Identifying Design Activities via Discourse and Content Analysis

Bonita Sharif, Natalia Dragan, Andrew Sutton, Michael L. Collard, Jonathan I. Maletic
Department of Computer Science
Kent State University
Kent Ohio 44242
bsimoes@cs.kent.edu, ndragan@cs.ken.edu, asutton@cs.kent.edu, collard@uakron.edu, jmaletic@cs.kent.edu

Abstract

A qualitative analysis of three video-taped and transcribed interviews of two-person developer teams solving a software design problem is presented. The transcripts are converted to XML and manually annotated with codes that identify engaged engineering activities and addressed requirements. The annotated transcripts are visualized, to support comparison, in a timeline matrix. The results are analyzed to provide insights into the various kinds of design activities involved. Observations and inferences about design activities and problem solving strategies are presented.

1. Introduction

How do we design software? This very short question typically results in many long and convoluted answers. The difficulties involved in giving one clear answer are many. The problem domain, the expertise of the designers, the quality and completeness of the requirements all impact how one designs a software solution. However, there are few studies, or basic understanding, of how software engineers actually design and what problem solving processes and activities are used.

The opportunity to conduct an in depth study of how engineers' design presented itself in September 2009. A group at UC-Irvine made video, audio, and transcripts of three separate two-person teams doing design for the same set of requirements, available. The open-ended objective was to develop a more sound understanding of the discipline for software design.

We decided to gather observations and insights into the design activities from a technical and cognitive perspective using discourse analysis [3] and a grounded theory [2] approach. More, specifically we attempt to identify the different decision-making and problem-solving activities being used by the different teams. Additionally, the particular requirement that was being addressed for each moment was determined. Then each of the three data sets were annotated with meta-information corresponding to the different types of activities taking place and requirement being addressed at any particular time during the design session.

This annotated data is then analyzed for trends and patterns in the design process over the three different teams. The research questions we attempt to address are:

- RQ1: What specific types of activities do the developers engage in?
- RQ2: What design strategies (activity sets) do the software designers use together?
- RQ3: Were all the requirements met or talked about in the design?
- RQ4: What was the level of interaction between the designers?
- RQ5: What are the similarities and differences between the three design sessions?

To address these questions we first need to generate, refine, and analyze the data from the developer interviews. The process involves the manual annotation of an XML formatted transcript and its visualization on a timeline. While the analysis of our data is still incomplete, we do have a number of interesting observations that partially address our research questions.

The paper is organized as follows. Section 2 details the process and tools used to organize, analyze, and visualize the video and transcript data. In Section 3, we present a sample of the resulting visualizations. Sections 4 & 5 discuss the analysis of the results. Conclusions and future work is presented in Section 6.

2. Methodology

In order to address the research questions of the study, we needed to develop a process and suite of tools to support the extraction and analysis of data from the source videos and their transcripts. We now give a brief overview of the process.

1. Transform the transcripts into an XML format in order to support data cleaning, annotation, queries and other transformations.

2. Review the videos to generate a set of codes related to activities observed. This step also attempts to fix any mis-transcribed or inaudible statements within the XML transcripts.
3. Review the videos again, annotating the XML transcripts with codes indicating the engineering activities being performed and requirements being addressed.
4. The annotated XML transcripts are used to generate timeline visualizations that depict how the studied participants engage the problem over the course of the session.

These timeline visualizations are manually studied to answer the research questions initially posed. The remainder of this section provides details on each step of the process.

2.1. Source Material

The analysis is conducted over three videos, each of which depicts a pair of professional software engineers discussing and designing a traffic signal simulation. Transcripts of the engineer's discussion are also analyzed. Summary information about the videos is presented in Table 1.

Table 1. Video Information

Video	Number of words	Total Time	Total # of inaudible
Adobe	10,043	1 hr 52 s	36
Amberpoint	12,328	1 hr 53 s	25
Anonymous	5,917	~1 hr	19

Before analyzing the videos and transcripts, we attempted to manually reduce any transcription errors. Typos, inconsistencies, or omissions between the videos and the transcripts were fixed. For example, if the transcript stated "[inaudible]", but we were able to recognize the spoken word from the video, the transcript was updated with the corrected text. There were also instances where the transcript changed the meaning of the spoken sentence completely. For example, on page 5 of the Adobe transcript (at 0:14:14.1), Male 2 uses the word "interface" which is transcribed as "beginner phase". Such incorrect transcriptions were also fixed. There were cases where we were not able to hear the speech, in which case we left the text as inaudible. Counts of inaudible text are also shown in Table 1.

In order to analyze periods of silence, additional timestamps were introduced into the transcripts when nothing was said for more than 20 seconds.

2.2. Activity and Requirement Coding

We developed two sets of codes that can be attributed to the speech and actions in the studied design sessions. The first set of codes identifies requirements from the problem domain that developers may address. The second set of codes corresponds to various kinds of software engineering activities in which the developers may engage during the sessions. These codes are used during the transcript annotation process (described in Section 2.3) to describe how the speech and actions of the participants engage different engineering activities and problem requirements.

Table 2. The set of requirement codes

Code	Description
map_roads_intersect	Map should allow for different arrangement of intersections.
road_len	Map should allow for roads of varying length.
intersect	Map should accommodate at least six intersections if not more.
light_seq	Describe and interact with different light sequences at each intersection.
set_light_timing	Students must be able to describe and interact with different traffic light timing schemes at each intersection.
set_lights	The current state of lights should be updated when they change
left-hand_turn	The system should be able to accommodate protected left-hand turns.
no_crash	No crashes are allowed.
sensor_option	Sensors detect car presence in a given lane.
sensors_light	A light's behavior should change based on input from sensors.
simulate	Simulate traffic flows on the map based
vis_traffic	Traffic levels conveyed visually to the user in real-time.
vis_lights	The state of traffic lights conveyed visually to the user.
spec_density	Change traffic density on any given road.
alter_update	Alter and update simulation parameters.
observe	Observe any problems with their map's timing scheme and see result of their changes on traffic patterns.
wait_time	Waiting is minimized by light settings for an intersection.
t_time	Travel time.
t_speed	Travel speed for cars

As the designers conversed the main topic of conversation was the problem given. Their discourse concerned parts of the software development process, including requirements, analysis, design, and even implementation. For each of these parts of the process specific items are discussed including use cases, actors, objects identified, ER or class diagrams, control and data flow, data structures, patterns and architecture. Different group of designers touch the different aspects of the design process with different degrees of details. The

purpose of identifying all the potential codes was to be able to annotate the transcript with the information about the requirements covered during the discussion, design strategies used in the software development process, and interpersonal communication.

In order to determine coverage and potential defects in the design, we needed a mechanism by which we could identify the requirements addressed during a design session. From the problem statement we first identified a set of basic requirements and assigned them an identifier or code. These requirements are shown in Table 2.

Additionally an examination of the videos and transcripts was performed to define a set of activities in which the participants engaged. We differentiate between two categories of activities: verbal and nonverbal. All speech belongs to the verbal category. Examples of non-verbal activities include drawing, reading, and analysis.

The verbal category consists of activities related to the design process and miscellaneous interpersonal communication shown in Table 3 and Table 4 respectively. The design process related verbal activities in Table 3 map to engineering activities including requirements, analysis, design, and implementation [4]. The miscellaneous activities in Table 4 include general conversational aspects such as consulting, asking or answering questions, and agreeing with the other participant. We also identified and annotated some activities as irrelevant to the design process. For example, in the Adobe video there was a 2 minute break where nothing transpired due to one of the developers leaving the room.

Table 3. Verbal activity codes

Verbal code	Description
v_asu	Making assumptions about the requirements.
v_req_new	Identifying other new requirements not specified in the design prompt.
v_UC	Define/refine use cases.
v _DD	Define/Modify the data dictionary/domain objects.
v _DD_assoc	Add associations to domain objects.
v _DD_attrib	Add attributes to domain objects.
v_class	Adding/Modifying a class.
v_class_attrib	Adding/Modifying a class attribute.
v _class_ops	Adding/Modifying a class operation.
v_class_rel	Adding/Modifying a class relationship.
v_ER	Work on the ER diagram.
v_arch	Identify architectural style. Identify design patterns.
v_UI	Defining/Modifying the UI.
v_DS	Identifying data structures.
v_code_logic	Specific implementation logic. Includes specific algorithms used, data flow and control flow.
v_code_structure	Designing code structure.

Table 4. Miscellaneous verbal activity codes

Misc. code	Description
v_consult	Consulting with other designer, asking questions, answering questions
v_agree	Agreement.
v_planning	Planning. This activity involves one or both designers deciding on the next step of the design process.
v_justify	Justification for a certain design decision.

Nonverbal activities are mainly related to reading, drawing or writing on the whiteboard, and analyzing that information. These activities are shown in Table 5 and include reading of requirements, drawing pictures, user interface, diagrams, and writing the code structure. While viewing the videos we decided to have a separate code for analyzing the whiteboard as a nonverbal activity, since this activity occurred with sufficient frequency. The following sections describe how the XML transcript was created, and how the annotations were performed.

Table 5. Nonverbal activity codes

Nonverbal code	Description
nv_read	Initial reading of problem statement.
nv_revisitreq	Revisit requirements by re-reading the design prompt.
nv_analyze_board	Both designers analyze the whiteboard. This is accompanied by nv_silent or other activity codes where appropriate.
nv_drawpic	Draw a picture. Can be paired with nv_silent.
nv_UC_draw	Draw/Modify use case.
nv_CD_draw	Draw/Modify class diagram.
nv_ER_draw	Draw/Modify an ER diagram.
nv_UI_draw	Draw/Modify a sketch of the UI.
nv_notes	Writing notes or rules/
nv_code	Writing code structure.
nv_silent	Silent. Usually combined with one or more of the above codes.

2.3. Transcript Annotation

The transcripts were annotated with the codes identified in the previous step. The time-stamped events from the transcript were converted into an XML format. The events were then annotated with the codes for verbal and nonverbal activities by reading the transcript and reviewing the videos respectively. The result is an annotated XML transcript with activity and requirement codes as metadata.

An example of an event element is given in Figure 1. Each such event corresponds to one time-stamped event. The XML element records the time down to the tenth of a second, a label for which participant was speaking, the number of inaudible tags in the text, and the transcription of the participant's speech.

```
<timestamp hr="0" min="06" sec="27"
sec_tenth="4" person="Female" num_inaudible="2">
  <sentence>So it's like a drawing tool that can
allow them to lay down these intersections. And
also some mechanism by specifying these
[inaudible] distances between intersections. I
suppose there's some traffic speeds at which
these cars travel... inaudible]</sentence>
</timestamp>
```

Figure 1. The initial XML transcript associates each sentence with its time spoken, speaker, a number of inaudible words or phrases.

Annotations were then added to the generated transcript based on manual analysis of the text and video. We developed a custom annotation tool to simplify the process of entering the annotations. A screenshot of this tool is shown in Figure 2.

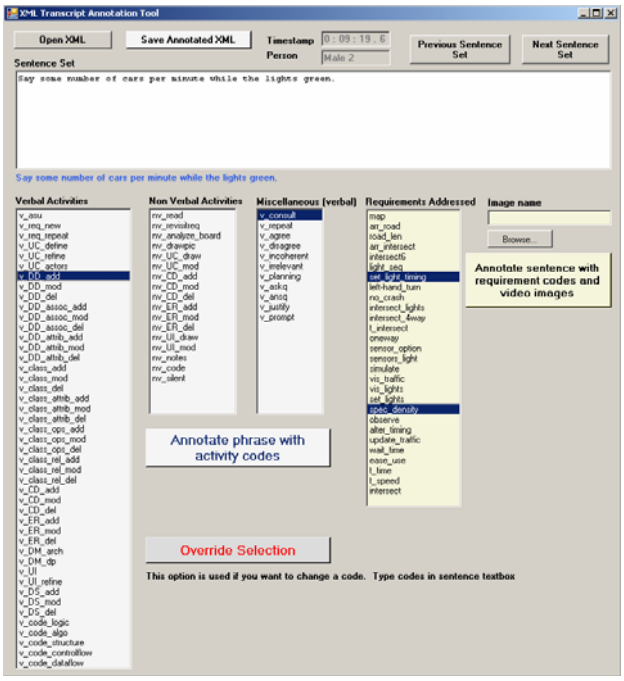

Figure 2. The XML transcript annotation tool enables the tagging of events with activity and requirements codes.

The process of annotation was performed in the following manner. First, for each event there was a *sentence set* that consists of one or more *sentences*. Each sentence consists of one or more *phrases*. Each phrase was coded with one or more activity codes. Not all phrases were coded. The resulting set of codes for the sentence is the union of the codes of all phrases in that sentence, and likewise for the sentence set. Overlapping phrases with activities were not allowed. In addition, each sentence set was also annotated with requirement codes, and a relevant image such as a class diagram or a still frame from the video (if available). An activity code based on keywords used in the phrase was also assigned. These keywords (in the problem and solution domain) are identified during the initial review of the videos. This decision process was done manually for each timestamp using the annotation tool. An example of the annotated XML from Figure 1 is shown in Figure 3.

The keywords we identified can also be used to query the XML transcripts if need be and is left as a future exercise. For example: One query could be "When did the first instance of the word *queue* occur?". The keyword queue is part of the solution domain. The keyword set also includes similar word mappings. For example, the word "signal" and "light" mean the same thing in this domain context. We do not list the keywords in this paper. However, they are implicit in the phrases that are annotated. And since we don't lose any information in the XML annotation process, we can easily query the XML document for these keywords.

```
<timestamp hr="0" min="06" sec="27"
sec_tenth="4" person="Female" num_inaudible="2">
<sentence req="map, arr_intersect"
          image_name="vlcsnap-00003.png">
  <activity>
    <phrase>So it's like a drawing tool that can
allow them to lay down these
intersections.</phrase>
    <code type="verbal">v_UC </code>
    <code type="verbal">v_UI</code>
  </activity>
  <activity>
    <phrase>And also some mechanism by
specifying these [inaudible] distances between
intersections.</phrase>
    <code type="verbal">v_UC</code>
  </activity>
  <activity>
    <phrase>I suppose there's some traffic
speeds at which... [inaudible]</phrase>
    <code type="verbal">v_asu</code>
  </activity>
</sentence>
</timestamp>
```

Figure 3. The annotated XML transcript decomposes sentences into phrases and includes requirement and activity codes and linked stll frames.

Some events and activity codes were inserted manually in the XML transcript, specifically events having to do with silence and reading activities—non-verbal activity codes *nv_silent* and *nv_read*. Events for both of these were added before the first utterance, i.e., before the first entry in the transcript. In the anonymous video, timestamps were also added for long periods of silence.

The analysis of videos and transcripts and their annotation is time consuming. One of the authors annotated the Adobe video and another author annotated the Amberpoint and Anonymous video. It took 30, 32, and 15 hours to code the Adobe, Amberpoint, and

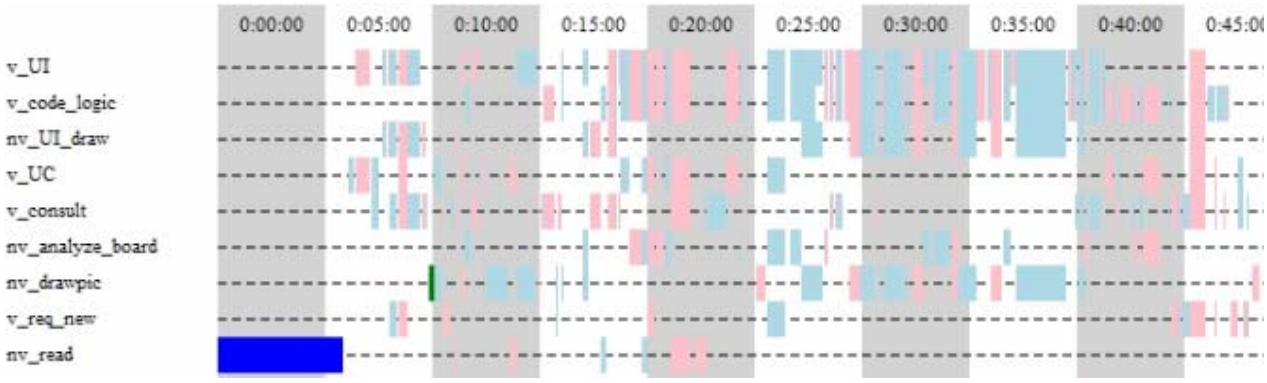

Figure 4. The sorted timeline matrix for the Amberpoint video presents actions and speech mapped to various event codes. Events are colored blocks indicating the duration of the participation, with the color of the block indicating the speaker or actor.

Anonymous sessions respectively. There were also challenges where one developer was drawing and the other was talking, which occurred quite frequently, and were put into the same event.

Since each video was coded by only one person, we were not able to calculate an inter-coder reliability measure [1]. While left as a future exercise, this may pose a threat to the coding accuracy. We use fully annotated XML transcripts for subsequent analysis, queries, and visualization.

2.4. Visualizing the Data

In order to help study and further analyze the data, we used timeline presentation for the annotated XML transcripts. Discussion and activities are visualized as a *timeline matrix* in which actions and speech are mapped to event codes (i.e., those described in Section 2.2).

An example of the engineering activities during the Amberpoint design session is shown in Figure 4. Here the software engineering activities are shown along the *y*-axis (here, in sorted order). The *x*-axis represents the duration of the session and is divided into blocks of five minutes. Activities and speech are colored blocks indicating the presence duration of participation. The color of the block indicates the speaker or actor engaging the activity or requirement. In this example, a light blue color denotes the first designer, pink denotes the second designer, dark blue denotes simultaneous participation by both designers, and green denotes the interviewer.

To generate the visualization, we processed the XML transcripts via a suite of Python scripts, extracting each event and associating it with both a requirement code (from Table 2) and an activity code (from Table 3 through Table 5). Recall that events are frequently annotated with multiple activity and requirement codes. Events associated with each key are ordered sequentially by start time. Note that transcripts are structured so that there are no overlapping time segments for a single event. In the event that an overlaps do occur, the most recent activity simply "interrupts" the previous.

Each of these mappings (event-to-activity and event-to-requirement) is stored in a *timeline* file that describes the sequence of events associated with each code. A second Python script renders the timeline file into the Scalable Vector Graphics (SVG) format. These images can then be rendered in a browser or converted to bitmap graphics for manual inspection.

To further improve readability, we manually ordered the timelines in the source data sets. The ordering attempts to group logically related activities. We also inserted missing activity or requirements codes with an empty timeline if the XML data file did not include them. These modifications are intended to facilitate visual comparisons between design sessions.

A second set of timeline was generated in which the activity and requirement codes were sorted such that the most-engaged in activity or addressed requirement would appear at the top of the matrix. We refer to these as the *sorted timeline matrices*.

3. Results

In this section, we present selected timeline matrices generated from the data analysis, specifically the Adobe design sessions. The fully annotated XML transcripts and timeline matrices are published online[1].

[1] Annotated transcripts and timeline visualizations are published online at http://www.sdml.info/design-workshop

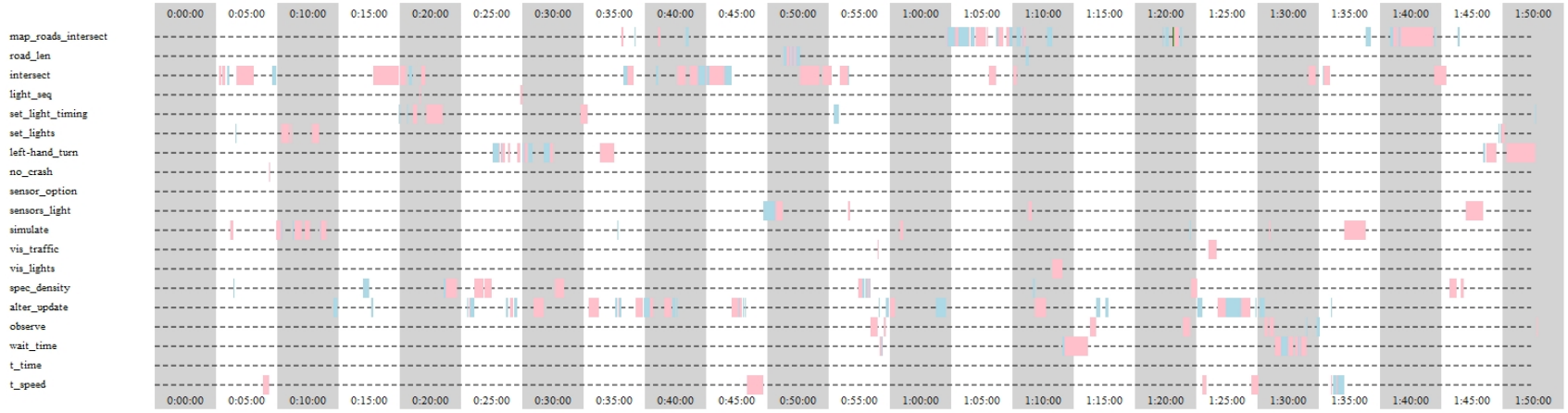

Figure 5. Requirements addressed during the Adobe session.

Figure 6. Activities engaged in during the Adobe session.

Figure 5 depicts the timeline matrix for requirements addressed in the Adobe session. From this matrix, we can see that the Adobe participants were most actively engaged in addressing problems related to the intersections and the altering or updating of simulation parameters.

Figure 6 depicts the timeline matrix for engineering activities in the Adobe session. From this diagram, we can see that the developers spent some time early on constructing a class model for the problem, and spent much of the remaining part of the hour discussing code logic.

The next section attempts to partially address the research questions we posed in the Introduction. These are our preliminary findings.

4. Data Analysis

From the annotated transcripts and their timeline matrices we can make a number of observations and comparisons about processes and strategies employed by each design team.

4.1. Activity Analysis

Our first research question seeks to determine the types of activities designers engage in while solving a problem. The mapping and visualization of phrases and statements to activity codes directly supports our ability to answer this question. The types of activities the designers engage in map nicely to the main phases of software development such as Requirements, Analysis, Design and Implementation (See Table 6). The sequence and iteration aspects of these steps were different across the three design teams. Amberpoint had a lot of activities engaging with the UI aspect of the design. In contrast the Adobe and Anonymous sessions looked more into the design of objects and associations between them. The Amberpoint session (448 events) engaged in a lot more verbal activity compared to the Adobe session (414 events).

Table 6. Activity codes are mapped to traditional phases engineering process.

Phase	Activities
Requirements	v_asu, v_req_new, nv_read, nv_revisitreq, v_UC, nv_UC_draw
Analysis	v_DD, v_DD_assoc, v_DD_attrib
Design	v_class, v_class_attrib, v_class_ops, v_class_rel, nv_CD_draw, v_ER, nv_ER_draw, v_arch, v_UI, nv_UI_draw
Implementation	v_code_logic, v_DS, v_code_structure, nv_code, nv_drawpic, nv_analyze_board, nv_notes, nv_silent

Curiously, participants from only one team (Amberpoint) discussed aspects of verification or validation with regards to their solution.

Our second research question seeks to determine design strategies or activity sets used by the designers. Examining the distributions of the activities in the timeline matrix provides some insights into answering this question. We can determine design strategies or activity sets by asking the following subset of questions. We address each of these sub-questions next.

- Which design activity took the longest?
- What was the sequence of activities for each session?
- How many of the activities were interleaved?
- Did the designers add additional features and make valid assumptions where appropriate?

Table 7. Activities on which the most and least time was spent in decreasing order for most, increasing order for least. Bold activities occurred in all sessions.

	Most Time Spent	Least Time Spent
Adobe	**v_code_logic** nv_analyze_board **v_consult** **v_UC** nv_CD_draw	nv_ER_draw v_ER nv_UC_draw v_DD_assoc nv_code
Amberpoint	v_UI **v_code_logic** nv_UI_draw **v_UC** **v_consult**	nv_code v_code_structure v_DS v_arch nv_CD_draw
Anonymous	nv_analyze_board **v_code_logic** **v_consult** nv_silent **v_UC**	nv_UI_draw nv_ER_draw v_ER nv_UC_draw v_req_new

Table 7 presents the first five activities that took the most time and the least time. This was determined by examining the sorted timeline matrices. Three activities (*v_code_logic*, *v_consult*, and *v_UC*) were common in all three videos but did not appear in the same order. In other words, in all the three videos, decisions about the logic of the code, discussions of use cases, and interaction between designers played major roles. Other top time-consuming activities include analyzing the white board, drawing class diagrams, talking about and drawing the user interface (*nv_analyze_board*, *nv_CD_draw*, *v_UI*, and *nv_UI_draw* respectively). Silence was much more prevalent in the Anonymous video, where they spend the time analyzing the white board.

The sequence of activities and the interleaving of those activities for each session varied. This can be attributed to personal differences in approaches to problem solving and previous expertise or experiences. In order to analyze the sequence of activities, we split the

timeline into 4 sections of 30 minutes (15 minutes for the Anonymous session since they took half the amount of time for the task). We also exclude the top five activities in this analysis. Table 8 shows the high level activities engaged in during each quarter.

For the Adobe session, the first 30 minutes is spent on identifying the objects in the system (data dictionary and domain model) and drawing pictures of the intersection. The next 30 minutes was spent on revisiting the requirements (*nv_revisitreq*), use cases (*v_UC*), and specifying properties of objects. In the third interval they concentrate on the UI and use cases. Finally they focus on drawing the UML class diagram and writing the structure of the code.

For the Amberpoint session, the first 30 minutes are spent talking about the UI, drawing pictures of the UI, introducing domain objects and use cases. The second quarter focuses much more on the UI with some discussion about the use cases and scenarios. The third quarter also focuses on the UI and use cases. Finally in the last quarter, they draw an ER diagram.

For the Anonymous session, the first 15 minutes is spent identifying domain objects, and drawing pictures. In the next 15 minutes, they do the same with a lot of silence between the members of the team. The next interval's focus was on drawing a diagram, specifying attributes, relationships, as well as on use cases. In the final interval, use cases were discussed including some discussion about the UI.

If we observe the activity chart for Adobe in Figure 6, we can get a visual indication of the types of interleaved activities in each quarter. The activities that were interleaved were usually related. For example, in the Adobe session, when they talk about the UI, they also talk about possible cases of user interaction (use cases). The activities of drawing pictures and identifying domain objects were also interleaved in all three videos. This points to the fact that drawing a visual representation actually helps in identifying the data dictionary and the thought process of the designers. Discussing the logic of the code was talked about almost consistently throughout and hence interleaved most with all other activities.

With respect to coming up with new requirements (*v_req_new*), Amberpoint did this more than Adobe. Anonymous did not address that subject too much. In the Adobe session, one new requirement was the ability to resize the simulation window.

All three design sessions make some assumptions about the requirements. For example, in the Adobe session, assumptions were made about roads having just two lanes to address the left-hand turn requirement. Adobe and Amberpoint make more assumptions than Anonymous. Some of these assumptions were critical to moving the design forward. One prevalent behavior in the Adobe video was to concentrate on the current topic and if something was not clear or if they had trouble with it, they went on to the next point and returned back to it at a later stage. This is basically the iteration aspect of design.

Table 8. Sequence of main (not necessarily interleaved) activities across time shown in 30 minute increments (15 minutes for Anonymous)

	Interval	Activities
Adobe	30 m	Identifying objects, drawing pictures
	1 h	Attributes of objects, revisiting requirements, use cases
	90 m	UI and use cases
	2 h	UML class diagram, code structure
Amberpoint	30 m	Drawing and talking UI, identifying objects, use cases
	1 h	Drawing UI, use cases, scenarios
	90 m	UI and use cases
	2 h	UI and ER diagram
Anonymous	15 m	Identifying objects, drawing pictures
	30 m	Identifying objects, drawing pictures, silence
	45 m	Drawing a diagram with attributes and relationships, use cases
	1 h	Use cases, domain objects, UI

4.2. Requirements Analysis

Our third research question seeks to determine requirements coverage. In other words, did the designers take into account all or some of the requirements while designing the system? From the annotations of the requirements in the transcript we can determine if all the requirements were met in the final design. The requirements coverage over time for the Adobe session is presented in Figure 5.

We can also determine which requirements were discussed the most. There might be a connection between the time taken for a requirement and the inherent complexity involved. There is one exception to this: a requirement that was never talked about would take the least amount of time but it is not necessarily easy to implement.

Table 9 presents the first five requirements that took the most time and the least time. We determine this by sorting the y-axis of our time series matrix according to time spent. Two requirements (*intersect*, *alter_update*) were common in all three videos but did not appear in the same order. The 'intersect' requirement dealt with the approach they used to describe and represent the intersection. It also included constraints such as *no one-way roads* or *T-junctions*.

In the Adobe and Anonymous design sessions, the intersection approach was the top requirement discussed. Since the Amberpoint design session focused more on

the UI, their top activity shows up as *alter_update*. This requirement basically involves interaction by the student using the software. It deals with setting simulation parameters and updating the simulation (See Table 2).

The Adobe and Amberpoint session both discuss the wait time requirement, which state that ideally the wait time needs to be minimized. This was the sixth most talked about activity for the Adobe session. The Anonymous session does not pay much attention to this non functional requirement.

The Adobe session addressed all the requirements in the design prompt. The Amberpoint session failed to address the *t_time* requirement, however they did address the *t_speed* requirement (seventh highest talked about requirement). These are both related events. The Anonymous session did not address the *no_crash*, *vis_lights*, and the *observe* events. Their design did not cover aspects of the UI. The *vis-lights* requirement states that the state of the lights should be visually conveyed to the user. The observation requirement deals with the students observing problems with traffic patterns and timing schemes. Since Anonymous did not touch the UI aspect in detail, they missed these requirements.

Table 9. Requirements on which the most and least time was spent in decreasing order for most, increasing order for least. Bold requirements were addressed in all sessions.

	Most Time Spent	Least Time Spent
Adobe	**intersect** **alter_update** map_roads_intersect left-hand_turn spec_density	t_time sensor_option no_crash light_seq vis_lights
Amberpoint	**alter_update** map_roads_intersect set_light_timing **intersect** wait_time	t_time road_len set_lights light_seq simulate
Anonymous	**intersect** set_lights **alter_update** left-hand_turn set_light_timing	wait_time observe vis_lights no_crash light_seq

Besides requirements coverage, if the requirements distribution across time is compared side by side (or overlaid) with the activity distribution across time, we can determine the activities involved to satisfy the given requirement. We discuss some instances of this below.

In the Adobe session, time spent on the *intersect* requirement involved a lot of consultation between the designers as well as identification of domain objects in the first ten minutes of the video. The *intersect* requirement is revisited after a while, but this time the designers describe the objects and their properties in greater detail and even start sketching out a preliminary UML class diagram. We also see that the *left-hand_turn* requirement was interleaved mostly with discussing the logic involved in its implementation.

4.3. Interpersonal Communication

We can also examine the interactions between the designers (RQ4), and its effect on the potential quality of the design. What was the distribution of silence and communication between the members of the team? How closely did the two designers interact? Again, we can derive this information from the timeline matrices by examining the amount of time spent consulting, planning, questioning, or answering each other. These are shown as the last five rows in the activity time series

The Adobe video session (see Figure 6) contained the most agreement between the two designers. It also had the consulting activity scattered almost uniformly throughout the video. They also engaged in a lot of planning in the last three intervals and also justify their choices. The Amberpoint video also had consulting throughout the session with a lesser level of agreement between the designers. They did much more planning almost at the end of the video session with not much justification involved. The Anonymous video also had consulting throughout the session. Their agreement was also consistent throughout. However, there were not many justifications and not much planning involved. There were no silent episodes in the Adobe or Amberpoint videos. The Anonymous video session had silent episodes spread throughout the design session.

4.4. Similarities and Differences

All the three videos had a different approach to solving the problem. We will briefly point out similar and different approaches between the design teams (RQ5). We base our comparison of the video sessions on the two main deliverables asked for in the design prompt: design interaction/UI and basic structure of the code.

The Adobe session took a very structured approach to the problem. This can be seen from the density of the activity graphs. They systematically planned what their next moves would be and followed through. They designed both the interaction and the code structure. There was a lot of consulting and agreement between the designers. The Amberpoint video focuses on the UI and so they did present an interaction scheme for students. However, they did not touch the structure of the code or even identify associations between main objects in the system. The Anonymous video went through it pretty quickly. Their chart is much less dense compared to the Adobe and Amberpoint activity charts.

We believe a just-graduated student for whom this was intended would not be able to implement it based on the description given by Anonymous and Amberpoint. A student with a software engineering undergraduate degree would get more information out of the Adobe video.

5. Preliminary Observations

Based on the timeline matrices we can start to make some general observations. In this section we will describe some of them. We observe clusters that clearly correspond to the design process stages, i.e., requirements, business modeling, design, and implementation.

- The requirements and use cases stage is spread throughout the whole session.
- The data dictionary and domain model is done at the beginning for Adobe and Anonymous and scattered throughout for the Amberpoint video.
- The class diagram/design model was discussed next for the Adobe and Anonymous session and in the case of Amberpoint they focused on the ER diagram.
- Discussion about the UI was done mostly in the second part of the design sessions for Adobe and Anonymous but scattered throughout the Amberpoint session
- Code logic/implementation was also scattered through the whole session for all groups.

For Requirements, at this time we are still analyzing the data. Overall among the three groups, the most time was spent on the requirements: *intersect*, *alter_update*, *map_roads_intersect*, and *left-hand_turn*. An average amount of time was spent on *sensor_option* and *sensor_light*, and the least amount of time was spent on *light_seq*.

Without a gold standard for the design, it is not possible to state which was the overall best or worse design. However, we can make some general points. The Adobe and Anonymous teams did a better job at the code structure and system design, while Amberpoint did a better job describing and outlining the UI.

One open question is the relationship between the problem statement and the OO design process. Did this process help or hinder the final design? Also, did the domain they worked on effect the design? Note, the teams were not asked to come up with an OO design.

6. Conclusions and Future Work

In this paper, we presented an analysis of the three problem design videos and their associated transcripts. In order to support the analysis we have developed an XML format for embedding dialogs and dialog metadata and developed a tool to assist in the manual annotation of speech with that metadata. We have also presented a means to visually compare the data with a timeline matrix that has been used to analyze and draw inferences about transcribed and video-taped conversations.

With regards to the video itself, we have posed a number of research questions and partially addressed them individually. In the future, we plan to further develop the visualizations, and create an interactive timeline exploration matrix that supports drill-down capabilities and overlapping event structures.

We also plan to continue developing this method of dialog analysis and the tools used to support it. We believe that finer-grain analysis of these interactions is both possible and valuable. We are also interested in using these analysis methods to evaluate the effectiveness of certain problem solving paradigms. This could provide a method of empirically evaluating problem solving strategies or mental models of software engineering and program comprehension.

7. References

[1] Artstein, R. and Poesio, M., "Inter-Coder Agreement for Computational Linguistics", *Computational Linguistics*, vol. 34, no. 4, 2008, pp. 555-596

[2] Bryant, A. and Charmaz, K., "Grounded Theory": Sage Publications, 2007, pp. 623.

[3] Gee, J. P., "An Introduction to Discourse Analysis", 2005.

[4] Larman, C., "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process".