also controlled at a constant level the remaining independent variables such as experience of the participating subjects, experiment materials, environment, and complexity of the experiment objects. We also blocked a further independent variable, the type of documentation, because are not were not interested in investigating this dimension.

3.3.4 Subjects at the URJC

Twenty five students belonging to the last course of the Master Degree in computer science, at the University of Rey Juan Carlos (Madrid, Spain) participated in the experiment. At least around the 60% of the subjects work for software companies, so they can be considered as software professionals; some of these are junior (less than 2 years of experience). 40% of the participants don't work for a company yet.

3.3.5 Design

We modeled five different roles or types of stakeholders: authentication, human interface, operative system, communication protocol, and data storage. Then, subjects expressed their preference for each role, according to their previous experience and level of confidence with the responsibilities of a role, but at the end we equally distributed five students per role. We selected five DDRD use-cases to investigate and five decisions for each role. Each experiment consists in a DDRD-documented decision already made by an architect, who virtually left, and a set of new requirements. Since a decision (and its DDRD) is compatible with the five DDRD UC(s), we used the twenty-five decisions with each DDRD UC. All subjects enacted all the five DDRD UC on all the five decisions belonging to his role. Moreover, in order to limit the occurrences and mitigate the influence of several threats to validity, we balanced the experiment design as we explain.

- Each decision has been adopted the same number of times.
- Each treatment (on each decision) has been applied the same number of times.
- Each subject applied all DDRD UC (on all decisions).
- All treatments (i.e. DDRD UC(s)) have been applied the same number of time (i.e. 25) in all the available orders (i.e. as first, second, third, fourth, and fifth).

3.3.6 Experimental material

In order to replicate the context of the real world, we used a synthetic software project concerning to a public transportation system with ambient intelligent features like heterogeneous sensors. Each subject received the material containing (1) the experiment rules, (2) the system main characteristics, (3) five different decisions (with related DDRD and new requirements), (4) a description of which DDRD UC enacts on which decision, and in which order, (5) the form to fill data in. In particular for each of the five DDRD UC(s) to enact, subjects had to perform the following steps: 1) to understand the current DDRD UC to enact, 2) to enter the form with the current time (initial time), 3) to read the DDRD related to a specific decision, 4) to enact the DDRD UC (write the requested answer), 5) to write the final time on the form, 6) to describe the level of perceived utility for each category of DDRD information.

Figure 1 shows the form the subjects had to fill in during the experiment; the first two columns describe the DDRD UC order of execution. The following three columns describe the initial time, the answer to the DDRD UC (described in the same row, column 1), and the final time when the DDRD UC completed, respectively. The columns from 6 to 18 describe thirteen perceived levels of importance: a column identifies a DDRD

information category; a row identifies an enacted DDRD UC. Each subjects used five of this form, one form for each decision.

3.3.7 Training

Previous to the execution of the experiment we perform an overview of typical software architecture concepts and we trained the subjects about the notion and the importance of design decisions, design rationale, techniques for capturing decisions, and their relationship to other products of the software life-cycle. We provide some examples about the concepts taught to the students.

We spent four hours in two sessions in such training activity. All the students attended to the training sessions so none of the subjects was exclude in the experiment. We also motivated the students about the utility of capturing and documenting design decisions for software architecture.

Finally, we ran a "pilot experiment" to check the conformance of material and let the subjects familiarize with the task to be done as well as with the materials. Therefore, the errors and doubts were solved during such a "pilot experiment".

3.3.8 Execution

After the subjects were familiarized with the tasks of the experiment and with the materials as well as once we answered the questions made during the "pilot experiment", the students had to fill the table of Figure 1. Then, we carried out the experiment in one single session of up to five hours maximum and we collected the results. Each subject had to fill five excel sheets with the results of applying each DDRD UC to the five selected decisions (DDRD) according to the role selected by the subject (e.g.: hardware, wireless, etc.).

Order		Answer			Utility level of DDRD information (0=Useless, 1=Optional, 2=Required)												
ID UC	ID D	Initial Time	Answer	Final Time	Issue	Decision	Status	Assumptions	Constraints	Positions	Argument	Implications	Related decisions	Related requirements	Related artifacts	Related principles	Notes
1	1																
2	2																
3	3																
4	4																
5	5																

Figure 1. Form filled by the subjects in the experiment

4. RESULTS

In general, the more a category of DDRD information is perceived as "Required", the more it is valuable for DDRD consumers. Table 1 shows the mean and the variance of the subjects who perceived a specific category of DDRD information as required while enacting the DDRD UC(s). Thus, Table 1 helps us to understand which parts of the documentation are interesting for the readers in case they have to enact all the DDRD UC(s).

Table 1. Percentage of subjects that felt as "required" a specific category of DDRD information

DDRD Information	Mean
Issue	70,72
Decision	94,24
Status	46,88
Assumptions	43,04
Constraints	22,08
Positions	53,76
Argument	64,64
Implications	37,76
Related decisions	56,16
Related requirements	67,84
Related artifacts	9,44
Related principles	12
Notes	0,48

Figure 2 shows the percentage of subjects who marked a specific category of DDRD information as "Required". Figure 2 supports our understanding of the relationship, if any, between the importance of a specific category of DDRD information and specific DDRD UC(s). More specifically, DDRD producers can use Figure 2 as a reference to choose whether to include or disregard a particular of DDRD information.

These results seem to confirm that the level of perceived utility does change both in relation to the category of DDRD information and the DDRD UC enacted. Note that measurements made in the experiment are repeatable in real-world projects; if you disagree with these results, you could repeat then the measurements in your own organization.

The results of Figure 1 complement those obtained at the University Tor Vergata, which are almost similar in most cases; we remand to future works the discussion of the differences between the two experiments results.

4.1 Analysis of Results

Because we are interested in the feasibility of our VB-DDRD approach, the results we observed form the experiment carried out are the following.

Issue. We confirmed our initial expectations about how useful users perceive the issue as part of the DDRD information. As we see in Table 1 we noticed that such a category have been perceived quite always (i.e. 71% of the time) as "required" and, as Figure 2 shows, independently of the UC enacted.

Decision. This information has been proven as the most used and useful item as it describes the decision itself. This item has been required frequently in almost all the DDRC UC with the highest percentage of use compared to the other information categories analyzed.

Status. The status of a decision is quite important for maintenance tasks as it describes the current state of a decision (e.g.: approved, rejected, pending). Subjects perceived this item not enough useful as they didn't carry out further maintenance activities. Moreover, we observed from Figure 2, that the status

appears important for DDRD UC 3 (i.e.: decision verification) while is not so important for others (e.g.: conflict detection).

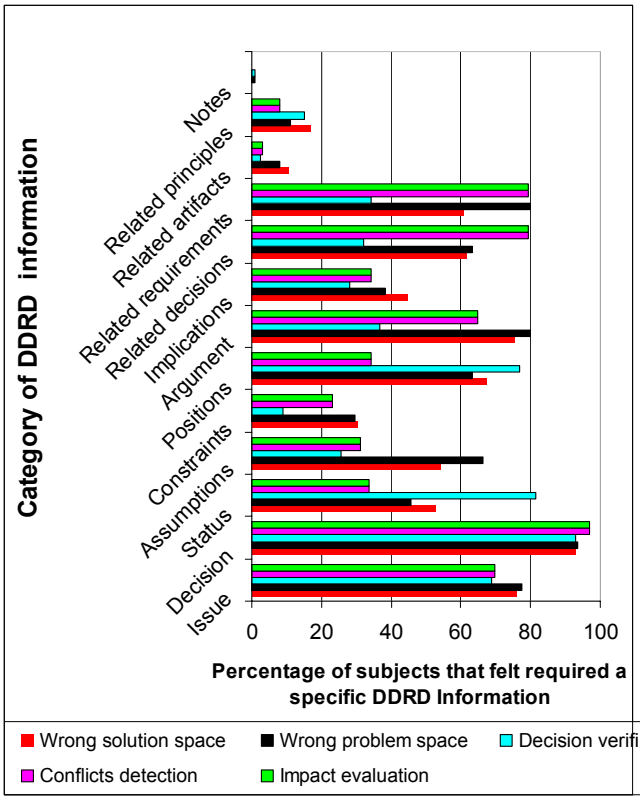

Figure 2. Percentage of subjects that felt as "required" a specific category of DDRD when enacting a specific DDRD UC

Assumptions. By analyzing Table 2 we observe that only half of the time this information has been perceived as required. In particular, by observing Figure 2 we notice that the importance of this item is particularly high when used to verify the knowledge of the problem space (DDRD UC 2) rather than in the other UC(s). Hence, we believe that subjects are not too familiarized with assumptions and they concentrate on the requirements and arguments.

Constraints. This information has been perceived required only sometimes. We deduce that subjects don't know how to deal with constraints in the context of using design decisions, and how these relate to software requirements as well. Constraints are often seen as barriers for making decisions, as the main reason to disregard this item. For subjects, constraints have a very little influence on verifying the decisions made.

Positions. The description of the alternatives considered for making a particular decision has being perceived in general quite useful for enacting DDRD UC 1-3 (wrong solution space, wrong problem space, and decision verification) and less important for DDRD UC 4-5 (conflicts detection, and impact evaluation). Thus, positions clearly show the list of alternatives ordered by significance for the responsible of the decisions on the early stages of the reasoning process as it is needed to approve the decision among different alternatives.

Arguments. They constitute the underlying motivation that takes during the reasoning phase. Subjects perceived useful this item a number 65% of times, but as opposite to the status category, the importance of the arguments have been recognized for all the DDRD UC(s) excepting for DDRD UC 3 (decision verification) which is less important.

Implications. In general, implications had a low importance for the subjects for all use cases. Subjects didn't perceive it enough much useful in the context of the design decision.

Related decisions. Related decisions had a moderate level of importance for subjects. Even the participants perceived this item important they don't visualize the whole picture of the overall related decisions. Therefore, they can't imagine how to manage the set of intertwined decisions in the contexts of a software system. Related decisions exhibit similar results like the argument and DDRD UC 3 exhibits lower importance with respect to the other use cases.

Related requirements. As users are more familiar with requirements, related requirements are considered more important than related decisions. Subjects learned that in order to make a good decision, they need to understand the problem space by means of the related requirements that might impact on the rationale that guides such decision. The results and the importance for the UC(s) is similar to the related decisions, as Figure 2 shows.

Related artifacts and related principles. Most of the decisions didn't provide information for these two items. Therefore, the users didn't feel important enough the utility of related artifacts and in particular the principles that guide the decision. Related principles seem to be a little bit more relevant for all UC(s) rather than the related artifacts, and the results are equally distributed for all use cases.

Notes. Almost all the users considered this field as useless; we believe this item should be considered more valuable as it allows representing other concerns not described by the rest of attributes that characterize a decision. Form Figure 2, we can observe that the results are insignificantly for all UC(s). Only some value is assigned to DDRD UC(s) 2 and 3.

As a summary, some information categories show significant differences in the perceived utility level while enacting different DDRD UC(s). Hence, the utility of such items may depend on the specific DDRD UC to enact. This information seems to be useful to estimate the type and the amount of information users would perceive more useful to be captured. Results suggest that we can save some effort by tailoring the DDRD into any organization, according to their business needs (DDRD UC).

5. CONCLUSION

In this paper we have described a replicated experiment on the feasibility of a value-based approach for documenting the reasons behind design decision (VB DDRD). Such a VB DDRD approach tries to mitigate the inhibitors that actually prevent the practical usage of DDRD in industry.

In this work we described a controlled experiment as a replica of one enact at the University "Tor Vergata" of Rome (Italy) to demonstrate that the importance of DDRD may vary depending of the use cases enacted. For the five DDRD UC(s) we enacted 125

executions and all the subjects provide different utility levels for each category of DDRD. Initially, the subjects had some difficulties to understand how to use the design rationale for each UC and which utility level was the most appropriate for each item of Figure 1.

We observed several difficulties in the reasoning activity to answer the questions that motivated each DDRD UC. Thus, from the reasoning activity of the subjects we deduced that a good description of the decisions is needed in order to avoid confusion when reusing decisions made by others.

A big number of information categories may increase the effort spent by the subjects in understanding the DDRD. Hence, we believe that flexible approaches able to reduce and adapt the different categories of information should be used to tailor DDRD.

Additionally, we learned that the level of utility related to the same category of DDRD information, significantly changes according to each DDRD UC. This means that DDRD consumers require specific categories of DDRD information according to the DDRD UC to enact. Consequently, this result suggests that the DDRD producer should include only the information required for the specific DDRD UC(s) that is expected to be enacted (i.e. the value-based approach). We finally advocate for the use of DDRD that should be captured and used according to the user or organizational needs and use lightweight and agile methods to reduce the effort both in creating and consuming such relevant architectural knowledge. Documenting this architectural knowledge explicit may be seen as a "new" cross-cutting architecture view [19] that complements the information documented in the other traditional architecture views.

Future works will include analysis and discussion about the differences and similarities among the results of presented experiment versus the previous one [5].

6. ACKNOWLEDGMENTS

The work of Rafael Capilla is partially funded by the PILOH project of the Spanish Ministry of Education and Research programme under grant number URJC-CM-2006-CET-0603.

The work of Davide Falessi is partially funded by the Intl. Cooperation Project of the University of Rome Tor Vergata with universities of Spain.

7. REFERENCES

[1] A. Jansen, and J. Bosch. Year. Software Architecture as a Set of Architectural Design Decisions. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 5),(Pittsburgh Nov. 6-9), IEEE CS.

[2] D. E. Perry, and A. L. Wolf, 1992. Foundations for the Study of Software Architecture. ACM Software Engineering Notes. 17, 4. (October), 40-52.

[3] P. Kruchten. 2003 The Rational Unified Process: An Introduction 3rd ed. Addison-Wesley Professional.

[4] P. Kruchten, 1995. The 4+1 View Model of Architecture. IEEE Software. 12, 6. (November), 45-50.

[5] D. Falessi, G. Cantone, and P. Kruchten. 2008. Value-Based Design Decision Rationale Documentation: Principles and Empirical Feasibility Study. In Proceeding of the Seventh Working IEEE / IFIP Conference on Software Architecture (WICSA 2008), (Vancouver, Canada).IEEE Computer Society.

[6] J. Tyree, and A. Akerman, 2005. Architecture Decisions: Demystifying Architecture. IEEE Software. 22, 2. (March/April), 19-27.

[7] P. Kruchten. Year. An ontology of architectural design decisions in software intensive systems. Proceedings of the 2nd Groningen Workshop on Software Variability.

[8] R. Capilla, Nava, and J. C. Dueñas. 2007. Modeling and Documenting the Evolution of Architectural Design Decisions. Proceedings of the 2nd Workshop on Sharing and Reusing Architectural Knowledge, ICSE Workshops, IEEE DL.

[9] P. Kruchten, P. Lago, and H. van Vliet. Year. Building up and Reasoning about Architectural Knowledge. 2nd International Conference on the Quality of Software Architectures,(Vaesteras, Sweden June 2006), LNCS 4214, Springer Verlag.

[10] A. Tang, M. A. Babar, I. Gorton, and J. Han, 2007. A Survey of Architecture Design Rationale. Journal of Systems & Software. 79, 12. 1792-1804

[11] L. Bratthall, E. Johansson, and B. Regnell. Year. Is a Design Rationale Vital when Predicting Change Impact? A Controlled Experiment on Software Architecture Evolution. International Conference on Product Focused Software Process Improvement,(Oulu , Finland 20/06/2000).

[12] L. Karsenty. 1996. An empirical evaluation of design rationale documents. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Common Ground, (Vancouver, British Columbia, Canada).ACM Press.

[13] M. Heindl, and S. Biffl. 2005. A case study on value-based requirements tracing. Proceedings of the 10th European Software Engineering Conference, (Lisbon, Portugal).ACM Press.

[14] D. Falessi, G. Cantone, and M. Becker. Year. Documenting Design Decision Rationale to Improve Individual and Team Design Decision Making: An Experimental Evaluation. Proceedings of the 5th ACM/IEEE international symposium on International Symposium on Empirical Software Engineering,(Rio de Janeiro, Brazil).

[15] J. Lee, 1997. Design Rationale Systems: Understanding the Issues. IEEE Expert: Intelligent Systems and Their Applications. 12, 3. 78-85.

[16] S. Biffl, A. Aurum, B. Bohem, H. Erdogmus, and P. Grünbacher. 2006 Value-Based Software Engineering. Springer.

[17] The-Standish-Group, CHAOS Report 1995, www.standishgroup.com, 1995.

[18] V. Basili, G. Caldiera, and D. Rombach, 1994. Goal/Question/Metric Paradigm. Encyclopedia of Software Engineering. 1, John Wiley & Sons. 528-532.

[19] J. C. Dueñas, and R. Capilla. Year. The Decision View of Software Architecture. 2nd European Workshop on Software Architecture,(Pisa, Italy June 13-15, 2005).