Achieving Usability Through Software Architectural Styles

Len Bass and Bonnie E. John
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pa 14213
{ljb,bej}@sei.cmu.edu

ABSTRACT
Design decisions at the architecture level can have far-reaching effects on the qualities of a computer system. Recent developments in software engineering link architectural styles to quality attribute analysis techniques to predict the effects of architectural design decisions on the eventual manifestation of quality. An Attribute-Based Architecture Style (ABAS) is a structured description of a particular software quality attribute, a particular architectural style, and the relevant qualitative and quantitative analysis techniques. Thus, it is a description that is meaningful to software engineers as they design or analyze proposed software architectures. We are producing a collection of ABASs that speak to the *usability* quality attribute. These ABASs will enable software engineers make early architectural design decisions that achieve specific usability functions.

Keywords
Software design, architectural styles, usability evaluation

INTRODUCTION
Despite almost two decades of research exploring usability and usability design methods, barely usable computer systems are still appearing on our desktops and elsewhere in our lives. The CHI community advocates design processes that bring usability considerations to early design decisions, but most of these processes stop short of explicitly mapping usability quality to specific software design patterns. This leaves the mapping to a software engineer who may not know much about usability, to a usability specialist who may not know much about software engineering, or to a multi-disciplinary team whose members have difficulty communicating. In other words, the mapping may often remain incomplete, or implicit in architectural decisions and thereby unexamined. We propose to explicitly map usability considerations to design constructs in language familiar to software engineers and, thus, bridge this communication gap.

Recent developments in software engineering explicitly link architectural styles [5] to software quality attributes like performance, reliability, and modifiability. The link is established through analysis techniques that predict the effects of architectural design decisions on the eventual manifestation of quality. An Attribute-Based Architectural Style (ABAS) is a structured description of a measurable quality attribute, a particular architectural style, and the relevant qualitative and quantitative analysis techniques [3]. Some aspects of usability fit well into this framework. Therefore, we are producing a collection of ABASs that address these usability aspects of computer systems.

Usability ABASs can fill several rolls in an analysis or design process. They provide an enumeration of specific usability features, serving as a checklist for consideration by a design team. They present implementation solutions for those features. Finally, they include methods for performing a cost/benefit analysis for particular usability features. Thus, an ABAS speaks to software engineers about measurable aspects of usability in terms that allow the information to be inserted into the architectural design process. In the remainder of this paper, we propose some aspects of usability that are candidates for ABASs and summarize one particular ABAS.

CHARACTERIZATION OF THE USABILITY QUALITY ATTRIBUTE
To generate ABASs, we must first characterize the usability quality attribute into *stimuli* (typically what users want to do) and *responses* (measurable behavior of a computer system). To do this, we are distilling the definitions and characteristics of usability presented by many authors over the past two decades. From Nielsen's heuristics (e.g. provide "clearly marked exits") [4] to the properties enumerated by IFIP's Working Group 2.7 [2], these sources have suggested many stimuli and responses that are specific enough to be written as operational requirements for software engineers and are likely to have architectural-level solutions. Thus, we are not proposing new aspects of usability, rather, we are compiling the collective wisdom of the field, choosing those usability aspects with architectural implications, and putting them in the language of operational requirements.

This work supported by the U.S. Department of Defense

For example, we have identified several types of users who should be considered when architectural decisions are being made, including end-users, system maintainers, and developers. Each user type may have similar needs (e.g., to be able to undo actions) and each may have some special needs. Some stimuli we have identified are: users will sometimes want to cancel their last operation at some point prior to the operation's completion, users will sometimes want to undo prior operations, and users will need to do repetitive commands on groups of items. Typical responses include: the ability to fulfill these user needs, the time to provide feedback to user about the status of their commands, and the accuracy and salience of feedback.

SUMMARY OF A USABILITY ABAS

The *Cancel ABAS* is used to reason about whether a proposed software architecture will facilitate users being able to cancel their last operation. The ABAS itself is over eight pages long, so we have only space here to capture the main ideas. There are four parts in an ABAS.

Problem Statement. The problem statement includes a brief statement of the problem, in this case, that a user wants to stop an operation he or she has requested before it is complete. It also includes a description of when it is appropriate to consider this ABAS in architectural decisions. In this case, people will always make mistakes and/or change their mind, so as long as the system includes any operations that take longer than one second to complete, the design team should use the Cancel ABAS in their deliberations.

Stimulus/Response measures. The stimulus is the user issuing the "cancel" command. The response has five measurable elements: 1) the amount of time it takes to acknowledge the "cancel" command, 2) the amount of time to complete the cancellation, 3) the extent to which the system state prior to the issuance of the cancelled command is restored, 4) the accuracy of the feedback to the user about the restored state, and 5) accuracy and salience of feedback to the user on the progress of the cancellation (if the cancellation will take more than one second).

Architecture Style. The architectural style we present in the Cancel ABAS assumes that it is not always possible for the process being cancelled to recognize that a cancel command has been issued. The process may be blocked or in an infinite loop. The style, instead, relies on having a separate cancel process. This process listens for the user to issue the cancel command, cancels the active command and informs collaborating processes of the cancellation so that they can, in turn, take appropriate action. The cancel process is responsible for generating the appropriate feedback to the user. The process being cancelled must save enough of its initial system state so that it can be restored on cancellation. The process being cancelled must also inform the cancellation process of any non-preemptable resources it acquires. These resources must be freed on cancellation.

Analysis. The analysis section describes how to reason about a solution in terms of the five measurable responses. The first two responses involve performance (immediate feedback acknowledging the cancel command and the time to perform the cancellation. The Cancel ABAS points to a performance ABAS for details of quantitative analysis techniques for calculating latency. The remaining three responses can be analyzed in a more qualitative manner, using scenarios. Several different scenarios that provide different aspects of system usage must be examined to verify that the system is returned to its original state, that resources are returned and that collaborating processes are informed, as well as whether accurate and appropriately salient feedback is given to the user.

FUTURE WORK

We are working with other software engineers to develop a handbook of ABASs, including those pertaining to the usability quality attribute. These ABASs draw from a full range of analysis techniques. The example given here used quantitative analyses of performance and qualitative analysis via scenarios, but other ABASs use models such as the Keystroke-Level Model [1] to determine the human-performance benefit of providing a particular function. This can be traded off against the cost to the developing organization of providing that function.

In order for software engineers to include usability features in the systems they develop, they must know what these features are, they must know how to implement the features, and they must know how to perform cost/benefit trade-off analysis to determine the utility of the features. Furthermore, all of this must be explained to them in their language. These are the goals we hope to achieve via usability ABASs. We also need to fit the use of ABASs into processes of architectural design and analysis that mesh with existing software development processes. All these goals are being addressed in the Architectural Tradeoff Analysis Initiative at the Software engineering Institute (URL: http://www.sei.cmu.edu/ata/ata_init.html).

REFERENCES

1. Card. S. K., Moran, T P., & Newell, A. (1983) *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, NJ.
2. Gram, C. & Cockton, G. (1996) *Design Principles for Interactive Software.* Chapman & Hall, London.
3. Klein, M., Kazman, R., Bass, L., Carriere, J., Barbacci, M., and Lipson, H. "Attribute-Based Architecture Styles", Software Architecture pp. 225-243. Proceedings of the First Working IFIP Conference on Software Architecture, San Antonio, TX, Feb, 1999.
4. Nielsen, J. (1993). Usability Engineering. Morgan Kaufman Publishers, Inc., San Francisco.
5. Shaw, M. & Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, Upper Saddle River, NJ.