Model-Centered Customizable Architectural Design Decisions Management

Lianping Chen
Lero—The Irish Software Engineering Research Centre
University of Limerick
Limerick, Ireland
Lianping.Chen@lero.ie

Muhammad Ali Babar
Software Development Group
IT University of Copenhagen
Copenhagen, Denmark
malibaba@itu.dk

Haiqi Liang
China Software Development Lab
IBM
Beijing, China
lianghq@cn.ibm.com

***Abstract*—Architectural Design Decisions (ADD) form a key element of Architectural Knowledge (AK), which plays a vital role in the software architecture process. To help manage ADDs, several tools have been proposed. However, most of them have prescribed fixed data models to be followed and do not provide sufficient customizability. Mismatches between a tool's data model and users' specific needs make the tool less usable, or even unusable. We propose a highly customizable solution that enables users to define specialized ADD models according to the specific needs of their individual preferences and working situations to achieve perfect fitness between the required model by users and the provided model by the tool. The results of the initial evaluation of the proposed solution are encouraging.**

Architectural design decisions; architectural knowledge; customizability; software architecture; knowledge management

I. INTRODUCTION AND PROBLEM STATEMENT

Recently, there has been an increased awareness that not only the architecture design itself is important to capture, but also the knowledge about the set of Architectural Design Decisions (ADD) leading to it [1-3]. Establishing ways to manage and organize ADDs is one of the key challenges in the field of Software Architecture (SA) [4]. To better manage ADDs, several tools have been developed (e.g., PAKME [5], ADDSS [6], Kruchten's Ontology Tool [7], ADkwik [8, 9], and EAGLE [10, 11]).

However, most of them fall short of meeting users' specific needs, because they prescribe a fixed ADD model (i.e., something that constrains the structure and formalization of captured ADDs) to follow without sufficient support for customization. It is worth clarifying that by customization we mean the modification of the ADD model to suit the specific needs of users without any programming involved. Practitioners usually have specific needs raised by their unique working situations. Different working situations usually have different requirements on the ADD model [7, 12-19]. The mismatch between a fixed ADD model and the required ADD model in a particular working situation can be quite large [12, 13].

Prescribing users to follow a fixed ADD model that does not fit to their needs can cause significant problems. People are forced to adapt their way of thinking and describing that thinking according to a particular prescriptive ADD model. This usually introduces extra effort in the process of converting from users' preferable and intuitive organization of ADDs to the imposed ADD model. Such conversion can be very hard to achieve in some cases (e.g., forcing users to capture ADDs in a more fine-grained way than what they are able to do at that time). Users' willingness and motivation of ADD externalization and documentation with the tool can negatively be affected in terms of frustration. As found by Poort et al., this negative emotion can dramatically affect the practice of Architecture Knowledge (AK) sharing in organizations [20]. Falessi et al. [12] performed a study based on a fixed ADD model (i.e. the template reported in [21]), the results indicate that if users are forced to follow a fixed model, there may be up to 50% extra cost incurred. Hence, we argue that lack of fitness of an ADD model to users' needs, a tool's perceived ease of use, perceived usefulness, and users' intention to use – which are the key predictors of technology acceptance [22, 23] – can be dramatically decreased as reported by Lago [24] and Thomas et al. [25].

It can be asserted that ADD Management (ADDM) systems must allow users to invent new ADD models and to arbitrarily modify them in order to accommodate their specific needs [26]. This paper proposes a solution aimed at addressing the shortcoming of the currently available ADDM tools.

The remainder of this paper is organized as follows. Section II presents a set of customization scenarios identified from a case study in order to characterize the needs of customizability. In Section III, existing ADDM tools have been evaluated to determine the extent to which they support the customization scenarios. Section IV introduces a solution to support ADD model customization. Section V describes

an initial evaluation of the proposed solution. Section VI presents few suggestions on ADD model customization. Section VII ends this paper with conclusions and future works.

II. ADD MODEL CUSTOMIZATION SCENARIOS

We have carried out a case study [13], which helped us to identify a set of scenarios to characterize the customization needs for ADD model. These scenarios can help assess the extent to which a tool supports customizability. Here we only provide a brief summary of the identified ADD model customization scenarios. We refer readers to [13] for more detail.

Delete an attribute: It is possible or even common that some attribute (we refer a data field of an ADD model as an attribute) of a prescribed model is not useful for a specific context in which users are working. Thus, the users need to delete that attribute.

Add an attribute: Users may need a new attribute to represent, e.g., some piece of context specific information. Thus, the users need to add an attribute.

Change the name of an attribute: A user may need to change the name of some attribute of a prescribed model to fit to the terms that are most familiar to the user. We found using a proper name is important in order to avoid any misunderstandings.

Change the description of an attribute: Some attributes may need to have contextualized or adapted meanings of the attributes. Thus, the attributes' description needs to be changed.

Change the property of "mandatory or optional": An attribute may be mandatory in situation A but optional in situation B, and vice versa. Thus, users need to change the property of "mandatory or optional" for some attributes of a prescribed model. Capilla et al. reported similar observation in [27].

Change value range of an attribute: some attribute may have a fixed set of values. The values in the set can vary from situation to situation. For instance, the attribute "decision status" may have a value range {Approved, Obsolete, Pending, Rejected} in one situation, and a value range {Idea, Tentative, Decided, Approved, Challenged, Rejected, Obsolesced} in another situation. Thus, users need to change the value range of an attribute.

Change the position of an attribute in the attribute list: A suitable order of the attributes can vary in different working situations. Thus, users need to adjust the order of the attribute.

Merge two attributes: A coarse-grained ADD model may be more suitable for users' context. Thus, users may need to merge two attributes to form a coarse grained attribute.

Split an attribute: A fine-grained ADD model may be more suitable for users' context. The users may need to split an attribute into two or more attributes.

We organized these scenarios in a fine-grained manner to concretely characterize the ADD model customization. However, some scenarios can be grouped into course-grained scenarios.

III. CUSTOMIZATION SUPPORT IN EXISTING TOOLS

We have investigated existing ADD management tools to determine the extent to which they support ADD model customization. To the best of our knowledge, there are no concrete criteria on ADD model customization support reported. Thus, we used the ADD model customization scenarios identified in the previous section as criteria in this investigation. The tools we investigated include PAKME [5], ADDSS [6], Kruchten's Ontology Tool [7], and ADkwik [8, 9]. They represent the current state-of-the-art in ADDM tools.

PAKME is a web-based ADDM tool built upon an open source groupware platform called Hipergate [28]. ADDSS is a research web-based tool for storing, managing, and documenting architectural design decisions during the architecting process [6]. Kruchten's Ontology Tool is developed to capture design decisions in a structured form using the ontology proposed in [29]. ADkwik [8] is an application wiki for collaboratively managing architectural decision knowledge. It is available on IBM's alphaWorks [30]. We did not include EAGLE [10, 11] in this investigation because it mainly captures ADD in free text.

TABLE I. THE SUPPORT OF ADD MODEL CUSTOMIZATION BY EXISTING ADDM TOOLS

Scenarios\Tools	PAKME	ADDSS	Kruchten's Ontology Tool	ADkwik
Delete an attribute	N	P	N	N
Add an attribute	N	P	N	N
Change attribute name	N	N	N	N
Change attribute description	N	N	N	N
Change attribute optionality (mandatory or optional)	N	N	N	N
Change value range of an attribute	N	N	N	N
Change attribute position	N	N	N	N
Merge attributes	N	N	N	N
Split an attribute	N	N	N	N

(N = not support; P = partially support; Y = support)

Table 1 presents the results, which show that except ADDSS, rest of the three tools do not support any of the ADD model customization scenarios. ADDSS only partially supports the scenarios of "delete an attribute" and "add an attribute" as it allows only a few attributes to be added to or deleted from its model. The users can not add any new attributes except the ones prescribed and delete any attributes except the ones prescribed. The findings reveal that the support for ADD model customization provided by existing tools is not satisfactory.

IV. CUSTOMIZABLE ADD MANAGEMENT

An ADD model usually serves as the base of an ADDM tool. Thus, we propose an ADD model-centered customizable solution for ADDM. This solution can enable users to get a personalized ADDM tool for their specific needs by configuring the ADD model. An infrastructure, called Customizable Architectural Design Decisions Management System (CADDMS) is used to support such an approach. In this section, we first give an overview of the ADD model customization part of CADDMS, then describe the underlying model that enables the customization, give an example of customization, and finally discuss other features of CADDMS.

A. Overview of ADD Model Customization

Fig. 1 presents the process for ADD model customization. The cube represents an operation, and other entities represent the outputs. A user first configures the ADD model. After a user submits his/her configuration, an ADD model definition is automatically generated. The system then takes the ADD model definition, and automatically transforms it into customized templates, storage, and search module.

The customized templates are presented as web-based forms, which represent the customized ADD model from two aspects: 1) the data structure prescribed by the ADD model, and 2) the constraints placed on the attributes. For example, a user can specify that for attribute A, a value should always be given and the value should be numerical within a certain range. These constraints are enforced when a user enters the architectural design decisions. Moreover, a user can also see the semantics, which are intended by the creator of the ADD model, of each attribute when providing values for different attributes of a design decision.

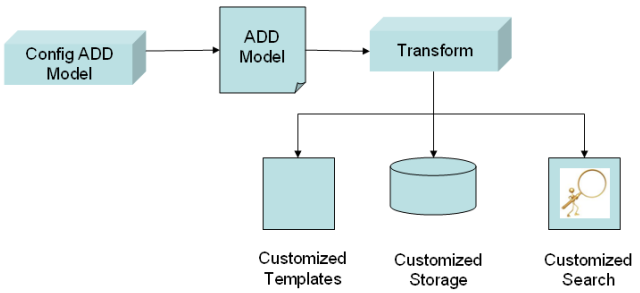

Figure 1. Overview of ADD model customization

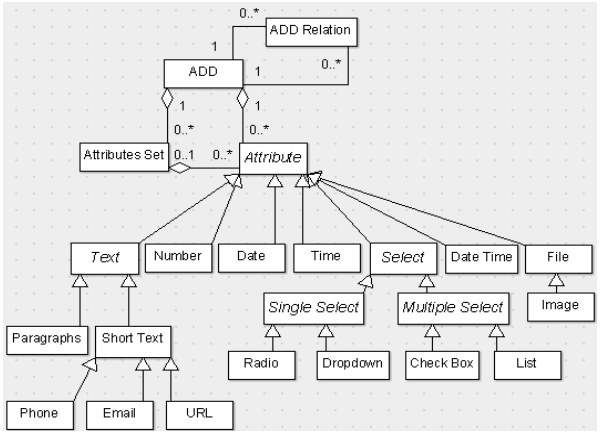

Figure 2. The meta-model for defining ADD model

The customized storage is a structured repository for storing the information captured by the customized template. This structured storage for ADDs can make the search and other data manipulations (e.g., analysis, visualization, and codification of ADDs) easier than using productivity applications (e.g., MS Word or Excel) or traditional wiki pages [31]. The machine processing of ADDs captured in Word documents, Excel sheets, or wiki pages is not easy [31].

The customized search enables a user to search their captured ADDs. Based on a customized ADD model definition, the system automatically generates a search facility that enables a user to specify his/her search criteria in the granularity of each attribute of the model. For example, a user can specify that "list all ADDs with value V1 for attribute A1, with value V2 for attribute A2, and with value V3 for attribute A3". The system fetches and shows only those ADDs that satisfy the search criteria. We assert that compared with keywords-based search, the customized search can be more fine-grained and accurate. Since, a keywords-based search may be sufficient for some situations, the system supports keywords-based search as well.

B. Underling Model to Enable Customization

Fig. 2 presents the underling meta-model that enables ADD model customization. We use the Unified Modeling Language (UML) notation for describing this meta-model. The italic font in the diagram indicates that the meta-class is abstract. In order to avoid any clutter, we do not show the attributes of the meta-classes. The ADDs (shown as the ADD meta-class) is the core entity of the model. An ADD meta-class is used to define an ADD class, which is used to prescribe the data structure that the ADDs should be captured and managed. An ADD captured based on the ADD class is an instance of the ADD class. The ADD meta-class has several properties, for example, name, label, description, namespace, and presentation style. A user can give a name to the ADD class (often the same as the name of the ADD model), for example, "MyADDModel". This name property is for machine processing purpose. In contrast, the label property allows users to give a human readable name to the

ADD model. The description property allows users to provide a short description of the ADD class. The namespace property gives each ADD class a unique identifier to avoid any conflict. The presentation style property is used to specify the look and feel of the web-form that users will use to fill in the ADD instances.

The ADD RELATION meta-class allows users to define the types of relationships that may be defined between ADD instances. Such as forbids, enables, subsumes, conflicts with, overrides, comprises, is alternative to, is bound to, is related to, and dependencies, as were described by Kruchten et al. [32]. The ADD RELATION has several properties, such as name, description, direction, from, and to.

An ADD meta-class is composed of several attributes and attributes sets. The concept of attribute set enables users to organize several tightly related attributes to a group. For example, users can organize attributes of "System scope", "Time scope", and "Organization scope" [32] into one group called "Scope". The attributes and attributes sets of an ADD class are defined by the ATTRIBUTE and the ATTRIBUTE SET meta-class, respectively.

The ATTRIBUTE meta-class has several properties such as name, label, description, multiplicity, and constraints. The properties of name, label and description have the same purposes as those of ADD meta-class. The property of multiplicity allows users to specify the number of occurrences of an attribute. The property of constraints allows users to impose some rules on the valid value of an attribute. The ATTRIBUTE meta-class has many sub-classes. The sub-classing is based on the data types of possible values for the attribute. These sub-classes were defined based on the types of data input elements used by web-based applications and HTML forms [33].

From the preceding description, we can see that the system supports very flexible customizability. The customizability ranges from very simple customization (e.g., changing names of some attributes) to enable users to invent an arbitrary ADD model, which can be the extreme customizability.

C. *An Example of Customization*

The example is based on the architecture decision description template reported in [21]. Since that template has been reported based on real projects, it is expected to bring some practical implications to this example. As shown in Fig. 3 and Fig. 4, the system provides a WYSIWYG (What You See Is What You Get) style user interface for users to define or customize ADD model. To add a new attribute to an ADD model, a user needs to click on the parent node, a drop down menu pops up with options "add field", "add field set". When a user chooses the "add field" option, a property specification page as shown in Fig. 3 appears. Once a user has specified the properties of the required fields of an ADD model, the left tree view outline and the preview page (shown in Fig. 4) is updated automatically. So whenever a user adds or updates a model, he/she can immediately see the exact template, including the look and feel of the web form, to be used to capture/update ADD information.

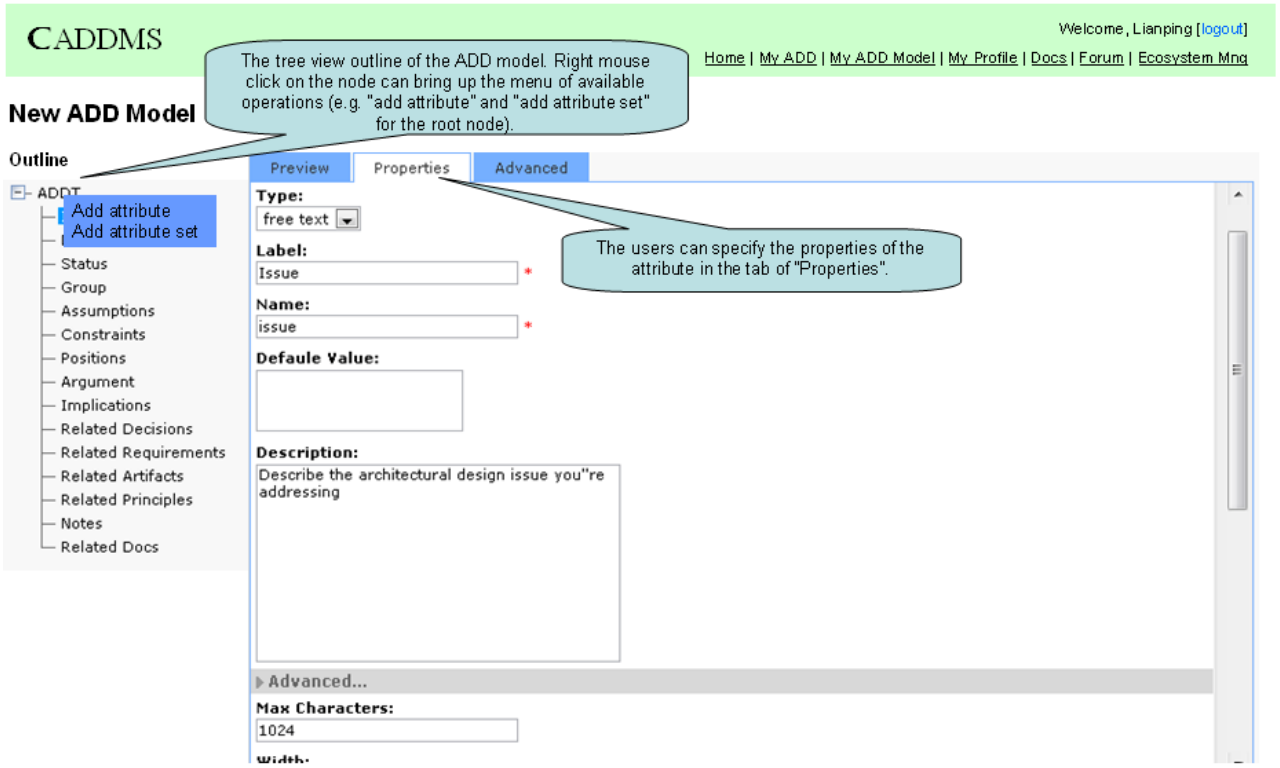

Figure 3. A screenshot for defining an ADD model

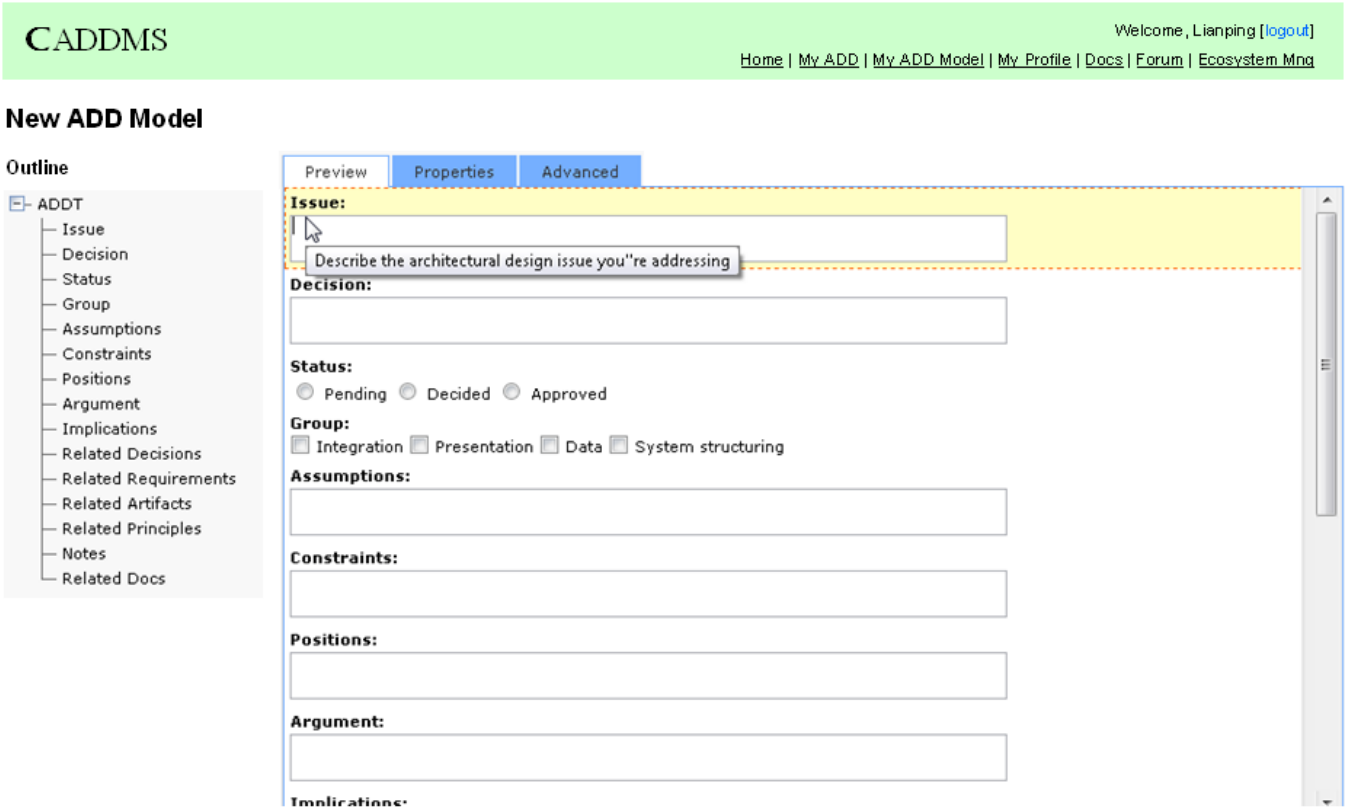

Figure 4. ADD model definition – ADD capture template preview

To change the order of the fields, which is also an important factor influencing the usability of a template, a user merely needs to drag the node representing the field on the left tree view outline and drop it at the desired position. This change is automatically reflected in the data model as well in the ADD template based on that data model. Other operations like modify field definition and delete field can also be performed quite intuitively. Once a user has defined a model, a customized ADDM tool support is automatically generated with a click. The tool consists of, among other functionalities, a customized ADD template, a customized storage, and a customized search facility (as shown in Fig. 5).

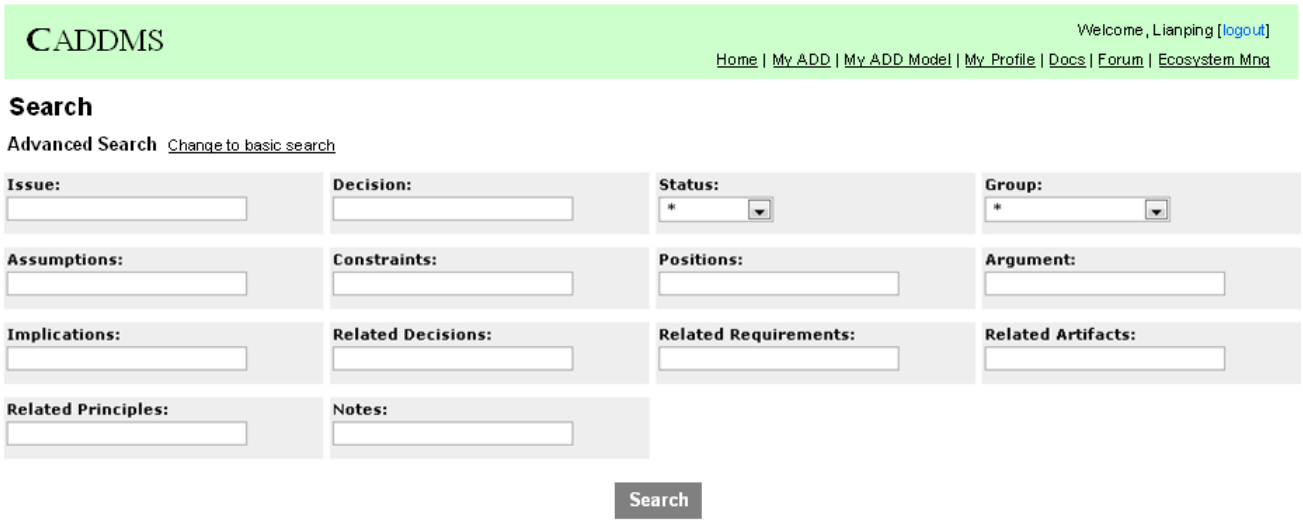

Figure 5. Personalized search of captured ADD

We have concluded that the ADD template presented in [21] has several attributes sensitive to working situations [13]. For example, the attribute "status" indicates the status of a decision. The possible values of "status" attribute are usually fixed in a specific project. As stated in [32], this attribute is analogous to problem reports. Taking a look at the available bug tracking systems, the diversity of status definition is obvious, which can also be anticipated in the "status" attribute of design decisions. The status definition is heavily determined by the process that a software team follows. For some large scale systems developed in a distributed organizations, the "scope" attribute [32] is very useful that is why a user needs to add this attribute to the model. Whereas in many small or medium sized projects, people may find it annoying if "scope" attribute is present in the ADD description template.

We can assume that an architect searches for a suitable ADDM tool support within CADDMS and finds the model that we have just defined close to her needs. However, some attributes of the model may not fit to her needs. She can easily customize the existing model and CADDMS will instantly generate a personalized ADDM tool support.

D. Other Features of CADDMS

Only supporting ADD model customization can not make a usable ADDM tool. CADDMS should also support several other features. Elaboration on these features is beyond the scope of this paper; however, we briefly describe some important features here.

Subscription and notification: Users can subscribe to the types of ADDs they are interested in (they can also use search strings to specify their interests in a fine-grained manner), and CADDMS notifies them whenever new ADDs are entered.

Sharing expertise and experience: CADDMS maintains each user's profile based upon which he/she could be searched for sharing knowledge, especially for the knowledge that is hard to articulate. A user can configure his/her profile not to be visible to members outside of her/his team.

ADD based discussion: Users can express their opinion on an ADD by posting comments on and rating a particular ADD.

Meta-data-enriched document management: The platform enables users to store documents with sufficient meta-data attached, which can be used for searching the documents.

Flexible classification of the content: The system enables users to make free classification of the content by using tags.

Scope-of-interest based views: the platform organizes ADDs based on the scope-of-interest, which can be a project, a department, or even the whole community. It is hierarchically organized. An inner scope-of-interest can be a member of an encompassing scope-of-interest. The members in an inner scope-of-interest can share information proprietary to them, they can also share some more general knowledge with a broader community consisting of the members of the encompassing scope-of-interest. A project can share some proprietary information within them. They can also share some more general knowledge with other teams in the same department. The knowledge in the innermost scope-of-interest will be dominant in the view that shows to the members of the scope-of-interest. For example, the system shows the knowledge pertaining to the project as the dominant part of the view that shows to the members of the project.

Role-based views: Within each scope-of-interest, different roles may have different views. For example, the views for the managers contain more summary information in the form of different types of reports.

V. EVALUATION

Customizability is the first criterion that CADDMS is expected to be evaluated against. We have explained how CADDMS can enable users to define arbitrary ADD models according to their specific working situations in previous sections. All the scenarios described in Section II are supported. Thus, we will not focus on customizability in this section. Instead, according to our conversations with practitioners in the industry, the aspects of the efforts needed for customization, the learnability and usability of CADDMS are important. Thus, we will first describe a user study that was performed to evaluate these aspects. Besides, there are general criteria for evaluating ADDM tools reported in the literature (i.e. [34]). We evaluate CADDMS against these criteria.

A. A User Study

The early feedbacks from industrial practitioners suggest that aspects such as the customization efforts required, usability and learnability of the tool are important to evaluate. This is because a customizable approach usually increases the initial effort of deploying and adopting. The more flexible an approach is, the higher the effort and skills usually required for customization. The effort required for performing the customization, and the usability and learnability of CADDMS will determine if we can achieve the goal that users are able to easily create a customized ADDM tool. Thus, we designed a user study to evaluate these aspects. Specifically, the main objectives of the study are as follow:

- determine the efforts required for obtaining a personalized ADDM tool support;
- evaluate the usability of CADDMS;
- evaluate the learnability of CADDMS.

We also intended to collect the suggestions on improvements of the system from participants.

The study involved eight participants: two post doctoral researchers, two Ph.D. students, and four undergraduate students. All of the participants were from Computer Science or Software Engineering backgrounds. The selection of the participants was based on available volunteers. All participants had developed software in research or commercial environments. One of the participants (Postdoc) had worked in industry for 6 years and had experience of architecture design. Another participant (PhD student) had 3 years of experience of software development in industry. For

the rest of the participants, they had on average more than 3 years of experience of software development, but mainly in academic environment.

The participants were asked to build a personalized tool support for managing ADDs. Our evaluation assumed that the users of CADDMS would have a clear vision of the ADD model they would use for managing ADDs. Hence, this study did not measure the efforts for coming up with a suitable ADD model. The clear vision of the required model was mimicked by giving the participants a description of an ADD model[1], which was based on the architectural decision description template proposed in [21]. The participants were asked to perform the task using CADDMS and MS Word. We selected MS Word mainly because of two reasons: (1) none of the existing ADD management tools provides similar level of customizability, as described in previous sections; (2) MS Word is the most commonly used productivity tool for drawing templates for architecture documentation in industry. We randomly selected half of the participants to use CADDMS first (group 1) and half of them to use the MS Word first (group 2).

The study was conducted with each participant one by one. Each of the participants was provided with a short (around 15 minutes) introduction to the study and CADDMS. The introductory session was designed to set up a concrete context for the task to be performed (e.g., the participant is a lead architect in a development team, she/he is going to ask the members of her/his team to manage architectural design decisions following the ADD model).

After the introductory session, the participants were asked to perform the task using MS Word as well as CADDMS. The sequence of using each of tools was randomized. The researcher observed the whole process of the participant's activities of performing the task. During this observation, the researcher noted the time taken[2] by each of the participants for performing the assigned task with each of the tools (i.e., MS Word and CADDMS). After finishing the assigned task, each participant was asked five questions about their experience of performing the assigned task. The questions include:

1. Have you learnt how to use CADDMS?
2. Is CADDMS easy to use?
3. Is the tool-generated template satisfactory?
4. Do you have any suggestions to improve CADDMS?
5. Which tool would you prefer to use for managing ADD in your team and why?

The researcher initially took all the notes on a notebook during the study. These notes were further codified to several tables using MS Excel Spreadsheet. The time taken was analyzed using descriptive statistics. We analyzed the content of the notes.

[1] The ADD model used in this user study is available from this link: http://193.1.97.13/wikisac/images/The_ADD_Model_Used_in_the_User_Study.pdf.

[2] Time taken measures the amount of time a user spent to finish the task. The researcher used a stopwatch to measure this time.

TABLE II. SUMMARY OF TIME TAKEN

	Average time taken (in minutes)		
Tools	***Group 1***	***Group 2***	***Overall***
CADDMS	16:36	16:51	16:44
MS Word	20:57	21:23	21:10

Group 1: Use CADDMS first; Group 2: Use MS Word first

As shown in Table 2, on average the participants took 16:44 minutes for CADDMS and 21:10 minutes for MS Word in order to perform the assigned task. It can also be observed that the time taken is not affected by the sequence of the use of the two tools significantly. These results show that the effort required for obtaining a customized tool support for ADDM with CADDMS is often less than for building the template with MS Word. It is also worth mentioning that the participants built the customized tool support for ADDM from scratch, customizing an existing model is expected to take less effort.

All the participants answered "yes" to the first three questions presented above. These findings indicate that CADDMS is easy to learn and the usability of CADDMS is satisfactory. The positive answer to third question indicates that the automatically generated personalized ADD capturing template was satisfactory.

Except one participant, each of the participants stated at least one point of improvement in response to the fourth question. The points of improvements mentioned by the participants include: auto-save the ADD model during the model construction process, display the attribute description in more manners, easily selectable default value of the properties, spellchecker, exporting the captured decisions to MS Word and PDF format, and more explicit operation of creating a new attribute. Most of these improvements were proposed by more than one participant.

All of the participants responded that if he/she were the team lead, he/she would prefer using the CADDMS to manage ADDs. We summarize the reasons of their preferences reported by the participants here: Formatting of the ADD template was automatically done by CADDMS; specifying constraints on attributes with MS Word was hard and the textual description of the constraints could easily be overlooked by a user of a template. It is hard to manipulate, analyze, and perform fine-grained search on the ADDs captured in Word documents. With MS Word, extra efforts are needed for setting up the facilities (e.g., shared folder) for storing the Word documents filled with captured ADDs. A user only needs a browser to access the captured ADDs from anywhere with CADDMS.

The findings from this preliminary evaluation are quite encouraging. The effort required for building a customized tool with CADDMS is often less than building a template with MS Word. The learnability and usability of CADDMS were also affirmed by the positive comments from the participants. Although investigator bias is inevitable with such studies, we tried to reduce its impact by (a) defining a study protocol prior to the study execution, and rigorously following the protocol; (b) blinding the participants from

knowing who the developers of CADDMS are, and encouraging them to give critical comments.

B. Evaluation against the Criteria

Farenhorst et al. proposed seven desired properties of ADDM tools based on empirical study in a large software development organization, and state-of-the-art literature in software architecture [34]. These properties have been used as general criteria for evaluating existing tools by several researchers (e.g. Shahin et al. [35]). We will also use these properties to evaluate CADDMS. In the following paragraphs, we briefly describe how CADDMS addresses the seven desired properties.

Stakeholder-specific content (C1). The features of scope-of-interest based views and role-based views provide specialized views on the available content for different stakeholders. The feature of subscription and notification also enables users to use search strings to specify interested content in a fine-grained manner.

Easy manipulation of content (C2). Users can easily manipulate the ADDs with the web-based interface. No special skills are needed.

Descriptive in nature (C3). CADDMS does not prescribe how architects should manage ADDs. The ADD model is up to architects' customization. Thus CADDMS is descriptive, instead of prescriptive.

Support for codification (C4). Users can codify and store ADDS in the customized repository. The feature of meta-data-enriched document management enables users to store related design artifacts as well.

Support for personalization (C5). The personalization strategy [36] emphasizes the interaction among users. The contributor of any ADD entry will be automatically recorded by the system and displayed to the users. The feature of sharing expertise and experience enables users to find each other and communicate personally. The feature of ADD based discussion enables users to discuss asynchronously.

Support for collaboration (C6). CADDMS allows stakeholders to collaborate with each other. The feature of notification and subscription helps increase the stakeholders' awareness of each other's work. The features of ADD based discussion and sharing expertise and experience also facilitate collaboration.

Sticky in nature (C7). Being sticky is important for a tool to keep users using it [34, 37]. The intuitive user interface, the features of notification and subscription, ADD based discussion, and sharing expertise and experience are helpful for increasing the level of stickiness of CADDMS. The fitness of the ADD model to the specific needs achieved by customization can also help lower the unwillingness of using the tool.

The results, as summarized in Table 3, of our analysis of CADDMS with respect to the given criteria suggest that all the desired properties are addressed by CADDMS. To reduce investigator bias, a second researcher checked the analysis independently. However, further field studies are needed to evaluate its effectiveness in the real industrial settings.

TABLE III. SUMMARY OF EVALUATION AGAINST THE CRITERIA

Criteria	C1	C2	C3	C4	C5	C6	C7
Results	√	√	√	√	√	√	√

VI. FEW SUGGESTIONS ON CUSTOMIZATION

When using the ADD model-centered customizable solution, one key decision is to decide the scope within which a consistent ADD model should be used. We call this scope 'ADD model enforcement scope'. Using a consistent ADD model in a certain scope is important. Because, if different teams in the same project use different ADD models for capturing ADDs, the communication between them and reuse of the ADDs are likely to be negatively affected. The scope of an ADD model can be as small as an individual, and as large as the whole software development community. The decision about the scope is a trade-off point. If too small, there can be too many customized ADD models, which may be incompatibles. If too large, the specific needs of different stakeholders may not be appropriately accommodated. Hence, a decision on the scope of an ADD model should optimize the potential benefits of customization with least amount of effort required for resolving incompatibilities. A complete guideline on defining an ADD model scope is not the objective of this paper. However, we suggest that the following two factors be considered:

- Have a consistent ADD model within a set of stakeholders who communicate with each other frequently.
- Share the same ADD model in similar working situations.

For deciding the information items to be included in an ADD model, the value-based ADD documentation approach proposed by Falessi et al. [12] can be useful.

VII. SUMMARY AND FUTURE WORK

Several tools have been developed for managing ADDs. However, most of them do not provide sufficient flexibility and adaptability to cater users' specific ADDM needs. Hence, the fitness and usability of such tools are limited, which may hinder their successful industrial adoption. In this paper, we have reported our attempt to fill this gap by an ADD model-centered customizable ADDM solution that can enable users to get a personalized ADDM tool to meet their specific needs raised by their unique working situations. We have also performed preliminary assessments of the proposed solution and the results are encouraging.

Our future work focuses on further evaluation of CADDMS. The current evaluation is very preliminary. It only involved 8 participants. It is hardly to draw a definitive conclusion from such a small study. We plan to run the user study with a larger sample of more relevant participants. We also plan to trial the approach in an industrial setting.

In addition, we plan to extend the system to a service-centric ADD sharing infrastructure, within which the tool support of ADDM is delivered as a service over the Internet. The envisioned infrastructure will not only facilitate the

sharing of ADD instances, but also the sharing of ADDM services (i.e. sharing the way people employed to manage ADD) in a social network environment.

ACKNOWLEDGMENT

The work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1. The work was inspired and partially based on a research project in IBM China Research Lab, where the first author worked on the project for 9 months during his internship. We sincerely thank the participants of our empirical studies. We would also like to thank the anonymous reviewers for their constructive comments on improving this paper.

REFERENCES

[1] J. Bosch, "Software Architecture: The Next Step," in Software Architecture: Springer, 2004, pp. 194-199.

[2] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," in 5th Working IEEE/IFIP Conference on Software Architecture, 2005, pp. 109-120.

[3] M. A. Babar and P. Lago, Design decisions and design rationale in software architecture, Journal of Systems and Software, vol. 82, pp. 1195-1197, 2009.

[4] M. Shaw and P. Clements, The Golden Age of Software Architecture, IEEE Softw., vol. 23, pp. 31-39, 2006.

[5] M. A. Babar and I. Gorton, "A Tool for Managing Software Architecture Knowledge," in 2nd Workshop on Sharing and Reusing Architectural Knowledge, 2007, pp. 11-11.

[6] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, A web-based tool for managing architectural design decisions, SIGSOFT Softw. Eng. Notes, vol. 31, p. 4, 2006.

[7] L. Lee and P. Kruchten, "Customizing the capture of software architectural design decisions," in Canadian Conference on Electrical and Computer Engineering, 2008, pp. 93-98.

[8] N. Schuster, O. Zimmermann, and C. Pautasso, "ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering," in International Conference on Software Engineering & Knowledge Engineering Boston, Massachusetts, USA: Knowledge Systems Institute Graduate School 2007, pp. 255-260.

[9] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, Managing architectural decision models with dependency relations, integrity constraints, and production rules, Journal of Systems and Software, vol. 82, pp. 1249-1267, 2009.

[10] R. FARENHORST, P. LAGO, and H. V. VLIET, EAGLE: EFFECTIVE TOOL SUPPORT FOR SHARING ARCHITECTURAL KNOWLEDGE, International Journal of Cooperative Information Systems vol. 16, pp. 413-437, 2007.

[11] R. Farenhorst, R. Izaks, P. Lago, and H. van Vliet, "A Just-In-Time Architectural Knowledge Sharing Portal," in 7th Working IEEE/IFIP Conference on Software Architecture, 2008, pp. 125-134.

[12] D. Falessi, G. Cantone, and P. Kruchten, "Value-Based Design Decision Rationale Documentation: Principles and Empirical Feasibility Study," in 7th Working IEEE/IFIP Conference on Software Architecture, 2008, pp. 189-198.

[13] L. Chen and M. A. Babar, Architectural Design Decision Model Needs Customization, Lero at UL 2009. http://193.1.97.13/wikisac/images/Tr_add.pdf

[14] S. L. Kogan and M. J. Muller, Ethnographic study of collaborative knowledge work, IBM Syst. J., vol. 45, pp. 759-771, 2006.

[15] A. Akerman and J. Tyree, Using ontology to support development of software architectures, IBM Syst. J., vol. 45, pp. 813-825, 2006.

[16] A. Agostini, G. d. Michelis, M. A. Grasso, and S. Patriarca, "Reengineering a business process with an innovative workflow management system: a case study," in Proceedings of the conference on Organizational computing systems Milpitas, California, United States: ACM, 1993.

[17] M.-M. Raul, W. Terry, F. Rodrigo, and F. Fernando, "The action workflow approach to workflow management technology," in Proceedings of the 1992 ACM conference on Computer-supported cooperative work Toronto, Ontario, Canada: ACM, 1992.

[18] T. Winograd and F. Flores, Understanding Computers and Cognition: A New Foundation for Design: Addison Wesley, 1987.

[19] M. Hammer and J. Champy, Reengineering the corporation: A manifesto for business revolution: HarperBusiness, 1994.

[20] E. R. Poort, A. Pramono, M. Perdeck, H. Viktor Clerc, and H. van Vliet, "Successful Architectural Knowledge Sharing: Beware of Emotions," in Software Architecture Knowledge Management: Theory and Practice, M. Ali-Babar, T. Dingsoyr, P. Lago, and H. v. Vliet, Eds.: Springer, To be Published in Summary 2009, pp. 247-265.

[21] J. Tyree and A. Akerman, Architecture decisions: demystifying architecture, Software, IEEE, vol. 22, pp. 19-27, 2005.

[22] F. D. Davis, PERCEIVED USEFULNESS, PERCEIVED EASE OF USE, AND USER ACCEPTANCE OF INFORMATION TECHNOLOGY, Mis Quarterly, vol. 13, pp. 319-340, Sep 1989.

[23] V. Venkatesh and F. D. Davis, A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies, Manage. Sci., vol. 46, pp. 186-204, 2000.

[24] P. Lago, "Establishing and Managing Knowledge Sharing Networks," in Software Architecture Knowledge Management, 2009, pp. 113-131.

[25] D. M. Thomas, R. P. Bostrom, and M. Gouge, Making knowledge work in virtual teams, Commun. ACM, vol. 50, pp. 85-90, 2007.

[26] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrik, "An Architectural Framework," in Rationale-Based Software Engineering: Springer Berlin Heidelberg, 2008, pp. 241-254.

[27] R. Capilla, F. Nava, and J. C. Dueñas, "Modeling and Documenting the Evolution of Architectural Design Decisions," in SHARK/ADI: Workshop on Sharing and Reusing Architectural Knowledge / Architecture, Rationale, and Design Intent, 2007.

[28] Hipergate, An open source CRM and Groupware system, 2007.

[29] P. Kruchten, "An Ontology of Architectural Design Decisions in Software-Intensive Systems " in 2nd Groningen Workshop on Software Variability, 2004.

[30] N. Schuster and O. Zimmermann, Architectural Decision Knowledge Wiki, http://www.alphaworks.ibm.com/tech/adkwik

[31] R. Farenhorst and H. v. Vliet, "Experiences with a Wiki to Support Architectural Knowledge Sharing," in Wiki4SE at WikiSym'08, Porto, Portugal, 2008.

[32] P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning About Architectural Knowledge," in Quality of Software Architectures, 2006, pp. 43-58.

[33] "HTML 4.01 Specification," in W3C Recommendation, D. Raggett, A. L. Hors, and I. Jacobs, Eds., 1999.

[34] R. Farenhorst, P. Lago, and H. van Vliet, "Effective Tool Support for Architectural Knowledge Sharing," in Software Architecture, 2007, pp. 123-138.

[35] M. Shahin, P. Liang, and M. R. Khayyambashi, "Architectural Design Decision: Existing Models and Tools," in the Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA), 2009. Accepted.

[36] M. T. Hansen, N. Nohria, and T. Tierney, What's Your Strategy for Managing Knowledge, Harvard Business Review, vol. 72, pp. 106-116, 1999.

[37] A. A. Bush and A. Tiwana, Designing sticky knowledge networks, Commun. ACM, vol. 48, pp. 66-71, 2005.