Benefits of Matching Domain Structure for Planning Software: The Right Stuff

Dorrit Billman[*+]
dorrit.billman@nasa.gov

Lucia Arsintescu[*+]
lucia.arsintescu-1@nasa.gov

Michael Feary[+]
michael.s.feary@nasa.gov

Jessica Lee[*+]
jessica.c.lee@nasa.gov

Asha Smith[*+]
asha.h.smith@nasa.gov

Rachna Tiwary[*+]
rachna.tiwary@nasa.gov

*San Jose State University Research Foundation (SJSURF)
[+]NASA Ames, Moffett Field, CA 94035-1000

ABSTRACT
We investigated the role of domain structure, in designing for software usefulness and usability. We ran through the whole application development cycle, in miniature, from needs analysis through design, implementation, and evaluation, for planning needs of one NASA Mission Control group. Based on our needs analysis, we developed prototype software that matched domain structure better than did the legacy system. We compared our new prototype to the legacy application in a laboratory, high-fidelity analog of the natural planning work. We found large performance differences favoring the prototype, which better captured domain structure. Our research illustrates the importance of needs analysis (particularly Domain Structure Analysis), and the viability of the design process that we are exploring.

Author Keywords
Complex work domains, planning, aeroastronautics.

ACM Classification Keywords
H5.2. [Information interfaces and presentation]: User interfaces – *evaluation/methodology*.

General Terms
Design, Experimentation, Human Factors, Performance.

I. INTRODUCTION

Building useful software depends on understanding what is needed. Work applications typically consist of a bounded domain and set of required functions. If these are identified and if software enables meeting these needs, the software will be useful. Multiple methods may be used to identify needs (e.g., Cognitive Work Analysis, Contextual Design), and such methods typically precede formal requirements specification. This paper reports on one approach to needs analysis. This is part of a larger research agenda of developing tools and methods: we aim to reduce costs of the information needed to ensure good human-system integration, particularly in high-risk domains [11,12].

Our approach to needs analysis focuses on capturing the abstract structure of the work domain; we refer to the product of such an analysis as a Domain Structure Analysis (DSA). Our approach contrasts with task analyses that capture the specifics of tasks carried out within the existing configuration of tools and procedures. Our goal is to identify high-value, low-cost information useful for design and evaluation. A Domain Structure Analysis (DSA) can guide design and evaluation of software to support work in the analyzed domain.

This paper reports an initial demonstration of the value of our approach to needs analysis, in a complex socio-technological domain: the planning work of one International Space Station (ISS) Mission Control group, Attitude Determination and Control Operator (ADCO). (*ADCO* refers to the group or an individual.) We run through a demonstration of the role of domain structure analysis from design through evaluation, "in miniature" for a limited scope of work. The logic of our demonstration is as follows: 1) We provide an example domain structure analysis. 2) We illustrate how the analysis can be used to select and design software by providing high-level constraints on key properties of the design. 3) We show how the analysis supports evaluation (as well as design) by assessing how well a candidate design matches the domain structure, and by making performance predictions based on the match. 4) We report a study that evaluated two planning tools, tested the predictions, and found dramatic performance differences as predicted.

The paper starts by setting the context for ADCO planning work. Then we report our research on the value of domain structure analysis; the focus of the paper is our experimental evaluation. Finally, we reflect on what this study suggests about needs analysis, our approach, and its role in producing useful and usable software.

Copyright 2011 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the U.S. Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.
Copyright 2011 ACM 978-1-4503-0267-8/11/05....$10.00.

II. ADCO WORK: FLYING THE SPACE STATION

ADCO controls the attitude (yaw, pitch, & roll) of the ISS. The operators monitor and command attitude real-time and also develop plans in advance of real-time operations. They are responsible for maintaining an efficient flight attitude during quiet phases of flight and for maneuvering into different attitudes for activities such as docking and undocking of vehicles or to support various tests. Supporting these activities means carrying out detailed sequences of actions that switch among mechanisms of controlling and applying force: the thrusters on the Russian capsules currently docked with the ISS, the momentum management system (large gyroscopes which can transfer torque), and in rare circumstances the space shuttle. These mechanisms and commanding software move the ISS to the appropriate attitude.

ADCO closely cooperates with Russian counterparts, to develop detailed plans supporting these activities. While the technical complexity of these operations requires good planning, the required international cooperation imposes an even greater need for accurate, detailed planning well in advance of execution.

The following characteristics of ADCO planning relate it to other work domains: 1) **High-Risk & Safety Critical**. The ADCO planning domain is part of a safety-critical, high-risk domain [10,15,18]. The planning functions help decouple [18] aspects of the system thus protecting real-time operation: ADCO intensively checks plans, identifies potential threats, and builds contingency plans. Planning favors reuse of old, safe plan components rather than exploration of less tested but possibly more optimal plans. 2) **Products are Plans**. The ADCO planning domain is information work and the products are represented in documents. ADCO gathers, integrates, and approves the information making up a plan. (*Plan* refers to both the abstract information and its expression in a particular document.) Planning documents select different information, for different users. 3) **Not Real-Time Control.** The ADCO planning domain is not subject to strong time pressure. There are planning deadlines, but in normal circumstances, the planning activity is not heavily driven by the real-time dynamics of unalterable, extrinsic, physical events. Indeed, the point of planning is to remove time pressure from decision making, thus allowing more successful execution. 4) **Complex Socio-Technical Domain**. Technically, the ISS is unique, changing, intensively modeled, yet incompletely understood. ADCO planning aims to ensure that planned activities are well away from technical constraints on how the ISS flies. ADCO draws information from several engineering support groups rather than conducting detailed engineering work internally. Social rather than technical constraints are the primary drivers of planning difficulty: when information becomes available from another group, or when others need ADCO products.

Current planning software evolved incrementally to serve a variety of the planning needs. Our research group became involved with the ADCO group because of their desire to improve the planning tool used to build and revise plans, particularly, the plan documents used in joint planning with their Russian counterparts.

Our observation, discussion, and document-gathering focused on this part of the planning process. We developed some understanding of the communication flows; however, our focus (of both analysis and experiment) is the operations done by an individual to build and revise the planning documents used to communicate with the Russian counterparts. Further, this is the aspect that key operators thought most needed improvement.

III. NEEDS ANALYSIS

Our Process & Comparison to Other Approaches

We began our needs analysis with task analysis [8,20] at a quite specific level; we shifted focus to capturing the relatively high-level functions, far removed from low-level, button-pressing. Our approach diverged further from task analysis as we saw that many tasks should disappear or change dramatically with better-designed tools. Although a high-level analysis of tasks [14] aims to avoid over-specificity, even a high-level task characterization seemed tied to current practices. It was difficult to identify the boundary between necessary tasks and those driven by history or limitations in current tools. (We suspect that in highly routine and proceduralized domains it may be easier to identify the functions, while analyzing the "declarative" information structure may be more tractable in open, generative, information work.)

We shifted to identifying the structure and constraints on ADCO planning. We sought to identify necessary elements both of the process of planning, and of the product, namely, the ADCO plan. A critical observation was that the ADCO planning work is primarily a matter of producing documents, appropriately vetted and with appropriate content. The primary constraints concern the required properties of the plan. Requirements on the product seemed to be a particularly important, clear guide to what is needed from planning software: it must support building and editing correctly structured plans. Our focus was on identifying the fundamental structure required in a plan, rather than constraints on its detailed content.

We are similar to Cognitive Work Analysis (CWA) [3,9,19,21] in prioritizing stable, structural aspects of the domain over tasks, but we diverged from CWA concerning the nature and source of constraint. For CWA, constraints are fixed and external to the work: the chemical plant operator does not re-engineer the size of the chemical tanks. For information work, rather than lying outside the work, the "constraints" or structure may be very much part of the work, constructed by the work activity, expressed in the work product, and changeable or negotiable as part of the

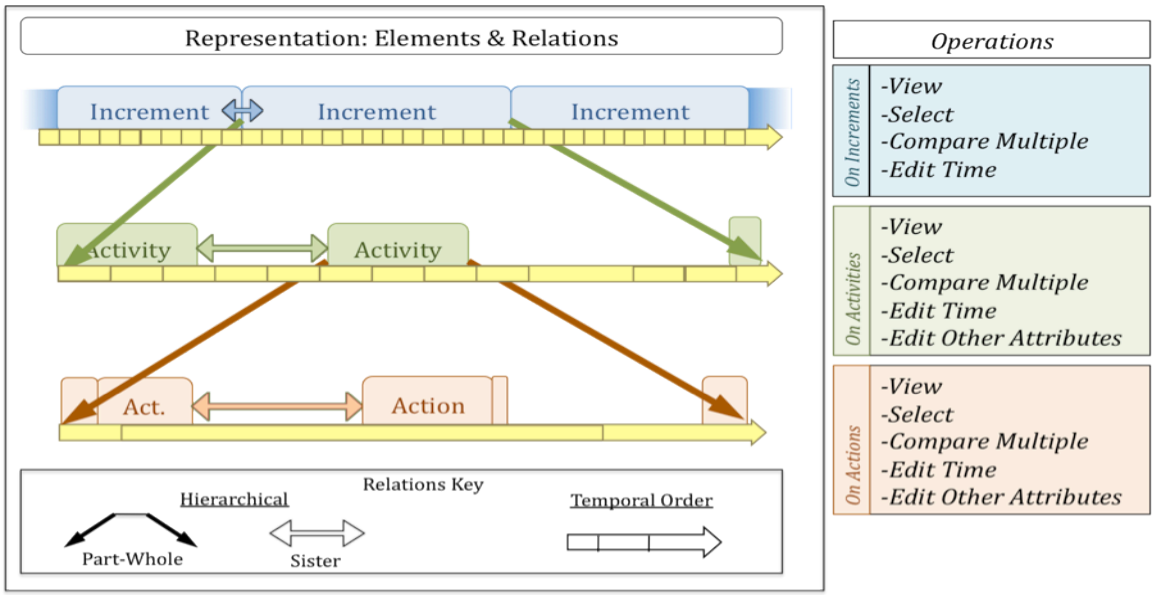

Figure 1. Schematic of the Domain Structure Analysis: Representations and operations in the ADCO planning domain. Increment, Activity, and Action elements are organized by Hierarchical (Part-whole, Sister) and Temporal (Order, contiguity) relations. Operations on these elements that are used by ADCO are shown on the right.

analysis of work products, in order to identify domain structure.

Ontology-based approaches also focus on identifying structural, "declarative" information about the work domain, and our approach is most similar to this [4,5,13]. Ontologies have been developed for information work domains, in which the structure and constraints are defined within the work, rather than outside its boundaries.

Contextual Inquiry (CI) [1] generates information about the needs that software should address. CI emphasizes gathering information about users, but also gathers information about structure or constraints, for example, in the artifact models. CI's emphasis on user observation may be most effective in less technical information work, where the domain structure can be easily understood by an observer adopting the role of a mentee [7].

We provide an example Domain Structure Analysis, for the ADCO planning domain. We believe this form of needs analysis will prove most directly applicable to 1) information work 2) for which accurate, novel, problem solving rather than speeded performance is paramount 3) and that is safety-critical or otherwise benefits from a very accurate design. This paper focuses on the product of analysis, not the process, which we continue to change significantly. Additional information on the method used here may be found in [2], but development is ongoing.

Need Analysis Results: Structure of the ADCO Domain.

A domain structure analysis identifies the organization of the work domain. The structure is expressed in terms of its elements and their relations; in addition, the operations on the domain are identified. For an information-work domain such as ADCO, domain structure is specified by the structure of ADCO plans. ADCO plans compose events into a part-whole hierarchy organized linearly in time. Primary operations on the plans and plan components are viewing, selecting, and editing. An overview of the high-level organization is shown schematically in Figure 1. The elements, relations, and operations from which the structure is built up are summarized in Table 1, and the plan elements described in Table 2.

The plan elements and types were not explicit prior to our analysis. Providing an explicit, external, representation of key concepts known implicitly by domain experts is a frequent, desirable outcome of many forms of work analysis. In our case, initially there was no general and standard term referring to *Activity*, to mark its importance as more than a series of Actions. Rather, events were often referred to very specifically, in two ways: 1) references were made to a specific docking mission (activity) or specific docking maneuver (action) in ways that glossed their relationship and 2) events were often referred to in terms of a particular format (e.g., "the UAF file") rather than the event it represents, which may appear in multiple formats and documents. The construct of *Increment* was explicit, both outside and within ADCO. Increments are the largest element for planning in ADCO, spanning the period between arrival and departure of ISS crew (i.e., from the

Elements	Increment, Activity, Action
Relations	part-whole {part-of: contains; sister} temporal{prior to: after; contiguous: gapped}
Operations	View, Select, Compare, Edit Time, Edit Other Attribute

Table 1. Components of the Domain Structure: Representation (Elements and Relations) and Operations.

Element	Description	Examples
Increment	Duration for a fixed ISS crew. One lead operator.	3+ months between Arv-Shuttle: Launch 129 Dpt- Soyuz 20.
Activity	Mission goal accomplished by integrated events	Docking, Undocking, Relocate, Reboost, Thruster Test
Action	Commanded operation. Attribute values specify parameters. May be point or interval event.	Change Attitude: Maneuver to docking attitude Change Control: Handover to Russia; Momentum Managemt; Free Drift

Table 2. Elements of the Domain Structure

Shuttle or Soyuz). One ADCO lead is assigned to each increment. Actions were explicit; however, *types* of actions as well as *types* of activities were not explicit but identified through study of documents and discussion with experts.

The temporal properties of plans are defined in terms of the three elements. All elements have both a relative temporal ordering and an absolute time value. Early in planning, events may have only relative time established or be assigned estimated times. Events at the same level are disjoint and do not overlap. Increments and Activities have durations. Some Action types have durations and others are specified as a point in time.

An action belongs to one of a small number of types, and is specified in terms of attribute values. Key attributes include start and stop times, action type, the attitude attained or maintained (yaw, pitch, and roll), and the control mechanisms used at the beginning and end of the action.

To summarize, ADCO plans consist of structure built from Increments, Activities, and Actions. The important relations between these elements come from the part-whole hierarchy (part-within-whole and sister-to-sister), and the temporal structure (including order and separation of elements). This plan structure represents events as compatible both with how people often think about events and future/planning [22] and with how these specific plans are executed. Note that this plan structure is not the only possibility. For example, in some domains plans might be appropriately represented as regulating several continuous flows. In those cases, the plan need not decompose into temporally separated, internally structured hierarchical elements.

IV. DOMAIN STRUCTURE AND SOFTWARE DESIGN

Rationale

We claim that software that matches, or aligns with, work domain structure will function better than software that does not. While a software system that does match can be defeated by bad execution, we believe a mismatching system is unlikely to be effective, whatever the details of its execution. We used the domain structure to develop prototype, planning software (called NEW) that does a better job fitting the constraints specified by the domain structure than does the LEGACY software currently in use. By identifying the key function as planning (rather than form- or document-editing on the one hand, or engineering on the other), we selected a planning framework available at NASA Ames, SPIFe (Scheduling and Planning Interface for Exploration) [16] with time-line representation. This was configured and modified to provide hierarchical representation of Increments, Activities, and Actions.

NEW was designed to cover the plan revision functions of LEGACY. (NEW did not cover the process of sending and receiving edited files to the Russians.) All revision tasks possible in one are possible in the other, but with very different cost. The resulting design of NEW also supports "for free" functions beyond those provided by LEGACY. Specifically, work thinking about or designing a plan is better supported by NEW; in the current system, operators would need to do these functions "in the head" or with tools outside LEGACY. We mention this to fill out the 'big picture' view of the functionality of the two systems.

High-Level Comparison of Software-to-Structure Match

Figures 2A & 2B illustrate the difference in match for the NEW and LEGACY systems. Each shows the three key elements, the part-whole and temporal relations, and the operations on element, which were identified in the DSA. The left panel shows that only a subset of the elements and relations are supported for LEGACY. Representation of high-level structure is impoverished: no Increment representation, no relationships between Activities, and few operations on Activities. Further, representation of Activities and all temporal relations are minimal. The right panel summarizes the match provided by the NEW prototype. All the elements and relations are represented, and almost all the operations. Comparing the information architecture and interaction design of the two systems, and how each system aligns with the domain structure can flesh out this high level characterization.

Figure 3 shows the primary window of the LEGACY system, used by ADCO to build and revise plans. The plan for an activity is represented in a UAF (unified ACR file), which is the file that lists the ACR's (actions) making up the UAF; each ACR represents an action in the activity. LEGACY acts as a form editor of UAF files. The upper left panel displays and edits metadata about the activity represented in the open UAF file. The lower left panel is a view-only window displaying the descriptions of about two actions (ACR's); the active action is highlighted. The upper right panel displays the active action, allowing the user to change the action's status and to move through the file by selecting the next or previous actions. The bottom right panel is the primary work area for editing the plan. It displays the attribute values for the selected action and allows the user to type in or select values for each of the action's attributes. When finished editing one action, the

user saves that action to the buffer displaying the UAF; when finished editing one activity, the user saves the UAF to the file system. Aspects of managing versions and file names are handled automatically.

Mapping the representation and operations available in LEGACY to the abstract domain structure shows up its limitations. First, it provides no explicit representation of an increment, of multiple activities, or of relations between activities. The resulting lack of context means little support is provided for understanding or interpreting activities or their component actions. We lack detailed observation, but broadly LEGACY is used to integrate information determined other ways, rather than aiding in conceptual planning or checking. The interpretive work of the user takes place in the head or with other tools. Elements that are not represented cannot be operated on. In addition, though activities are minimally represented (as files), inside LEGACY files can only be selected or saved: it is not possible to edit an activity as a unit. For example, when an activity or set of activities slips, e.g., due to a delayed launch, the times of each action must be individually changed. This is a major frustration point. ADCO operators report trying to build their own calculators that will at least derive the values needed when times are offset by some increment. Second, there is no visual representation of time. While real numbers are indeed ratio scale as needed to appropriately represent time, representation only as text rather than also in an analog, perceptual representation means that understanding or changing time relations is a high-effort cognitive calculation rather than one supported by perception [6,23]. In sum, representation of the overall structure is very incomplete and representation of structure components (e.g., temporal relations) is far from ideal.

We illustrate how LEGACY is used with a typical revision task, of the sort used in our study: shift the time of the docking maneuver in a Soyuz docking activity an hour later. 1) Using LEGACY'S browsing window, navigate through the file system to find the file for the correct activity. 2) Select the file to open the edit window in Figure 3. 3) Scroll through the file to find and select the intended action. 4) In the edit panel change the Start Time and End Time to an hour later. 5) Click the Revise-ARC button in the lower right panel to enter the action; then click the Generate-UAF button in the lower left panel to save the activity file.

Figure 4 shows the window of NEW. An increment is represented as a plan, activities are expandable hierarchical events in a plan, and actions are component events. 1) The top tool bar includes functions such as zoom and undo. 2) The left panel displays the available plans, selectable by clicking. 3) The central panel provides timeline views of an increment. The top two colored time-lines are relevant to

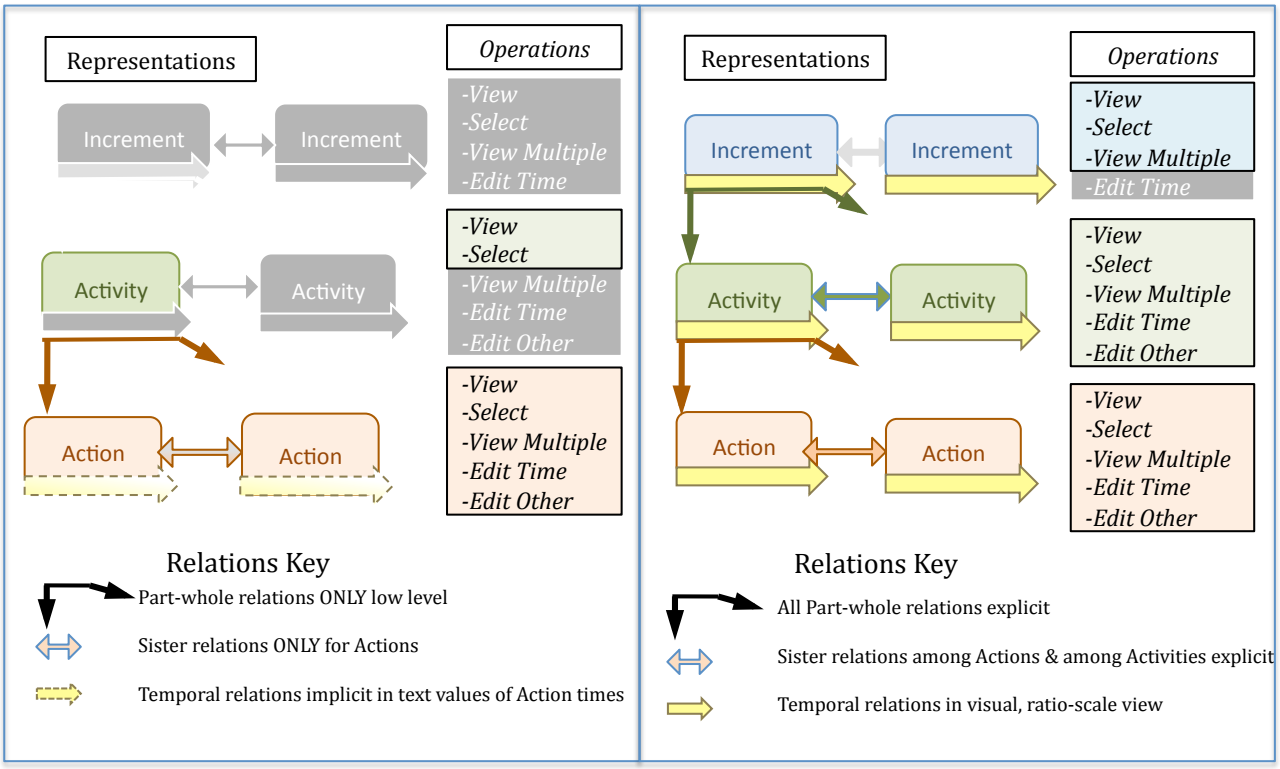

Figure 2. LEGACY system represented in Left Panel, and NEW system represented in Right Panel. Dimmed representations or operations show structure in the domain that is not expressed in the software. The NEW system provides much better match to domain structure than does the LEGACY system.

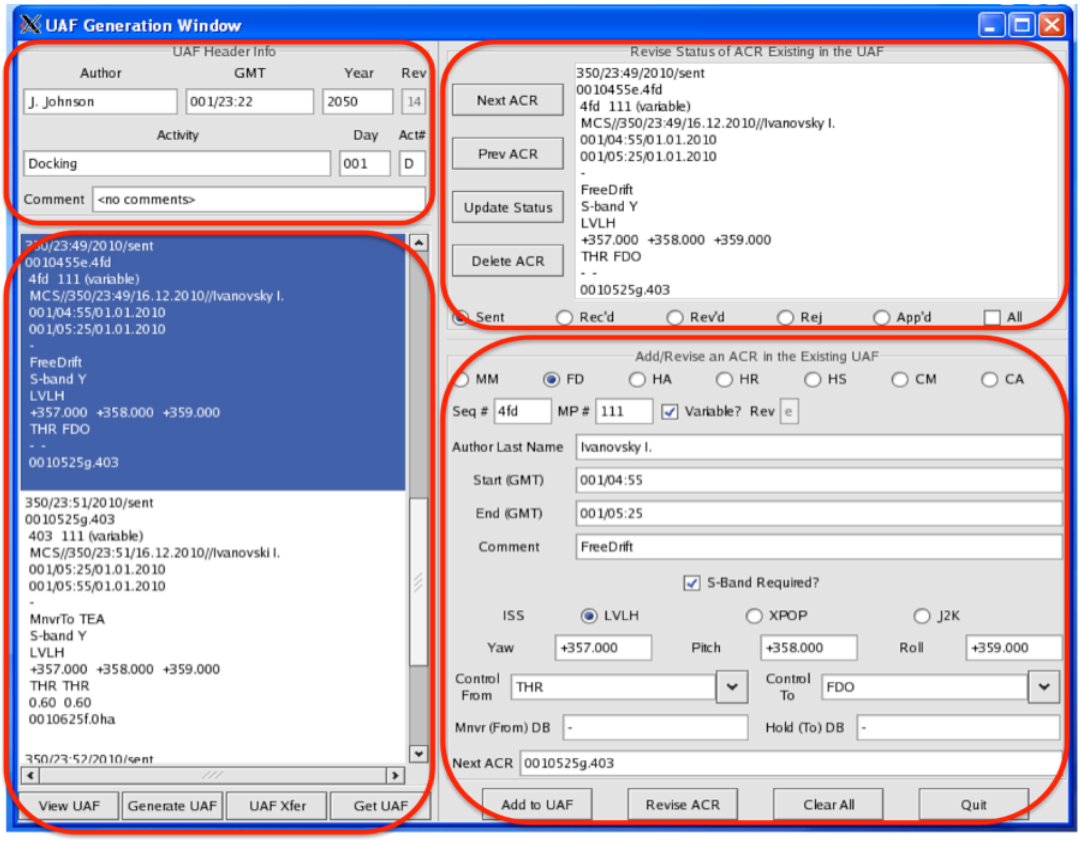

Figure 3. Screenshot of LEGACY system showing the four main function panels for editing plans. (The Actions attribute values shown here are invented, do not reflect a real event, but illustrate the types of individually possible values .)

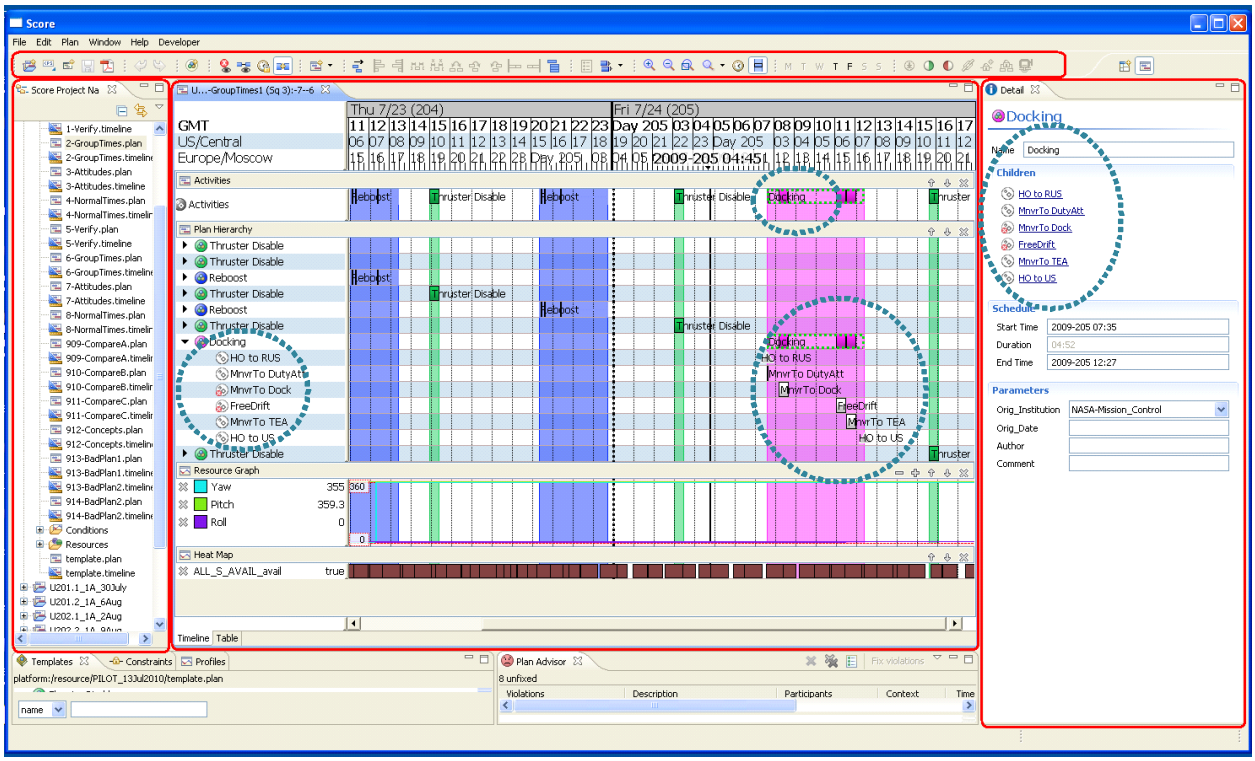

Figure 4. Screenshot of NEW system, showing the four main function panels for editing plans outlined in red. Dotted circles show 4 of the 5 repesentations of Activities

the tasks reported here. The top, *Activities* time line, summarizes events at the activity level. The second, *Plan Hierarchy* time line, provides two ways of accessing actions from activities: a) the list on the left expands the row to list the actions; b) the activity name within the timeline can be clicked to expand to show the component actions. The duration of activities and actions is indicated by their display size in the timeline. Actions and activities can be dragged and dropped to new times; a text field displays the precise time of the current start point to guide dropping. (Implementation issues limited the ease with which events could be precisely dropped.) 4) The right, Details-Edit panel allows display and edit of the attribute values of a selected event. For a selected activity, its component actions, time information, and meta-data are displayed. For a selected action, the engineering data (attitude, control, mass properties index, etc) and meta-data are displayed. In addition, collections of actions can also be selected and edited: if a constant value, say an updated Mass Properties Index, is needed for all selected actions, this can also be set through the Details Editor.

The following steps illustrate how to shift the time of the docking maneuver in a Soyuz docking activity an hour later in NEW. 1) Scan the increment plan for the intended activity, and if needed, double click on the Activity in the timeline or list at the left to open it. 2a) Drag and drop the intended action in the timeline one hour later OR 2b) Click-select the intended action, then type in the new start and end time in the Details Edit Panel. [3) Plans are automatically updated over an editing session and saved at the end of an editing session.]

Mapping the representation and operations of NEW's information architecture and resulting interface to the abstract domain structure shows the high-level correspondence. Increments are represented as Plans, setting the period displayed in the timeline. Activity and Actions have four parallel representations; Activity has an additional representation, in the *Activities* timeline. Four of the representations of activities are circled in dotted turquoise. The fifth, available for Actions as well, is a view that appears with tooltip rollover. The tooltip display (not shown) can be used to compare values of a second event with the values of a selected event, shown in the Details Edit Panel. NEW includes some additional functions not relevant to this report. In sum, the overall domain structure with all its elements and almost all relations is represented, and represented in a way that allows relationships to be quickly identified.

Predictions

The comparison of interaction designs with the domain structure generates two types of predictions. First, if one design provides better overall match than another, we predict better overall performance with the better matching system (particularly when the points of match of one system are a subset of the match points of the other). LEGACY provides a proper subset of the matches provided by NEW. Better overall match should enable tasks to be accomplished with fewer operations and should provide a more coherent model of the work domain, thus enabling better overall performance in NEW than in LEGACY. Second, the mapping identifies the locus of match or mismatch; the locus of differential match should predict the locus of greatest performance difference. Differences in representations and operations are greatest at the level Activity. In contrast to NEW, LEGACY provides only implicit representation of Activities (as a file), and no editing operations can be done at the Activity level. Rather, changes must be made individually to each action that requires a change. Further, LEGACY does not enable viewing at an Increment level or of any time span larger than one activity, thus preventing seeing an Activity in context. Thus, we predict that operations at the level of Activity will be particularly disadvantaged in LEGACY vs NEW.

We conducted a lab study comparing performance using NEW vs LEGACY. This study focused on revising plans, included a variety of tasks, and took two days. Tasks were designed both to represent the types of editing normally done by ADCO and to assess predicted differences between systems. This paper reports Day 1 data for one editing task. Preliminary analysis of the remaining edit data is consistent with the pattern of findings reported here.

V. EXPERIMENT METHOD

Participants

Participants were technical students and thus rough surrogates for ADCO trainees. The design was between-subjects with nine LEGACY and eight NEW condition participants. Degree of expertise varied within and was matched between conditions. Each condition included 3 graduate students in aerospace engineering and 1 doctoral student in another physical science, with upper level undergraduates in science or engineering as the remaining participants.

Materials & Tasks

To build the plan content used in the experiment, we reconstructed the series of events of one as-flown increment from ADCO documents. The content of our plans exactly matched the actual plan, with the following exceptions designed to make it easier for our users: 1) we divided that increment into 3 smaller, "mini-increment" plans, 2) we compressed the timing between activities, 3) we regularized the timing and naming of actions within activities. In addition, for users of LEGACY, we tried to make the file directory structure as clear, simple, and best matched to task demands as we could.

Revision of times and attitudes in response to updated information forms the bulk of actual ADCO editing tasks. Our experimental tasks also focused on time and attitudes. Our goal was to have our users engage in tasks typical of ADCO plan revision work and that also might best reveal differences between systems.

Procedure

Participants began with training about the ADCO domain, studying text and diagrams and answering questions. After roughly an hour of ADCO training in the lab, participants were trained for roughly a half hour on their respective tool (less time on the simpler LEGACY). The tool training was interactive and included "now you do it" actions for all of component functions needed. After training, the participant did two blocks consisting of 4 core editing tasks, including Group Times. After these two blocks, users did a variety of more conceptual tasks. They returned for a second day, repeating and extending tasks from Day 1. This paper reports Day 1 Group Times data.

The 12-item Group Times Task required users to reschedule the times of four different collections of events: of an activity, of several adjacent actions within an activity, of an action, and of several adjacent actions spanning activities. Table 3 details this structure. Presentation was blocked in an effort to make it easier.

We predict different patterns of difficulty in NEW vs APU. In NEW changing time of an Action or an Activity will be comparable and fast, because the user can operate on a single element. Changing the time of a group of actions, within or between activities, will be slower. In APU, changing the time of a single Action will be much faster than changing the time of an Activity. Time to make a change will be primarily a function of number of actions to change, with some cost of switching Activities.

VI. EXPERIMENT RESULTS

Differences in Overall and in Pattern of Performance

Performance on Group Times items was better in NEW than LEGACY, with average times of 59 seconds (SE=8.6) versus 124 seconds (SE=8.2) per trial. We anticipated that users would work as slowly as needed to be accurate, but in fact error rates were high at 13% for NEW and 27% for LEGACY. We did analyses with and without error responses, with and without dropping outliers, aggregating and separating blocks. We got very similar results.

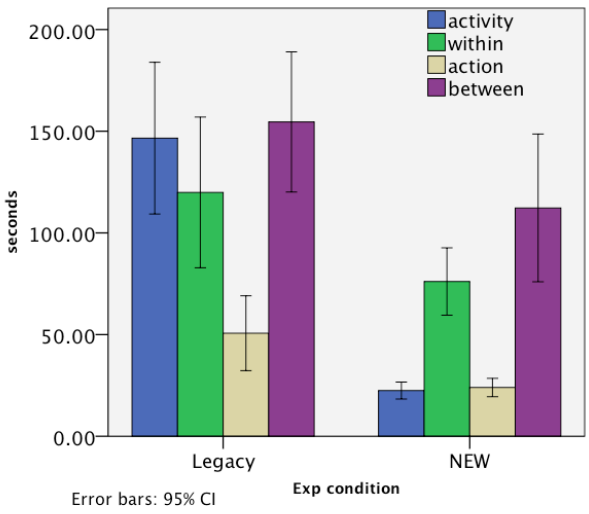

Figure 5. Response time in seconds for the LEGACY and the NEW conditions, on the 4 Group Times item types: shift an Activity, shift Actions within an Activity, shift an Action, and shift Actions spanning Activities.

Analysis with errors dropped, outliers included, blocks pooled are reported, from a 2 (condition) x 4 (item type) MANOVA.

The first question was whether overall performance was better for participants who used NEW versus LEGACY. LEGACY users took twice as long, and the condition effect on time was strong, $F(1,26)=23.20$, $p<.001$. Overall greater ease in NEW is mirrored in error rates half that of LEGACY, as well.

The second question is whether the pattern of performance differs between conditions as predicted. Response times are shown in Figure 5. The condition by item type effect is also strong, $F(1.9,50.10)=9.9$, $p<.001$, Greenhouse-Geisser adjusted. Main effect of Item type is also strong $F(1.9,50.10)=32.43$, $p<<.001$,Greenhouse-Geisser adjusted. Most importantly, the relative difficulty (measured by time) of the item types is very different, in the predicted pattern. For LEGACY changing the time of a whole activity is much slower (156 sec) than changing the time of a subset of actions (119 sec), $t(14)=2.46$, $p=.02$; while for NEW the reverse holds, with 23 sec. to change an activity and 76 secs for a subset of actions within an activity $t(14)=-6.39$, $p<.001$. Further, changing an activity takes the same time as changing an action in NEW, but is much harder than an action in LEGACY.

	Activity			Actions in Activity			Action			Actions across Activity		
trial	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Block 1	**6**	**2**	**5**	**4**	**2**	**3**	**1**	**1**	**1**	**6**	**5**	**3**
Block 2	**5**	**2**	**8**	**4**	**2**	**4**	**1**	**1**	**1**	**5**	**4**	**5**
Ave #Act	**4.67**			**3.17**			**1**			**4.67**		

Table 3. Group Times Items: *Activity* trials shift an activity. *Actions-in-Activity* shift a subset of actions in an activity. *Action* shifts one action. *Actions-across-Activities* shift consecutive actions running across activities. Cell values are number of actions in each item to be shifted, with average for each type of trial listed below. Trial shows item order.

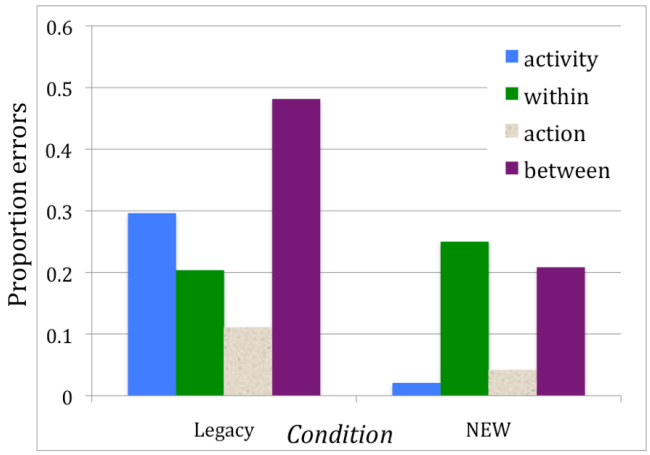

Figure 6. Proportion errors for the LEGACY and the NEW conditions, on the 4 Group Times item types: shift an Activity, shift Actions within an Activity, shift an Action, and shift Actions spanning Activities.

The error data divided by activity type and condition, in Figure 6, show the same, predicted pattern as the response times. Average times when error data were included were very slightly longer than with errors excluded.

Results Summary

We found overall faster, more accurate performance for participants using the system that better matches the domain structure (NEW), than for the system with poor match (LEGACY). Most importantly, we found the differential patterns of performance predicted by DSA-match: NEW was (equally) fast shifting Actions and Activities, and slower when multiple individual actions within an activity had to be selected; LEGACY was slow shifting Activities, faster when only a subset of Actions in an Activity needed to be changed, and, broadly, performance was faster the fewer actions to move. This NEW advantage depends on the availability of the Activity element.

Match to domain structure predicted clear, substantial impact on usefulness and usability, both overall and in determining where points of relative ease or difficulty will be encountered. In turn, this suggests the value of prioritizing domain structure analysis in conducting a needs analysis. The study provides initial evidence that the DSA approach is worth further development.

Our approach a) identified/created systems with large differences in match to DSA and b) identified specific performance patterns based on the differences in matching. Support for these predictions provides one step in supporting the claim that DSA distinctively contributes to building better systems.

From a practical perspective we found that time needed for editing tasks of the sort done by ADCO was cut in half and errors comparably reduced with the NEW system. These dramatic differences in performance are important, and we hope they will contribute toward building better tools for ADCO.

VII. CONCLUSIONS

Implications

The research we reported here illustrates how DSA can guide the whole development cycle. We ran through the cycle, in miniature, from needs analysis through design, development, and evaluation. A domain structure analysis identifies domain organization and the elements and relations composing that organization. An explicit, external, shared representation of this information is critical to develop successful software. Interaction structure of a candidate design can be matched to the DSA; alternative designs can be compared for extent and locus of how each matches to the DSA. Match and mismatch can generate predictions about performance, which can be tested in the target domain or in analogs that preserve domain structure.

When we carried out this cycle, we found large differences, predicted by DSA match, in an experimental analog of ADCO planning work. If validation accumulates, DSA could be applied to novel design problems without resorting to empirical test of predictions.

Relation to Prior Research

We share with many researchers the recognition that some form of Needs Analysis is critical to guide the design and/or evaluation of software and other socio-technological systems. Several interrelated approaches have been proposed, which prioritize different types of information yet overlap considerably, e.g., task analysis may reveal information about domain ontology through the objects and agents involved in the tasks. We believe Domain Structure Analysis can be efficiently and explicitly derived and is particularly valuable for high-stakes, safety-critical, problem-solving information work.

Our empirical evaluation compared a legacy system to one designed to match the DSA, and found dramatic advantages as predicted by DSA match. The study was ambitious in that it sought evidence about the overall design process and resulting system. A similar goal motivated an empirical comparison of systems with displays that provided better and worse match to the information needs in process control, e.g. [17] cited in [3]; here the intent was to support ecological interface design and the work analysis that motivates it. Because the systems compared and the process of developing them may differ in many ways these system-wide comparisons cannot uniquely specify the reason for a found benefit: multiple explanations for a supported prediction are possible. In contrast, studies which vary an isolated aspect of match to domain structure can make more precise claims about the limited aspect investigated.

Our approach compared a heuristic pair of applications: a legacy system and a novel system that was feasible to construct from pre-existing components. Heuristic comparison is feasible in circumstances where assessing minimally contrasting sets of applications is not.

Limitations

Our goal is to assess whether system performance improves from better match to the DSA. The support from this study is tempered because other differences between systems might have contributed. We did not sample multiple designs that matched versus violated the DSA, nor did we assess minimally contrasting pairs. In a more ideal –and very expensive study-- the effect of match to domain structure would be assessed by testing multiple designs with high versus low match to DSA. Further, our study did not investigate relative effectiveness of different approaches to needs analysis. Nor have we explored whether and when a domain structure analysis may provide a sufficient needs analysis and when it must be integrated with other information. Additional investigation is called for.

Finally, a practical limitation concerning the ADCO domain must be noted. The DSA addressed a limited scope of work: that carried out by one individual. However, communication, information exchange, and negotiation are critical parts of ADCO planning work. While the prototype we developed is a very promising basis for software redesign, an extended Domain Structure Analysis for the larger scope of work would be required prior to designing a prototype adequate for actual planning.

Future work

Our underlying motivation is to understand how the process of needs analysis can be made as efficient as possible. The foundational step is understanding what information is important, in what contexts. Based on this, we are working to convert this understanding to technology. For example, we are exploring how analysis of documents might bootstrap building a characterization of domain structure; we are also exploring how hypothesized domain-structure might best be represented. We are aware of the costs, as well as the value, of needs analysis for design and evaluation. By understanding needs analysis we hope to identify most efficient methods and tools, and to reduce the burden of these effective needs analysis methods.

ACKNOWLEDGMENTS

We thank Erin Reed and Tarik Ward, the ADCO's who worked with us, taught us, and checked materials; and Steven Hillenius, Melissa Ludowise, Mike McCurdy, and the SPIFe development team for help with the software. We thank Space Human Factors Engineering for funding.

REFERENCES

1. Beyer, H., and Holtzblatt, K. Contextual Design: Defining Customer-Centered Systems. San Francisco, CA: Morgan Kaufmann (1998).
2. Billman, D., Feary, M., Schreckenghost, D., & Sherry, L. (2010) Needs Analysis: The Case of Flexible Constraints and Mutable Boundaries. In *Proc. CHI 2009*, ACM Press pp. 4597-4612.
3. Burns, C. M., & Hajdukiewicz, J. R. (2004). Ecological interface design. Boca Raton, FL: CRC Press.
4. Butler, K.A, Jiajie Zhang, J., Esposito, C., Ali Bahrami, A. Ron Hebron, R., & David Kieras. Work-Centered Design: A Case Study of a Mixed-Initiative Scheduler. In *Proc. CHI 2007*, ACM Press (2007).
5. Butler & Zhang (2009) Design models for interactive problem-solving context & ontology, representation & routines. In *Proc. CHI 2009*, ACM Press pp. 4315-4320.
6. Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). Information visualization: Using vision to think. San Francisco, CA: Morgan-Kaufmann.
7. Chilana, P.K., Wobbrock, J.O., & Ko, A.J. (2010) Understanding Usability Practises in Complex Domains. In *Proc. CHI 2010*, ACM Press pp. 2337-2346.
8. Diaper, D. & Stanton, N.A. (2004) *The Handbook of Task Analysis for Human-Computer Interaction*. LErlbaum Associates, Mahway, NJ.
9. Elliott, G., Crawford, J., Watson, M., Sanderson, P., & Naikar, N. (2000). Knowledge elicitation techniques for modeling intentional systems with Cognitive Work Analysis. *Proceedings of the Fifth Australian Aviation Psychology Symposium.*
10. Ellis, S. R. (2000). Collision in Space. Ergonomics in Design: The Quarterly of Human Factors Applications, 8(1), 4-9.
11. Feary, M., Sherry, L., Polson, P., and Fennel, K. (2003) Incorporating Cognitive Usability into Software Design Processes. In Harris, D., Duffy, V., Smith, M., and Stephandis, C. (Eds.), *Human-Centered Computing: Cognitive, Social, and Ergonomic Aspects*, Volume 3, Lawrence Erlbaum, Mahweh, NJ, p. 427-431.
12. Feltovich, P. J., Hoffman, R. R., Woods, D., and Roesler, A. (2004). Keeping It Simple: How the Reductive Tendency Affects Cognitive Engineering, *IEEE Intelligent Systems*, May-June, IEEE Computer Society Publications Office, Los Alamitos, CA, p.90-95.
13. Johnson, H., & Johnson, P. (1991). Task knowledge structures: Psychological basis and integration into system design. Acta Psychologica, 78, 3-26.
14. Kieras, D. E. (1996). Task analysis and the design of functionality. In A. Tucker (Ed.) *The Computer Science and Engineering Handbook (2nd Ed)*. Boca Raton, CRC Inc.
15. Leveson, N. (1995) Safeware: System Safety and Computers. Addison-Wesley, New York, NY, USA.
16. McCurdy, M. (2009) Planning Tools for Mars Surface Operations: Humn-Computer Interactions Lessons Learne. IEEE AC.
17. Moradi-Nadimian (2003). Cited in Burns, C. M., & Hajdukiewicz, J. R. (2004). Ecological interface design. Boca Raton, FL: CRC Press.
18. Perrow, C. (1984). Normal Accidents: Living with High-Risk Technologies. New York: Basic Books, Inc., Publishers.
19. Rasmussen, J., Pejtersen, A. M., & Goodstein, L. P. (1994). Cognitive systems engineering. New York: Wiley.
20. Schraagen, J. M., Chipman, S. F., & Shalin, V. L. (2000). Cognitive task analysis. Mahwah, N.J.: L. Erlbaum Associates.
21. Vicente.K. (1999). *Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-based Work*. LErlbaum Associates, Mahway, NJ.
22. Zacks, J. M. T., B. (2001). Event structure in perception and conception. Psychological Bulletin, 127, 3-21.
23. Zhang J. (1996). A representational analysis of relational information displays. *International Journal of Human-Computer Studies*, 45, 59-74.