CHAPTER 10

Identifying and Analyzing Software Design Activities

Bonita Sharif
Youngstown State University

Natalia Dragan
Cleveland State University

Andrew Sutton
Texas A&M University

Michael L. Collard
The University of Akron

Jonathan I. Maletic
Kent State University

CONTENTS

10.1 INTRODUCTION

How do we design software? This very important question has implications for virtually all aspects of the software development life cycle, the participating software developers, and even the project managers. Good software design should promote programmer productivity, lessen the burden of maintenance, and reduce the learning curve for new developers. Despite the seeming simplicity of the question, the answers tend to be anything but simple. An answer may touch on aspects of the problem domain, the expertise of the designers, the quality and completeness of the requirements, and the process models and development methods that are followed. Much work has been done on how software *should be* designed, but there are comparatively few studies of how software *is actually* designed (Guindon et al. 1987; Curtis et al. 1988; Carlos et al. 1995; Robillard et al. 1998; Detienne and Bott 2001; Ko et al. 2006; Friess 2007).

To address the aforementioned question on how software is designed, we introduce specific research questions based on the types of activities that designers engage in during the early stages of design. These questions were chosen to cover the activities observed in the context of the requirements met by the design. We are also interested in designer interaction and how similar/different the team interactions were. The research questions we attempt to address are:

RQ1: What specific types of activities do the designers engage in?

RQ2: What design strategies (activity sets) are used together?

RQ3: Are all the requirements met or discussed in the design?

RQ4: What is the level of interaction between the designers?

RQ5: What are the similarities and differences between the different design sessions?

Since this is an *observational* study, we do not generate hypotheses for the research questions. Instead, we try to qualitatively assess the design sessions in a structured manner that is reproducible by others. In order to answer the earlier questions, two issues have to be addressed. First, we need data on the design practice of professional designers; such data are extraordinarily difficult to obtain. Individuals and groups inside companies do not always perform all design activities using a formal tool, but they often rely on paper or whiteboards, especially in the early interactive stages. This, coupled with the collaborative interaction between designers, is very challenging to capture for later study. Second, once the data have been collected, the additional questions of how the data should be analyzed and/or processed arise.

The opportunity to conduct an in-depth study of how professional designers engage in design presented itself in September 2009. A group at the University of California, Irvine (2010) made available video recordings and transcripts of three two-person teams engaged in design for the same set of software requirements. Given these data and our research questions, we decided to use an approach rooted in discourse analysis (Gee 2005): the analysis of language with the goal of identifying problem-solving and decision-making strategies.

In order to do this, we first need to generate, refine, and analyze the data from the design sessions. This process involves generating an XML-structured version of the transcripts,

the manual annotation of the XML-formatted transcripts, and its visualization on a time line. For each activity undertaken by the designers, we identified the requirement being addressed. We annotated each design session data set with metadata corresponding to the activities (such as drawing and agreement/disagreement between designers) taking place along with the requirement(s) being addressed. We also analyzed the annotated data for trends and patterns in the design process across the teams.

The results indicate that the types of design activities map nicely to the main phases of software development, such as requirements, analysis, design, and implementation. However, the sequence and iteration of these steps were different for each team. The decisions about the logic of the code, discussions of use cases, and the interaction between the designers played major roles for all three teams, and all teams spent the most time on the same three requirements. The team that took a structured approach with systematic planning, and with much consultation and agreement, gave the most detailed solution and covered the most requirements in the deliverables. A software designer engaged in these types of activities, whether an expert or a novice, may benefit from observing what does and does not work. The future goal is to develop theories (Bryant and Charmaz 2007) on how software is designed based on our observations and results.

10.2 METHODOLOGY

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

1. *Preprocessing*: In order to conduct a structured analysis on the data, we first transformed the transcripts from a simple text-based listing into an XML format for easy processing. This allowed us to clean up the transcripts, annotate them with additional data, and perform queries.

2. *Comprehension and Cleanup*: We reviewed the videos to generate a set of codes related to the activities we observed. This step also attempted to fix any ill-transcribed or inaudible statements within the XML transcripts. Almost all cases were fixed by listening to the audio multiple times or at a higher volume level. Very few inaudible sections remained.

3. *Annotation*: Once the data were cleaned up and in an initial annotated format, we reviewed the videos again, this time annotating the XML transcripts with codes indicating the activities being performed and the requirements being addressed at particular points in time.

4. *Visualization*: A time line visualization of the annotated XML transcripts was generated that allowed us to better see how the designers engaged the problem over the course of the sessions. These time line visualizations were manually studied to answer the research questions we posed.

10.2.1 Problem Statement and Source Material

The three teams were charged with designing a traffic signal simulation. They were given a two-page problem statement on paper and access to a whiteboard where they could design a solution. They were asked to produce the user interface (UI), a high-level design, and a high-level implementation. The design sessions ran from 1 to 2 hours. The audience for the design was listed in the problem statement as a team of software developers whose competency level could be compared to a basic computer science or software engineering undergraduate degree holder. Each team was debriefed at the end of their design session.

Our analysis was conducted on the videos of the three teams, each of which depicts a pair of professional software designers designing the traffic signal simulation. Summary information about the videos is presented in Table 10.1. The transcripts of the designers' discussion were also analyzed. Our approach is rooted in discourse analysis where we examine the videos, code the transcriptions, and analyze the coding. In prior related work, Robillard et al. (1998) used protocol analysis to build a cognitive model, devise a hierarchical coding scheme, and look at the influence of individual participant roles (d'Astous and Robillard 2001). We concentrate on finding the design activities and strategies that the designers used to address the requirements.

Typos, inconsistencies, or omissions between the videos and the transcripts were fixed prior to analyzing the videos. 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. For those cases where we were not able to hear the speech, we left the text as inaudible. The counts of inaudible text are also shown in Table 10.1. Furthermore, additional time stamps for periods of silence greater than 20 seconds were introduced into the transcripts.

AU: Please check that the edit made to the sentence 'For the cases... inaudible' retains the intended meaning.

10.2.2 Activity and Requirement Coding

After reviewing all three video sessions, two of the authors jointly developed two sets of codes that can be attributed to the speech and actions in the design sessions. The first set of codes identifies the requirements from the problem domain that the designers addressed. The second set of codes corresponds to various kinds of software engineering activities that the designers engaged in during the sessions. These codes are used during the transcript annotation process.

TABLE 10.1 Video Information

Video	Number of Words	Total Time	Total Number of Inaudible
Adobe	10,043	1 hour 52 seconds	36
AmberPoint	12,328	1 hour 53 seconds	25
Intuit	5,917	~1 hour	19

As the designers conversed, the main topic of conversation was the problem given. Their discourse can be categorized into different parts of the software development process, including requirements, analysis, design, and implementation. For each of these parts, specific artifacts and parts of artifacts were discussed, including use cases, actors, objects, entity–relationship (ER) or class diagrams, control and data flow, data structures, patterns, and architecture. The different teams of designers used different aspects of the design process with varying degrees of detail. From the problem statement, we first identified a set of basic requirements and assigned them a code. The fine-grained requirements identified are shown in Table 10.2.

Additionally, an examination of the videos and transcripts was performed to define a set of activities in which the designers engaged. We differentiated between two categories of activities: verbal and nonverbal. All speech belongs to the verbal category, while the nonverbal activities include drawing, reading, writing, silence, and analyzing the whiteboard. We also identified and annotated the activities that were irrelevant to the design process. For example, in the Adobe video there was a 2-minute break with no activity due to one of the designers leaving the room.

The verbal category consists of activities related to the design process and miscellaneous interpersonal communication, shown in Tables 10.3 and 10.4, respectively. The verbal activities related to the design process in Table 10.3 map to the engineering activities of requirements, analysis, design, and implementation. The activities presented in Table 10.4

TABLE 10.2 Requirement Codes Identified from the Problem Statement

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 the 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 the 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

AU: In the text 'simulate traffic flows on the map based', the phrase 'map based' is a little unclear. Please rewrite if possible.

TABLE 10.3 Verbal Activity Codes Related to the Design Process

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 10.4 Verbal Activity Codes Related to Interpersonal Communication

Miscellaneous Code	Description
v_consult	Consulting with other designers, asking questions, answering questions. No distinction is made between who asked or who answered the question.
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.

include general conversational aspects such as consulting, asking and answering questions, and agreeing or disagreeing with the other participant.

The nonverbal activities are mainly related to reading, drawing or writing on the whiteboard, and analyzing the information on the whiteboard. These activities are shown in Table 10.5 and include reading the requirements; drawing pictures, UIs, and diagrams; and writing the code structure. While viewing the videos, we decided to include a separate code for analyzing the whiteboard as a nonverbal activity because this activity occurred with sufficient frequency. The following sections describe how the XML transcripts were created and how the annotations were performed.

10.2.3 Transcript Annotation

The transcripts were annotated with the requirement and activity codes identified in the previous section. First, the time-stamped events from the simple text-based 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

TABLE 10.5 Nonverbal Activity Codes

Nonverbal Code	Description
nv_read	Initial reading of problem statement.
nv_revisitreq	Revisit requirements by rereading 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.
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.

metadata. An example of an event element is given in Figure 10.1 in nonbolded font. Each such event element corresponds to one time-stamped event. The XML element records the time down to a tenth of a second, a label for which participant was speaking, the number of inaudible tags in the text, and the transcription of the designer's speech.

Annotations were then added to the generated transcript based on a manual analysis of the text and video. As can be imagined, just entering the annotations is very labor intensive and prone to errors. To help alleviate this, we developed a custom annotation tool, *TransAnn*, to simplify the process of entering the annotations. A screenshot of TransAnn is shown in Figure 10.2.

```
<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 10.1 XML format for an event in the video. The nonbolded text was generated from the original text-based transcript that associates each sentence with the time spoken, the speaker, and any inaudible words or phrases. The bolded text shows the added annotations to the transcript that decomposes sentences into phrases and includes requirement codes, activity codes, and linked still frames.

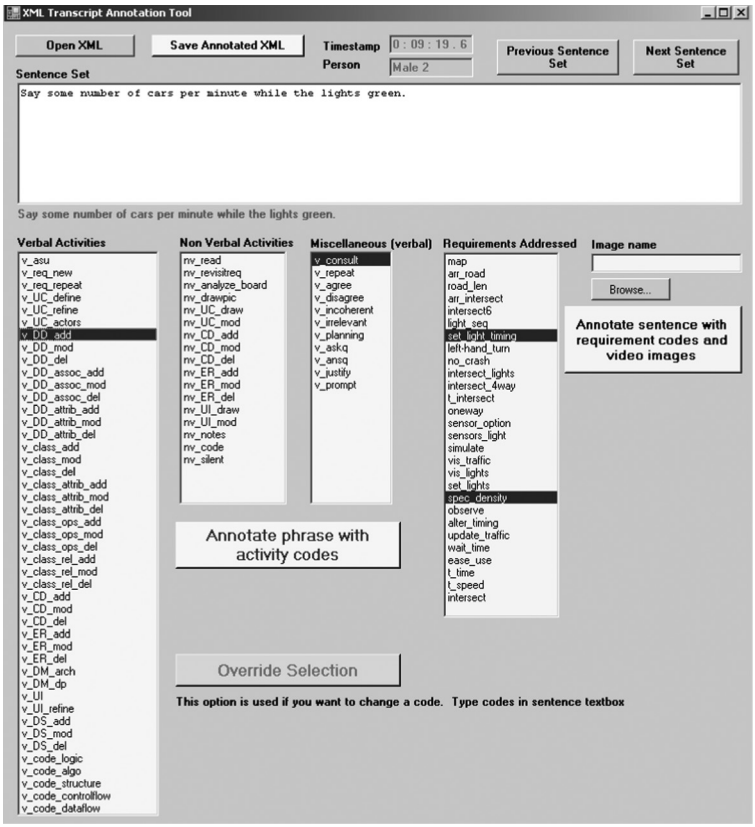

FIGURE 10.2 XML transcript annotation tool *TransAnn* was created to enable the tagging of events with activity and requirements codes.

The process of annotating was performed in the following manner (see Figure 10.3 for the structure of the XML annotations). First, each event (time stamp) was associated with a *sentence*. Each sentence consisted of one or more *phrases*. Each phrase was coded with one or more activity codes. Some phrases did not fit into any activity and were not coded. The resulting set of codes for the sentence is the union of the codes of all phrases in that sentence. Each sentence was additionally annotated with requirement codes, and a relevant

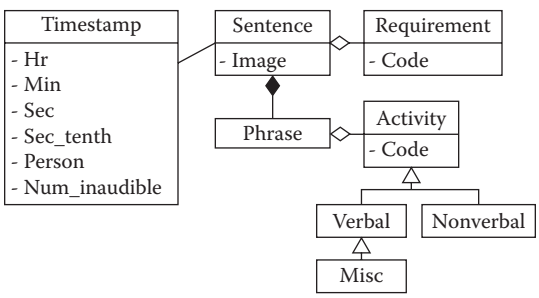

FIGURE 10.3 Class diagram for the annotated XML structure.

image such as a class diagram or a still frame from the video (if available). An activity code based on key words used in the phrase was also assigned. These key words (in the problem and the solution domains) were identified during the initial review of the videos. The annotation decision process was done manually for each time stamp using the annotation tool. An example of the annotated XML is shown in Figure 10.1.

The key words we identified can also be used to query the XML transcripts and this is left as a future exercise. For example, one query could be, "When did the first instance of the word *queue* occur?" The key word *queue* is part of the solution domain. The key word set also includes similar-word mappings. For example, the words "signal" and "light" mean the same thing in this domain context. We do not list the key words here however; they are implicit in the phrases that are annotated. Additionally, since we do not lose any information in the XML annotation process, we can easily query the XML document for these key words.

Some events and activity codes were inserted manually into the XML transcript, specifically events having to do with silence and reading activities, the nonverbal activity codes *nv_silent* and *nv_read*, respectively. Events for both of these were added before the first utterance, that is, before the first entry in the transcript since the designers started the session by reading the problem statement. In the Intuit team video, time stamps were also added for long periods of silence because this was unique to their interaction.

Even with the annotation tool, the analysis of the videos and transcripts and their annotation were quite time consuming as it took 30, 32, and 15 hours to code and annotate the Adobe, AmberPoint, and Intuit sessions, respectively. One of the authors annotated the Adobe video session and another author annotated the AmberPoint and Intuit video sessions. Initially, both annotators worked together to agree on a common coding rules convention. Other challenges included such things as when one designer was drawing and the other was talking, which occurred quite frequently. These instances were put into the same event. Since each video was coded by only one person, we did not calculate an intercoder reliability measure (Artstein and Poesio 2008). However, the two coders consulted each other on coding that seemed unclear and they reached a consensus, which reduced the threat to the validity of the coding scheme. The fully annotated XML transcripts were used for subsequent analyses, queries, and visualizations.

10.2.4 Visualization

In order to help study and further analyze the data, a time line presentation for the annotated XML transcripts is used. Discussion and activities are visualized as a *time line matrix* in which actions and speech are mapped to the event codes described earlier. Examples of engineering activities are shown in Figures 10.5 through 10.7. The duration of the session is shown on the *x* axis divided into 5-minute chunks. The activities and speech are colored blocks indicating the presence duration of participation. The color of the block indicates the speaker or actor engaging in the activity or requirement.

AU: The text 'presence duration of participation' is a little unclear. Please rewrite if possible. Can 'presence' be deleted?

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 10.2) and an activity code (from Tables 10.3 through 10.5). Recall that events are frequently

annotated with multiple activity and requirement codes. Events associated with each code are ordered sequentially by start time. Note that the transcripts are structured so that there are no overlapping time segments for a single event. In the event that an overlap does occur, the most recent activity simply "interrupts" the previous one.

Each of these mappings (event-to-activity and event-to-requirement) is stored in a *time line* file that describes the sequence of events associated with each code. A second Python script renders the time line 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 *y* axis of the time lines in the source data sets. This ordering attempts to group logically related activities. We also inserted missing activity or requirements codes with an empty time line. These modifications are intended to facilitate visual comparisons between design sessions. For example, one design team might not have touched on a specific requirement (thus not appearing on the time line), while another team did. A second set of time lines was generated in which the activity and requirement codes were sorted such that the most engaged activity or the most addressed requirement would appear at the top of the matrix. We refer to these as the *sorted time line matrices*.

10.3 RESULTS AND ANALYSES

In this section, we present selected time line matrices generated from the data analysis. The fully annotated XML transcripts and the time line matrices (including sorted matrices) are published online.* Figure 10.4 depicts the time line matrix for requirements addressed in the Adobe session. From this matrix, we can see that the Adobe team was most actively engaged in addressing problems related to the intersections (*intersect* and *map_roads_intersect*) and the altering or updating of simulation parameters (*spec_density*, *alter_update*, and *t_speed*).

Figure 10.5 depicts the time line matrix for engineering activities in the Adobe session. It is evident from this diagram that, early on, these designers spent time constructing the data dictionary (*v_DD_**), use cases (*v_UC* and *nv_UC_draw*), and class model for the problem (*v_class_**). They spent a great deal of time discussing code logic (*v_code_logic*) even while they were building the class model. They were also consistent in analyzing the whiteboard (*nv_analyze_board*), after designing the class structure and the UI (*nv_UI_draw* and *v_UI*).

AU: Please check that the edits made to the sentence 'It is... *).' convey the intended meaning.

The next sections attempt to address the research questions we posed in Section 10.1. From the annotated transcripts and their time line matrices, we can make a number of observations and comparisons about the processes and strategies employed by each design team.

10.3.1 Activity Analysis

Our first research question (RQ1) seeks to determine the types of activities that the designers engage in while solving a problem. The mapping and visualization of phrases to activity codes directly supports answering this question. The types of activities that the designers engage in map clearly to the main phases of the software development, such as requirements, analysis, design, and implementation (see Table 10.6). This mapping was not

* Annotated transcripts and time line visualizations are published online at http://www.sdml.info/design-workshop.

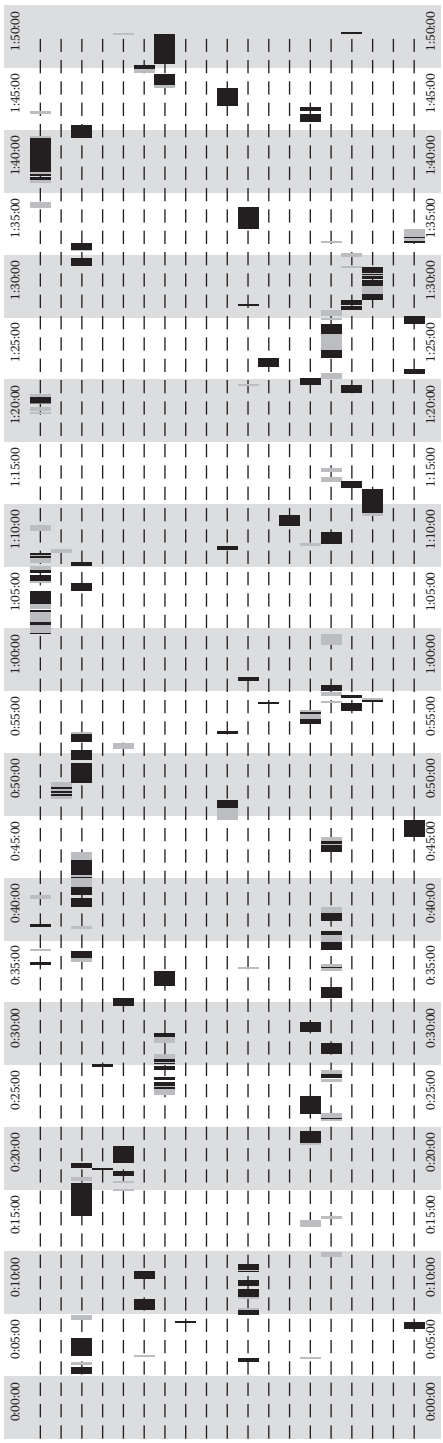

FIGURE 10.4 Requirements addressed during the Adobe session. Events are colored blocks indicating the duration of the participation, with the color of the block indicating the speaker.

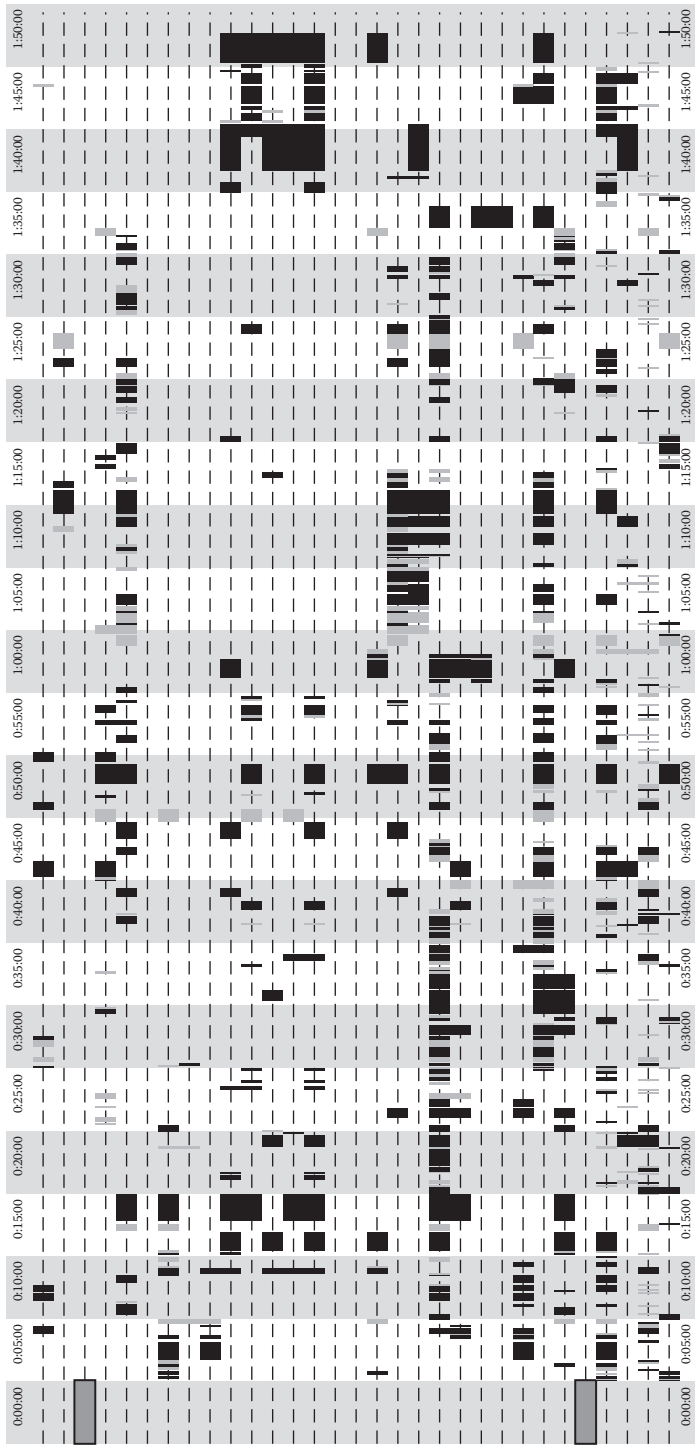

FIGURE 10.5 Engineering activities engaged in during the Adobe session. Events are colored blocks indicating the duration of the participation, with the color of the block indicating the speaker.

TABLE 10.6 Activity Codes Mapped to Phases of a Traditional Software 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

determined a priori, rather it was observed during the analyses of the design sessions. The main phases are shown circled in the time line matrices of Figures 10.6 (Intuit activities) and 10.7 (AmberPoint activities).

We observed that the sequence and iteration of these steps were different for the different teams. The AmberPoint team spent a lot of time designing the UI, whereas the Adobe and Intuit teams spent more time designing the object model. The AmberPoint team engaged in more verbal activity (448 events) compared to the Adobe team (414 events). The Intuit team finished in half the time of the other teams and had the least verbal activity. The participants from only one team (AmberPoint) discussed aspects of verification or validation with regard to their solution. Figures 10.6 and Figure 10.7 group together

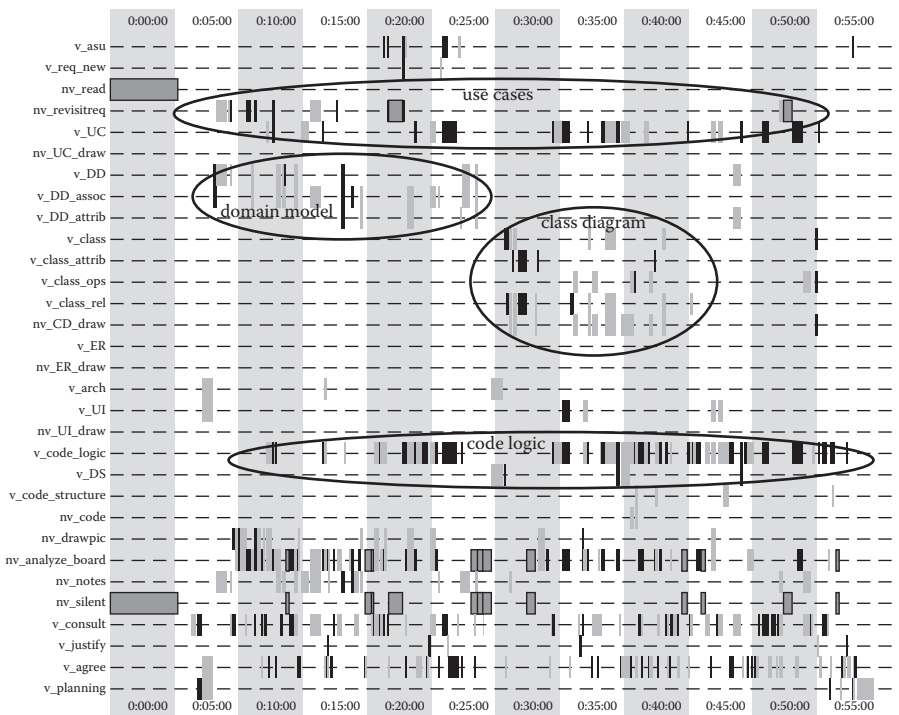

FIGURE 10.6 Time line matrix of activities for the Intuit session shows actions and speech mapped to various event codes. The events are colored blocks indicating the duration of the participation, with the color of the block indicating the speaker. The use cases, domain model, class diagram, and code logic are circled.

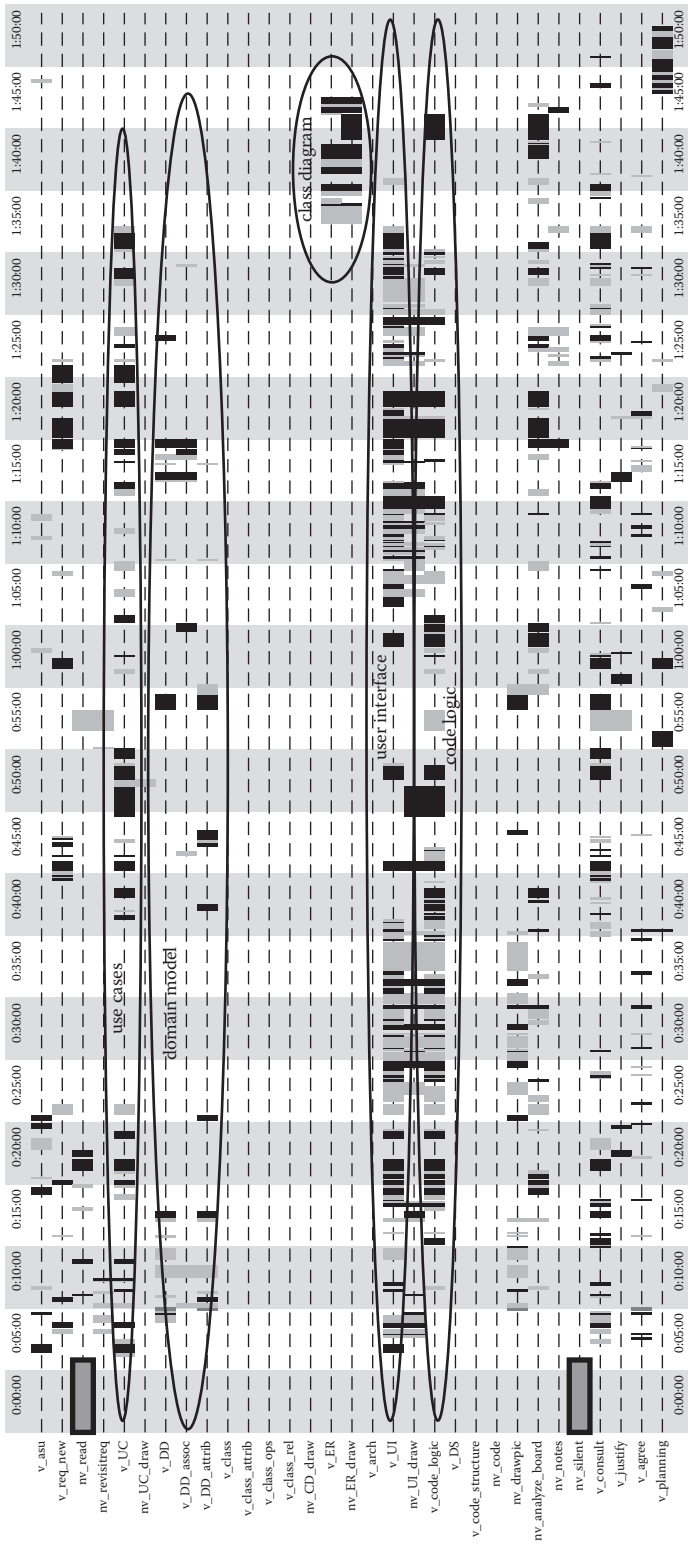

FIGURE 10.7 Time line matrix of activities for the AmberPoint session shows actions and speech mapped to various event codes. The events are colored blocks indicating the duration of the participation, with the color of the block indicating the speaker. The use cases, domain model, class diagram, code logic, and UI are circled.

chunks of activity on the *x* axis, as shown by the circled areas. In Figure 10.6, we observe that the designers discussed code logic (*v_code_logic*) starting at 20 minutes through to the end of the session; however, it is important to see how the logic evolved. At 20 minutes, they were discussing the logic with respect to the data dictionary (as seen by the overlapping domain model oval), whereas at 30 minutes, they had progressed to building the class model. During the entire session, they focused on the requirements, referring back to them at regular intervals.

Our second research question (RQ2) seeks to determine the design strategies or activity sets used by the designers. Examining the distributions of the activities in the time line matrix provides some insights into answering this question. We can determine the design strategies or activity sets by asking the following subquestions. We address each of these subquestions next.

RQ 2.1: Which design activity took the longest?

RQ 2.2: What was the sequence of activities for each session?

RQ 2.3: How many of the activities were interleaved?

RQ 2.4: Did the designers add additional requirements/features and make valid assumptions where appropriate?

The first five activities that took the most and least time are shown in Table 10.7. These are the first five and last five in the sorted time line matrices. Three activities (*v_code_logic*, *v_consult*, and *v_UC*) were common in all three sessions but they did not appear in the same order. In other words, in all three sessions, decisions about the logic of the code, discussions of use cases, and interactions between the designers played major roles. Other top time-consuming activities (RQ 2.1) included analyzing the whiteboard, drawing class diagrams, and talking about and drawing the UI (*nv_analyze_board*, *nv_CD_draw*, *v_UI*, and *nv_UI_draw,* respectively). Silence was much more prevalent in the Intuit session, where they spent the time analyzing the whiteboard.

TABLE 10.7 Top Five and Bottom Five Activities on which the Most and Least Time Was Spent

	Adobe	**AmberPoint**	**Intuit**
Most time spent	*v_code_logic*	v_UI	nv_analyze_board
	nv_analyze_board	*v_code_logic*	*v_code_logic*
	v_consult	nv_UI_draw	*v_consult*
	v_UC	*v_UC*	nv_silent
	nv_CD_draw	*v_consult*	*v_UC*
	:	:	:
Least time spent	nv_code	nv_CD_draw	v_req_new
	v_DD_assoc	v_arch	nv_UC_draw
	nv_UC_draw	v_DS	v_ER
	v_ER	v_code_structure	nv_ER_draw
	nv_ER_draw	nv_code	nv_UI_draw

Note: Activities in italic occurred in all sessions.

TABLE 10.8 Sequence of Main (Not Necessarily Interleaved) Activities Across Time Shown in 30-minute Increments (15-minute Increments for Intuit)

	Interval	Activities
Adobe	30 minutes	Identifying objects, drawing pictures
	1 hour	Attributes of objects, revisiting requirements, use cases
	90 minutes	UI and use cases
	2 hours	UML class diagram, code structure
AmberPoint	30 minutes	Drawing and talking UI, identifying objects, use cases
	1 hour	Drawing UI, use cases, scenarios
	90 minutes	UI and use cases
	2 hours	UI and ER diagram
Intuit	15 minutes	Identifying objects, drawing pictures
	30 minutes	Identifying objects, drawing pictures, silence
	45 minutes	Drawing diagram attributes and relationships, use cases
	1 hour	Use cases, domain objects, UI

The sequence of activities and the interleaving of those activities for each session varied, which can be attributed to individual differences in approaches to problem solving, expertise, or prior experiences. In order to analyze the sequence of activities (RQ 2.2), we split the time line into quarters (see Table 10.8). For the Adobe and AmberPoint sessions, each quarter was 30 minutes, while in the Intuit session each quarter was 15 minutes since they took only an hour for the complete session. We excluded the top five activities in this analysis in order to focus our attention on more specific tasks; the top five tasks were fairly general activities and were common to all sessions.

For the Adobe session, the first quarter was spent identifying the objects in the system (data dictionary and domain model) and drawing pictures of the road intersection. The next quarter was spent revisiting the requirements (*nv_revisitreq*), use cases (*v_UC*), and specifying the properties of the objects. In the third quarter, they concentrated on the UI and use cases. Finally, they focused on drawing the unified modeling language (UML) class diagram and writing the structure of the code.

For the AmberPoint session, the first quarter was spent talking about the UI, drawing pictures of the UI, and introducing domain objects and use cases. The second quarter focused much more on the UI with some discussion about the use cases and scenarios. The third quarter also focused on the UI and use cases. Finally, in the last quarter, they drew an ER diagram.

For the Intuit session, the first quarter was spent identifying domain objects and drawing pictures. In the next quarter, they continued with long periods of silence. The focus of the next quarter was on drawing a diagram, specifying attributes, relationships, as well as use cases. In the final quarter, use cases were discussed, including some discussion about the UI.

Observing the activity chart for Adobe in Figure 10.5, we get a visual indication of the types of interleaved activities in each quarter (RQ 2.3). The interleaved activities are often related. For example, in the Adobe session, when they talked about the UI, they also discussed user interaction (use cases). The activities of drawing pictures and identifying domain objects were also interleaved in all three sessions. This is evidence that drawing a

visual representation actually helps in identifying the data dictionary and conveying the thought process to the designers. In the Adobe session, the logic of the code was discussed almost consistently throughout and hence it interleaved the most with all other activities.

Next, we address RQ 2.4: coming up with new requirements and making valid assumptions. With respect to coming up with new requirements (*v_req_new*), the AmberPoint team did this more than the Adobe team, while the Intuit team addressed the subject very little. In the Adobe session, one of the new requirements was the ability to resize the simulation window. We consider this to be an important feature of the UI and the ability of the team to think ahead in terms of simulation usability.

In all three design sessions, assumptions (*v_asu*) were made about the requirements. For example, in the Adobe session, the assumption was made that roads have two lanes to address the left-hand turn requirement. The Adobe and AmberPoint teams made more assumptions than the Intuit team, some of which were critical to moving the design forward. One prevalent behavior in the Adobe session 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 to it at a later stage. This observation demonstrates the iterative process of design.

10.3.2 Requirements Analysis

We now address RQ3: whether the designers took into account all or only some of the requirements. The answer can be determined from the annotations for the requirements in the transcript. For example, the requirements coverage over time for the Adobe session is presented in Figure 10.4. Similar time line matrices showing the requirements coverage for the AmberPoint and Intuit sessions can be found on our companion website. Furthermore, we can determine which requirements were discussed most frequently (in terms of the amount of time spent). There might be a connection between the time taken for a requirement and the inherent complexity involved. However, a requirement that was never discussed is not necessarily less complex or easy. Due to the lack of a gold standard, we do not relate completeness with quality.

The top and bottom five requirements that took the most time and the least time are shown in Table 10.9. Three requirements (*intersect*, *alter_update*, and *light_seq*) were common in all three sessions but did not appear in the same order. The *intersect* requirement dealt with the approach used to describe and represent the intersection of the roads. It also included constraints such as *no one-way roads* or *T-junctions*.

In the Adobe and Intuit sessions, the intersection was the top requirement discussed. Since the AmberPoint session focused more on the UI, their top activity shows up as *alter_update*. This requirement involves interaction by the student using the software (as indicated in the problem statement), and deals with setting simulation parameters and updating the simulation (see Table 10.2). Finally, the *light_seq* requirement was common but all teams spent the least time on it: a scheme for updating the lights.

The Intuit and AmberPoint sessions both discussed the *wait_time* requirement, which states that ideally the wait time needs to be minimized. This was the sixth most talked about activity for the Adobe session. The Intuit team did not pay much attention to this

TABLE 10.9 Top Five and Bottom Five Requirements on Which the Most and Least Time Was Spent

	Adobe	AmberPoint	Intuit
Most time spent	*intersect*	*alter_update*	*intersect*
	alter_update	map_roads_intersect	set_lights
	map_roads_intersect	set_light_timing	*alter_update*
	left-hand_turn	*intersect*	left-hand_turn
	spec_density	wait_time	set_light_timing
	:	:	:
Least time spent	vis_lights	simulate	*light_seq*
	light_seq	*light_seq*	no_crash
	no_crash	set_lights	vis_lights
	sensor_option	road_len	observe
	t_time	t_time	wait_time

Note: Requirements in italics occurred in all sessions.

nonfunctional requirement. Since they finished earlier than the other two teams, they might have glazed over nonfunctional requirements as well as some functional requirements as stated in the following text.

The Adobe session addressed all the requirements of the design problem. The AmberPoint session failed to address the *t_time* requirement; however, they did address the related requirement *t_speed* (seventh highly discussed requirement). The Intuit session did not address the *no_crash*, *vis_lights*, and *observe* requirements. The *vis_lights* requirement states that the state of the lights should be visually conveyed to the user. The observation (*observe*) requirement deals with the students observing problems with traffic patterns and timing schemes. Since the Intuit team did not touch the UI aspect in detail, they missed these requirements. The requirements distribution across time can be compared side by side (or overlaid) with the activity distribution across time, to determine the activities involved to satisfy a given requirement. We discuss some instances of this next.

In the Adobe session, the first 10 minutes of the session was spent on the *intersect* requirement, which involved a lot of consultation between the designers as well as the identification of domain objects. The *intersect* requirement was revisited after a period of time, but then the designers described the objects and their properties in greater detail and even started sketching out a preliminary UML class diagram. We also see that the *left-hand_turn* requirement was interleaved with a discussion of the logic involved in its implementation.

In the AmberPoint session, the *intersect* requirement was talked about intensively in the second part of their discussion for ~20 minutes, which also overlapped with drawing an ER diagram. The first part of the session dealt with traffic light timing schemes (*set_light_timing*), which maps to the nonverbal activities of talking or drawing the UI (*nv_UI_draw* and *v_UI*) and discussing use cases and code logic. During the middle of the session, they discussed the altering of the simulation parameters (*alter_update*) for ~20 minutes, which maps nicely to analyzing the whiteboard, defining use cases, and discussing code logic. It was also interesting to note that they came up with new requirements (building multiple simulations and measuring traffic complexity) during the analysis of the *alter_update* requirement.

In contrast, the Intuit session's talk about the *intersect* requirement spread throughout the session for a total of ~20 minutes. This requirement maps to creating use cases, a domain model, and a class diagram; analyzing and drawing pictures on the whiteboard; and code logic. In the beginning, they discussed updating the current state of the lights (*set_lights*) simultaneously with the *intersect* requirement. These requirements map to creating use cases and the domain model. Finally, they talked about altering the simulation parameters (*alter_update*) for ~10 minutes at the end of the session, which maps to creating a class diagram and discussing code logic. It is important to note that even though each team worked in a different way, there was some commonality in the requirements met and the activities discussed, as presented in the previous sections.

10.3.3 Interpersonal Communication

We also examine the interactions between the designers (RQ4), and their effects on the resulting design. Again, we derive this information from the time line 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 (Figures 10.5 through 10.7).

The Adobe session (see Figure 10.5) contained the most agreement between the two designers. It also had consulting activity scattered almost uniformly. They engaged in a lot of planning in the last three intervals and also justified their choices. The AmberPoint team also consulted throughout their session with a lesser level of agreement between the designers. They did much more planning almost at the very end of the session with justification provided for each decision made. The Intuit team also consulted throughout their session and they were consistently in agreement. However, there were few justifications made for their decisions and little planning involved. There were no silent episodes in the Adobe or AmberPoint sessions, while the Intuit session had periods of silence spread throughout. To conclude, we can hypothesize that planning, as well as the high degree of agreement between the two designers of the Adobe session, played a significant role in delivering the most detailed design that covered the most requirements.

10.3.4 Similarities and Differences in Design

We will briefly point out the similarities and differences between the approaches used between the design teams (RQ5). The comparison of the sessions is based on the two main deliverables asked for in the design problem: (1) the design interaction/UI and (2) the basic structure of the code.

The Adobe team 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 structures. There was a lot of consulting and agreement between the designers. The AmberPoint team focused on the UI and did produce the deliverable of "an interaction scheme for students." However, they did not spend any time on the structure of the code or even identify associations between the main objects in the system. The Intuit team went through the session very quickly. Their design activity time line matrix is much less dense compared to

the other two teams. While they did not design a UI or outline a code structure, they did discuss the code logic (*code_logic*).

The problem that the designers were given was to complete enough of a design such that a student who had recently graduated could implement it. The results of the sessions for teams Intuit and AmberPoint did not achieve this. Conventional wisdom leads us to believe that a software developer would be likely to produce a correct and conforming implementation based on the design provided by the Adobe team because it gives the most detail and covers the most requirements.

10.4 OBSERVATIONS

The three teams took very different approaches to problem solving. Based on the time line matrices, we observed clusters that clearly correspond to the design process stages, that is, requirements, business modeling, design, and implementation; which are shown in Figures 10.6 and 10.7. Some of the observations based on the foregoing analysis are as follows.

- The requirements and use cases were spread throughout the session.
- The data dictionary and the domain model were done at the beginning for the Adobe and Intuit teams and were scattered throughout for the AmberPoint team.
- The class diagram/design model was discussed next by the Adobe and Intuit teams while the AmberPoint team focused on the ER diagram.
- The discussion about the UI was done mostly in the second part of the design sessions for the Adobe and Intuit teams, but it was scattered throughout the AmberPoint session.
- Code logic/implementation was also scattered throughout the session for all teams.

Overall among the three teams, 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 worst design. However, we can make some general points. Two teams (Adobe and Intuit) did a better job at the code logic and system design, while the third team (AmberPoint) did a better job describing and outlining the UI. Curiously, only one team (AmberPoint) discussed the aspects of verification or validation with regard to their solution. We also observed that only two teams (Adobe and Intuit) identified the fact that they had to deal with queues in a traffic simulation problem. We consider the queue as an important part of the design of this system. AmberPoint did not mention this at all, possibly due to a lack of prior experience with simulation systems. In other words, the design teams with different experiences and backgrounds focused on different aspects of the design.

One open question is the relationship between the problem statement and the design process. Did the object-oriented design process help or hinder the final design? Also, did the domain affect the design? Note, the teams were asked to come up with a high-level design, they were not specifically asked to come up with an object-oriented design.

10.5 CONCLUSIONS AND FUTURE WORK

We presented an analysis of three design videos and their associated transcripts. Analyzing video recordings is a time-consuming process and requires an organized approach. As such, we needed to choose a guiding principle for our investigation and develop tools to alleviate some of the manual drudgery. In order to support the analysis, we developed an XML format for embedding dialogs and dialog metadata and we developed an annotation tool to assist in the manual annotation of speech with metadata. We also presented a means to visually compare the data with a time line matrix that was used to analyze and draw inferences about transcribed and videotaped conversations.

We posed a number of research questions on how developers actually design and we addressed them individually. We conducted a fine-grained analysis of the requirements as well as the design activities engaged in by the designers. Another contribution was the generation of various codes for both design and requirements (albeit problem specific). The design codes, however, span across various domains and may be reused.

As future work, we plan to further develop the visualizations, and create an interactive time line 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 a finer-grained 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 (Gentner and Stevens 1983) of software engineering and program comprehension.

REFERENCES

Artstein, R. and M. Poesio (2008). Inter-coder agreement for computational linguistics. *Computational Linguistics* **34**(4): 555–596.

Bryant, A. and K. Charmaz, Eds (2007). *Grounded Theory*. Sage, Los Angeles, CA.

Curtis, B., H. Krasner, and N. Iscoe (1988). A field study of the software design process for large systems. *Communications of the ACM* **31**(11): 1268–1287.

d'Astous, P. and P. N. Robillard (2001). Quantitative measurements of the influence of participant roles during peer review meetings. *Empirical Software Engineering* **6**: 143–159.

Detienne, F. and F. Bott (2001). *Software Design: Cognitive Aspects*. Springer, London.

Friess, E. (2007). Decision-making strategies in design meetings. In *CHI '07 Extended Abstracts on Human Factors in Computing Systems*. ACM Press, New York, pp. 1645–1648.

Gee, J. P. (2005). *An Introduction to Discourse Analysis*. Routledge, New York.

Gentner, D. and A. L. Stevens, Eds (1983). *Mental Models*. Erlbaum, Hillsdale, NJ.

Guindon, R., H. Krasner, and B. Curtis (1987). Breakdowns and processes during the early activities of software design by professionals. In G. M. Olson, S. Sheppard, and E. Soloway (Eds), *Empirical Studies of Programmers: Second Workshop*. Ablex, Norwood, NJ, pp. 65–82.

AU: Ref. Friess 2007: Please verify page range.

AU: AU: Ref. Guindon et al. 1987: Please verify third author's name and page range.

Ko, A. J., B. A. Myers, M. J. Coblenz, and H. H. Aung (2006). An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering* **32**(12): 971–987.

Maia, A. C. P., C. J. P. de Lucena, and A. C. B. Garcia (1995). A method for analyzing team design activity. In *Proceedings of the Conference on Designing Interactive Systems*. ACM Press, Ann Arbor, MI, pp. 149–156.

Robillard, P. N., P. d'Astous, F. Détienne, and W. Visser (1998). Measuring cognitive activities in software engineering. In *International Conference on Software Engineering*. IEEE Computer Society, Washington, DC, pp. 292–299.

University of California, Irvine (2010). Studying professional software design. An NSF-Sponsored International Workshop. University of California, Irvine, CA.

AU: Ref. Maia et al. 1995: Please verify authors' names, publisher's name and page range.

AU: Ref. Robillard et al. 1998: Please verify publisher's name and location and page range.