RC24949 (W1002-054) February 19, 2010
Computer Science

IBM Research Report

Are Flexible Modeling Tools Applicable to Software Design Discussions?

Harold Ossher[1], Bonnie John[1,2], Michael Desmond[1], Rachel Bellamy[1]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[2]Carnegie-Mellon University
Human-Computer Interaction Institute
5000 Forbes Avenue
Pittsburgh, PA

Are Flexible Modeling Tools applicable to Software Design Discussions?

Harold Ossher[1], Bonnie John[1,2], Michael Desmond[1], Rachel Bellamy[1]

[1]IBM T.J Watson Research Center
19 Skyline Drive
Hawthorne, New York 10532

[2]Carnegie-Mellon University
Human-Computer Interaction Institute
5000 Forbes Avenue, Pittsburgh, PA

Abstract

We have been developing an approach to modeling which we call flexible modeling. Our aim is to support the work of pre-requirements analysts whose activities are characterized by informal use of representations and evolution of both representations and concepts. However, we believe that the features of flexible modeling tools have more general applicability to tasks involving conceptual development. Furthermore we believe that design is such a task. This paper investigates whether the characteristics that make flexible modeling useful and usable for the work of pre-requirements analysis might also make it a good approach for supporting the kinds of discussion that take place around a whiteboard in the early phases of a software design project.

Keywords-flexible modeling; design discussions; concerns

I. Introduction

We have been developing an approach to modeling called flexible modeling [3][4]. Flexible modeling developed out of our observations that business users working on pre-requirements analysis, prefer to use office tools such as Microsoft Word, Powerpoint and Excel, rather than modeling tools. Office tools are usually used for many good reasons. However, these tools are semantics free, are limited in organizing information, and do not have an underlying domain model. Consistency can only be maintained manually, and even small changes can take a lot of effort to propagate. Practitioners reported that entire days are often spent on the mechanics of maintaining consistency, which is not only time-consuming, but also disrupts flow. Migrating results from these tools to downstream modeling tools, which are founded on underlying domain models, is also a manual process, and traceability is usually lost.

To support such users, we developed flexible modeling tools that blend the advantages of office and modeling tools. A flexible modeling tool would allow users to work freely and easily with visual elements, yet be able to attribute semantics to visual characteristics when necessary, enabling automatic construction and maintenance of an underlying model and management of consistency. Given suitable domain-specific definitions, the tool provides guidance and checking, but without requiring strict conformance to a rigid metamodel. Users can thus move smoothly between informal exploration and modeling with varying degrees of formality and precision.

Flexible modeling tools support this style of activity by providing specific features:

- A visual layer providing multiple views, in which the user can work with much of the freedom of office tools.
- An underlying model consisting of related visual and semantic sub-models, enabling visual cues to be given semantics.
- A forgiving approach to domain-specific guidance, with structure definitions specifying structural constraints used to check for structural violations and provide assistance. When provided as a package, they effectively define a meta-model. Models that violate them can, however, be created, manipulated and saved.
- Refactoring support to allow convenient reorganization.
- A presentation layer supporting the synthesis of presentations from working views.

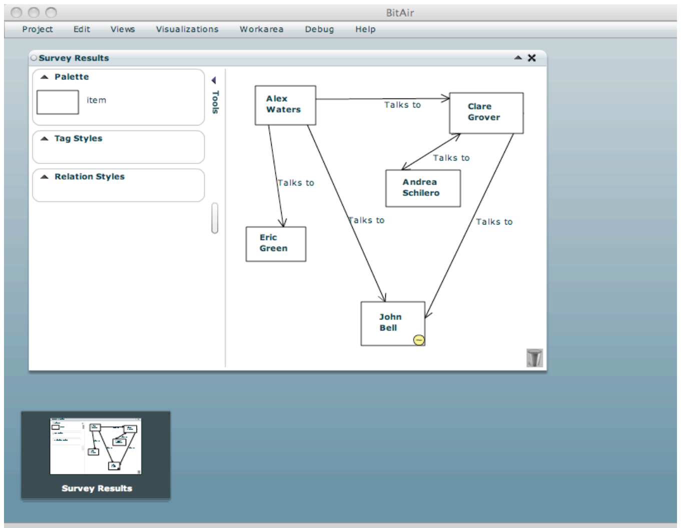
We believe that the style of activity flexible modeling tools aim to support—moving between informal exploration and more formal modeling—is characteristic of many kinds of early design activities. This paper investigates whether it is also characteristic of software design discussions that take place at a whiteboard. Our investigation consists of an informal analysis of three software design discussions that took place using a whiteboard. Our analysis focuses on how representations and the whiteboard is used during the identification and development of concerns. This focus limits the scope of the analysis such that obvious issues such as running out of space on the whiteboard are not discussed. Our consideration of flexible modeling is not concerned with the detailed issues of how to design a usable flexible modeling whiteboard tool, but focuses on whether flexible modeling would be a useful and suitable approach.

The structure of this paper is as follows. First we briefly describe BITKit, an example of a flexible modeling tool. Then we talk about the analysis methodology we used, next we discuss what we observed about the design process. We conclude by discussing implications of our findings for the application of flexible modeling tool features to software design discussions.

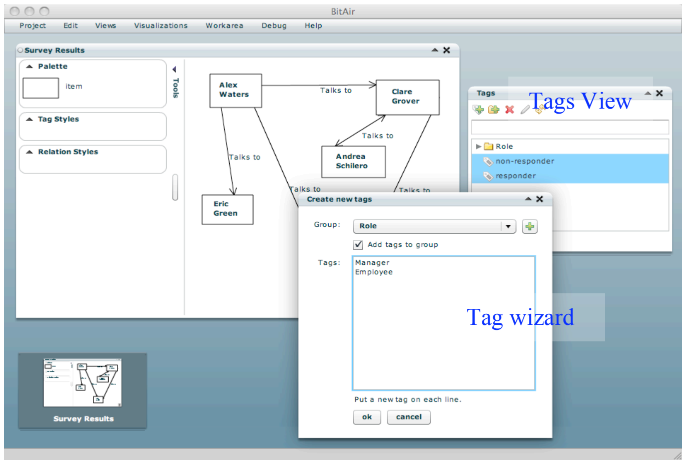
This research was sponsored in part by the National Science Foundation under grant CCF-0845840 .

II. BITKIT: AN EXAMPLE OF A FLEXIBLE MODELING TOOL

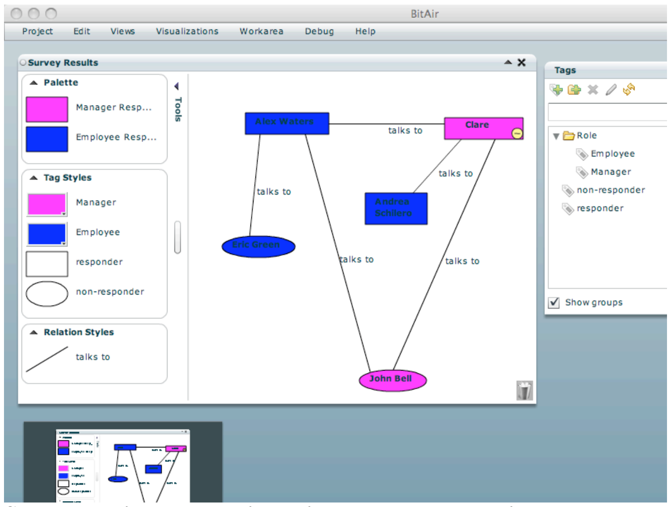
We describe BITKit, to illustrate how flexible modeling can be embodied in a particular tool. This is an example of flexible modeling concepts, **not** of a flexible modeling tool for an electronic whiteboard; differences in the affordances of whiteboards would demand a very different user interface. BITKit was designed to support the activities of pre-requirements analysts: gathering information, organizing it to gain insight, envisioning alternative possible futures, and presenting insights and recommendations to stakeholders. The storyboard frames below describe an hypothetical business user uses BITKit to explore communication issues within her organization. Her insights will later be used to help formulate requirements for new collaboration tools for the organization.



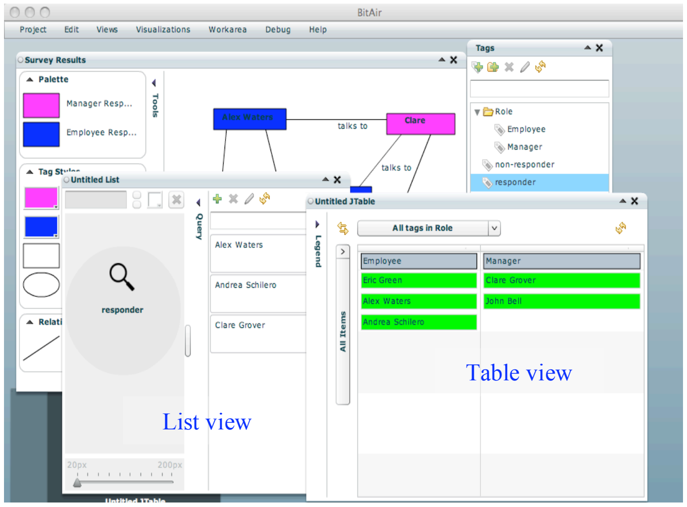
Creating views, just like office tools—In this typical BITKit workspace our user is creating a diagram just as she would using a typical diagramming tool. She is using the diagram to represent data about communication patterns. She collected this data from her group members using an informal survey. Just like in typical diagramming tools, she can add items to the diagram from a palette, and label them. She has added items representing all the people who responded to the survey and all the people they said they talked to. She labeled these items with the people's names. She has also added lines between the people items to show who talks to whom.



Emergent structure—By tagging items in the diagram, our user can readily start to add structure. So when she realizes she needs to distinguish between people who answered the survey and those who didn't, she does so by creating a 'responder' tag and a 'non-responder' tag. When she also realizes that she wants to distinguish the managers from the employees, she does so by creating two additional tags, 'Employee' and 'Manager'. To structure her tags further, she creates a tag group called 'Role' and places the 'Employee' and 'Manager' tags into this group. In the previous screenshot we see her creating the 'Role' tag group and its tags in a tag wizard, which she accessed after first opening the tags view.



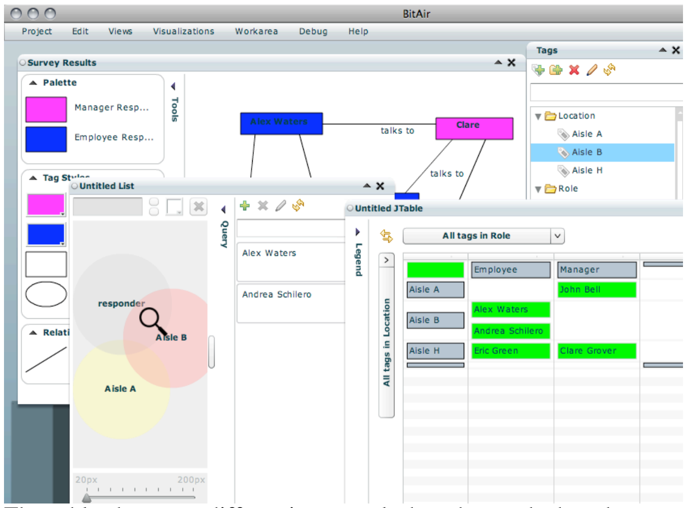
Style mapping to associate visual appearance with model structure—Tags and visual styles can be associated with each other. One way this is done is using the legend on the left side of the diagram, where a user can add styling elements such as color and line width, and then select the tag to associate with that style. Here we can see that our user has associated colors with the 'Manager' and 'Employee' tags, so that manager items and employee items are visually distinct in her diagram.



Employee	Manager
Eric Green	Clare Grover
Alex Waters	John Bell
Andrea Schilero	

Multiple views—To get a list of just those people who responded, for example, our user opens a list view and uses the Venn Diagram visual query to view just those items that are tagged with 'responder'. She also opens a table view and sets the columns to show the tags in the tag group 'Role'. This allows her to tabulate which people are employees and which are managers

Using tags and tag groups, she introduces another semantic distinction, the office location; which she shows as rows in her table. Using the drag and drop features of the table, she drags each of the people into the row that corresponds to their office location. Without having to think about the underlying model, but just by manipulating the table structure, she has effectively categorized the people items in the underlying model.



The table does not differentiate people based on whether they responded or not. Because her list is also a view on the same underlying model, she can now modify the query, as shown above, to explore the office locations of the people who responded to her survey.

In this paper we explore whether these same underlying flexible modeling approach embodied within BITKit (not the same UI) might be appropriate to support the activities of software designers as they converse around a whiteboard.

III. Methodology

We decided to analyzed the design sessions with respect to concerns because in software engineering *concerns* capture the essential concepts of a design. *Identifying concerns* is thus a process of conceptual development. The business users we designed flexible modeling tools to support were similarly engaged in identifying and developing concepts. By focusing our analysis on this same process we hope to identify opportunities for flexible modeling to support design conversations and also reveal potential mismatches.

Concerns are the primary motivation for organizing and decomposing software into manageable and comprehensible parts. Many kinds of concerns may be relevant to different developers in different roles, or at different stages of the software lifecycle. For example, the prevalent concern in object-oriented programming is the *class*, which encapsulates data concerns. Appropriate separation of concerns has been hypothesized to reduce software complexity, improve comprehensibility, promote traceability, facilitate reuse, non-invasive adaptation, customization, and evolution, and simplify component integration.

Though all software engineers are familiar with separation of concerns, definitions of the term "concern" are hard to come by and are often inadequate. Following Sutton and Rouvellou [9], we use the IEEE definition of concerns as "... those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders" [2]. In studying concerns in the design sessions, we will therefore be studying such "interests," including key concepts in the domain (e.g., road), user interface elements (e.g., visual map), model elements (e.g., Road class), quality of service issues (e.g., number of intersections supported), and so on.

In the next section, we describe how we produced an initial concern model from the design prompt and encoded the design sessions using the concern model.

IV. Concern Model and Encoding

Many concerns that must be addressed by developers arise during requirements analysis; others arise as development proceeds. The design prompt provided to all three design teams takes the place of a requirements document in this context, and was therefore treated as the initial source of concerns. The design sessions both dealt with concerns raised in the design prompt and introduced new ones.

In this section, we describe the concern model and how it was produced initially from the design prompt, and the encoding of the design sessions and how the concern model evolved during this process.

A. Producing an Initial Concern Model

We began by producing an initial concern model [6][9] from the design prompt. One author analyzed the prompt phrase by phrase, using both the language of the prompt and knowledge of the domain to identify concerns to produce a draft. As subsequent discussion will reveal, there turned out to be considerable subtlety involved in this process, despite the simplicity of the design prompt.

Initially, a simple list of concerns was produced, mapping closely to the phrases in the design prompt. For example, consider the first phrase describing requirements: "Students must be able to create a visual map of an area." Taken as a whole, this phrase expresses a concern that the user interface provide users with the ability to create a visual map. The sub-phrase "visual map" expresses a concern that there *be* a visual map. Consideration of the domain indicates that the visual map depicts a network of roads and intersections. We thus ended up with five concerns in our first version of the concern model derived from this single phrase:

1. Road
2. Intersection
3. Network of roads and intersections
4. Visual map
5. Creation of visual map

It quickly became clear that concerns pertain to different *domains*. This is illustrated by the example above. The first three concerns pertain to the *physical domain*: roads and intersections in the real world, knowledge of which allowed everyone to understand the problem. The last two concerns are in the user interface domain, since they involve the visual map presented to the user. The fourth involves the *presentation* to the user whereas the fifth involves giving *control* to the user.

After initial discussion and refinement of the draft, two of the authors examined short portions of two of the session transcripts to perform calibration encoding, associating concerns with utterances as described below. It soon became clear that the list of concerns was becoming too clumsy. The designers often dealt with the same concept across different domains. The last three concerns shown earlier are of this form: they all address the concept of a map (or the underlying network of roads and intersections), but in three domains: physical, user interface presentation and user interface control. The design prompt was short enough that the list based on it was manageable, but once we began examining the transcripts, the list started becoming the unwieldy cross-product of the set of concepts and the set of domains.

We therefore decided to restructure the concern model to reflect this two-dimensional nature directly. Together, we went through the concerns we had already identified, and separated them into (concept, domain) pairs. We then discussed the concepts to determine which were subsumed by others, and pared the list accordingly. In the end, we reduced the initial concern model to 15 concepts and 5 domains, shown in Table 1 as originating from the design prompt. A concern henceforth means a (concept, domain) pair.

Concepts	Domains
▪ avoid crash producing combinations ▪ intersection ▪ lane ▪ lane sensors ▪ map ▪ protected left-hand turn ▪ road ▪ road length ▪ traffic density ▪ traffic flow ▪ traffic flow simulation ▪ traffic light ▪ traffic light behavior ▪ ui layout ▪ ui modes	▪ physical domain ▪ model ▪ ui control ▪ ui presentation ▪ quality of service

TABLE I. ORIGINAL CONCEPT AND DOMAIN CONCERN CODES

B. Encoding the Design Sessions

We encoded the design sessions for three kinds of information: concerns being dealt with, activities being performed by the designers, and types of representations drawn by them on the whiteboard. The transcripts were used to encode the concerns, with the videos themselves consulted when necessary to resolve uncertainty. The videos were used directly to encode the activities and representations. After describing these in the next three sections, we discuss issues of consistency and reliability.

1) Concern Encoding

With the initial concern model in hand, three of the authors each encoded a transcript of a design session (Adobe, Amberpoint or anonymous). The encoding involved associating one or more concerns with each utterance.

Two different granularity issues arose immediately. The transcripts provided are broken up into fairly large chunks, divided only by speaker transitions, and time boundaries were provided only at this level of granularity. A single such chunk often deals with multiple concerns. However, for this exploratory study we simply mapped each chunk to all applicable concerns. This resulted in some potentially-interesting detail being lost, such as the sequence in which the concerns were mentioned and which speaker activities accompanied which of the concerns. It is worth noting that, even in the case of fine-grained utterances, it is sometimes necessary to associate multiple concerns with a single utterance, as in the case of the design prompt phrase discussed earlier.

The second granularity issue is granularity of concepts, and hence concerns. For example, consider the following statement about traffic lights in the anonymous session: “So each one’s got a state either red, yellow, green.” *Traffic light state* makes sense as a concept, introduced by this statement. However, the statement is also covered by the more general concept *traffic light behavior.* In this and similar situations, we chose to use the existing, coarser-grained concept, and only to introduce new concepts in cases that did not seem to be covered at all by concepts identified previously. This made the encoding task more manageable, but at the cost of losing information. A fine-grained approach, in which the *traffic light state* concept would be introduced and modeled as *contributing to* [9] [7]*traffic light behavior* would lead to a more precise concern model, and would be of benefit in a study of concern lifecycle: it would be interesting to examine what fine-grained concerns are raised in the course of discussing a coarse-grained concern, and whether and how they become important or ignored. This is left for future work.

The other author encoded the summary presentations given by the groups at the end of each session, to determine what concerns were surfaced during these presentations.

Encodings of concerns were performed using spreadsheets, with one utterance per row. Times, speakers and utterance text were imported from the transcripts. The end time of an utterance was defined as the start time of the next utterance, an approximation that allows us to study durations, but again looses detail about silences. Additional pairs of columns were used as needed to record concerns as (concept, domain) pairs. This was done with reference to lists of concepts and domains – the concern model – maintained in a shared GoogleDocs document. In two cases the document was updated with new concepts and, very occasionally, domains, so that only concepts and domains in the concern model were used as codes; in the third case, a separate list of new concepts was maintained.

2) Activity Coding

We wished to capture the activities performed by designers as they were working at the whiteboard, to enable us to correlate these with concerns being discussed. Table 2 shows the activity codes we used. These were developed incrementally during the encoding process, and shared using the GoogleDocs document mentioned above.

Activity	ActivityDomain	Representation
▪ Reading design prompt ▪ Pondering ▪ Marking ▪ Gesturing ▪ erasing	▪ physical domain ▪ model ▪ ui ▪ use cases ▪ metrics ▪ presentation	▪ list ▪ diagram ▪ paragraph ▪ sketch ▪ table ▪ ui controls

TABLE I. FINAL ACTIVITY, ACTIVITY DOMAIN AND REPRESENTATION CODES

To perform this encoding, the design session videos were imported into ELAN [http://www.lat-mpi.eu/tools/tools/elan], a video annotation tool. The videos were viewed in ELAN, and the activities coded. Unlike the coding of the transcript, the coding of the activities was not restricted to the timings of utterances, but could use begin and end time as needed.

The activity domains were encoded based on what was represented on the whiteboard, whereas the concern domains were encoded based on what was said, though actions and representations were consulted to resolve uncertainty. Also, a single UI domain was used for activities, rather than trying to distinguish presentation from control, since this could generally not be done based on what appeared on the whiteboard.

3) *Limitations*

We attempted to ensure that codes were being used consistently. We had several face-to-face discussions early on, discussing and refining the initial concern model. We then performed calibration encoding on small portions of the session transcripts, with face-to-face discussions of the results, leading to adjustments. As encoding proceeded, we had occasional discussions about what we were finding. We used the GoogleDocs document to share the evolving sets of codes as they grew, though in the case of the Adobe session, a separate list of new concepts was made. We used spreadsheet facilities (data validation or auto-complete) and post-processing as an aid to ensuring that only codes from the lists were used.

As noted above, however, concern encoding proved subtle, with most cases open to interpretation. The primary difficulties were with concepts, especially deciding whether an utterance really introduced a new concept or should be associated with an existing, coarser-grained concept. There were also cases, however, where the domain encoding was unclear. In particular, it was sometimes difficult to tell whether the designers were talking about the physical domain or the model. We consulted the videos themselves to try to resolve such issues, rather than relying just on the transcripts, but even so some cases were unclear. Similar issues of interpretation were also discovered for activity encoding.

The encoding that we did would really be just the first step of a full study. We would need considerable further discussion to produce an en*coding manual*, with definitive lists of codes and explanations of how to use them, how to handle granularity issues and simultaneous activities, and so on. This coding manual would then be used to encode a larger body of data, with further discussion and enhancement if too many cases were found not to be properly covered by the manual. Given the data set and timeframe of this exploratory study, such a full approach was not possible. We did have an agreed-upon initial concern model from the design prompt, but did not have time to iterate sufficiently during the encoding to have well-defined additional concerns. The result is that we do not have inter-rater reliability, and, since our understanding evolved considerably during the encoding process, there are probably some inconsistencies within the encoding done by each of us. Any patterns, similarities or contrasts observed in the visualizations should, therefore, be treated as suggestive only, pointing to areas that might be worthy of more rigorous study.

V. VISUALIZATIONS

The approach we took to exploring the coding we had done of the design transcripts was to create visualizations to help us see patterns in the data. We used several tools to create these visualizations. Our initial visualizations were done using ManyEyes [http://manyeyes.alphaworks.ibm.com/manyeyes/], which provides a variety of interactive visualizations. In some cases, we found that ManyEyes does not currently provide the visualizations we needed. The primary example was timelines, showing the sequences of concerns and related activities and representations. To visualize these, we used MacShapa [http://www.itee.uq.edu.au/~macshapa/].

With these visualizations in hand, we looked to see how concerns were used in the design process and how they related to specific design activities and representations used. Knowing that this study has many limitations, we looked for broad effects that we couldn't explain due to limitations in the method or differences between the coding done by the coders. Below we describe our exploration of the visualizations and what we noticed relevant to flexible modeling.

VI. CHARACTERIZING SOFTWARE DESIGN DISCUSSIONS

A. *Concerns evolve they do not only originate from requirements documents*

Visual inspection of timelines of concepts shown in Figure 1 reveal that concepts do not only originate in the design prompt, but evolve over time. In this Figure the concepts in the top section were inferred from the Design Prompt. The concepts in the second section were added by all three teams during their design sessions, the concepts in the third section were added by two of the three teams during their design session, and those in the bottom section were added only by the team in whose timeline they appear (the anonymous team had no such concepts). Although some variation is undoubtedly due to encoder variability, the concepts added by all three teams suggest that not all valid concepts can be inferred from design requirements (in this case, the Design Prompt) and any tool centered around concerns will have to allow design teams to add new ones as they work.

However, when the frequency with which concepts were actually discussed is examined (Figure 2), surprisingly the majority of concepts discussed arise from the design prompt. This suggests that initially populating a flexible modeling tool's model with concerns derived from a concern analysis of requirements documents may be helpful.

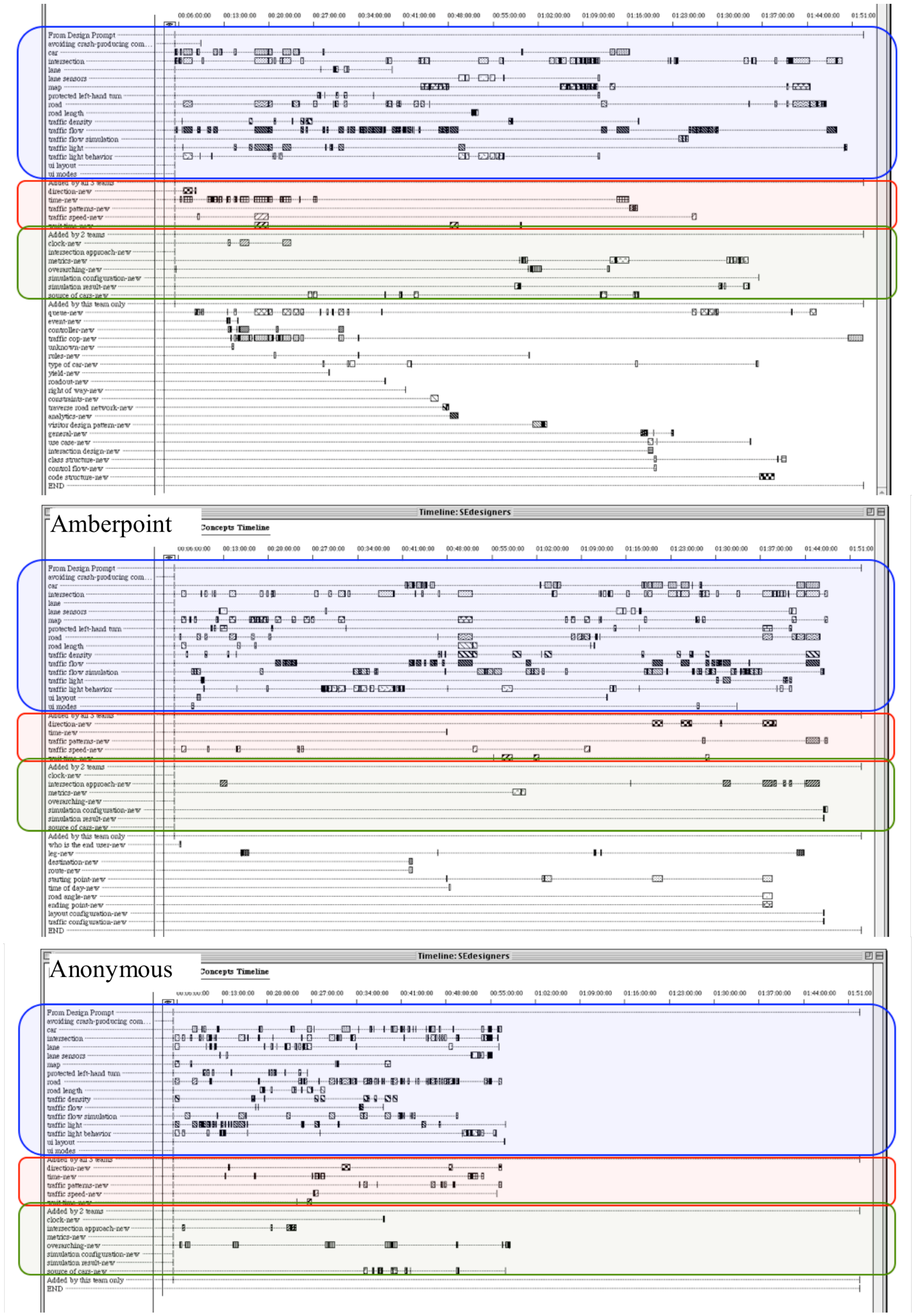


Figure 1. Timelines for concern concept codes. Showing which originated in the design prompt, which were added for partiuclar teams.

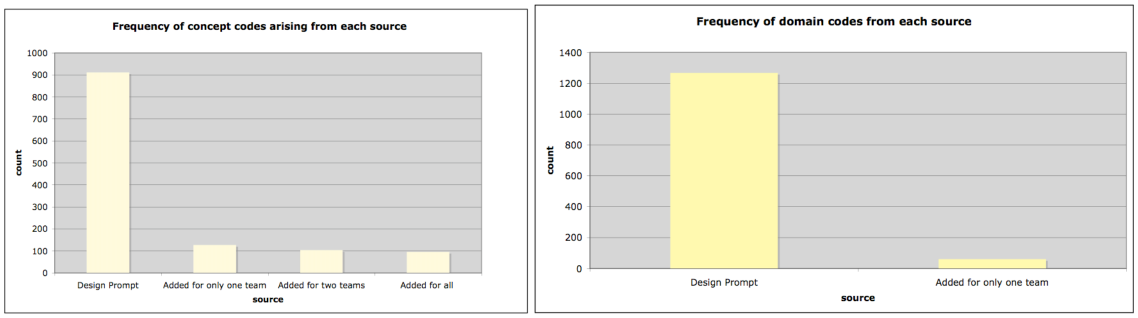


Figure 2. Frequencies for concern codes

The timelines for the concern domains (Figure 3) show that they do not evolve as dramatically as the concepts do over the course of the design sessions. In fact, only use-cases arose as a new domain in all three session and only the Adobe team discussed anything else: code complexity and the presentation they would have to give at the end of the design session. This observation that there were fewer concern domains, and most originated in the design prompt further reinforces the possibility of populating a flexible modeling tool's model with concern domains derived from requirements documents.

B. Concern concepts are revisted throughout the design process

Concern concepts were repeatedly revisited throughout the discussions of all the teams as can be seen in Figure 1. Examination of the concept timelines of all three sessions shows constant and frequent jumping around among concepts. This is especially true of the major concepts introduced in the design prompt. There are a few cases where the same concept is discussed for a few minutes at a stretch, but not for very long, and even then usually in part concurrently with other concepts.

Some concepts, however, were seldom discussed, in some cases only once. This is especially true of new concepts introduced during the design sessions. White space to the right (jagged right margin) indicates that discussion of a number of concepts is dropped fairly early during the session, never to be resumed. While some of this may be attributed to encoding differences, some concerns may be resolved and therefore not talked about again. The timelines help focus our attention on when these resolutions might have come about, but further work is necessary to confirm this hypothesis. If indeed decisions are made and not revisited, a flexible modeling tool may need to denote these decisions somehow.

Revisiting concepts, going back to concepts that have previously been discussed is characteristic of discussions in general. They are not linear and well-structured, but messy and ill-structured. During these design conversations, the designers were struggling to understand the domain, and create a well-structured representation of the key concerns. This going back and forth is possibly related to the highly interrelated nature of concerns. Establishing the best organization for the design is a challenge.

Flexible modeling tools would aid such conversations by automatically capturing concepts in an underlying semantic model as these concepts were externalized by designers in representations.

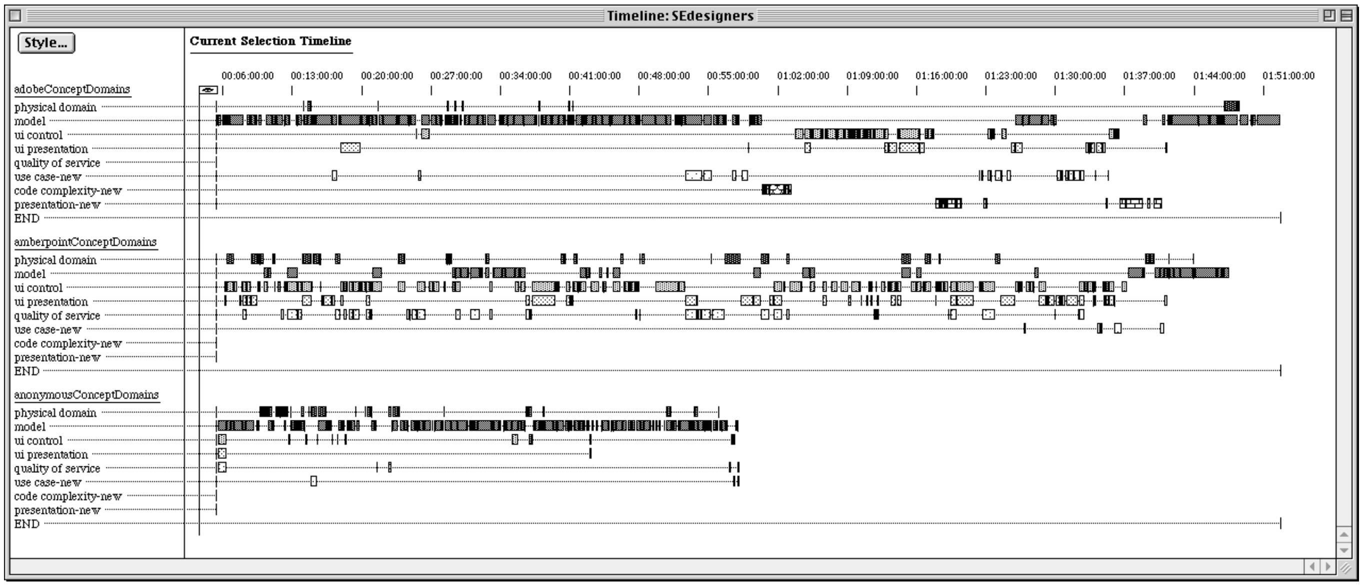
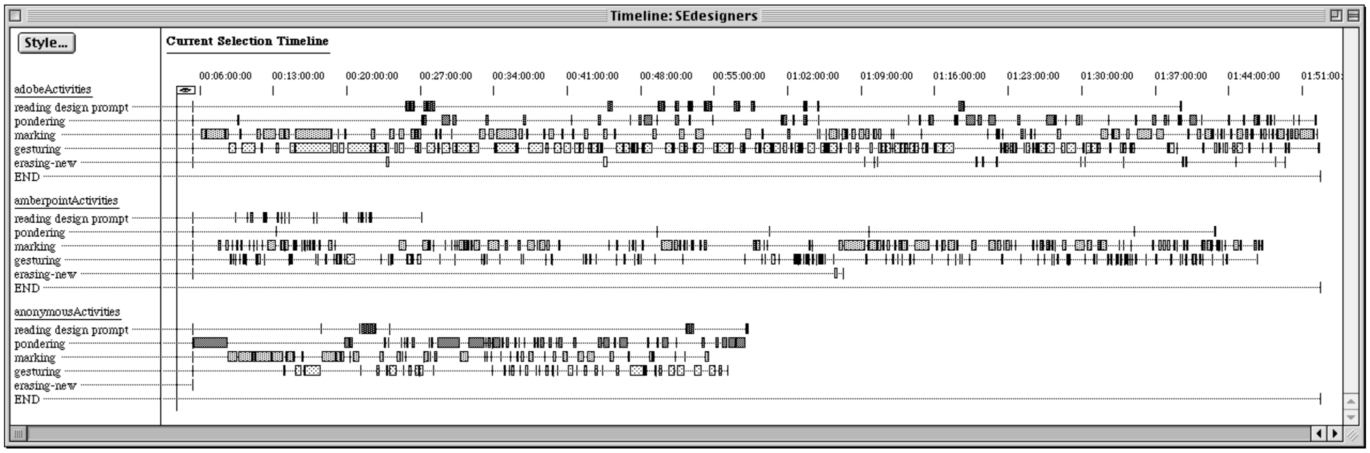


Figure 3. Timelines of the domains that were discussed by all three teams.

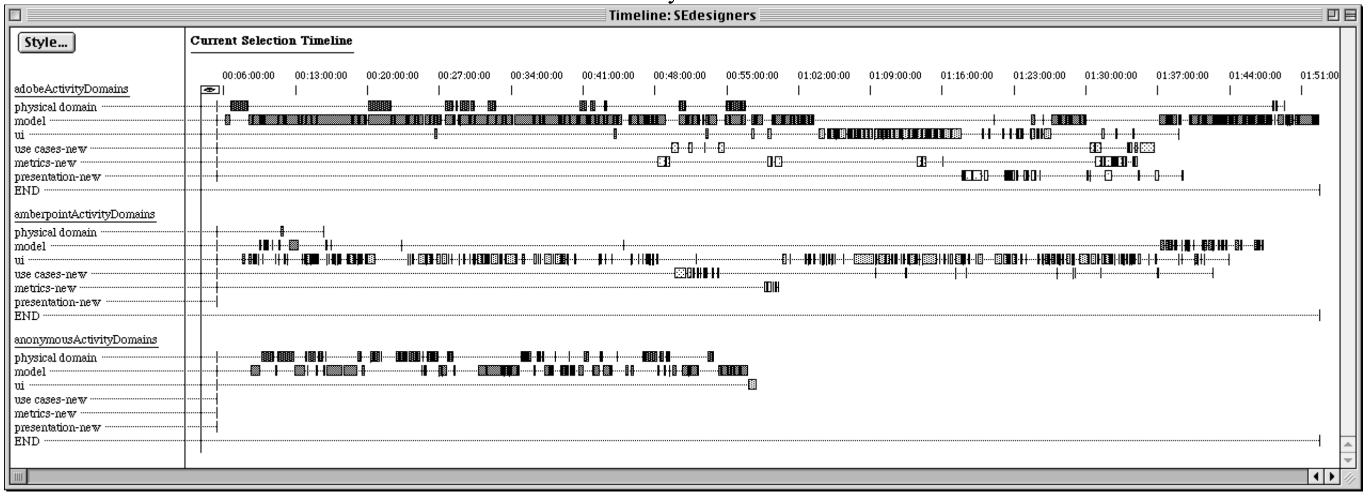
Although the designers went back and forth between concepts during the discussion, surprisingly they did not explore multiple possible organizations by drawing them on the board. All of the teams used different, but a single main organization for their design, and they developed this early on.

It is well-known that good designs result from exploration of multiple possible designs. We conjecture that flexible modeling would reduce the burden of creating a representation, and thus free-up designers to explore more possibilities in their discussions. It would also facilitate capturing and returning to previous organizations, something that is not easy to do using a traditional whiteboard.

Activities



Activity Domain



Representation

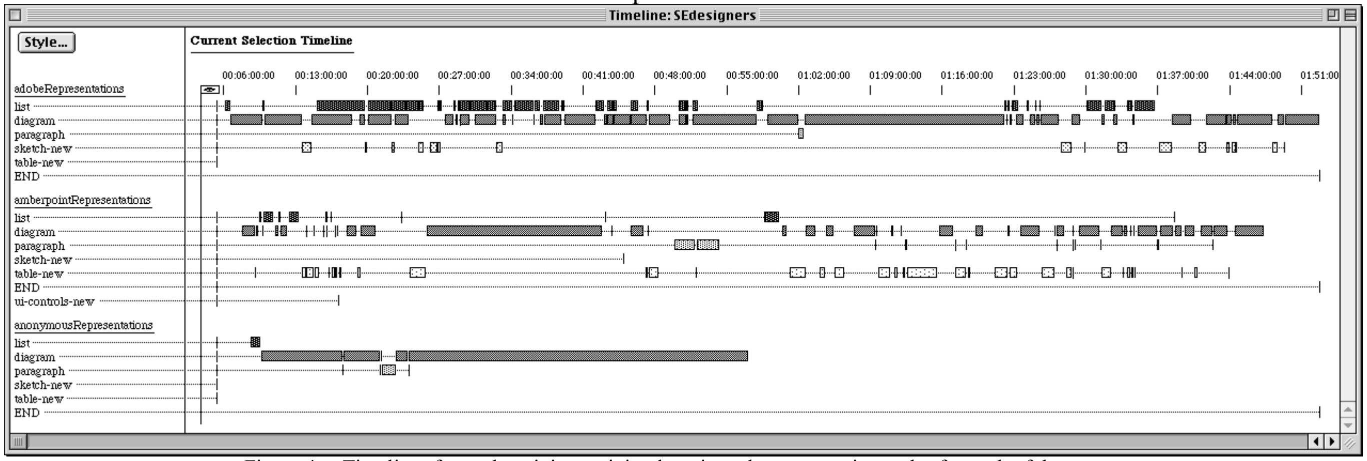


Figure 4. Timelines for each activity, activity domain and representation codes for each of the teams.

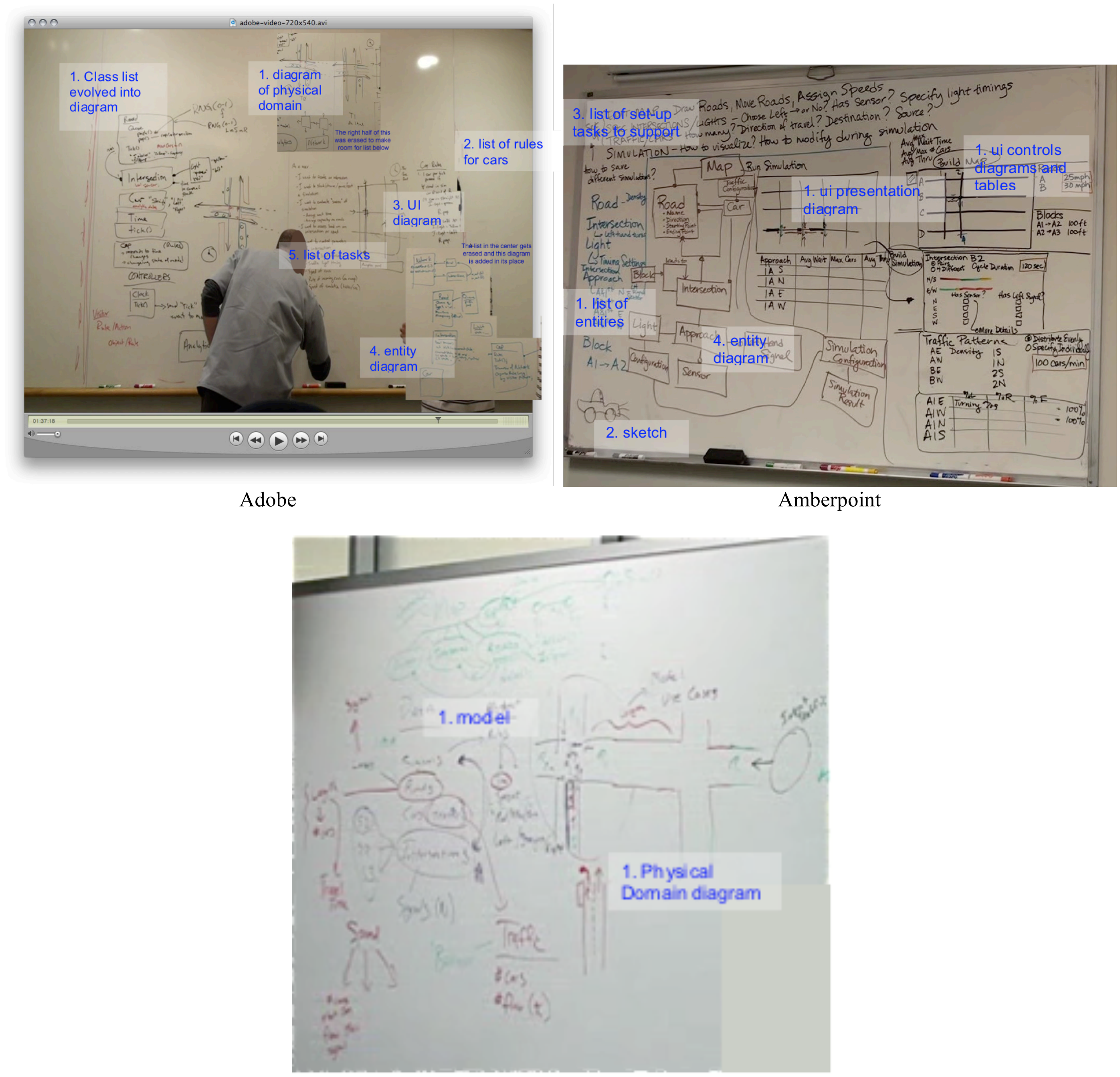

1. Class list evolved into diagram
1. diagram of physical domain
The right half of this was erased to make room for list below
2. list of rules for cars
3. UI diagram
5. list of tasks
The list in the center gets erased and this diagram is added in its place
4. entity diagram
Adobe
3. list of set-up tasks to support
1. ui controls diagrams and tables
1. ui presentation diagram
1. list of entities
4. entity diagram
2. sketch
Amberpoint
1. model
1. Physical Domain diagram
anonymous

Figure 5. Contents of whiteboards showing representations created. Numbers indicate order in which representations were predominantly used.

C. *Concern domains are different for teams with different backgrounds*

The timelines for the concern domains (Figure 3) reveal that the domains discussed are different across the three groups. The anonymous team barely touched on any aspect of the UI[1], mentioning it only at the very end. The Adobe team had clear phases of discussing mostly the model (Figure 3, 00:06:00:00–00:55:00:00), then mostly the UI (Figure 3 01:03:00:00–01:15:00:00, both control and presentation), then back to mostly the model (01:25:00:00)[2]. However, there is a much

[1] We inquired as to whether the video might have been cut short, but were assured that the data were complete.

[2] Although the anonymous team seemed to oscillate between discussing the physical domain and then the model and the Adobe team only touched on the physical domain, discussion between the encoders when looking at these timelines revealed that there might have been a difference in coding criteria for attributing an utterance to physical domain vs. model, and thus, this difference is not reliable.

larger difference between the Amberpoint team and the other two: Amberpoint discusses all of the domains inferred from the Design Prompt all the way through, with heavy emphasis on the ui control and presentation and substantial time in the physical domain, model and quality of service. After finding this pattern, we inquired as to the backgrounds of the different team members and indeed, although all participants had programming in their backgrounds, both members of the Amberpoint team currently held positions as "Interaction Designers" while the Adobe and anonymous teams were software engineers, software architects or software project managers.

By providing multiple views mapped to a single evolving semantic model, flexible modeling tools can allow teams to focus on a particular domain during any particular design conversation. That same model can then be used as a starting point for discussions by the same people (or different members of a team) that focus on a different domain. The support for mapping a single model to different views would be helpful here.

Typical design teams are multi-disciplinary with team members having very different backgrounds and often using different language and preferring different representations. Also their conversations take place across multiple meetings with different stakeholders. For such teams a single model that can be mapped to many different views allows people to work in their preferred domain, but helps maintain consistency.

D. *Mulitple representations are used*

It is clear from the analysis of the designers activities that they really did use the whiteboard heavily. This is shown in the timeline on Figure 4. Marking and gesturing occupied a considerable proportion of their time in all cases. In addition, much of the time was spent pondering, the designers looked at what was drawn on the whiteboard, though in some cases we could not tell what, if anything, they were looking at. A variety of different representations, from informal sketches and lists to somewhat more structured diagrams were used. They covered most the domains, with domain emphasis, unsurprisingly,

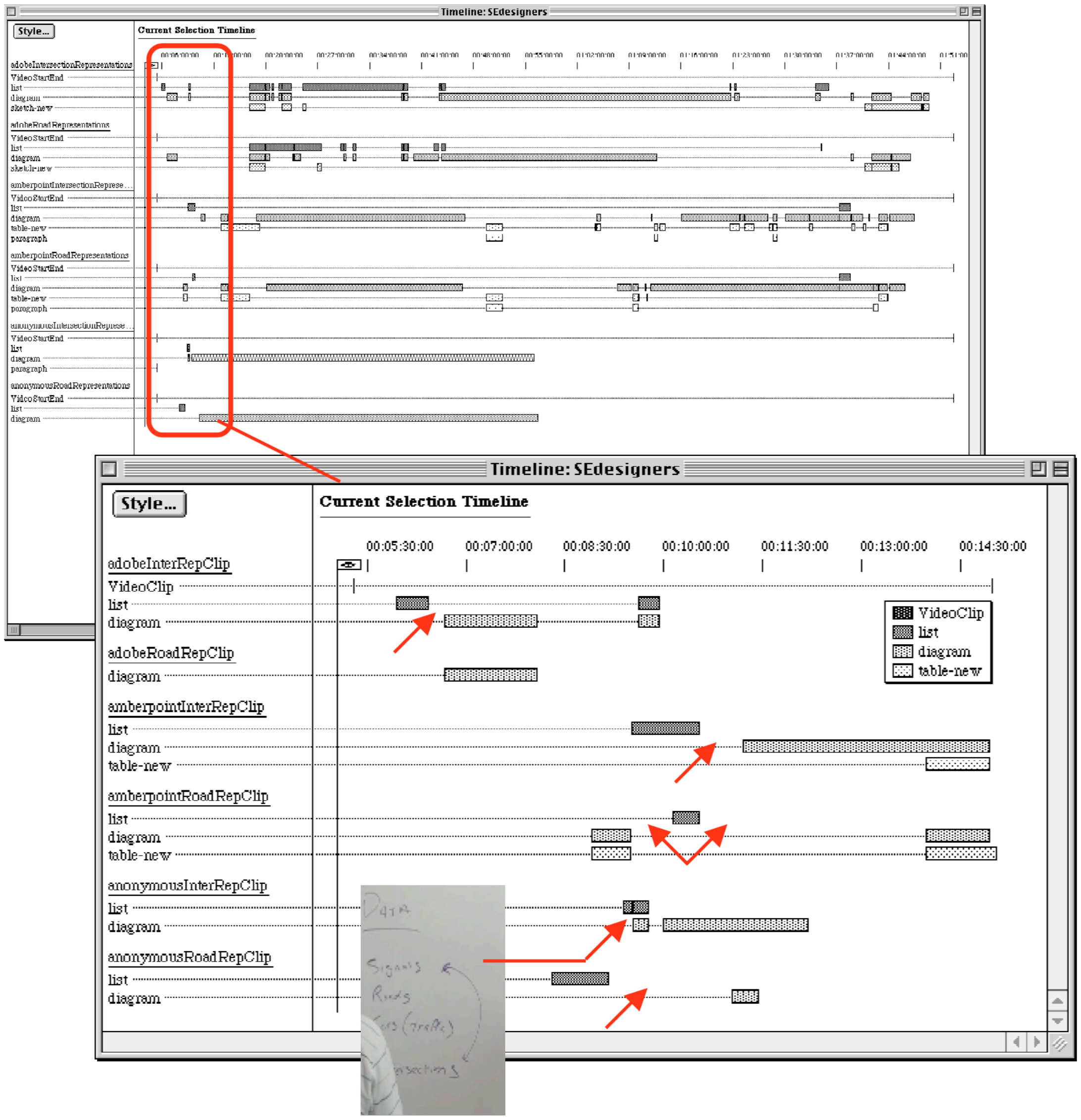


Figure 6. Timeline showing interleaved

seeming to match domain emphasis in the transcripts.

Flexible modeling tools provide multiple representations that are views onto an underlying model. A flexible modeling tool for a shared whiteboard would need to support use of many different representations as views, but we did not observe any that we have not previously considered and are not in common use. What is more challenging for flexible modeling tools is cross-representation annotation such as that used by the Amberpoint team. This example is visible in the Figure 5, where in the center of the Amberpoint board there is an labeled arrow annotation pointing between a diagram and a table. Flexible modeling tools would need to be extended to deal with such cross-representation annotations.

E. Representations evolved becoming increasingly formal and detailed

Designer's did not make a picture and then talk about it, people continually updated the picture as they talked. This interleaving of conversation and marking is typical and is shown in Figure 6. This figure shows for each team, the activities centered around list and diagrams that occurred while the design teams talked about a single concept. In the figure for each team we show two concepts, the 'intersection' concept first and the 'road' concept second.

We also observed that representations used by the designers evolved over time. We observed that lists commonly became trees and diagrams. Adobe's class list evolved into a class diagram. Anonymous' physical diagram became their UI diagram. Amberpoint's entity list became an entity tree.

Often details were added to the representation as the conversation led to deeper understanding. Figure 6 shows the interplay between two important concepts: 'intersections' and 'roads'. We focus on the beginning of the timeline where we see six instances of the designers moving between representations (indicated by the red arrows). Some of these are examples of use of multiple representations and some of them are where the designers were changing the form of a representation. The insert shows how the anonymous team changed their list into a diagram by drawing arrows. They came to realized this relationship existed through the conversation they were having as they made the list.

Representations also became more formal and detailed over time. For Amberpoint and Adobe teams, formality appeared as designers thought about what they needed to present. Returning to Figure 5, the numbers indicate the phase of activity in which each representation appeared. Notice that the entitiy-relationship diagram was the final diagram created by the Amberpoint team, and the Adobe team. The Adobe team also created a list of use cases to aid their presentation. It was clear that these two teams thought of the creation of the presentation as a separate step, and this was not integrated with the conceptual exploration they were doing on the whiteboard. When making diagrams and lists to present, they needed to refer to what they had previously done, but had to do the work of rewriting.

Flexible modeling seems highly appropriate to the process we observed here. Such tools allow the semantics and representation details to evolve in concert; guidance reminds users of possible structural violation, but does not enforce precision; and the presentation layer enables the synthesis of a presentation from the conceptual explorations that have already happened during the discussion.

VII. Conclusions

We learned that at least for the three examples of software engineering design discussions explored in this paper, the process of concept development and use of the representations is non-linear with concepts gradually evolving over-time. Designers make use of multiple different representation to capture their ideas during the design process, and use these representation informally and imprecisely. This style of work suggests that flexible modeling tools would offer the kinds of support that might facilitate this process.

We also noted that designers move towards more formal and precise representations as they think about how to present their designs to others. Thus the presentation layer provided by a flexible modeling tool would also benefit software engineers.

We should note however that this exploration is limited in several ways. Firstly, the three design discussions we observed are not real world instances of design discussions, but were artificial tasks staged specifically for the purpose of study. Time was limited and unrealistically focused, the problem was not real-world, designers were on-camera and knew they were being recorded for others to watch. There are also limitations to our analysis of the videos as discussed earlier. Our results are preliminary, there is no inter-rater reliability in the coding. All comparisons are only 'eyeball comparisons.

Despite such limitations we believe that flexible modeling tools would support software engineering discussions at the whiteboard.

VIII. Acknowledgment

We would like to thank Sandy Esch from CMU for advice on using MacShapa.

IX. References

[1] Edsger W. Dijkstra, "On the role of scientific thought." in Dijkstra, Edsger W., *Selected writings on Computing: A Personal Perspective*, Springer-Verlag New York, Inc., pp. 60–66.

[2] IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std. 1471- 2000. Approved 21 Sep. 2000. Page 4.

[3] H. Ossher *et. al.*, "Business Insight Toolkit: Flexible pre-requirements modeling." Informal demonstration paper in *ICSE 2009 Proceedings Companion*, May 2009.

[4] H. Ossher *et. al.*, "Using tagging to identify and organize concerns during pre-requirements analysis." Workshop paper in *ICSE 2009 Proceedings Companion*, May 2009.

[5] David L. Parnas, "On the criteria to be used in decomposing systems into modules." Comm. ACM *15*(12), ACM, December 1972, pp. 1053–1058.

[6] Martin P. Robillard and Gail C. Murphy, "Concern graphs: finding and describing concerns using structural program dependencies." In *Proceedings of the 24th International Conference on Software Engineering*, ACM, May 2002, pp. 406–416.

[7] Martin P. Robillard and Gail C. Murphy, “FEAT: a tool for locating, describing, and analyzing concerns in source code.” In *Proceedings of the 25th International Conference on Software Engineering,* ACM, 2003, pp. 822–823.

[8] Martin P. Robillard and Frederic Weigand Warr, “ConcernMapper: simple view-based separation of scattered concerns.” In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange,* ACM, 2005, pp. 65–69.

[9] Stanley M. Sutton Jr. and Isabelle Rouvellou, “Modeling of software concerns in Cosmos.” In *Proceedings of the 1st International Conference on Aspect-Oriented Software Development,* ACM, April 2002, pp. 127–133.

[10] Peri Tarr, Harold Ossher, William Harrison and Stanley M. Sutton, Jr., "N degrees of separation: multi-dimensional separation of concerns." In *Proceedings of the 21st International Conference on Software Engineering*, IEEE, May 1999. pp. 107–119.