Preparing Usability Supporting Architectural Patterns for Industrial Use

Pia Stoll
ABB Corporate Research
Forskargränd 6
SE 72178 Västerås, Sweden
Tel: +46 21 32 30 00
pia.stoll@se.abb.com

Bonnie E. John, Len Bass, Elspeth Golden
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA, USA 15213
Tel: 1+412 268 2000
bej@cs.cmu.edu, ljb@sei.cmu.edu, egolden@cmu.edu

ABSTRACT

Usability supporting architectural patterns (USAPs) have been shown to provide developers with useful guidance for producing a software architecture design that supports usability in a laboratory setting [7]. In close collaboration between researchers and software developers in the real world, the concepts were proven useful [2]. However, this process does not scale to industrial development efforts. In particular, development teams need to be able to use USAPs while being distributed world-wide. USAPs also must support legacy or already partially-designed architectures, and when using multiple USAPs there could be a potentially overwhelming amount of information given to the software architects. In this paper, we describe the restructuring of USAPs using a pattern language to simplify the development and use of multiple USAPs. We also describe a delivery mechanism that is suitable for industrial-scale adoption of USAPs. The delivery mechanism involves organizing responsibilities into a hierarchy, utilizing a checklist to ensure responsibilities have been considered, and grouping responsibilities in a fashion that both supports use of multiple USAPs simultaneously and also points out reuse possibilities to the architect.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**] User interfaces; D.2.11 [**Software Architectures**]: Patterns; H.5.2 [**User Interfaces**] Theory and Methods

General Terms

Design, Human Factors.

Keywords

Software Architecture, Usability, Human-Computer Interaction, HCI

I-USED'08, September 24, 2008, Pisa, Italy

1. INTRODUCTION

The Software Engineering community has recognized that usability affects not only the design of user interfaces but software system development as a whole. In particular, efforts are focused on explaining the implications of usability on software architecture design [3, 4, 5, 6, 10].

One effort in this area is to produce artifacts that communicate usability requirements in a form that can be effectively used for software architecture evaluation and design. These *usability supporting architectural patterns* (USAPS) have been shown to improve the ability of software architects to design higher quality architectures to support usability features such as the ability to cancel a long-running command [7, 8]. Other uses of USAPs in industrial settings have been productive [2] but have revealed some problems that prevent scaling USAPs to widespread industrial use. These problems include:

1. Prior efforts have involved personal contact with USAP researchers, either face to face or in telephone conversations. This process does not scale to widespread industrial use.

2. The original USAPs included UML diagrams modifying the MVC architectural pattern to better support the usability concern. Although intended to be illustrative, not prescriptive, software architects using other overarching patterns (e.g., legacy systems, SOA) viewed these UML diagrams as either unrelated to their work or as an unwanted recommendation to totally redesign their architecture.

3. The original use of USAPs was as a collection of individual patterns. Even using one pattern involved processing a large amount of detailed information. Applying multiple USAPs simultaneously is likely to overwhelm software architects with information.

In this paper, we introduce a pattern language [1] for USAPs that reduces the information that architects must absorb and produces information at a level that applies to all architectures. We also discuss the design of a delivery mechanism suitable for industrial-scale adoption of USAPs.

2. BACKGROUND

A USAP has six types of information. We illustrate the types with information from the cancellation USAP [9]

1. A brief scenario that describes the situation that the USAP is intended to solve. For example, "The user issues a command

then changes his or her mind, wanting to stop the operation and return the software to its pre-operation state."

2. A description of the conditions under which the USAP is relevant. For example, "A user is working in a system where the software has long-running commands, i.e., more than one second."
3. A characterization of the user benefits from implementing the USAP. For example, "Cancel reduces the impact of routine user errors (slips) by allowing users to revoke accidental commands and return to their task faster than waiting for the erroneous command to complete."
4. A description of the forces that impact the solution. For example, "No one can predict when the users will want to cancel commands"
5. An implementation-independent description of the solution, i.e., responsibilities of the software. For example, one implication of the force given above is the responsibility that "The software must always listen for the cancel command."
6. A sample solution using UML diagrams. These diagrams were intended to be illustrative, not prescriptive, and are, by necessity, in terms of an overarching architectural pattern (e.g., MVC).

It is useful to distinguish USAPs from other patterns for software design and implementation. USAPs are not user interface patterns, that is, they do not represent an organization or look-and-feel of a user interface [e.g., 11]; they are software architecture patterns that support UI concerns. Nor are USAPs structural software architecture patterns like MVC; they are patterns of software responsibilities that can be applied to any structure. As such, they can be applied to any legacy architecture and can support the functionality called for in UI patterns.

3. A PATTERN LANGUAGE FOR USAPs

Through collaboration among academic and industrial researchers and usability engineers, we are constructing three USAPs for process control and robotics applications. The first author and her colleagues in the industry research team drafted an "Alarms, Events and Alerts" USAP while, independently, the last three authors drafted a "User Profile" USAP and an "Environment Configuration" USAP. When these three USAPs were considered together, it was clear that there was an enormous amount of redundancy in the responsibilities necessary for a good solution. In addition, in preliminary discussions, industry software architects reacted negatively to the large amount of information required to implement any one of the USAPs.

Our early work [4] recognized the possibility of reusing such software tactics as separating authoring from execution and recording (logging), but our subsequent work had not incorporated that notion, treating each USAP as a separate pattern. A consequence of focusing on industrial use is that reuse in constructing and using USAPs was no longer an academic thought experiment, but a necessity if industrial users are to construct and use USAPs themselves.

We observed that both the industry research team and the academic research team independently grouped their responsibilities into very similar categories. This led us to construct a pattern language [2] that defines relationships between USAPs with potentially reusable sets of responsibilities that can lead to potentially reusable code. Our pattern language relating "Alarms, Events and Alerts", "User Profile" and "Environment Configuration" is shown in Figure 1.

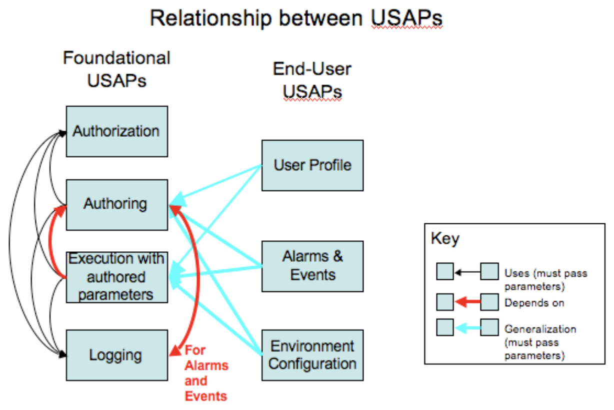

Relationship between USAPs
Foundational USAPs
End-User USAPs
Authorization
Authoring
Execution with authored parameters
Logging
User Profile
Alarms & Events
Environment Configuration
For Alarms and Events
Key
Uses (must pass parameters)
Depends on
Generalization (must pass parameters)

Figure 1 USAP Pattern Language for "User Profile", "Alarms, Events and Alerts", and "Environment Configuration"

The pattern language has two types of USAPs. "End-User USAPs" follow the structure given in Section 2. Their purpose from a user's point of view can be expressed in a small scenario, they have conditions under which they are relevant, benefits for the user can be expressed and they require the fulfillment of software responsibilities in the architecture design. End-User USAPs are used by the requirements team to determine which are applicable to the system being developed. In this example, they are "User Profile", "Alarms, Events and Alerts", and "Environment Configuration".

The pattern language also contains what we are calling "Foundational USAPs". These do not have the same six portions as the End-User USAPS. For example, there is no scenario, no description of conditions, and no benefits to the user for the Foundational USAPs. Rather, they act as a framework to support the construction of the End-User USAPs that make direct contact to user scenarios and usability benefits. For example, all of the End-User USAPs that we present have an authoring portion and an execution portion, that is, they are specializations of the Authoring Foundational USAP and the Execution with Authored Parameters Foundational USAP. These foundational USAPs make use of other foundational USAPs, Authorization and Logging. We abstracted the commonalities of the End-User USAPs to derive the responsibilities of the Foundational USAPs. The responsibilities in the Foundational USAPs are parameterized, where the parameters reflect those aspects of the End-User USAPs that differ.

An example of the parameterization is that the Authoring Foundational USAP and the Execution with Authored Parameters Foundational USAP each have a parameter called SPECIFICATION. The value of SPECIFICATION is "Conditions for Alarm, Event and Alerts", "User profile", and "Configuration description" for the three End-User USAPs, respectively.

In addition to parameterization, End-User USAPs explicitly list assumptions about decisions the development team must make prior to implementing the responsibilities. For example, in the "Alarms, Events and Alerts" End-User USAP, the development team must define the syntax and semantics for the conditions that will trigger alarms, events or alerts. End-User USAPs may also have additional responsibilities beyond those of the Foundational USAPs they use. For example, the "Alarms, Events and Alerts" End-User USAP has an additional responsibility that the system must have the ability to translate the names/ids of externally generated signals (e.g., from a sensor) into the defined concepts. Both the assumptions and additional responsibilities will differ for the different End-User USAPs.

There are three types of relationships among the Foundational USAPs and these are shown in Figure 1 as different color arrows. The Generalization relationship (turquoise) says that the Foundational USAP is a generalization of part of the End-User USAP. The End-User USAP passes parameters to that Foundational USAP and, if there are any conditionals in the responsibilities of the Foundational USAP, the End-User USAP may define the values of those conditionals. The Uses relationship (black) also passes parameters, but the USAPs are at the same level of abstraction (the foundational level). The Depends-On relationship (red) implies a temporal relationship. For example, the system cannot execute with authored parameters unless those parameters have first been authored. The double headed arrow between authoring and logging reflects the possibility that the items being logged may be authored and the possibility that the identity of the author of some items may be logged.

Foundational USAPs each have a manageable set of responsibilities (Authorization has 11; Authoring, 12; Execution with authored parameters, 9; and Logging 5), as opposed to the 21 responsibilities of the Cancel USAP that seemed to be too much for our experiment participants to absorb in one sitting [7]. These responsibilities are further divided into groups for ease of understanding, e.g., Authoring is separated into Create, Save, Modify, Delete and Exit the authoring system. This division into manageable Foundational USAPs simplifies the creation of future USAPs that use them. For example, the User Profile End-User USAP requires only the definition of parameters and the values for one conditional, and pointers to the Authoring and Execution Foundational USAPs.

4. DELIVERING A SINGLE USAP TO SOFTWARE ARCHITECTS

The roadblocks to widespread use of USAPs in industry identified in the introduction were (1) the need for contact with USAP researchers in the development process, (2) reactions to examples using a particular overarching architectural pattern (MVC) and (3) an overwhelming amount of information delivered to the software architect. Data from our laboratory study and the pattern language outlined above put us in a position to solve these problems.

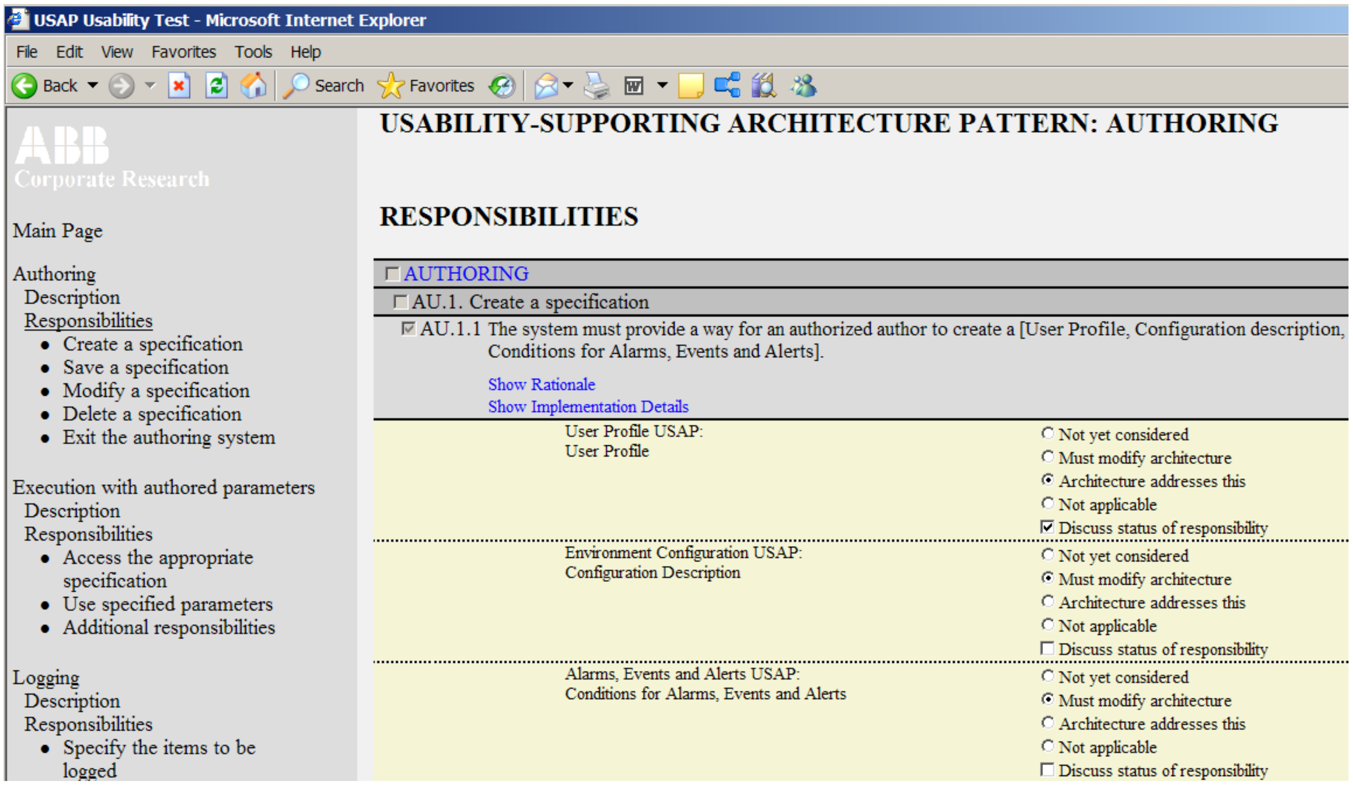
Our laboratory study [7] showed that a paper-based USAP could be used by software engineers[1] without researcher intervention, to significantly improve their design of an architecture to support the users' need to cancel long-running commands. Although significantly better than without a USAP, these software engineers seemed to disregard many of the responsibilities listed in the USAP in their designs. To enhance attention to all responsibilities, we have chosen to design a web-based system that presents responsibilities in an interactive checklist (Figure 2). The design includes a set of radio buttons for each responsibility that are initially set to "Not yet considered." The architect reads each responsibility and determines whether it is not applicable to the system being designed, already accounted for in the architecture, or that the architecture must be modified to fulfill the responsibility. If "Not applicable", "Must modify architecture to address this" or "Architecture addresses this" is selected, then the responsibility's checkbox is automatically checked. If "Not considered", "Must modify architecture or "Discuss status of responsibility", is selected, the responsibility will be recorded in To-Do list generated from the website (Figure 3). We expect this lightweight reminder to consider each and every responsibility will not be too much of a burden for the architect, but will increase the coverage of responsibilities, which is correlated with the quality of the architecture solution [8].

As Figure 2 show, the responsibilities are arranged in a hierarchy, which reflects both the relationship of End-User and Foundational USAPs and the internal structure within a Foundational USAP. This hierarchy divides the responsibilities into manageable sub-parts. The checkboxes enforce this structure by automatically checking off a higher-level box when all its children have been checked off, and conversely, not allowing a higher-level box to be checked when one or more of its children are not. Thus, this mechanism simultaneously addresses the problems of providing guidance without intervention by USAP researchers and simplifying the information provided to the software architect.

Another mechanism for simplifying the information delivered to an architect is that each responsibility has additional details available only by request of the architect. These details include more explanation, rationale about the need for the responsibility and the forces that generated it, and some implementation details. This information is easily available, but not "in the face" of the software architect. As well as simplifying the presentation, this mechanism de-emphasizes the role of illustrative examples situated in reference architecture like MVC. We expect that this presentation decision will reduce the negative reactions to generic example UML diagrams. When using the tool in-house in industry, the reference architecture used in example solutions could be changed to an architecture used by that industry. This would both accelerate understanding of the examples and increase the possibility of re-using the sample solution. This presumes that the tool is constantly managed and updated by in-house usability experts and software architects, a presumption facilitated by delivering the examples in separate web pages.

Although the hierarchy of responsibilities reflects the relationship of the End-User USAPs and the Foundational USAPs, the difference between the types of USAPs is not evident in the presentation of responsibilities. It was a deliberate design choice to express each responsibility in terms of the End-User USAP's vocabulary. Thus, the responsibilities in Figure 2 are couched in terms of "User Profile", "Configuration Description", "Conditions for Alarms, Events, and Alerts" and this string replaces the parameter SPECIFICATION in the Foundational Authoring USAP.

[1] The participants in our lab study had a Masters in SE or IT, were trained in software architecture design, and had an average of over 21 months in industry.


USAP Usability Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
ABB
Corporate Research

Main Page

Authoring
- Description
- Responsibilities
  - Create a specification
  - Save a specification
  - Modify a specification
  - Delete a specification
  - Exit the authoring system

Execution with authored parameters
- Description
- Responsibilities
  - Access the appropriate specification
  - Use specified parameters
  - Additional responsibilities

Logging
- Description
- Responsibilities
  - Specify the items to be logged

USABILITY-SUPPORTING ARCHITECTURE PATTERN: AUTHORING

RESPONSIBILITIES

☐ AUTHORING

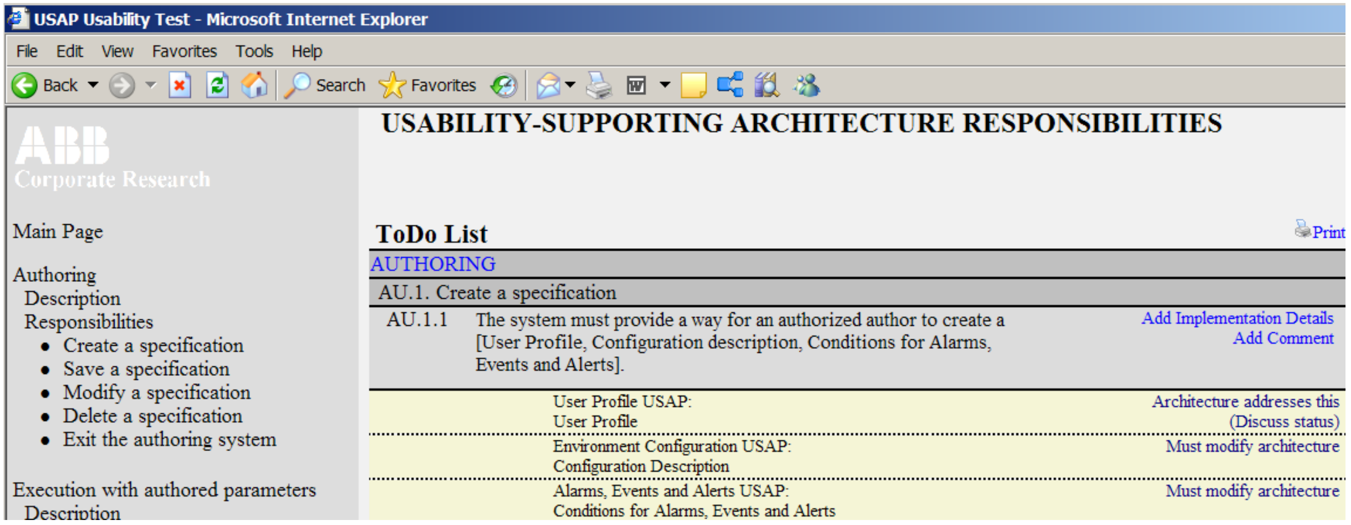
☐ AU.1. Create a specification

☑ AU.1.1 The system must provide a way for an authorized author to create a [User Profile, Configuration description, Conditions for Alarms, Events and Alerts].

Show Rationale
Show Implementation Details

User Profile USAP: User Profile	Not yet considered [✓ Architecture addresses this] Must modify architecture Not applicable [✓ Discuss status of responsibility]
Environment Configuration USAP: Configuration Description	Not yet considered [✓ Must modify architecture] Architecture addresses this Not applicable ☐ Discuss status of responsibility
Alarms, Events and Alerts USAP: Conditions for Alarms, Events and Alerts	Not yet considered [✓ Must modify architecture] Architecture addresses this Not applicable ☐ Discuss status of responsibility

Figure 2: Prototype of a web-based interface for delivering USAP responsibilities to industry software architects.


USAP Usability Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
ABB
Corporate Research

Main Page

Authoring
- Description
- Responsibilities
  - Create a specification
  - Save a specification
  - Modify a specification
  - Delete a specification
  - Exit the authoring system

Execution with authored parameters
- Description

USABILITY-SUPPORTING ARCHITECTURE RESPONSIBILITIES

ToDo List — Print

AUTHORING

AU.1. Create a specification

AU.1.1 The system must provide a way for an authorized author to create a [User Profile, Configuration description, Conditions for Alarms, Events and Alerts]. — Add Implementation Details / Add Comment

User Profile USAP: User Profile	Architecture addresses this (Discuss status)
Environment Configuration USAP: Configuration Description	Must modify architecture
Alarms, Events and Alerts USAP: Conditions for Alarms, Events and Alerts	Must modify architecture

Figure 3: Prototype “to do” list produced from those responsibilities that are marked as requiring architectural modification.

In the next section, we discuss how we anticipate managing the situation when the architect chooses multiple USAPs as being relevant to the system under construction. This will allow distribute architecture teams both to record rationale for their choice and to discuss potential solutions. Attaching design rationale and discussion is optional so our delivery tool will support discussion, but not require it, keeping the tool lightweight.

At any point in the process of considering the different responsibilities, the architect can generate a "to do" list. This is a list of all of the responsibilities that have been checked as "Not yet considered" or "Must modify architecture". See Figure 3 for an example. The list can then be entered into the architect's normal task list and will be considered as other tasks are considered.

Supporting world wide distribution of the architecture team in the use of USAPs has two facets.

- Enable world wide access
- Reduce the problems associated with simultaneous updates by different members of the team.

The use of the World Wide Web for delivery allows world wide access with appropriate access control. Standard browsers support the concept of check lists and producing the "to do" lists.

Allowing simultaneous updates is not supported by standard browsers. Some Wikis do support simultaneous updates, e.g. MediaWiki [www.mediawiki.org], but we do not yet know whether these wikis directly support checklists and the generation of "to do" lists. We are currently investigating which tool or combination of tools will be adequate for our needs and what modifications might have to be made to those tools.

5. DELIVERING MULTIPLE USAPS TO SOFTWARE ARCHITECTS

Our motivation for developing the USAP Pattern Language was partially to simplify the delivery of USAPs when multiple USAPs are relevant to a particular system. We also want to indicate to the architect the possibilities for reuse. In this section, we describe how we anticipate accomplishing these two goals.

Recall that the Foundational USAPs are parameterized and each End User USAP provides a string that is used to replace the parameter. For instance, consider a responsibility from the Authoring Foundational USAP "The system must provide a way for an authorized user to create a SPECIFICATION". When three End User USAPs are relevant to the system under design, such as "User Profile", "Environment Configuration", and "Alarms, Events and Alerts", the three responsibilities are displayed to the architect as "The system must provide a way for an authorized user to create a [User Profile, Configuration description, Conditions for Alarm, Event and Alerts].

This presentation satisfies two goals and introduces one problem. Presenting three responsibilities as one reduces the amount of information displayed to the architect since every Foundational USAP responsibility is displayed only once, albeit with multiple pieces of information. This presentation also indicates to the architect the similarity of these three responsibilities and hence the reuse possibilities of fulfilling them through a single piece of parameterized code.

The problem introduced by this form of the presentation is that now the radio buttons becomes ambiguous. Does the entry "Architecture addresses this" mean that all of the three responsibilities have been addressed or only some of them? We resolve this ambiguity by repeating the radio buttons three times, once for each occurrence of the responsibility. Thus, the three responsibilities will be combined into one textual description of the responsibility but three occurrences of the radio buttons.

6. CURRENT STATUS AND FUTURE WORK

At this writing, we have developed the pattern language for three End User USAPs and four Foundational USAPs (Figure 1) and have fleshed out all the responsibilities for these seven USAPs. We have constructed a prototype delivery tools for a browser based checklist and "to do" list generator.

We plan to test the delivery mechanism in an ongoing industrial development effort. This will demonstrate strengths and weaknesses of our approach and we will iterate to resolve any problems or capitalize on any opportunities. One suggestion put forth in early industry feedback is to enhance the to-do list by assigning expected effort to each responsibility. One requirements engineer at ABB said that her perception of the effort needed to implement a scenario had been thoroughly revised just be looking at the to-do list. By adding estimated hours to the responsibilities, industry would get a better estimate of the usability improvements' translation into software implementation cost. These estimates would vary depending on many factors such as underlying architectural style, implementation language, skill of programmers, etc. but a large organization may have enough data from previous projects to make such estimates for their organization. In addition, such a feature could emphasize the savings realized by reuse; responsibility-implementations that serve multiple End-User USAPs would show up as requiring very little effort after the first implementation.

The delivery platform that we have described here, to be used by software architects, is envisioned to be the final portion of a tool chain. There are two additional roles involved in the development and use of USAPs. First, USAP developers will have to create USAPs within the stylized context of the USAP Pattern Language. Tool support for USAP developers will greatly simplify the creation of USAPs.

The second role is the requirements definers; often a team comprised of technologists and human factors engineers, usability engineers, designers, or other users or user advocates. Figure 4 shows how we envision a tool supporting this role.

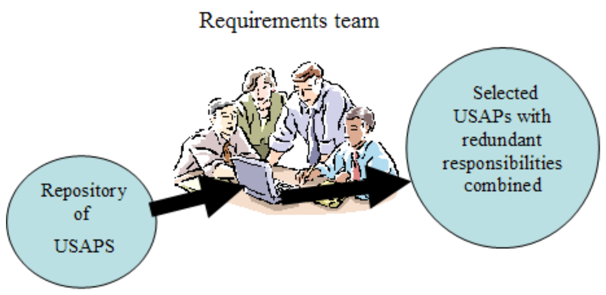


Figure 4: Tool to support the requirements elicitation process

The requirements team has available to them a repository of USAPs. They select the ones that are appropriate for the system being constructed. In our experience, the USAP end-user scenarios are very general and can be used to invoke ideas about how they apply to the system at hand. However, industrial teams would like to tailor these scenarios to match their everyday usability issues. Thus, the tool supporting requirements definers will allow them to re-write the general scenarios to suit their specific application.

The tool then creates input for the delivery tool while simultaneously combining redundant responsibilities. The output of the requirements definition process will then be presented to software architects, as described in this paper, to aid in their architecture design process.

In summary, USAPs have been proven to be useful to software architects but have also demonstrated some problems that hinder industrial use. Definition of a USAP Pattern Language and an appropriate selection of tools supporting the roles involved in the creation and use of USAPs should simplify industrial use. We are currently constructing versions of these tools and testing the extent to which they do, in fact, enable the industrial use of USAPs.

7. ACKNOWLEDGMENTS

We would like to thank Fredrik Alfredsson and Sara Lövemark for their contributions to the “Alarms, Events and Alerts” USAP. This work was supported in part by funds from ABB Inc. The views and conclusions in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of ABB.

8. REFERENCES

[1] Alexander, C. (1977). *A Pattern Language: Towns, Buildings, Construction*. USA: Oxford University Press. ISBN 978-0195019193.

[2] Adams, R. J., Bass, L., & John, B. E. (2005) Applying general usability scenarios to the design of the software architecture of a collaborative workspace. In A. Seffah, J. Gulliksen and M. Desmarais (Eds.) *Human-Centered Software Engineering: Frameworks for HCI/HCD and Software Engineering Integration*. Kluwer Academic Publishers.

[3] Bass, L., John, B., & J. Kates, (2001), “Achieving Usability through Software Architecture,” Technical Report CMU/SEI-2001-TR-005, Software Eng. Inst., Carnegie Mellon Univ., 2001.

[4] Bass, L., & John, B. (2003) “Linking Usability to Software Architecture Patterns through General Scenarios,” *The J. Systems and Software*, vol. 66, no. 3, pp. 187-197, 2003.

[5] Folmer, E. (2005) *Software Architecture Analysis of Usability*, Ph.D. thesis. Department of Computer Science, University of Groningen, Groningen.

[6] Folmer, E., van Gurp, J., Bosch, J. (2003) A Framework for capturing the relationship between usability and software architecture; *Software Process: Improvement and Practice*, Volume 8, Issue 2. Pages 67-87. April/June 2003.

[7] Golden, E, John, B. E., & Bass, L. (2005) The value of a usability-supporting architectural pattern in software architecture design: A controlled experiment. *Proceedings of the 27th International Conference on Software Engineering, ICSE 2005* (St. Louis, Missouri, May, 2005).

[8] Golden, E., John, B. E., Bass, L. (2005) Quality vs. quantity: Comparing evaluation methods in a usability-focused software architecture modification task. *Proceedings of the 4th International Symposium on Empirical Software Engineering* (Noosa Heads, Australia, November 17-18 2005).

[9] John, B. E., Bass, L. J., Sanchez-Segura, M-I. & Adams, R. J. (2004) Bringing usability concerns to the design of software architecture. *Proceedings of The 9th IFIP Working Conference on Engineering for Human-Computer Interaction and the 11th International Workshop on Design, Specification and Verification of Interactive Systems*, (Hamburg, Germany, July 11-13, 2004).

[10] Juristo, N., Moreno, A. M., Sanchez-Segura, M. (2007), “Guidelines for Eliciting Usability Functionalities”, *IEEE Transactions on Software Engineering, Vol. 33*, No. 11, November 2007, pp. 744-758.

[11] Tidwell, J. (2006), *Designing Interfaces: Patterns for Effective Interaction Design*. O’Reilly Media: Sebastopol, CA.