Quality vs. Quantity: Comparing Evaluation Methods in a Usability-Focused Software Architecture Modification Task

Elspeth Golden
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA, USA
egolden@cs.cmu.edu

Bonnie E. John
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA, USA
bej@cs.cmu.edu

Len Bass
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
ljb@sei.cmu.edu

Abstract

A controlled experiment was performed to assess the usefulness of portions of a Usability-Supporting Architectural Pattern (USAP) in modifying the design of software architectures to support a specific usability concern. Results showed that participants using a complete USAP produced modified designs of significantly higher quality than participants using only a usability scenario. Comparison of solution quality ratings with a quantitative measure of responsibilities considered in the solution showed positive correlation between the measures. Implications for software development are that usability concerns can be included at architecture design time, and that USAPs can significantly help software architects to produce better designs to address usability concerns. Implications for empirical software engineering are that validated quantitative measures of software architecture quality may potentially be substituted for costly and often elusive expert assessment.

1. Introduction

Software engineering attacks difficult development projects by bringing method to the chaos of requirements, design, coding, teamwork, and deadlines. Given the multitude of methods available, developers should choose based on the needs of their projects and the efficacy of the methods. Empirical software engineering often delivers results demonstrating that a method is an improvement along a particular measure of interest, but those measures are sometimes difficult to relate to desired outcomes in real-world projects. Some measures are easy to collect, like time on task or number of compilation errors, but these measures are less compelling when the tasks are more open-ended, as in software architecture design. Measures of quality may have more external validity but are more difficult to collect. For example, expert assessment of quality is particularly problematic; experts are costly, difficult to find and corral, and agreeing on and calibrating to a coding scheme is time consuming.

This paper has two goals related to the validity of a test of new software architecture design materials. The primary goal is to increase the construct validity [6] of a previous study by reporting a measure of quality in addition to the measure of coverage already in the literature [8]. The second is to investigate convergent validity of our coverage metric, which is relatively easy to collect and analyze, with the much more costly expert assessment of quality.

In this paper, first we justify the importance of usability as it relates to software architecture, and briefly discuss Usability-Supporting Architecture Patterns (USAPs), which address the handling of specific usability concerns in the software architecture design process. Second, we describe our experiment to test the effectiveness of USAPs. We report briefly on objective results, including time on task, and a coverage metric that counted how many software responsibilities were considered in an architecture design modification task [8]. Next, we turn to our use of expert evaluators, describing the results found when software architecture experts assessed the quality of the same participant solutions to which we had independently applied our coverage metric. We then investigate the convergence of our easily-collected coverage metric with the more costly expert quality assessments. Finally, we describe the implications of our results for future work.

2. Background

Usability is an important quality attribute of interactive software systems, but it been largely ignored in the context of architecture design for software development. Since the 1980s, usability has been treated as a subset of modifiability, in which architects commonly separate the user interface from the functionality of the system, and assume that usability issues that arise during user testing can be handled with localized modifications to the user interface portion of the system [13]. As recently as 1999, usability has been labeled not "discernable" at architecture design time [10]. Although usability is now beginning to receive some recognition as a quality attribute of software architecture, existing heuristic guidelines for usability [11] are sufficiently openly worded as to provide limited guidance for software engineers.

Unfortunately, simply separating the interface from the functionality is a poor way to support usability concerns, many of which reach deeply into a system's architecture design [3]. When deep usability issues are not considered early in the design process, support for them is not designed into the architecture. As a result, usability problems may be found in user testing which would require extensive and costly re-architecting of software systems. Usability issues have been seen to comprise more than 60% of requirements-related defects in some professional software development projects [14].

Our research on the relationship between usability and software architecture has led to the development of Usability-Supporting Architecture Patterns (USAPs), each of which addresses a usability concern that is not addressed by separation alone [9]. Each USAP consists of
- an architecturally sensitive usability scenario,
- a list of general responsibilities derived from forces inherent in the task and environment, human capabilities and desires, and software state, which must be considered by any software implementation for which the usability scenario is relevant, and
- a sample solution implemented in a larger separation-based design pattern such as MVC.

These patterns could prove of great benefit to software architects responsible for developing large-scale software systems, and to the enterprises of all kinds for which these software systems are developed. In our research efforts, we applied an early version of USAPs to the design of the architecture of the MERBoard, a wall-sized tool that supports shoulder-to-shoulder collaboration of the engineers and scientists on NASA's Mars Exploration Rover mission [1]. However, USAPs are quite detailed and complex, which might make them difficult for software architects to apply to their own design problems without personally consulting with the USAP developers. Since software architects are already burdened with complex tasks and extensive methodologies, it is important to determine whether all parts of a USAP are useful or necessary before widely disseminating USAPs to these professionals.

We performed a controlled experiment to assess the value of the different parts of a USAP in modifying a software architecture design. Reported in [8] and summarized below, we asked software engineers to apply one of three different versions of an important USAP (canceling a long-running command) to the redesign of an architecture that had not originally considered the ability to cancel. The experiment measured whether the architectural solutions produced as a result of using all three components of a USAP more fully supported the needs of a usable cancellation facility than those produced by using certain subsets of the USAP components. However, it is not sufficient to consider whether a software architecture design has taken into consideration all the responsibilities needed to implement a particular usability scenario. The quality of the architecture designed to solve the problem is also a critical concern. Therefore, we did further analysis on the solutions produced by our participants, and assessed the quality of those solutions using a panel of expert evaluators, discussed in section 4.3.

3. Description of the Experiment[1]

3.1 Participants

18 computer science graduate students at the Carnegie Mellon West Coast campus participated in the experiment. All 18 participants had completed work for a master's degree in software engineering and/or information technology, and were working toward an additional practice-based Professional Development Certificate. The 15 male and 3 female participants ranged in age from 23 to 30. Fifteen of the participants averaged 25.7 months of industrial programming experience, with a range from 6 to 48 months; the other 3 had no programming experience in

---

[1] The description of the experiment is a synopsis of that appearing in [8], included here so that our additional analyses can be understood. Readers of [8] can skip to Section 4, Results, where new analyses appear.

industry. Fourteen of the participants averaged 15.3 months of software design in industry, with a range of 4 to 36 months; the other 4 participants had no industrial experience in software design. At the time of the experiment, participants reported spending between 5 and 50 hours per week programming, with an average of 22.9 hours per week, and from 0 to 30 hours per week on software design, with an average of 11.4 hours per week.

3.2 Materials

To test the value of different parts of a USAP, we developed three versions of a "Training Document" and a software architecture modification task. For all instructional and task materials described below, eight academic and industry software architecture experts evaluated the contents for correctness and completeness with respect to software architecture.

The scenario-only (S) Training Document containing only a usability scenario describing circumstances under which a user might need to cancel an active command [Fig. 1]. This scenario is similar to the report that a typical usability expert might submit to a development organization, to recommend that cancellation capability be added to an application.

The scenario-plus-general-responsibilities (S+GR) Training Document contained the usability scenario, and also a list of 19 general responsibilities that should be considered in designing any software implementation of a cancel command. For example's sake, Table 1 shows the first 4 of these 19 responsibilities; a full list appears in [8]. This list of responsibilities was derived from an analysis of forces generated by characteristics of the task and environment, the desires and capabilities of the user, and the state of the software itself [9] and vetted by the panel of architecture experts mentioned above. Since this was a list of general responsibilities designed to be considered in any implementation of cancel functionality, the S+GR Training Document stipulated that not all responsibilities might be applicable to solving any given design problem.

The final Training Document (S+GR+SS) included the scenario, the list of general responsibilities, and a sample solution for adding cancellation to a software architecture design, based on MVC. The sample solution contained a "before and after" Component Diagrams of the MVC architecture, "before" meaning a model which did not consider cancellation, and "after" meaning a model which did explicitly consider cancellation. A set of numbered responsibilities displayed in the components in the "before" diagram corresponded to a numbered list of the responsibilities of each of the MVC components.

	Usability Scenario: Canceling a Command
	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. It doesn't matter why the user wants to stop; he or she could have made a mistake, the system could be unresponsive, or the environment could have changed.

Figure 1. Usability Scenario in Training Document

Numbered responsibilities displayed in the "after" diagram additionally allocated the general cancellation responsibilities (CRs) in the Training Document to the MVC components, and added several new components required to fulfill the cancellation responsibilities. Although the example solution is not the only arrangement of components and responsibilities that would support cancellation, our eight external expert architects came to a consensus that it was a reasonable solution.

Because of these differences in content, the three Training Documents varied in length. The S Training Document consisted of a single paragraph, the S+GR Training Document was three printed pages of prose and the S+GR+SS Training Document was eight pages of both prose and software architecture diagrams that related to the prose.

A software architecture redesign task was given to each participant: to redesign an existing architecture that was not designed to support cancellation, so that the resulting design did support cancellation.

Table 1. Sample of Responsibilities of Cancel

CR1	A button, menu item, keyboard shortcut and/or other means must be provided, by which the user may cancel the active command.
CR2	The system must always listen for the cancel command or changes in the system environment.
CR3	The system must always gather information (state, resource usage, actions, etc.) that allow for recovery of the state of the system prior to the execution of the current command.
CR4	The system must acknowledge receipt of the cancellation command appropriately within 150 ms. Acknowledgement must be appropriate to the manner in which the command was issued. i. For example, if the user pressed a cancel button, changing the color of the button will be seen. ii. If the user used a keyboard shortcut, flashing the menu that contains the command might be appropriate.

143

For this task, we chose the Plug-in Architecture for Mobile Devices (PAMD), an architecture design for a plug-in controller for the Palm OS4. PAMD has been used in a software architecture design and analysis course at the Software Engineering Institute at CMU [7]. This architecture design was chosen for several reasons. First, PAMD is simple enough that a participant already trained in software architecture can understand it in a relatively short period of time. Additionally, since the Palm OS4 is a single-threaded architecture, adding the ability to cancel a long-running command is a nontrivial task. PAMD was also sufficiently different from MVC that the participants who received the MVC-based sample solution had to extrapolate and generalize in order to create a specific solution for adding cancel to PAMD.

The Task Instructions included seven elements:

- a general description of the PAMD architecture;
- an example scenario of how PAMD works (without considering cancellation);
- a numbered list of responsibilities of the PAMD components for normal operation;
- a numbered list of Component Interaction Steps detailing the run-time operation of PAMD while calling a plug-in;
- a Component Interaction Diagram, showing the components and connectors involved in the PAMD architecture, and assigning numbered responsibilities from element (3), and numbered steps from element (4), to each component;
- a Sequence Diagram of PAMD run-time component interaction while calling a plug-in, utilizing the numbered steps from element (4) and the components from element (5);
- a final page instructing the participant to add the ability to cancel a plug-in to the PAMD architecture design. The final page instructed participants to only modify the architecture to address cancellation, without considering or preserving other usability concerns or quality attributes. They were instructed to indicate their changes by modifying diagrams and written materials on the Answer Paper provided.

We designed an Answer Paper where the participants could easily and efficiently record the information relevant to their redesign. Because we did not want the participants to waste time drawing boxes and moving them around, or to struggle with computer-based tools which might have introduced confounding factors, the Answer Paper was paper-based and

contained a Component Interaction Diagram, a Sequence Diagram, and a list of Component Interaction Steps, all with sufficient white space for the participants to insert their designs. The participants were also given several blank sheets of paper to use as they wished. The Component Interaction Diagram and the Component Interaction Steps were identical to those provided in the Task Instructions, except that the assignments of numbered PAMD responsibilities and run-time steps were removed from the Answer Paper. The Sequence Diagram in the Answer Paper showed unnumbered execution steps only up through calling a plug-in, instead of continuing through the completion of normal plug-in termination as in the Task Instructions, because the user's request for cancellation would appear in the Sequence Diagram after the plug-in was called.

The participants were instructed to use these diagrams as the bases for their designs, to add any components, responsibilities, or steps as needed to express their ideas for supporting cancellation. They were asked to make the diagrams correspond with each other, just as the PAMD diagrams corresponded with each other.

In addition to the Training Documents and task instructions, we also designed a canonical solution to the redesign task. As with the MVC example, this solution was revised iteratively until the panel of experts agreed that it was a reasonable solution. The canonical answer was used as a point of comparison for the quality judgments reported in Section 4.3.

3.3 Procedure

Participants were randomly assigned to one of the three experimental conditions, and run in individual sessions of unlimited duration. In an introduction to the experiment, the participants were told that they were participating in a study about fixing one kind of usability problem in a specific software architecture design. They were informed that they would be given a handout to read, describing a usability scenario relevant to system architecture, that we would then read through a description of a system architecture needing the information described in the handout, and finally, that they would be asked to understand and modify the sample software architecture to meet the requirements of the usability scenario.

Participants were then given the appropriate Training Document for their experimental condition, and asked to read and understand it. After reading their Training Document, participants were given the Task Instructions. To minimize variation in time and comprehension level during this portion of the

144

experiment, the experimenter read the Task Instructions aloud, while the participant read along silently, and interrupted with any questions. During the reading of the final page of the Task Instructions, participants were given the Answer Packet. Each participant was allowed unlimited time in which to complete the redesign task. After completing their solution, the participant was asked to explain the details of the solution to the experimenter to disambiguate any hand-writing or diagrammatic difficulties.

4. Results

We were interested in whether the different Training Documents had an effect on the quality of the resulting software architecture design. To measure quality, we counted the cancellation responsibilities they considered in their design (Section 4.2) and had architecture experts assess quality on a Likert scale (Section 4.3). We also analyzed time on task to investigate any speed/accuracy trade-offs that might have resulted from the different Training Documents. Since time on task proved less interesting than the quality of the design, we present those results first.

4.1 Time on Task

There were three separate time intervals in each session: the time to read the Training Document, the time to read the task instructions, and the time to perform the task (Table 2). The mean time required to read each of the Training Documents varied significantly between the three experimental conditions in a one way ANOVA ($F(2,14) = 23.46$, $p < .01$). This result was expected because of the difference in length of the Training Documents (1 paragraph, 3 pages, and 8 pages, respectively). Since the Task Instructions were read aloud by the experimenter, variations in time taken on this document were produced only if a participant interrupted with questions during the reading. Again, as expected, a one-way ANOVA revealed no statistical

Table 2. Results of Time Analyses

Time	Condition	Mean	Min	Max
Time to read Training Doc	S	1.5	1	3
	S+GR	8.0	5	11
	S+GR+SS	18.2	10	29
Time to read Task Instructions	S	15.2	14	18
	S+GR	13.7	13	14
	S+GR+SS	14.2	13	15
Time to perform Task	S	79.7	47	124
	S+GR	88.3	64	112
	S+GR+SS	84.6	39	138

main effect of condition on this measure ($F(2,14) = 2.85$, $p = .09$).

The time to complete the redesign task is more interesting because we had no a priori expectations of how the task instructions would effect time to perform the task. The scenario-only (S) condition might have taken more time than the others because the participants might have to think harder without guidance. The full USAP (S+GR+SS) might have taken longer because mapping the MVC example to the PAMD might be difficult. However, a one-way ANOVA revealed no statistical main effect of condition on this measure ($F(2,15) = .21$, $p < .10$). Although our small sample size and high variance contributes to the lack of significance on this measure, this is preliminary evidence that the improvement in quality we will report in the next sections is not due to additional time on task.

4.2 Coverage Metric

Analysis of the results for the cancellation responsibilities variable have been presented elsewhere [8], but will be summarized here for purposes of comparison with the quality measure discussed in the next section.

Similarly to the method used in Prechelt et al. for counting the degree of requirements fulfillment within subtasks of a programming maintenance task [12], we developed a scoring system that counted the union of all cancellation responsibilities found to have been considered in elements of the participant solution. That is, we counted a responsibility as having been considered if it appeared in the Component Interaction Diagram, or the Sequence Diagram, or the Component Interaction Steps, or in any list of Additional Responsibilities added by the participants. Each specific cancellation responsibility that appeared in the participant solution was counted only once, independently of the number of solution elements in which it appeared.

Using the results in Table 3, analysis of variance showed a significant main effect of the USAP subset given to participants on the number of cancellation responsibilities considered ($F(2,15) = 5.26$, $p < .05$). Pairwise comparisons made using Tukey's HSD indicated significant mean difference between giving the scenario alone (S) and giving the full USAP (S+GR+SS), while mean difference between other pairs of conditions was not significant.

4.3 Quality of Solution

To collect a more direct measure of quality than the coverage metric defined above, we employed the panel of eight software architecture experts who had

evaluated the experimental materials and canonical solution prior to the experiment. The canonical solution was not used as a definitive "answer sheet" by the evaluators, rather, the process of creating it assured that the evaluators were calibrated as to what constituted a reasonable solution. Participant solutions were anonymized and randomly distributed to these evaluators. Evaluators were asked to rate the overall quality of each participant solution as an architectural solution for adding cancellation, on a Likert scale from 1 to 7, where a rating of 1 indicated that the "participant solution is a substantially poor architectural solution for adding cancellation," a rating of 4 was an "adequate architectural solution", and 7 was a "substantially good architectural solution."

In addition to the overall judgments of quality, evaluators were asked to determine whether each individual cancellation responsibility, out of the 19 possible responsibilities, had been addressed at all in the participant's solution. If a specific responsibility was judged to have been addressed, evaluators were asked to specify where in the participant solution the cancellation responsibility had been addressed, and to rate, on a Likert scale from 1 to 7, how well the responsibility was handled in the solution, with a rating of 1 indicating "the responsibility was handled very poorly," and a rating of 7 indicating "the responsibility was handled very well." These evaluators were not given access to the scoring system we had devised, above, but were expected to perform their own independent assessment of whether each specific cancellation responsibility had been addressed in any given participant solution. Thus, expert evaluator assessments were not confounded by our earlier assessments using this counting method. Finally, the evaluators were invited to add written comments on the overall quality of the participant solution.

Five experts returned a total of 45 evaluated solutions, distributed such that each participant solution received from one to four independent evaluations. Additionally, three of the eight original experts withdrew from the study before completing evaluation of the participant solutions. Most participant solutions

Table 3. Cancellation Responsibilities Considered

Cancellation Responsibilities Considered	Scenario (S)	Scenario & General Responsibilities (S+GR)	Scenario & General Responsibilities & Sample Solution (S+GR+SS)
Mean	3.17	7.67	9.50
Std Dev	0.75	4.41	4.04
Lowest	2	4	5
Highest	4	15	15

was therefore assessed by between two and four expert evaluators; two participant solutions received only one expert evalution apiece. One of the original eighteen participant solutions was omitted from the final analysis, due to incorrect application of anonymization procedures; this solution was in the condition which received the full set of USAP materials.

Analysis of variance showed a significant main effect of the USAP subset given to participants on the quality of the solution $(F_{(2,14)} = 8.64, p < .01)$. Pairwise comparisons made using Tukey's HSD indicated significant mean difference between giving the scenario alone (S) and giving the full USAP (S+GR+SS), while mean difference between other pairs of conditions was not significant [Fig. 2]. These results mirror the results of the coverage metric.

Paired-sample t-tests did not show a significant effect of time on task on the quality of the solution, and Pearson's correlation between groups did not find significant correlation between the quality of the solution and any factor other than the experimental condition, such as industrial experience in programming or software design, age, or gender.

4.4 Comparison with Consideration of Cancellation of Responsibilities

In this phase of data analysis, we compared the experts' judgments of solution quality with our earlier count of responsibilities considered. One goal of this analysis was to investigate convergent construct validity [6], that is, whether these different measures of the quality of a solution are measuring analogous effects of the training condition. If so, our relatively easily collected and analyzed coverage metric might be used alone in future experiments, saving months of work.
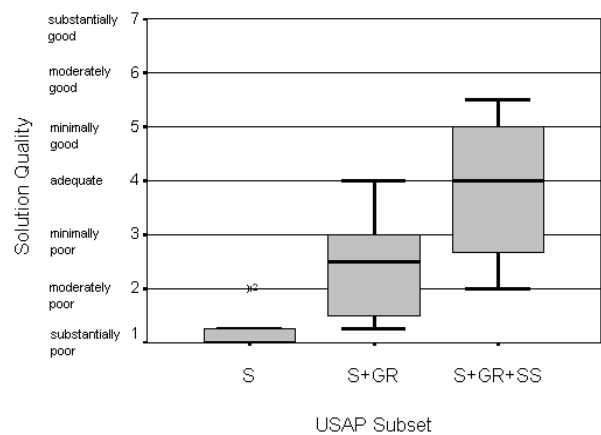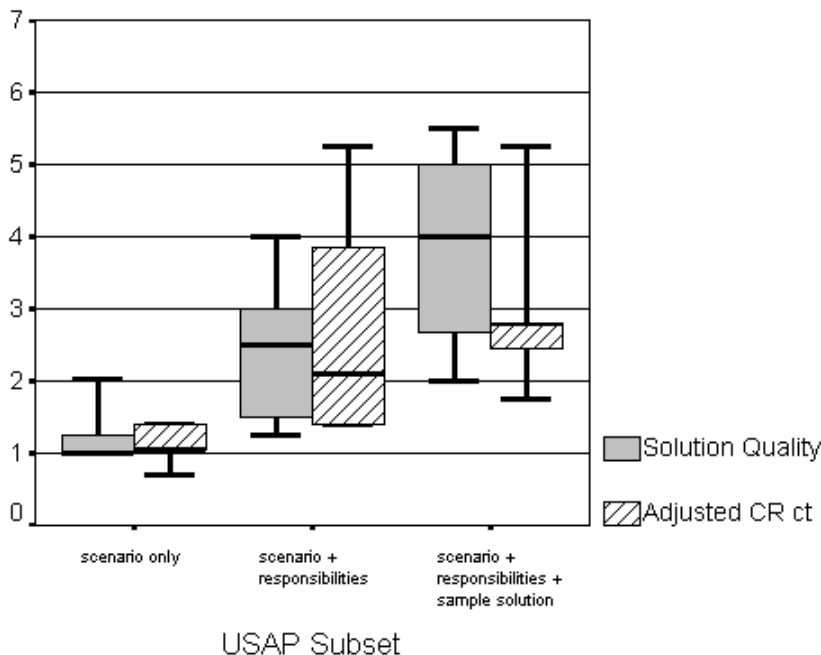

Figure 2. Solution Quality by USAP Subset

Figure 3. Solution Quality vs. Adjusted Count of Responsibilities by USAP Subset

The coverage metric was first adjusted for scalar consonance with the quality measure, using the formula Adj CR Count = CR Count * 7 /20. This adjustment was performed to allow easier visual comparison of the two outcome measures. Fig. 3 shows a side-by-side comparison of the mean solution quality and adjusted cancellation responsibilities count measures for the three different subsets of the cancellation USAP provided in the participants' Training Documents. The long tails on the Adjusted CR count boxplots in the Scenario + General Responsibilities (S+GR) and the Scenario + General Responsibilities + Sample Solution (S+GR+SS) conditions indicate that there were two participants in each of those conditions who considered more responsibilities in their solutions than did the remainder of participants in those experimental conditions.

Pearson's correlation between groups found significant correlation between the number of responsibilities considered by participants and the quality of solution as rated by expert evaluators ($F_{(1,15)}$, r = .775, p < .01). We performed a linear regression to determine whether the solution quality measure could be predicted from our coverage metric. Linear regression of the adjusted count of responsibilities against solution quality also showed significant association between the count of responsibilities and quality of the solution (F = 22.56, p < .01). Fig. 4 shows the relationship between the adjusted count of responsibilities and the solution

quality as rated by expert evaluators; the regression equation is displayed on the graph.

5. Discussion

On the other hand, solutions produced in the scenario plus general cancellation responsibilities (S+GR) condition were not significantly better than those produced with the scenario only. The list of general responsibilities is akin to a checklist, which are commonly accepted memory aids used in software development practice. Adding the example solution makes the S+GR+SS Training Document akin to the materials suggested by design patterns advocates. Our current results, with a sample size of 6, are good news for patterns advocates; a checklist memory aid alone is insufficient to produce statistical difference in coverage and quality, but the addition of an example that applies the checklist, although it adds some benefit [2], does not produce statistically significantly improvement in software designers' performance. We are currently collecting additional data to see if this result holds with a larger number of participants, or if the trend toward statistically significant difference between each of the three conditions bears fruit once the sample size has been increased.

The experiment also showed a significant positive relationship between the count of cancellation responsibilities and the overall quality of the solution. This is important because, although our previous

147

analysis showed significant difference between the scenario-only recipients and those who had the full USAP, it was unclear whether simple consideration of cancellation responsibilities would correlate with overall quality of solution. Our results are encouraging, accounting for 60% of the variance in the quality judgments.

In an attempt to understand whether the unaccounted for variance is noise in the data or some systematic variation we plot Predicted vs. Actual Solution Quality in Fig. 5. Each point is a single participant, ordered by condition (S, S+GR, then S+GR+SS) and by ascending actual solution quality within condition. Fig. 5 shows an apparent tendency of the model toward centrality. That is, the model more often appears to over-predict when the solution quality is low (condition S), and under-predict when the solution quality is high (condition S+GR+SS). Although we do not have sufficient data to analyze this statistically, we can hypothesize why this might have occurred. One hypothesis is that experts may value some cancellation responsibilities more highly than others in arriving at their quality ratings, while the coverage metric weights all cancellation responsibilities equally. This could result in high quality ratings for solutions which consider responsibilities that are more highly valued by experts, even when coverage is not especially high, and lower quality ratings for solutions that leave out expert-valued responsibilities.

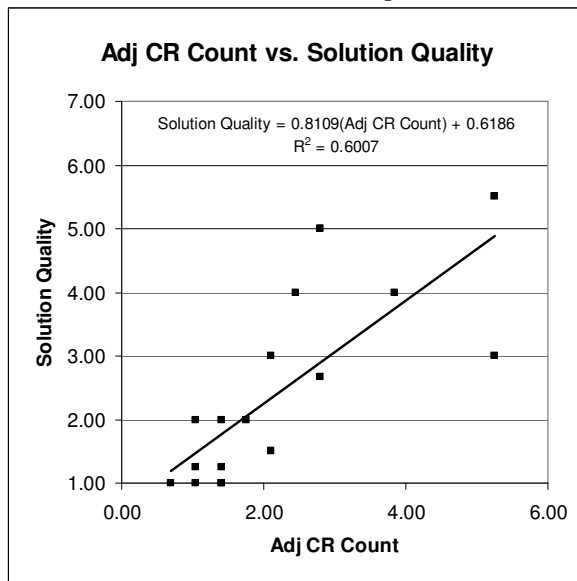A first step towards teasing out this sort of information is to determine if the expert evaluators had

Figure 4. Adjusted Count of Responsibilities vs. Solution Quality

identified any responsibilities as being addressed with higher quality in the S+GR+SS condition than in the S condition. In our post-hoc analysis of the evaluators' judgment of coverage (using Tukey's HSD), we encountered six such cancellation responsibilities. Three of these responsibilities concerned the treatment of system state: CR3, which states that the system must always gather information that allows for prior system state recovery, and CR10 and CR11, both of which detail responsibilities for restoring prior system state when the active command is cancelled. The remaining three were unrelated: CR2, which points out the responsibility of the system to listen for a cancel command or changes in the system environment, CR4, which allows for acknowledgement of a cancel command, and CR17, which stipulates that the system must estimate the time cancellation will require. These six responsibilities were considered by the expert evaluators to have been handled poorly or ignored altogether in solutions created by participants who received only the general scenario, while having the full USAP significantly increased the quality and frequency with which they were handled in participant solutions. Although we do not have enough data to statistically correlate these responsibilities with solution quality, and determine weighting parameters, this is an indication that there are some specific areas in which the full USAP is particularly useful in adding cancellation to an architecture design, and these areas may warrant further investigation.

Finally, the disparity between the predicted and actual solution quality seems to be larger in the full-USAP condition than in the other two. Although we have no clear explanation at this time, further analysis using the free-form comments written by the expert evaluators may shed light on the phenomenon.

6. Conclusions and Future Work

Using a full USAP increased the quality of solution that participants created in an architectural redesign to add cancellation to the existing architecture design for the PAMD system. Participants who used all three parts of the cancellation USAP were able to create a solution of significantly higher quality, as well as to identify and address three times as many cancellation responsibilities, on average, as participants who received only a general usability scenario, in the same amount of time, and without having more work experience or formal training prior to the task. Thus, the USAP for canceling a command can already be considered a valuable tool for modifying software architecture designs to address a specific usability concern.
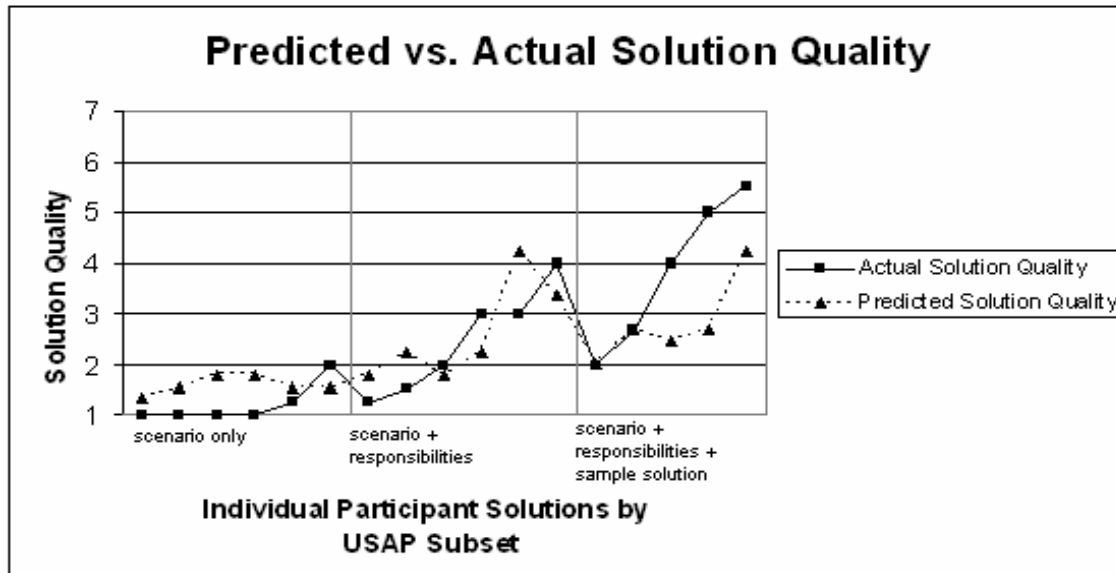
Figure 5. Predicted vs. Actual Solution Quality

More work is necessary, however, since the USAP brought these participants up to an average of "adequate"; we would like to bring the average software designer up to "good". In particular, the high variability in quality of solutions produced using the full USAP indicates that the sample solutions might benefit from an improved format, or perhaps the provision of additional training with the USAP.

Quality judgments performed by expert software architects external to our research project allowed us to examine correlations between coverage of cancellation responsibilities, and the overall quality of the software architecture solution. The positive correlation between the externally evaluated quality of participant solutions and our method of counting coverage of cancellation responsibilities has implications for evaluation of other architecture design techniques that can be expressed in terms of responsibilities. Expert evaluation is high in temporal and organizational costs; if a coverage metric of responsibilities considered in a solution could be used as an effective discount alternative to expert evaluation, these costs would be significantly reduced. At present, we are in various stages of developing more than two dozen USAPs. This experiment focused on a single usability scenario, and a single USAP; future work will seek to expand our results to additional scenarios and additional USAPs. Having an empirically validated convergent evaluation method which does not rely on external experts will be an aid to further research in this area.

7. Acknowledgements

8. References

[1] Adams, R.J., Bass, L., and B.E. John, (in press) 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.

[2] Anderson, J.R. and J.M. Fincham, "Acquisition of Procedural Skills from Examples", *Journal of Experimental Psychology: Learning, Memory and Cognition*, 20, 1322-1340.

[3] Bass, L. and B.E. John, "Supporting Usability Through Software Architecture", IEEE Computer, 34 (10), October, 2001, 113-115.

[4] Bass, L., John, B.E., and J. Kates, "Achieving Usability Through Software Architecture", Carnegie Mellon University/Software Engineering Institute Technical Report No. CMU/SEI-TR-2001-005, 2001.

[5] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. & J. Stafford, Documenting Software Architectures: Views and Beyond, Addison-Wesley, 2003.

[6] Cook, T.D, and D.T. Campbell. Quasi-Experimentation: Design & Analysis Issues for Field Settings, Houghton Mifflin, 1979.

[7] Eguiluz, H., Govi, V., Kim, Y.J. and A. Sia, PAMD: Developing a Plug-In Architecture for Palm OS-Powered Devices Using Software Engineering, Technical Note CMU/SEI-2002-TN-020, Carnegie Mellon University, Pittsburgh, PA, 2002.

[8] Golden, E., John, B.E., and L. Bass. The Value of a Usability-Supporting Architectural Pattern in Software Architecture Design: A Controlled Experiment. *Proceedings of the 27th International Conference on Software Engineering*, May, 2005, St. Louis, MO.

[9] John, B.E., Bass, L., Sanchez-Segura, M-I., and R.J. Adams, "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, German, July 11-13, 2004.

[10] Naveda, J. Fernando, "Teaching Architectural Design in an Undergraduate Software Curriculum", *ASEE/IEEE Frontiers in Education Conference Proceedings*, 1999, 12b1-4.

[11] Nielsen, J., Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods, John Wiley & Sons, New York, NY, 1994.

[12] Prechelt, L., Unger, B. Philippsen, M. and W. Tichy, "Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance", *IEEE Transactions on Software Engineering*, 28, 6, 595 – 606, June 2002.

[13] Van der Veer, G. and H. van Vliet, "A Plea for a Poor Man's HCI Component in Software Engineering Curricula," *Proceedings of the 14th Conference on Software Engineering Education and Training*, Charlotte, NC, February 19-21, 2001.

[14] Vinter, O., "Experience-Based Approaches to Process Improvement," *Proceedings of the Thirteenth International Software Quality Week*, Software Research, San Francisco, CA, 2000.