Reasoning of novice and experienced software designers on creating UML class diagrams

Master of Science Thesis in the Programme Software Engineering

TIANJIE MA

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, November 2013

Reasoning of novice and experienced software designers on creating UML class diagrams

Tianjie Ma

Examiner: Patrizio Pelliccione
Supervisor: Michel Chaudron

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden November 2013

Abstract

Currently, UML class diagrams are widely used in Software Engineering, especially on software design process. Simultaneously, people realize that the reasoning plays a significant role in software design process. This study is investigating the reasoning of novice and experienced software designers on creating UML class diagrams by interviewing student and expert participants. It concludes that (1) the reasoning of novice and experienced software designers on creating UML class diagrams can be described using seven reasoning activities {RS, DC, DA, DM, DAM, CD, NO}
(2) experienced software designers perform more reasoning activities and spend more time on creating UML class diagrams than novices ones (3) some beneficial phenomena relevant to the reasoning process are found on reasoning structure, reasoning confidence, reasoning sources and reasoning principles.

Key words: reasoning, UML class diagram, novice participant, experience participant

Acronyms, Abbreviations and Terms

UML	**Unified Modeling Language**
TAP	**Think-aloud-Protocol**
SE	**Software Engineering**
RS	**Read the user specifications and use cases**
DC	**Determine the class and its name**
DA	**Determine the attributes of the class**
DM	**Determine the methods of the class**
DAM	**Determine the associations and its multiplicities**
CD	**Check the created class diagram**
NO	**Notations related to the design**

Table of Contents

1 Introduction

Nowadays, software has become essential in daily life as it is generally applied in smart phones, computers as well as the other intelligent terminals. As is known to all, the software design is the key process during the whole lifeline of software development, which is directly having an effect on whether the software succeeds or not [6, 18]. UML, as one of the most widely used software design modelling tools, is frequently mentioned in software design area. Its class diagram, which is describing the whole structure of the software at the early stage, especially is widely used in software design process [5, 16, 30]. Moreover, during the software design process, the reasoning plays a more significant role [7, 25]. In terms of Reasoning, by definitions, is “the process of forming conclusions, judgments, or inferences from facts or premises” [11]. Thus, the reasoning process is extremely vital to make satisfied software designing in SE [17, 18, 19].

In recent years, people have been making efforts to find a better approach for better software designs on UML class diagrams modelling. In other word, the way of creating a better class diagram is desirable. However, due to the different educational background and different practical experience, the software designers have different thought and behaviours even on facing the same domain problems. Meanwhile, to my knowledge there are little research papers on investigating the reasoning of software designers on creating UML class diagrams. As a result, understand the reasoning of software designers on creating UML class diagrams can be helpful and meaningful for people doing a better software design in industry as well as improving the software quality in the future.

This thesis study is mainly for investigating the reasoning of novice and experienced software designers in order to understand their thoughts and behaviours while facing the specific domain problems. Also, this study will compare novice and experienced software designers to find out the difference between them, understanding the different software design process of different level designers. Therefore, the study will address the following research questions:

RQ1: *What is the reasoning of novice and experienced software designers on creating UML class diagrams?*

RQ2: *What is the difference between novice and experienced software designers on creating UML class diagrams?*

To answer the research questions, the interviews, with the selected participants who are treated as novice and experienced software designers, have been carried out. The participants would make a UML class diagram about tour/reservation online system as the task during the interview process and after that, they were asked few questions related to their designs. Simultaneously, owing to the interview process applied TAP method, all interviews were video-recorded for data analysis. Later, the collected interview data was both translated into the coded transcripts as well as its reasoning activity sequence diagram and timing distributes. Finally, the results or findings were summarized based on these interview transcripts of the participants.

The paper is structured as follows: Section 2 introduces backgrounds of UML class diagram and TAP method; Section 3 lists some selected literature studies relevant to this study; Section 4 highlights the research process and Section 5 presents the results and findings; Section 6 makes discussions based on the results and findings; Section 7 explains the threats of the study; finally, Section 8 suggests the further research and Section 9 draws the conclusions.

2 Background

2.1 UML class diagram

UML, as one of the design modeling tool for graphical representations of software system, has been widely used and well-accepted on software design in software engineering industry as well as daily teaching in software engineering education [5, 16, 30, 32]. For UML modeling in practice, the class diagram is the most frequently used [16]. Therefore, the role of class diagram becomes even more important in depicting the problem domain clearly [33].

For UML class diagram, which is the type of static structure diagram, it is the foundational UML model. UML class diagram is mainly used to describe the whole structure of target software system or application, consisting of classes, their attributes, methods or operations and the associations (including association name and multiplicity) between classes [5, 16].

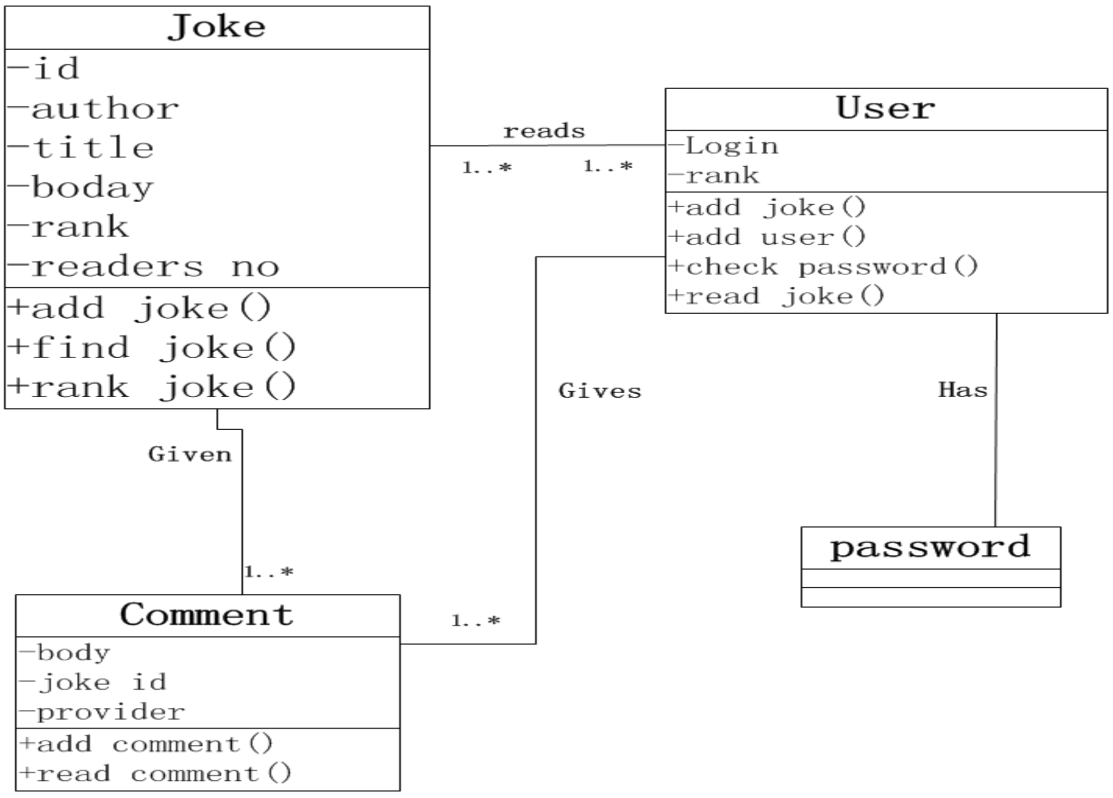


Figure 2-1 Example of a class diagram

This is a small example of Joker-teller system’s class diagram from book – “UML by examples” [5] (see Figure 2-1), explicitly showing the classes- Joke, User, Comment, password and the relationships between them. Moreover, the example clearly presents the attributes and methods of the classes, as well as the association names and its multiplicities.

2.2 Think-aloud Protocol

During this thesis study, the Think-aloud Protocol (TAP) is adopted as the experiment method. It is known that TAP is a good way of understanding the thoughts of the subjects while they perform a set of tasks for researchers [12, 26]. Simultaneously, according to Van Someren et al, it has two main strengths than other similar research methods:

1. **"THE THINK ALOUD METHOD AVOIDS INTERPRETATION BY THE SUBJECT AND ONLY ASSUMES A VERY SIMPLE VERBALIZATION PROCESS" [20].**
2. **"THE THINK ALOUD METHOD TREATS THE VERBAL PROTOCOLS THAT ARE ACCESSIBLE TO ANYONE, AS DATA THUS CREATING AN OBJECTIVE METHOD" [20].**

Here is one typical definition of TAP method:

"By thinking aloud while attempting to complete the task, user can explain their method of attempting to complete the task, and illuminate any difficulties they encounter in the process." [12]

In brief, TAP method allows subjects thinking aloud as they are doing the specific tasks and the subject need to say whatever they are seeing, thinking and feeling through the process of doing tasks. The researchers cannot interrupt the thought process of the subjects. Actually, the researchers only can keep remainder of "Please keep talking" if the participants become silent while doing the tasks. In general, the experiment should be audio-recorded or video-recorded such that the researchers can observe what the subjects did or reacted again and again. As a result, it is easy and convenient to analyze the experiment data in order to make a better research. [12, 20, 26]

3 Related work

In fact, there is little study published for researching the reasoning of software designers on creating UML class diagrams so far. But, there are many relevant research papers providing beneficial inspirations for this study. The selected literature reviews are interpreted briefly as follows:

- Gero and McNeill described an approach which was using protocols to make an analysis on the process of designing. Coding schemes were used as design protocols, making the collected data more structured in order to understand design process deeply and better. The results were proved that this approach was indeed gaining some insight into the analysis of how designers designed.[9]

- Schenk, vitalari and Davis carried out a research about investigating the difference in the problem-solving methods of novice and expert system analysts while they were facing the information requirements tasks. For findings, the verbal protocols explicitly indicated that differences in the problem-solving processes of novices and experienced systems analysts were presented in the quantity of verbalizations and task duration, as well as level of detail. [1]

- Zannier, Chiasson and Maurer took a research on that how design decisions made in the software design process by interviewing 25 software designers in SE industry. They concluded that the structure of the design problems determined the using of rational and naturalistic decision making. In short, they summarized “the more structured the design decision, the less a designer considers options” [8].

- Atman et al. conducted an in-depth research, which was comparing the student and expert participants on the engineering design process. The results supported that the major differences of them in the engineering design process on two aspects – problem scoping and information gathering. [10]

- Tang and Aleti investigated how human reasoning worked in the process of problem domain and solution domain during the software design. Reasoning techniques used by software designers were found, such as design structuring and contextualization, which were directly having a deep impact on software design results. Meanwhile, these found reasoning techniques could improve the understanding of how software designers thought and solved problems during the design process, in partly.[7]

4 Methodology

4.1 Goal of the research

The goal of this study is to investigate how novice and experienced software designers creating UML class diagrams while meeting the concrete practical domain problems in order to understand the reasoning of them creating class diagrams generally, as well as to be clear about their differences. Simultaneously, the other findings during the study related to the reasoning are also valuable.

4.2 Research process

The study is primarily to interview the selected subjects to understand the reasoning how they creating UML class diagrams as well as their differences. During the interview process, the study is employed the TAP method to realize the research goals.

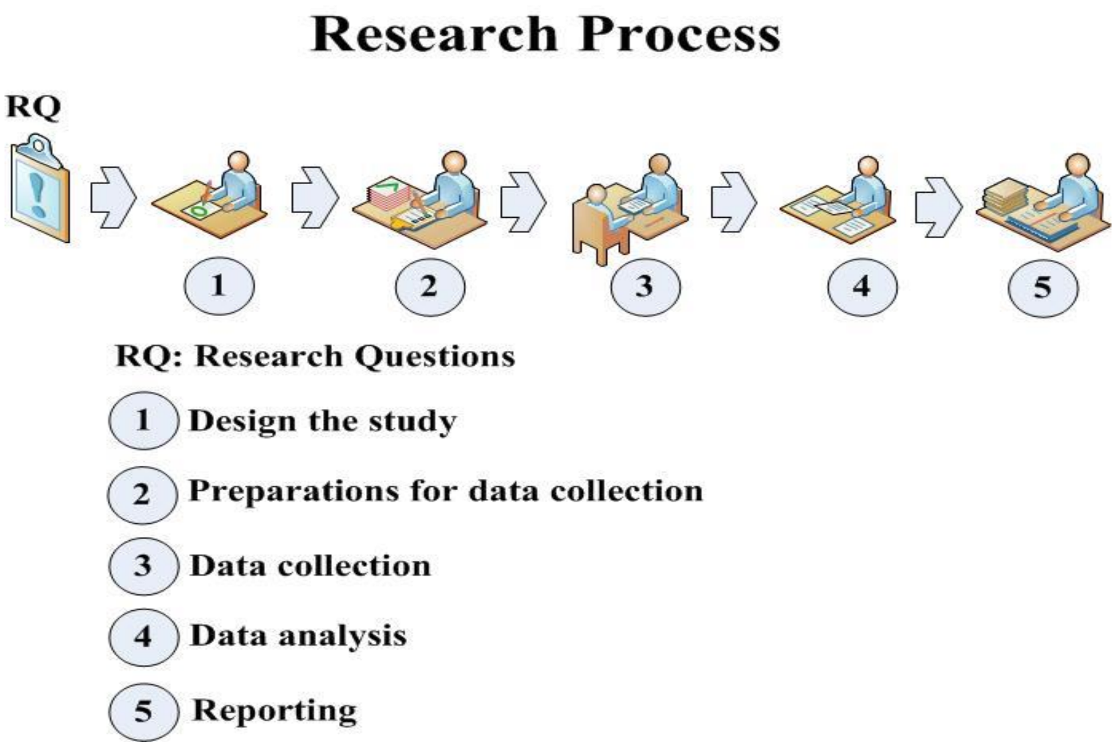


Figure 4-1 Research process of this study

For the whole research process, it can be divided into these following steps (see Figure 4-1):

- **DESIGN THE STUDY**
- **PREPARATIONS FOR DATA COLLECTION**
- **DATA COLLECTION**
- **DATA ANALYSIS**
- **REPORTING**

4.2.1 Design the study

As mentioned earlier, this study is applied with TAP method to interview the representatives of novice and experienced software designers. And during the interview process, the subjects are performing the design tasks by using the interview guide.

-Subject

In total, this study carried out 16 interviews but 2 interviews were not involved for final data analysis because of the invalid interview data, which means that the participants did not offer efficient information to the study. Interviewee 15 felt difficult on the task because he had not used UML for modelling for a long time. Interviewee 16 did not understand the interview guide well so that he only spent five minutes to complete the task without thinking aloud. Perhaps owing to the language, the subject did not clarify what he should do. (Participant #16 is Chinese but the interview guide is written by English)

The selected subjects are divided into two types – novice software designers and experienced software designers. Both of participants have previous experience with the UML modelling, who coming from China, Ethiopia, Germany, Iceland, India, Jordan and Sweden.

There were 9 samples of which were 3 females and 6 males as novice software designers for data analysis. All of the participants as novice software designers were studying students, whose diplomas were master or doctor levels. 8 people are studying master students from Chalmers University of Sweden, and the major of who is both software engineering. Another one (participant #11) is a PHD student from Leiden University of Netherland. His major is software engineering, too.

On other hand, there were 5 participants and both of them were males as experienced software designers for data analysis. All of them had enough software designing experience in industry. 4 people have more than ten years experience and come from Ericsson and Knowit Company of Sweden. Another one (participant #10) is a current PHD student from Chalmers of Sweden but he is an assistant teacher of teaching UML courses, in order that he also has enough experience on UML modelling. For their majors, 3 people (participant #9, #12, #13) are electronic engineering. The other two are respectively computational linguistics (participant #10) and computer science (participant #14).

Here is the table including the basic information for these 14 participants.

No	***Sex***	***Category***	***Nationality***	***Current Situation***	***Working experience***	***UML project***
#1	F	Novice	India	Master student	Around one year	**1 – 5**
#2	F	Novice	Germany	Master student	None	**1 – 5**
#3	F	Novice	China	Master student	Around one year	**1 – 5**
#4	M	Novice	Sweden	Master student	Around one year	**1 – 5**
#5	M	Novice	Ethiopia	Master student	Around half year	**1 – 5**
#6	M	Novice	China	Master student	None	**1 – 5**
#7	M	Novice	Sweden	Master student	None	**1 – 5**
#8	M	Novice	Iceland	Master student	Around one year	**1 – 5**
#9	M	Experienced	Sweden	Working at Knowit	More than ten years	**>=10**
#10	M	Experienced	Sweden	PHD student	Around two years	**>=10**
#11	M	Novice	Jordan	PHD student	Around three years	**1 – 5**
#12	M	Experienced	Sweden	Working at Ericsson	More than ten years	**>=10**
#13	M	Experienced	Sweden	Working at Ericsson	More than ten years	**>=10**
#14	M	Experienced	Sweden	Working at Knowit	More than ten years	**>=10**

Table 4-1 Basic statistical information for participants

-Interview guide

During the interview process, the participants were using an interview guide to drive the experiment. The interview guide was mainly containing two parts – one was the background questionnaire and the other one was the task specification. With the help of interview guide, the participants could easily and clearly understand the purpose of the study, how the interview was going on as well as some essential information about the study.

For the first part of background questionnaire, all the questions were simple and easy to answer. Its type is the multiple-choice question so as to be well accepted by the participants. The questions covered three aspects, which were study background, work experience and UML experience. Certainly, if the interviewees had comments or suggestions, they could write down their opinions on the comment part.

For the second part of task specification, the participants were going to draw the initial UML class diagram based on the user requirements and use case models of a reservation/tours online system. This case was chosen from the book- “UML by Example” [5]. The reason why selected this case was that the case domain was simple enough in order that the participants had no troubles of performing the tasks, and also people might have this kind of ordinary experience in daily life, such as going to travel abroad.

Here is the reservation/tours online system user requirement as presented.

> *"Software for a travel agency provides reservation facilities for the people who wish to travel on tours by the travel bureau. The application software stores information about tours. Users can access the system to make a reservation on a tour and to view information about the tours available without having to go through the trouble of asking the employees at the agency. The third option is to cancel a reservation that was made. Any complaints or suggestions that the client may have could be sent by email to the agency or stored in a complaint database. Finally, the employees of the corresponding agency could use the application to administrate the system's operations. Employees could add, delete, and update the information on the customers and the tours. For security purposes, employees should be provided a login ID and password by the manager to be able to access the database of the travel agency".* [5]

Here is the use case model of reservation/tours online system as presented. [5]

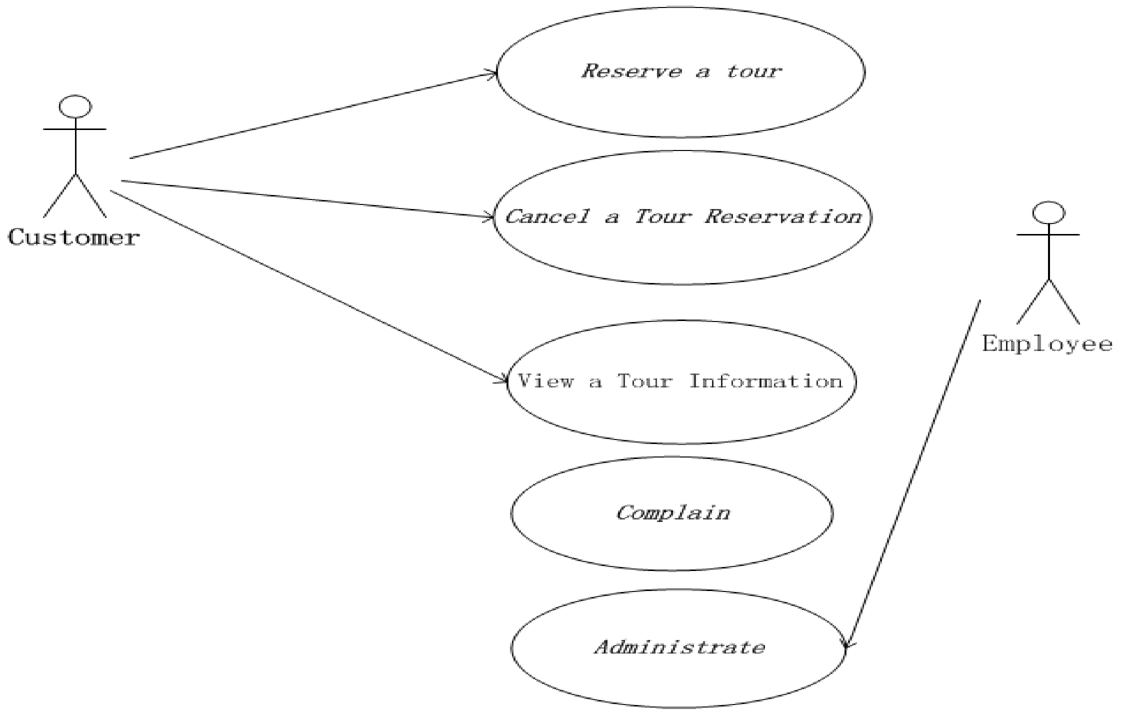


Figure 4-2 Use case model of reservation/tours online system

The complete interview guide document is in the appendices section. (Please see Appendix A)

4.2.2 Preparations for data collection

After designing the study, it is the time to prepare the needed materials and information for study. For preparations, it is necessary to print the interview guide documents for the participants and find an appropriate place, book a suitable time in order to interview the selected subjects. Meanwhile, because the interviews are video-recorded for data analysis in the further, it is essential to prepare the digital devices, such as smart phones or cameras.

4.2.3 Data collection

The primary job in this step is to collect interview data and relevant information from the interviews of the subjects. The interview process had been conducted from April to August in 2013 of which mostly were held in Gothenburg of Sweden and only one was in Jinan, Shandong of China. As explained earlier, not all the interviews were involved in the final data analysis which was that interview data of participant #15 and #16 were obsolete. Generally, the average interview time was one hour, and one and half hour at most and 40 minutes at least. The majority of interview place was in meeting room which located in Chalmers University of Sweden and the others were in conference room at Ericsson or Knowit Company of Sweden.

The interview process was based on the interview guide and video-recorded. The participants started a background questionnaire and next, the interviewees faced the concrete domain problem including specific tasks they needed to complete in a limited time. After that, the participants were asked by a set of questions based on their designs. Finally, the subjects needed to summarize and write down the common steps on how they created the class diagram of this case. For the whole interview process, the attitude of all participants was serious and they tried their best to make a better solution.

In terms of the questions, there were five questions relevant to the created class diagrams.

Here are these five questions:

1What reasoning did you use for deciding what should be classes in your diagram?

2What reason did you use for determining attributes and methods of the classes?

3What reasoning did you use for deciding associations between classes?

4Did you think of any design principles while designing the class diagram?

5What's your naming strategy? i.e. how did you think of the names of your class diagram?

In summary, during the interview, the main task for participants was to create the initial class diagram of reservation/tours online system. Moreover, the process was applied in TAP method and the interviewees should keep talking during the whole interview in order to be understood for how they were creating UML class diagram as well as the reasoning when they were performing the task. All the interviews were recorded as videos for further and deep analysis with the digital camera. It was easy and convenient of going back to check what participants performed and thought during the interview time. In addition, as the conductor of the study, I could not disturb the interviewees and just kept reminder of "please keep talking" if the subjects became silent as the request of TAP method. In the other time of interview, I just made notes and listened to the words of subjects.

The complete interview data document is in the appendices section. (Please see Appendix B)

4.2.4 Data analysis

By having collected interview data from the interviews of the participants, the main task of this step is to analyses, explain, discuss and summarize the collected data. Based on the goals of this study, it should find the behaviours and thoughts of different software designers while they create UML class diagram such that clarifying the reasoning of them how to create class diagrams when meeting the specific domain problems as well as their differences.

The results and findings of this study are illustrated in the result section.

4.2.5 Reporting

The final job of the research is to write this paper and present the findings, suggestions and future improvements.

5 Results

In this section, the results of this research will be presented to illustrate the reasoning of how novice and experienced software designers created UML class diagrams as well as their differences and other relevant findings.

5.1 General reasoning process

To elaborate the reasoning of the participants created UML class diagrams, I will present the collected interview data of one selected novice and experienced participant respectively, showing their reasoning process generally.

5.1.1 Novice software designer

There are 9 participants as novice software designers for data analysis. It is impossible to present all the subjects' data so one of them is chosen to present it. Selected participant#3 as representative and the results will be interpreted by four aspects, which are interview transcript, coded interview transcript, reasoning activity diagram and reasoning activity timing distributes.

5.1.1.1 Interview transcript

During the interview, the participants were keeping talking on how they thought about on creating UML class diagram, in order to be understood by their reasoning on it. Their talking words were written down as interview transcript and the interview transcripts were both based on the interview videos of each participant

Here is the participant #3 original interview dialogue transcript.

[Read the user requirements] First, we need to make sure that how many entities we should have in this class diagram. Of course, the first is the user and then I define the user as an abstract class. For this abstract class, it has two subclasses which are the customer and the employee. Next, the entity is the reservation. [Read the user requirements]Mm… I guess there should be another entity called tour. [Read the user requirements]I guess it should be an entity named as complain. Also, let's see the manager. I think it is the type of the user and I add it called administrator. [Find and speak out the evidence from the user requirements to support the existing of the created entities]OK. The customer can add reservation and one customer has several reservations. The tour and the reservation should be also connected. One reservation could be only to one tour but one tour, you can make several reservations depending on your information about the tour. (Ask whether the attributes can be defined freely) OK. As the user, I want to have the name, gender and perhaps the userID. As the customer, I don't think I need to add more attributes. Let's go back the tour. It should contain the duration, destination and how many reservations could make about the tour. How should I name it? Amount? Ok,

that's it. [Check the created class diagram]And the customer could also view tour and cancel reservation. Oh, I forgot something important. The tour should have TourID and also the reservationID is in the reservation class. And in the reservation, it should contain the total amount of the reservations and the attribute is the amount. [Read the user requirements]OK. The customer could also add complaints. One customer can make several complaints and for the class of complain, it should have the category, like I want to complain your website or complain your service in different areas. And also it should have the attribute of the comment. [Check the created class diagram] The employee could add/delete/update both users and tours. So, I add these methods to the class of employee. And the employee would be connected customer and the tour. I think one employee could manage several customers and one customer could be managed by several employees. Also, one employee can manage several tours and one tour could be managed by several employees. [Read the user requirements] The employee could have the attributes of login ID and password. Because the employees should be provided a login ID and password by the administrator so the entity of administrator should also have the same attributes of login ID and password. The difference is that the administrator can add/update/delete the employees. [Specify the user means to the customer in the class of employee] And only one administrator could manage several employees. The customer? The customer can view tour information such that there should be an association between them. (Think for a while) Maybe, the tour should have the attribute of status, like the tour is available or is closed. And the tour should have the action of viewing available tour which only returning the available tours, but I am not quite sure. (Think for a while and finish the left association's name) [Check the created class diagram]Mm…OK, I think everything is done.

This transcript reflects the whole thought process of participant #3 while she created UML class diagram of reservation/tours online system as well as the behaviors among it. Her talking words are truly presented, and among the words, it is filled by relevant behaviors which are marked by symbol [] and (), such as [Read the user requirements] and (Think for a while). No matter the talking words and the relevant behaviors, they are both belonging to the corresponding reasoning activity of how to create UML class diagrams. The reasoning activity will be detailed discussed in the next part.

5.1.1.2 Coded interview transcript

Reasoning activity

As just mentioned, to understand the reasoning of how subjects created UML class diagram as well as to analyze the interview transcripts better, the interview transcripts of them are coded, which are translated into seven reasoning activities. The reasoning activities are consisting of RS (Read the user specifications and use case), DC (Determine the class and its name), DA (Determine the attributes of the class), DM (Determine the methods of the class), DAM (Determine the associations and its multiplicities), CD (Check the created class diagram) and NO (Notations related to the design). On one hand, the classification for these seven reasoning activities is mainly based on the composition of one UML class diagram, for instance, DC, DA, DM and DAM. On the other hand, the other three reasoning activities (RS, CD and NO) are related to what the participants thought and performed during the interview process.

RS – Read the user specifications and use cases

This is the fundamental reasoning activity of creating UML class diagrams. It is obviously known that reading the user requirements and relevant information about the problem domain is the first step of each software designer does in software design. This is also a good way of capturing the whole picture of the problems for problem solvers, and it is the primary source of getting inspired for software designers to make a better solution in the end. In the study, the user requirements and use case model are offered to the participants in order that they just need to refer to them freely.

DC –Determine the class and its name

This is the beginning reasoning activity of creating UML class diagram, which is meaning that the UML class diagram is going to set out at the moment. The class is the core part of the UML class diagram and this reasoning activity will affect the other reasoning activities directly and deeply, especially on DA and DM.

DA – Determine the attributes of the class

This reasoning activity is close connected to DC. As one composition of the class, attributes are needed to be considered once the classes are built. The attributes information may clearly exist in the requirement specifications, or need to be derived by the participants.

DM – Determine the methods of the class

This reasoning activity is similar to activity DA, and also it has a close relationship with the DC. As another important part of the class, the methods/operations are needed to be thought about while the classes are created, too. The sources of methods information are the same as the attributes of the class.

DAM – Determine the associations and its multiplicities

This reasoning activity is significant, which decides the relationships among the created classes and its corresponding multiplicities. The relationship is another core composition of the UML class diagram, like the class. Owing to the relations, it is clear and easy to understand how different classes connect for each other and their mapping associations.

CD – Check the created class diagram

This reasoning activity is a type of verification behavior, and it is a necessary job to check the created classes and their relationships. Finishing the class diagram is not meaning that the work is completely done. To make a better solution of the problem, the design should be improved again and again. The CD reasoning activity can be taken place at any moment of creating UML class diagram once the first entity is built. In a word, this reasoning activity guarantees the software designers make a better design as far as possible.

NO – Notations related to the design

Different from the six reasoning activities mentioned above, NO is an optional reasoning activity so that only a few participants performed it during the interview process. This reasoning activity is to describe the notation, opinions or ideas relevant to the design of the class diagrams which were come up with by the participants. It is helpful to understand what they really considered and how their decisions were made in the problem solving.

Now, present the coded interview transcript of participant #3. (Different reasoning activities are signed as different colors in order to be distinguished)

RS: [Read the user requirements]

DC: First, we need to make sure that how many entities we should have in this class diagram. Of course, the first is the user and then I define the user as an abstract class. For this abstract class, it has two subclasses which are the customer and the employee. Next, the entity is the reservation.

RS: [Read the user requirements]

DC: Mm… I guess there should be another entity called tour.

RS: [Read the user requirements]

DC: I guess it should be an entity named as complain. Also, let’s see the manager. I think it is the type of the user and I add it called administrator.

CD: [Find and speak out the evidence from the user requirements to support the existing of the created entities]

DM, DAM: OK. The customer can add reservation and one customer has several reservations. The tour and the reservation should be also connected. One reservation could be only to one tour but one tour, you can make several reservations depending on your information about the tour.

DA: (Ask whether the attributes can be defined freely) OK. As the user, I want to have the name, gender and perhaps the userID. As the customer, I don’t think I need to add more attributes. Let’s go back the tour. It should contain the duration, destination and how many reservations could make about the tour. How should I name it? Amount? Ok, that’s it.

CD: [Check the created class diagram]

DM: And the customer could also view tour and cancel reservation.

DA: Oh, I forgot something important. The tour should have TourID and also the reservationID is in the reservation class. And in the reservation, it should contain the total amount of the reservations and the attribute is the amount.

RS: [Read the user requirements]

DM: OK. The customer could also add complaints.

DAM, DA: One customer can make several complaints and for the class of complain, it should have the category, like I want to complain your website or complain your service in different areas. And also it should have the attribute of the comment.

CD: [Check the created class diagram]

DM: The employee could add/delete/update both users and tours. So, I add these methods to the class of employee.

DAM: And the employee would be connected customer and the tour. I think one employee could manage several customers and one customer could be managed by several employees. Also, one employee can manage several tours and one tour could be managed by several employees.

RS: [Read the user requirements]

DA: The employee could have the attributes of login ID and password. Because the employees should be provided a login ID and password by the administrator so the entity of administrator should also have the same attributes of login ID and password. The difference is that the administrator can add/update/delete the employees.

DAM: And only one administrator could manage several employees. The customer? The customer can view tour information such that there should be an association between them.

DA: (Think for a while) Maybe, the tour should have the attribute of status, like the tour is available or is closed.

DM: And the tour should have the action of viewing available tour which only returning the available tours, but I am not quite sure.

DAM, CD: (Think for a while and finish the left association's name) [Check the created class diagram]

Mm…OK, I think everything is done.

The coded interview transcripts are translated into the corresponding reasoning activities. Reasoning activity NO is not shown up inside such that this transcript is merely translated by six reasoning activities. What's more, the finishing sentences are loyally presented and there is no special color to mark it.

Except for coded interview transcripts, each participant has own reasoning activity sequence diagram and corresponding timing distributes of creating UML class diagrams.

-Reasoning activity sequence diagram

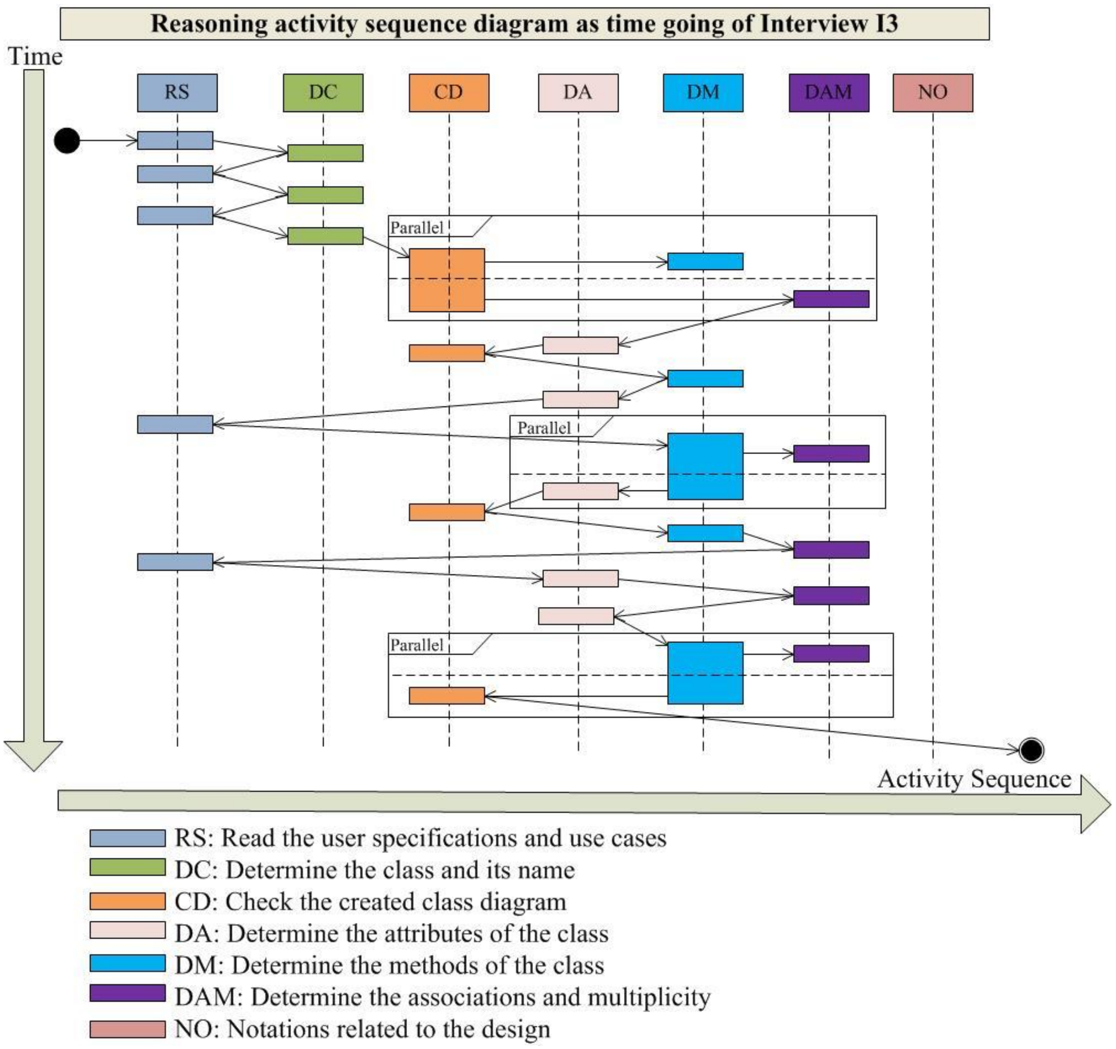

Reasoning activity sequence diagram as time going of Interview I3
Time
RS
DC
CD
DA
DM
DAM
NO
Parallel
Parallel
Parallel
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Figure 5-1 Reasoning activity sequence diagram of participant #3

It reflects the happening sequence of reasoning activity as time going based on the coded interview transcript of participant #3. (see Figure 5-1) Each reasoning activity has own timeline by its color and the colorful rectangles just stand for corresponding reasoning activities. The reasoning activities are sequentially connected, which is starting from symbol● and ending with symbol◉. The full line with an arrow points from one reasoning activity to the next happening activity and in some situations, two reasoning activities take place at the same time after the former one presented in the parallel combine fragments. Moreover, the arrows of coordinates show the happening sequence of the time and reasoning activity.

-Reasoning activity timing distributes

Reasoning Activity	***Start Time***	***Finish Time***	***Duration***	
			Minutes	**Seconds**
RS	***Default time***		***5m***	
DC	[00:00:00]	[00:01:50]	***1m***	***50s***
RS	[00:01:50]	[00:02:30]		***40s***
DC	[00:02:30]	[00:02:50]		***20s***
RS	[00:02:50]	[00:03:18]		***28s***
DC	[00:03:18]	[00:05:06]	***1m***	***48s***
CD	[00:05:06]	[00:05:27]		***21s***
DM, DAM	[00:05:27]	[00:07:23]	***1m***	***56s***
DM: [6s]		DAM: [1m50s]		
DA	[00:07:23]	[00:08:59]	***1m***	***36s***
CD	[00:08:59]	[00:09:12]		***13s***
DM	[00:09:12]	[00:10:11]		***59s***
DA	[00:10:11]	[00:11:11]	***1m***	
RS	[00:11:11]	[00:11:31]		***20s***
DM	[00:11:31]	[00:12:07]		***36s***
DAM, DA	[00:12:07]	[00:12:57]		***50s***
DAM: [5s]		DA: [45s]		
CD	[00:12:57]	[00:13:13]		***16s***
DM	[00:13:13]	[00:14:25]	***1m***	***12s***
DAM	[00:14:25]	[00:16:00]	***1m***	***35s***
RS	[00:16:00]	[00:16:13]		***13s***
DA	[00:16:13]	[00:18:13]	***2m***	
DAM	[00:18:13]	[00:21:00]	***2m***	***47s***
DA	[00:21:00]	[00:21:37]		***37s***
DM	[00:21:37]	[00:22:20]		***43s***
DAM, CD	[00:22:20]	[00:24:20]	***2m***	
DAM: [10s]		CD: [1m50s]		

	Minutes	**Seconds**
RS	**6m**	**41s**
DC	**3m**	**58s**
CD	**2m**	**40s**
DA	**5m**	**58s**
DM	**3m**	**36s**
DAM	**6m**	**27s**
Total Time	**29m**	**20s**

Table 5-2 Reasoning activity timing distributes of participant #3

Table 5-2 shows the reasoning timing distributes of participant #3 based on the interview videos which were obtained from her interview. For each participant, they were allowed to have five minutes to read the user requirements at the beginning, as a result, the time of the first RS is both set five minutes as default for each participant.

5.1.2 Experienced software designer

There are 6 participants as experienced software designers for data analysis. The situation of interview transcripts is similar to novice ones and here is no more interpretation of it. Now, it is just showing coded interview transcript, reasoning activity sequence diagram and reasoning activity timing distributes of participant #9, who is one selected participant as experienced software designers.

5.1.2.1 Coded interview transcript

RS: [Read the user requirements]

NO: Mm… this is a tricky task because there are lots of freedom things here such that I think I have to make quite many design choices. The first question I would like to discuss with the stakeholders is how the customers actually access to the software would be built though web-service or something like that. We could skip that part and just to say it doesn't matter because it's belonging to the interface. We just focus on the core software, I guess. The other thing is that do we have any ideas about the reservation of the tour they like? What information are we going to request to the user? Normally, in the use case, you write down the general sequence things and very verb description of different steps. We have a lot of choices to make of each use case. Normally, I know what people usually do is to read the descriptions and pick out all the objects you find. Without thinking about the database, we want to identify the entities which the software needs to be related to. So, we can make a reservation, cancel a tour, view tour information, add complaints and the employees can administrate it. Mm… the reason I am quiet is that I am thinking do I want to model actual software piece and the data structure which the information that software handle? For me, what I normally do is to make problem-domain model.

DC: So, I have customer first to take care of. Then, I have the tour.

DAM: One customer can have many tours and one tour can be associated to many customers as well.

RS: [Read the user requirements]

DC: OK. Perhaps it is the reservation rather than tour. This is the tour. (Change the tour as the reservation and create the new tour)

DAM: The reservation is belonging to one tour and one tour has many reservations. That could be one way to look at it. For the travel agency, it has several and many tours. One tour can be related many reservations and many reservations can be done by one customer. I could connect the customer and tour directly and the reservation can be treated as an association class. It depends on how you want to do it and it would be the same thing.

RS: [Read the user requirements]

DC: Of course, I need to have complaint object taken care of.

DAM: One customer can make several complaints and in the description, it doesn't say anything about the complaints related to the reservations or tours. But, we properly would like to the complaint related to the reservation, I guess. The association between them should be an option thing so I just put the multiplicity 0…1.

RS: [Read the user requirements]

NO: What I do now actually is not the software design and I am trying to identify the things the software manages.

DA: The customer has public attributes id and password. I am not adding the employee here right now because the employee is not the actual system, who will use the system through the software parts and the employee here is just self.

DC: Mm… OK, I understand it. Perhaps we should do this way. I just create the user as abstract class which the customer as well as the employee is both the user here and this is clear.

CD: [Check the created class diagram] I need to draw this again.

DAM: Mm…I think the customer should be directly associated to the tour in order to view tour information.

DM: The tour should also have the method of creating reservation which returns reservation, of course. The reservation should be cancelled.

RS: [Read the user requirements] I really need to read the text over and over again.

DC: Mm… I add the email object and the customer creates it. The email object can create complaints.

NO: All right and I could re-draw it to make it nicer.

DC: In the description, it talks about the complaint database such that I could add this object –complaint database. The complaint database consists of all the complaints made by customers and also I need to have tour database containing all the tours information.

DAM: And the association related to the databases is the composition instead of aggregation. The aggregation means kind of logic ownership but for the composition, it means like physical ownership.

NO: OK. We have nine classes now. I will change some relationship as my understanding of the problems going on.

DAM: Mm… the customer could access the tour database directly to view tour information and the association between the customer and the tour is kind of wrong such that I remove it. I think the tour is related to the reservation and the reservation is related to the customer. The association should be like this way, not like the tour self connected to the customer, I am quite sure about it.

DA: The reservation class should have the negotiation-price and advanced-payment as the attributes.

NO: Mm… the implementation of this system should be web-based interface in order that the customer can access to the information about tours via it. Mm… I mean this is the initial class diagram for me to understand the actual problems. This is my sketch. The next step is to add software components to identify how we do now. The description doesn't say anything about how to do. I cannot do the software design now because I am lack of constraints and non-functional requirements. What I can get now is just functional requirements and the actual implementation of the system should be based on the non-functional requirements. From the user requirements and the use cases, I just can do this.

CD: [Check the created class diagram]

All right, I think this is my initial one. (Re-draw the initial class diagram)

-Reasoning activity sequence diagram

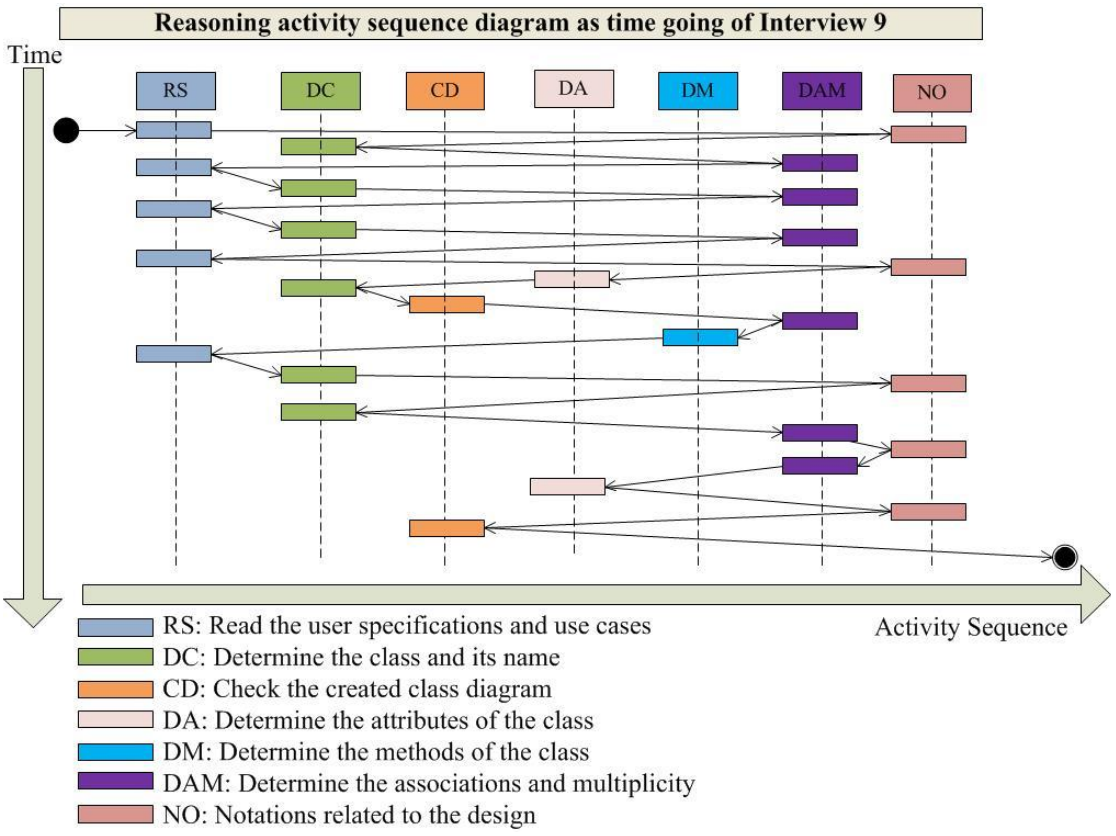

Reasoning activity sequence diagram as time going of Interview 9
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Figure 5-3 Reasoning activity sequence diagram of participant #9

-Reasoning activity timing distributes

Reasoning Activity	***Start Time***	***Finish Time***	***Duration***	
			Minutes	**Seconds**
RS	***Default time***		***5m***	
NO	[00:00:00]	[00:04:20]	***4m***	***20s***
DC	[00:04:20]	[00:04:55]		***35s***
DAM	[00:04:55]	[00:05:25]		***30s***
RS	[00:05:25]	[00:06:50]	***1m***	***25s***
DC	[00:06:50]	[00:07:07]		***17s***
DAM	[00:07:07]	[00:09:00]	***1m***	***53s***
RS	[00:09:00]	[00:09:19]		***19s***
DC	[00:09:19]	[00:09:27]		***08s***
DAM	[00:09:27]	[00:10:42]	***1m***	***15s***
RS	[00:10:42]	[00:11:54]	***1m***	***12s***
NO	[00:11:54]	[00:12:14]		***20s***
DA	[00:12:14]	[00:12:54]	***1m***	
DC	[00:12:54]	[00:13:36]		***42s***
CD	[00:13:36]	[00:14:40]	***1m***	***04s***
DAM	[00:14:40]	[00:15:23]		***43s***
DM	[00:15:23]	[00:16:12]		***49s***
RS	[00:16:12]	[00:16:49]		***37s***
DC	[00:16:49]	[00:17:39]		***50s***
NO	[00:17:39]	[00:17:47]		***08s***
DC	[00:17:47]	[00:18:21]		***42s***
DAM	[00:18:21]	[00:19:06]		***45s***
NO	[00:19:06]	[00:19:38]		***32s***
DAM	[00:19:38]	[00:21:03]	***1m***	***57s***
DA	[00:21:03]	[00:21:48]		***45s***
NO	[00:21:48]	[00:25:04]	***3m***	***16s***
CD	[00:25:04]	[00:28:04]	***3m***	

	Minutes	**Seconds**
RS	**8m**	**33s**
DC	**3m**	**06s**
CD	**4m**	**04s**
DA	**1m**	**25s**
DM		**49s**
DAM	**6m**	**31s**
NO	**8m**	**36s**
Total Time	**33m**	**04s**

Table 5-4 Reasoning activity timing distributes of participant #9

5.2 Differences between novice and experienced software designers

No matter the novice and experienced participants, all reasoning thoughts and behaviors can be described by the seven reasoning activities as mentioned above. However, there are still some differences between them and the comparisons will be discussed in this part from two aspects, which are based on reasoning activity sequence diagrams and reasoning activity timing distributes.

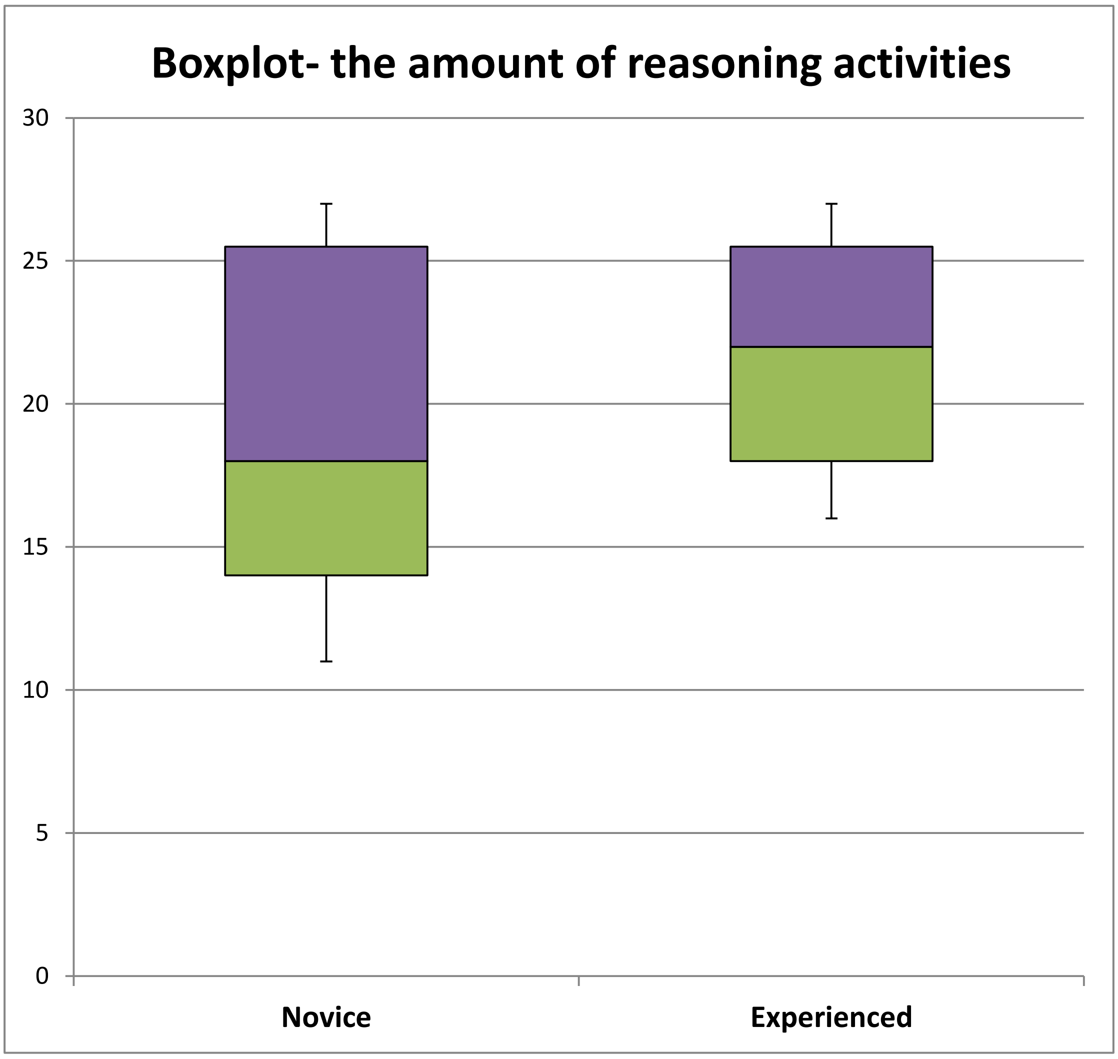
5.2.1 Comparison on the amount of reasoning activity

Participant	No of RS	No of DC	No of DA	No of DM	No of DAM	No of CD	No of NO	Total
#1	4	4	3	2	2	3	0	18
#2	6	5	3	3	1	1	0	19
#3	5	3	4	5	5	5	0	27
#4	2	2	1	1	2	2	2	12
#5	3	3	2	4	1	3	0	16
#6	1	1	3	3	2	1	0	11
#7	4	5	5	4	3	2	3	26
#8	4	5	6	4	3	2	1	25
#11	2	5	4	3	2	1	0	17
Total	***31***	***33***	***31***	***29***	***21***	***20***	***6***	***171***
Mean	***3.44***	***3.67***	***3.44***	***3.22***	***2.33***	***2.22***	***0.67***	***19.00***
Variance	***2.53***	***2.25***	***2.28***	***1.44***	***1.50***	***1.69***	***1.25***	***34.50***
Standard deviation	***1.59***	***1.50***	***1.51***	***1.20***	***1.22***	***1.30***	***1.12***	***5.87***

Table 5-5 Statistical data on reasoning activity sequences diagrams on novice software designers

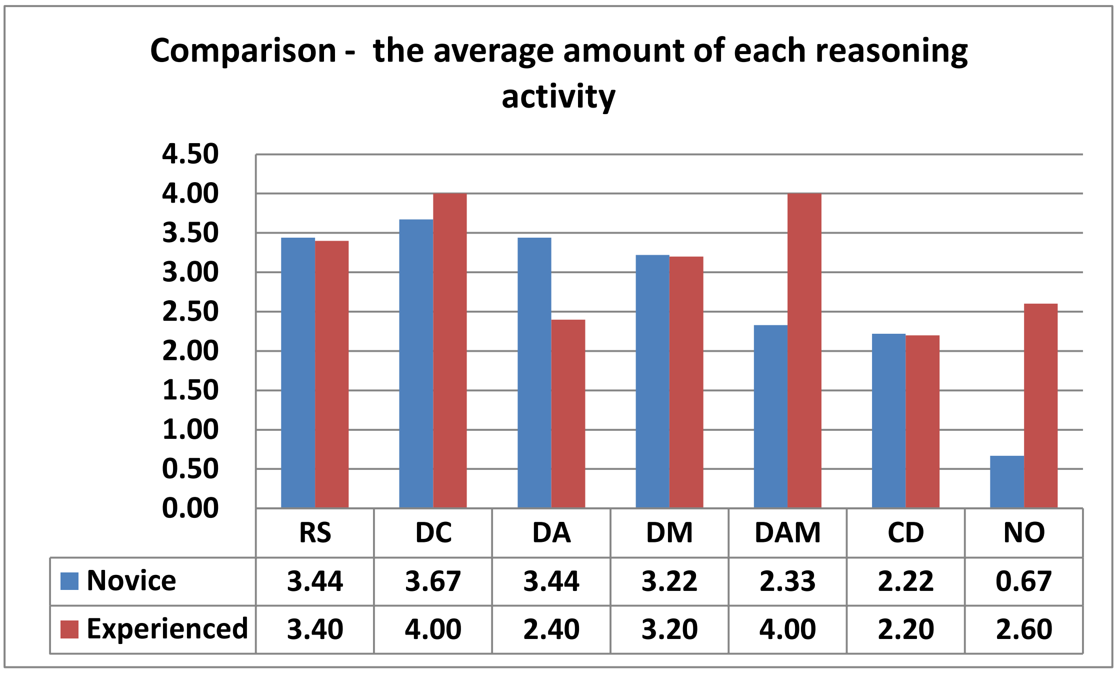
Participant	No of RS	No of DC	No of DA	No of DM	No of DAM	No of CD	No of NO	Total
#9	5	6	2	1	6	2	5	27
#10	3	2	2	4	5	2	2	20
#12	3	4	2	2	5	3	3	22
#13	3	3	3	3	2	2	0	16
#14	3	5	3	6	2	2	3	24
Total	***17***	***20***	***12***	***16***	***20***	***11***	***13***	***109***
Mean	***3.40***	***4.00***	***2.40***	***3.20***	***4.00***	***2.20***	***2.60***	***21.80***
Variance	***0.80***	***2.50***	***0.30***	***3.70***	***3.50***	***0.20***	***3.30***	***17.20***
Standard deviation	***0.89***	***1.58***	***0.55***	***1.92***	***1.87***	***0.45***	***1.82***	***4.15***

Table 5-6 Statistical data on reasoning activity sequences diagrams on experienced software designers



	Novice	Experienced
Minimum	**11**	**16**
Q1	**14**	**18**
Median	**18**	**22**
Q3	**25.5**	**25.5**
Maximum	**27**	**27**
Interquartile range	**11.5**	**7.5**

Figure 5-7 Boxplot – the total amount of reasoning activities for novice and experienced participants



	RS	DC	DA	DM	DAM	CD	NO
🟦 Novice	3.44	3.67	3.44	3.22	2.33	2.22	0.67
🟥 Experienced	3.40	4.00	2.40	3.20	4.00	2.20	2.60

Figure 5-8 Comparison – the average amount of each reasoning activity for novice and experienced participants

The tables truly present the amount of reasoning activities for novice and experienced participants and its relevant statistical data based on their own reasoning activity diagrams. (see Table 5-5 and 5-6) For each type, the original data is showed by individual interview number of the participants, which columns clearly showing the amount of seven reasoning activities separately and totally according to the corresponding reasoning activity sequence diagram. The data with black bold is the statistical data results, representing the sum, mean value, variance and standard deviation of each reasoning activity separately and totally, too. The figures reflects the boxplot which is standing for the amount of total reasoning activities on novice and experiences ones (see Figure 5-7) and the comparison of mean values on seven reasoning activities for them. (see Figure 5-8)

For reasoning activity of RS, DC, DM and CD, the mean values of novice and experienced software designers are the almost same (3.44 vs 3.40, 3.67 vs 4.00, 3.22 vs 3.20 and 2.22 vs 2.20), stating that these reasoning thoughts and behaviors are kind of similar while the novice and experienced participants creating UML class diagrams. But, the variances of RS, DC, DM and CD are different. In terms of RS and CD (0.80 vs 2.53 and 0.20 vs 1.69), variance values of the samples as experienced software designers are smaller such that the happening rate of them performing RS and CD reasoning activities is stable than novice participants as well as that its data distribution is also concentrated. On the contrary, the situations of DC and DM reasoning activity are different, which is meaning that the happening rate of novice ones is stable and its data distribution is concentrated (2.25 vs 2.50 and 1.44 vs 3.70).

For reasoning activity of DA, DAM and NO, the differences of mean values are more than 1, showing that novice participants were performing DA reasoning activity more (3.44 vs 2.40) but experienced ones were doing DAM and NO reasoning activity more (4.00 vs 2.44 and 2.60 vs 0.67). In other words, the novice participants might determine more attributes of the classes for their designs. Meanwhile, the experience ones paid more attention to the relationship of different created classes as well as more willing to express their opinions or ideas related to the design on their created class diagrams.

For the total amount of the reasoning activities they did on creating UML class diagrams, the experienced participants were likely performing more reasoning activities than novice ones in general (21.80 vs 19.00) and the happening rate of them performing is also stable as well as more concentrated for its data distribution (17.20 vs 34.50). Further, in terms of the boxplot, it is also proved that the data distribution of experience participants is concentrated and the interval between maximum and minimum is smaller than novice ones.

5.2.2Comparison on the time spent of reasoning activity

Participant	Time of RS	Time of DC	Time of DA	Time of DM	Time of DAM	Time of CD	Time of NO	Total
#1	0:05:51	0:01:48	0:02:21	0:02:32	0:01:23	0:02:10	0:00:00	*0:16:05*
#2	0:08:20	0:04:15	0:04:08	0:04:59	0:04:50	0:01:48	0:00:00	*0:28:20*
#3	0:06:41	0:03:58	0:05:58	0:03:36	0:06:27	0:02:40	0:00:00	*0:29:20*
#4	0:05:40	0:06:13	0:01:24	0:03:17	0:03:17	0:10:42	0:02:18	*0:32:51*
#5	0:08:24	0:00:57	0:01:53	0:03:52	0:00:20	0:04:43	0:00:00	*0:20:09*
#6	0:05:00	0:00:55	0:02:48	0:02:51	0:03:06	0:06:00	0:00:00	*0:20:40*
#7	*No available data*							
#8	0:06:12	0:02:41	0:03:08	0:03:16	0:03:21	0:04:02	0:00:43	*0:23:23*
#11	0:05:20	0:03:25	0:01:34	0:01:36	0:05:31	0:01:19	0:00:00	*0:18:45*
Total	***0:51:28***	***0:24:12***	***0:23:14***	***0:25:59***	***0:28:15***	***0:33:24***	***0:03:01***	***3:09:33***
Mean	***0:06:26***	***0:03:02***	***0:02:54***	***0:03:15***	***0:03:32***	***0:04:10***	***0:00:23***	***0:23:42***

Table 5-9 Statistical data on reasoning activity timing distributes of novice software designers

Participant	Time of RS	Time of DC	Time of DA	Time of DM	Time of DAM	Time of CD	Time of NO	Total
#9	0:08:33	0:03:06	0:01:25	0:00:49	0:06:31	0:04:04	0:08:36	*0:33:04*
#10	0:07:00	0:03:09	0:02:05	0:02:53	0:13:16	0:04:05	0:04:52	*0:37:20*
#12	0:07:43	0:01:28	0:02:27	0:00:39	0:06:18	0:07:42	0:07:03	*0:33:20*
#13	0:06:08	0:02:08	0:01:13	0:01:39	0:03:37	0:03:47	0:00:00	*0:18:32*
#14	0:07:30	0:03:48	0:03:26	0:05:50	0:05:09	0:07:27	0:05:54	*0:39:04*
Total	***0:36:54***	***0:13:39***	***0:10:36***	***0:11:50***	***0:34:51***	***0:27:05***	***0:26:25***	***2:41:20***
Mean	***0:07:23***	***0:02:44***	***0:02:07***	***0:02:22***	***0:06:58***	***0:05:25***	***0:05:17***	***0:32:16***

Table 5-10 Statistical data on reasoning activity timing distributes of experienced software designers

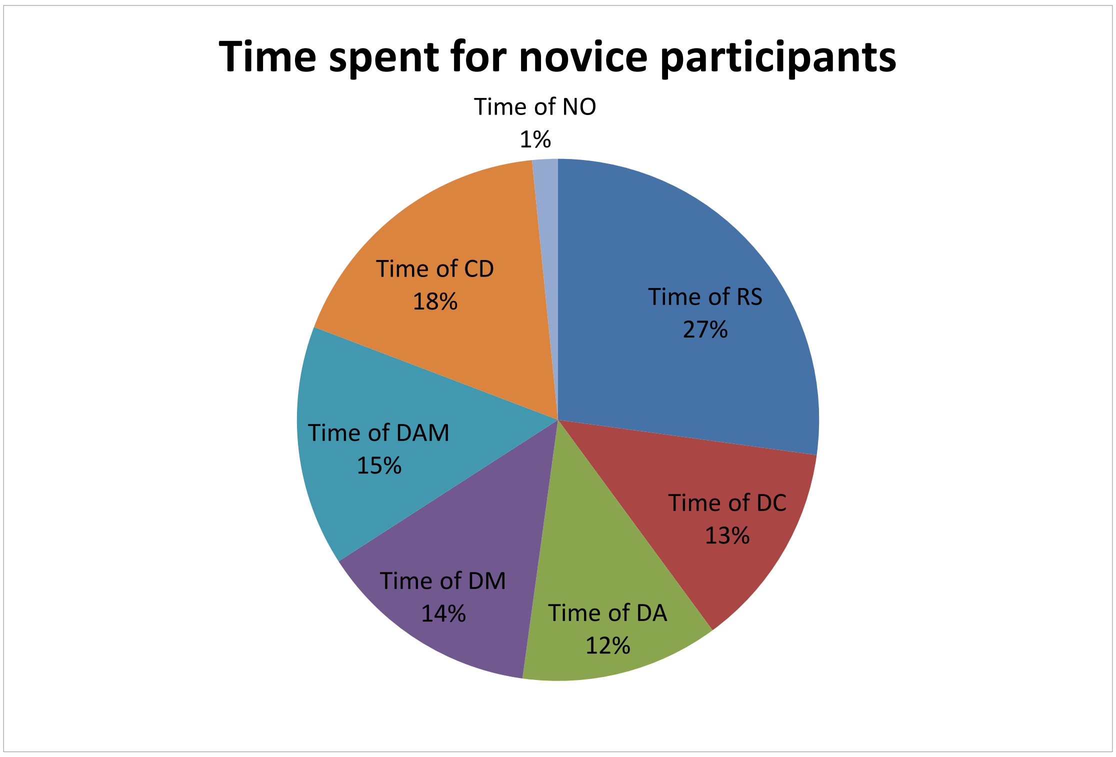


Figure 5-11 Pie chart – time percentage of novice participants on reasoning activities

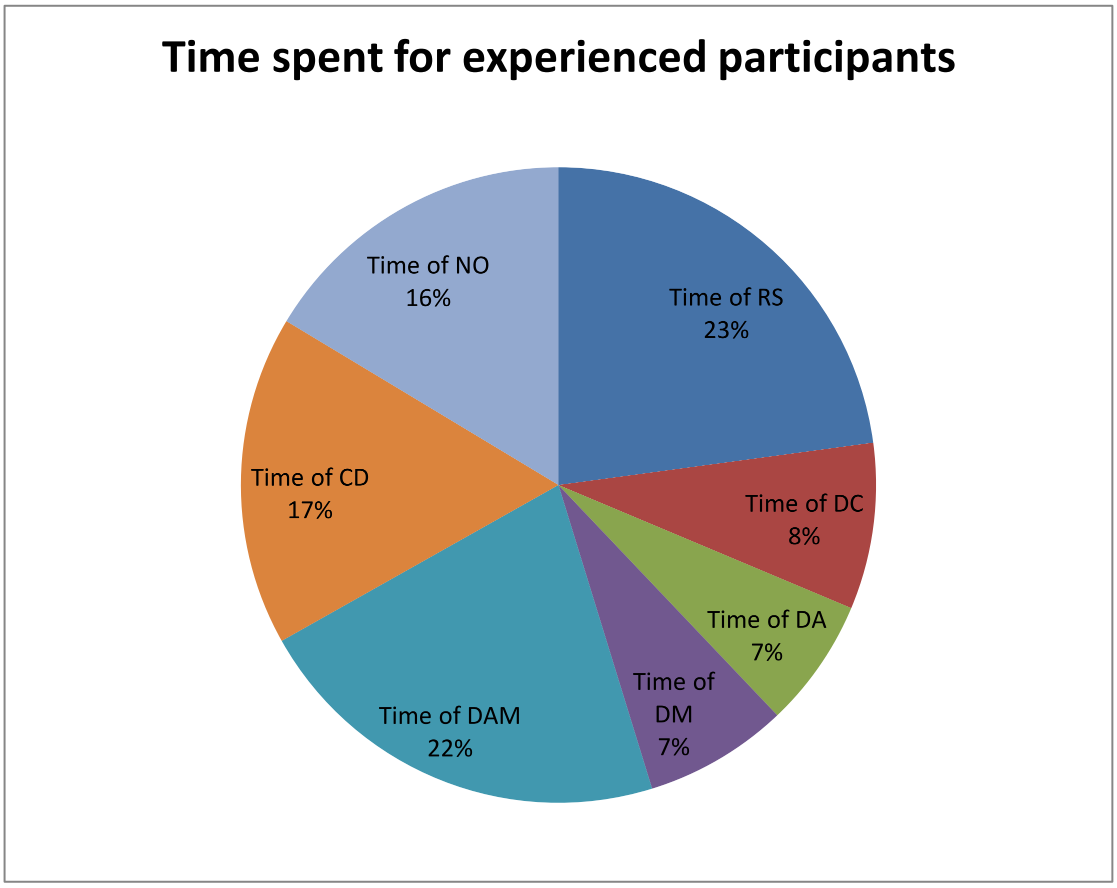


Figure 5-12 Pie chart – time percentage of experienced participants on reasoning activities

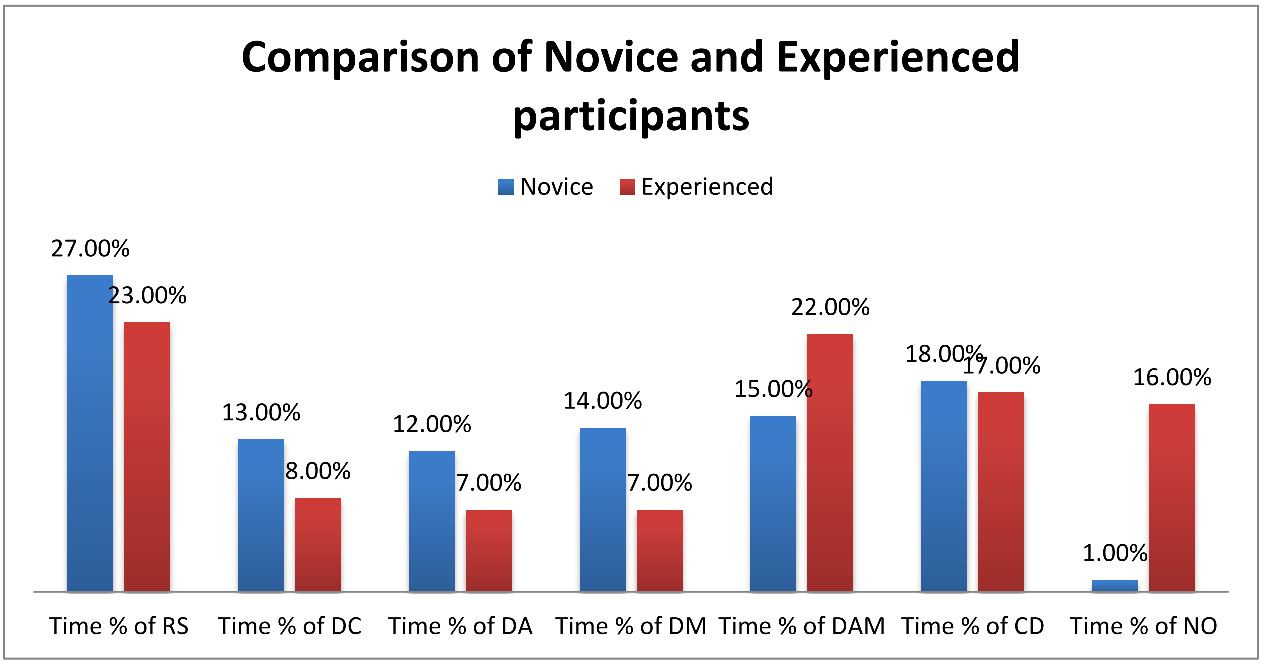
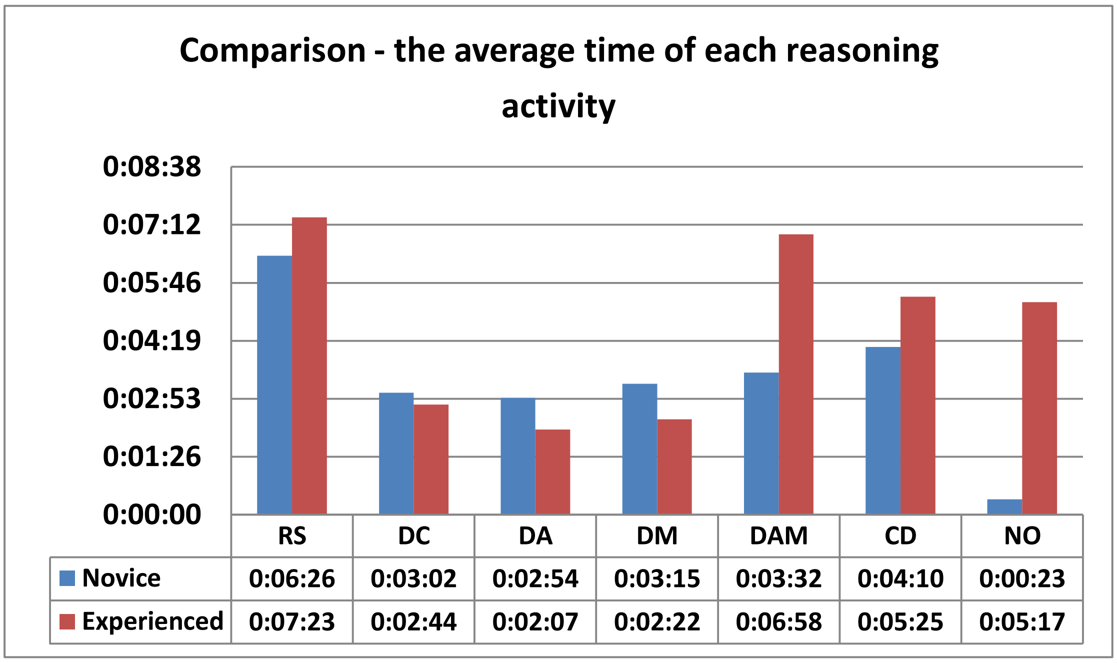


Figure 5-13 Comparison – time percentage of novice and experienced participants on reasoning activities



	RS	DC	DA	DM	DAM	CD	NO
🟦 Novice	0:06:26	0:03:02	0:02:54	0:03:15	0:03:32	0:04:10	0:00:23
🟥 Experienced	0:07:23	0:02:44	0:02:07	0:02:22	0:06:58	0:05:25	0:05:17

Figure 5-14 Comparison – the average time of each reasoning activity for novice and experienced participants

The tables present the raw data and its statistical data based on the reasoning activity time distributes of novice and experienced participants respectively. (see Table 5-9 and 5-10) These two forms clearly show the time spent on different reason activities as different participants. For participant #7, there is no collected data about his reasoning activity timing distributes because of the interview camera, which was broken at his interview time. The pie charts explicitly indicate the time percentage of each reasoning activity on total spent time. (see Figure 5-11 and 5-12) The comparisons of the time percentage and time mean values are shown in the Figure 5-13 and 5-14.

For reasoning activities of RS, DAM,CD and NO, the mean values of participants as experienced software designers are larger than novice ones (07:23 vs 06:26, 06:58 vs 03:32, 05:25 vs 04:10 and 05:17 vs 00:23), which means they probably spent more time on these reasoning activities than novice ones. In terms of DAM and NO, the differences of mean values are more than three minutes, clearly stating that experienced participants took more time on considering the relationships of different classes as well as expressing their own thoughts related to the problem domain. Moreover, the time percentage is also proved that experienced participants spent more time to determine the relationships among the classes and speak out their design ideas. (22% vs 15% and 16% vs 1%) Especially on NO activity, the diversity is too obvious that novice participants hardly talked about their thoughts related to the design on creating class diagrams. In terms of RS and CD, there is no big difference on mean values and time percentage charts.

For reasoning activities of DC, DA and DM, the mean values of participants as novice software designers are larger (03:02 vs 02:44, 02:54 vs 02:07 and 03:15 vs 02:22). But, the differences of mean values are under one minute. It is easily known that there is also no big difference on time spent of them. However, the time percentages of these three reasoning activities are not small. Comparing to experienced participants, novice ones were spending more time on determining classes as well as its attributes and methods in order that they might treat them more significantly. (13% vs 8%, 12% vs 7% and 14% vs 7%)

For average total time, the value of experienced participants is 32:16, which is comparing to 23:42 of novice ones. As a result, it is obvious that they were spending more time on creating UML class diagrams than novice ones. And the differences on DAM and NO, which experienced ones took more time to perform during the interview process, are mostly leading the approximate ten minutes' diversity.

5.2.3 Statistical significant difference

The differences on the amount and the time spent of reasoning activity for different software designers have been presented above. Now, it is discussing the statistical significant difference based on the differences.

Owing to the small number of the participants involved in this study, to test whether there is a statistical significant difference between novices and experienced ones, the non-parametric method - Mann-Whitney U test is applied.

The amount of reasoning activity

The amount of each reasoning activity kind and the total amount of reasoning activity the participants performed are tested. The Mann-Whitney U test for total amount of reasoning activity is showed as below.

The total amount of reasoning activity	**Category**
18	**N**
19	**N**
27	**N**
12	**N**
16	**N**
11	**N**
26	**N**
25	**N**
27	**E**
20	**E**
17	**N**
22	**E**
16	**E**
24	**E**
N:*Novice participants* **E:***Experienced participant*	

```
2 Sample Mann-Whitney – The total amount of Reasoning Activity

Test Information
H0: Median Difference = 0
Ha: Median Difference ≠ 0

Category                                           E        N
Count                                              5        9
Median                                            22       18

Mann-Whitney Statistic                         44.00
p-value (2-sided, adjusted for ties)          0.4227
```

As is shown above, the P-value (0.4227) is larger than confidence level (α =0.05) in order that it is adequate to accept null hypothesis H_0. As a result, it is clearly that there is ***no statistical significant difference*** between novice and experienced participants on the total amount of reasoning activity. In other word, even though experienced participants performed more reasoning activity in general than novice ones (21.80 vs 19.00), there is no statistical significant difference between them.

For other situations, the result is that there is also ***no statistical significant difference*** between novice and experienced participants on the amount of each reasoning activity RS, DC, DA, DM, DAM, CD and NO they performed.

The time spent on reasoning activity

The time spent of each reasoning activity category and the total time of reasoning activity the participant spent are examined to test whether is a statistical significant difference. The Mann-Whitney U test for DAM and NO reasoning activity is presented as below.

Category	**Time spent of reasoning activity DAM**	**Time spent of reasoning activity NO**
N	**0:01:23**	**0:00:00**
N	**0:04:50**	**0:00:00**
N	**0:06:27**	**0:00:00**
N	**0:03:17**	**0:02:18**
N	**0:00:20**	**0:00:00**
N	**0:03:06**	**0:00:00**
N	**0:03:21**	**0:00:43**
E	**0:06:31**	**0:08:36**
E	**0:13:16**	**0:04:52**
N	**0:05:31**	**0:00:00**
E	**0:06:18**	**0:07:03**
E	**0:03:37**	**0:00:00**
E	**0:05:09**	**0:05:54**

2 Sample Mann-Whitney - Time of DAM

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	E	N
Count	5	8
Median	0.004375	0.00230324 1

Mann-Whitney Statistic	49.00
p-value (2-sided, adjusted for ties)	0.0481

2 Sample Mann-Whitney - Time of NO

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	E	N
Count	5	8
Median	0.004097222	0

Mann-Whitney Statistic	50.00
p-value (2-sided, adjusted for ties)	0.0210

As is shown above, the P-values equal 0.0481 and 0.0210 respectively which are both smaller than confidence level (α =0.05) such that it is sufficient to reject null hypothesis H_0 but accept alternative hypothesis H_1. Therefore, it is explicit that there is ***statistical significant difference*** between novice and experienced participants on the time spent of reasoning activity DAM and NO. Obviously, experienced participants spent much more time to consider the relations between the created classes and took more time to express their own opinions about the design as well as to improve the design than novice participant.

For other cases, the result is that there is ***no statistical significant difference*** between novice and experienced participants on the time spent of reasoning activity RS, DC, DA, DM, CD and the total time they spent for the design.

5.3 Other findings

There are still some valuable and beneficial findings discovered during the interviews so as to present them as follows.

5.3.1 Reasoning structure

For creating UML class diagram, different participants have different deciding ways in order that there are types of reasoning structures on UML class diagram built. In this study, two reasoning structures are involved, which are "Depth First" and "Breadth First".

-"Depth first" structure

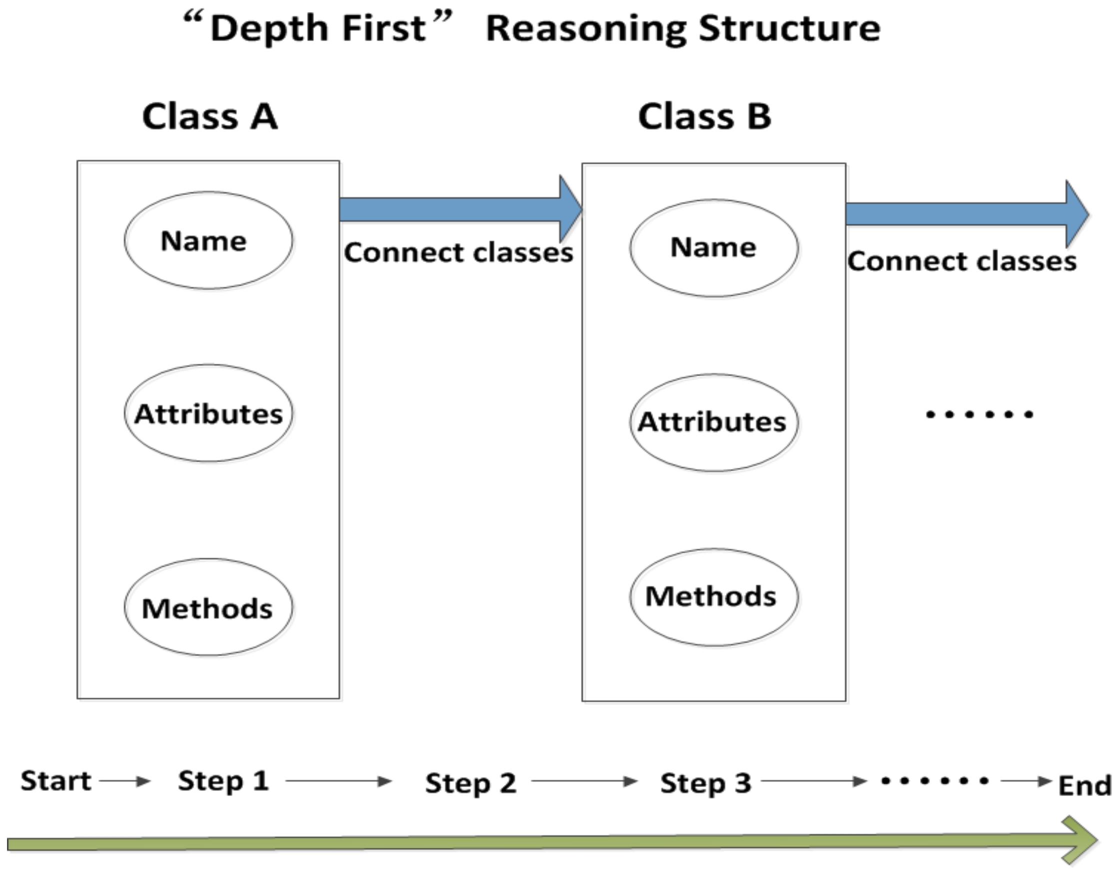


Figure 5-15"Depth First" reasoning structure on creating UML class diagram

It clearly shows what the "Depth First" reasoning structure is. (see Figure 5-15) In short, "Depth First" means that the participants determined one class completely and then, continued determining another class completely until completing all classes. In the figure, this behavior starts from deciding class A, which is containing to create class name, attributes as well as methods. Next, it is the time to connect class A to class B, deciding the relationship between them. Then, it should build class name, attributes and methods of class B. The whole process of "Depth First" will be an end until to accomplish all classes.

-"Breadth first" structure

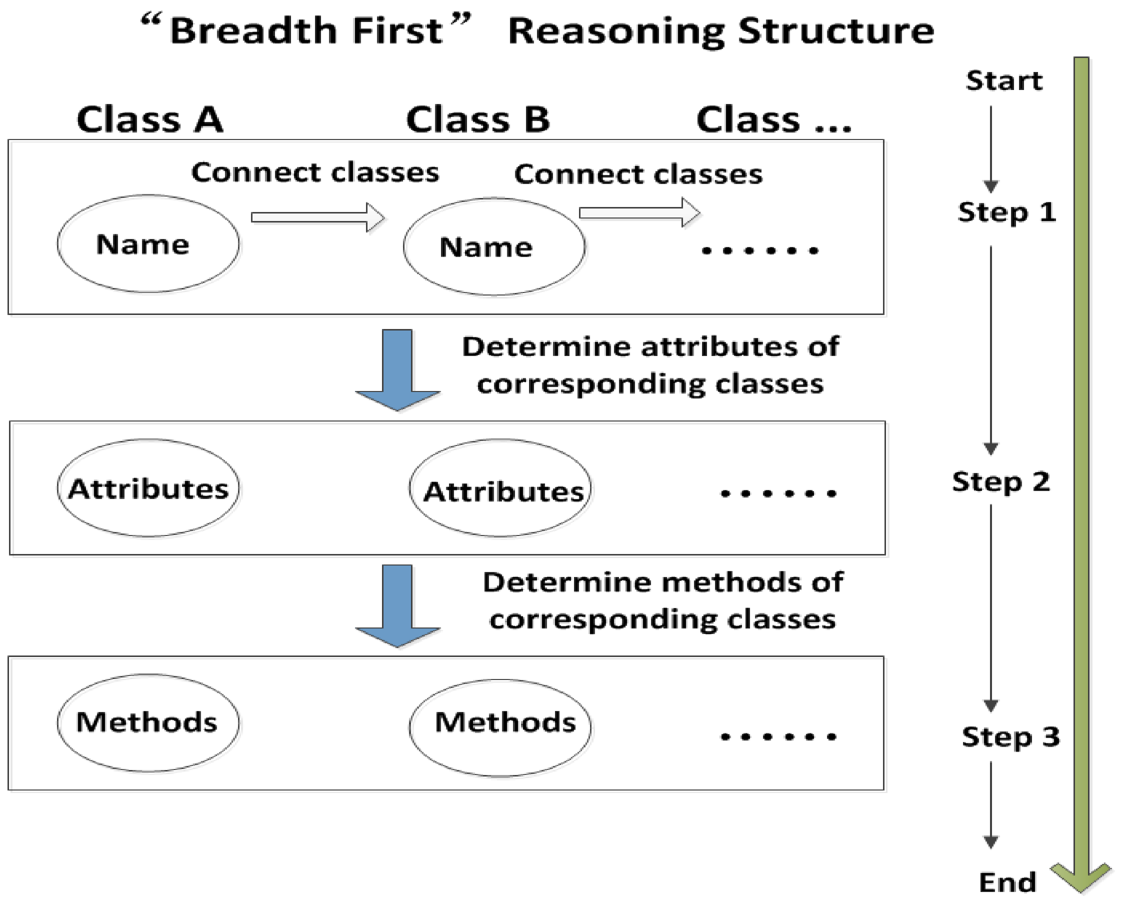
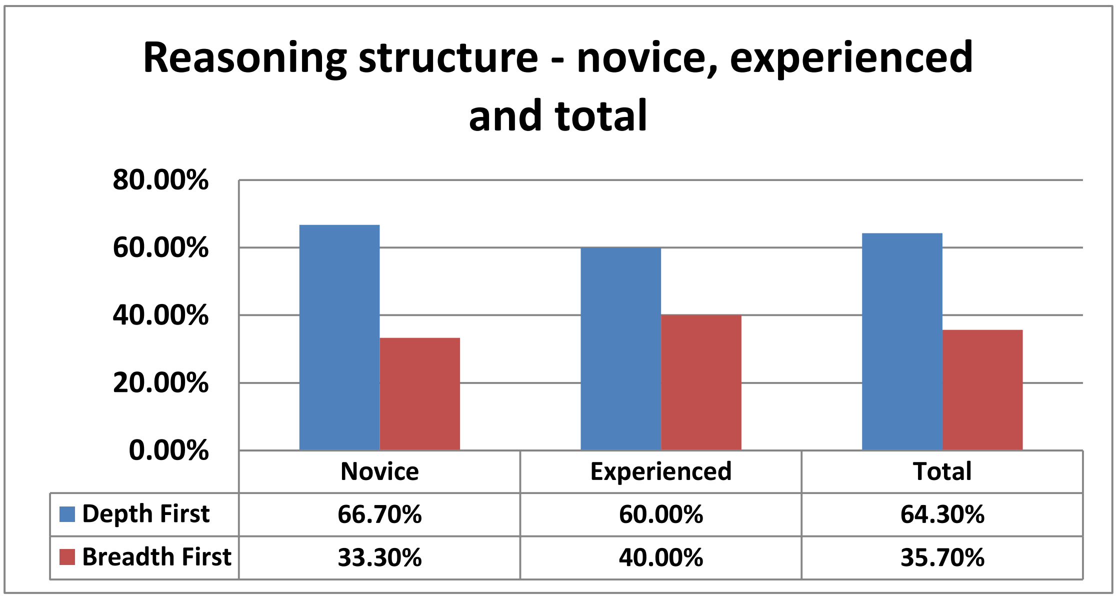


Figure 5-16 "Breadth First" reasoning structure on creating UML class diagram

It explicitly presents what the "Breadth First" reasoning structure is. (see Figure 5-16) In summary, "Breadth First" stands for that the participants determined all the classes first (mainly determine the class names) and then went on with finishing the attributes and methods of the corresponding classes. In the figure, it begins at deciding all the needed classes' names and making connections among them. Then, it is the time to create the attributes and methods of the classes. The entire process of "Breadth First" will be finished at the time of completing all the contents of the created classes in the first step.

Reasoning structure	**Participant**	
	Novice	**Experienced**
"Depth First	***#1, #2, #5, #7, #8, #11***	***#12, #13, #14***
"Breadth First	***#3, #4, #6***	***#9, #10***

Table 5-17 Statistical data on reasoning structures of participants



	Novice	Experienced	Total
🟦 Depth First	66.70%	60.00%	64.30%
🟥 Breadth First	33.30%	40.00%	35.70%

Figure 5-18 Percentage of reasoning structure usage on novice, experienced participants and their total

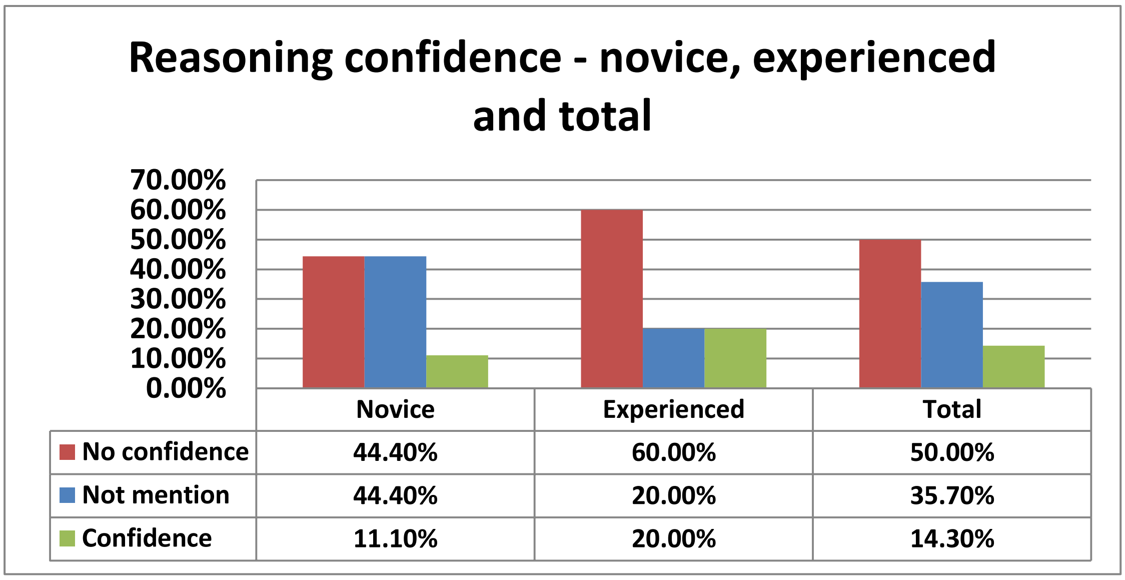
In the study, participants #1, #2, #5, #7, #8, #11, #12, #13 and #14 were using "Depth First" structure on creating UML class diagrams of which were 6 novice participants and 3 experienced ones. Participant #3, #4, #6, #9 and #10 were adopting "Breadth First" structure of which were 3 novice participants and 2 experienced ones. (see Table 5-17) As is shown (see Figure 5-18), majority of the participants (64.3%) were applying "Depth First" reasoning structure. For novice participants, 66.7% of them used "Depth Fist" structure, comparing to 33.3% using "Breadth First" structure. For experienced participants, 60% of them were for "Depth First" and 40% were for another structure. As a result, choosing which reasoning structure on creating class diagrams is based on the decision of the participants but for an inference, maybe experienced participants were more willing to use "Breadth First" structure but novices may prefer to adopt "Depth First" structure.

5.3.2 Reasoning confidence

As mentioned earlier, the participants both have previous experience on UML modeling so that creating UML class diagram for them maybe is not unfamiliar and not difficult. However, the situations are different because of reasoning confidence. Some are good at UML modeling, but some are not. As a result, the participants who have confidences are quite sure about what they should do and what they have done on their created class diagrams. On the contrary, the others are not confident and unsure about their designs. In this study, majority of the subjects were not confident enough for their designs while created UML class diagrams, and only few participants were not.

Reasoning confidence	**Participant**	
	Novice	**Experienced**
No confidence	*#2, #3, #7, #8*	*#12, #13, #14*
Not mention	*#4, #5, #6, #11*	*#10*
Confidence	*#1*	*#9*

Table 5-19 Statistical data on reasoning confidences of participants



	Novice	Experienced	Total
🟥 No confidence	44.40%	60.00%	50.00%
🟦 Not mention	44.40%	20.00%	35.70%
🟩 Confidence	11.10%	20.00%	14.30%

Figure 5-20 Percentage of showing confidence by novice, experienced participants and their total

Now, present some examples related to confidence and no confidence on creating UML class diagrams of the participants.

-Confidence

"The user can view information, complain, make and cancel a reservation. Mm…, <u>**good**</u>..." *-Participant #1*

"...I think the tour is related to the reservation and the reservation is related to the customer. The association should be like this way, not like the tour self connected to the customer, ***<u>I am quite sure about it</u>****." -Participant #9*

-No confidence

"...Reservations, travel agency bureau, tours and the users can access to the same system, ***<u>so I guess the system should be in the diagram</u>****...How can I name it?"*
-Participant #2

"Mm... ***<u>I guess there should be another entity called tour</u>****." -Participant #3*

"...The tour can have several reservations, ***<u>I guess</u>****." -Participant #12*

*"**I guess** we need to know how many reservations we have now…"* *-Participant #13*

*"And the tour should have the action of viewing available tour which only returning the available tours, but I am **not quite sure**."* *–Participant #3*

*"**I am not sure whether I have done or not.** I am doing android currently and that's why I doing like this way…"* *-Participant #7*

*"I think the tour also has the count-customer which means counting the number of the customers registered on the tour and properly price. Yes, I am **not sure**…"*
*"Mm… I am **not sure** that I am gone or in the right direction. "*

-Participant #8

*"Maybe the complain is not correct in the system because the complaints are sent by emails and stored outside of the system, so I am **not sure** whether I should include it or not."*

-Participant #14

As is shown above (see Table 5-19 and Figure 5-20), 50% (7 participants) of the participants were not confident enough for their created UML class diagrams. 35.7% (5 participants) of them did not clearly show their confidence or not and only the left 14.3% (2 participants) were confident on their designs. For novice participants, 88.9% (8 participants) of them felt not confident or not mentioned their confidence and only 11.1% (1 participant) showed confidence. For experienced participants, also only 20% (1 participant) had confident on the created class diagrams and 80% (4 participants) were not confident or not mentioned their confidence.

The key words, such as "not sure" and "guess", are obviously marked with black bold to show no confidence on the created class diagrams. On the contrary, only two participants were confident enough, using key words "sure" and "good" to express their confidence. Moreover, there is no direct relationship between participants' type and their reasoning confidence on creating UML class diagrams. For instance, participant #1 was selected as novice software designer, but she was confident for her design. Similarly, participants #12, #13 and #14 were treated as experienced software designers, and they were unexpectedly not confident on their designs. However, it is still a good way of gain reasoning confidences on creating UML class diagrams by acquiring experience on UML modeling. In summary, reasoning confidence should be determined by many factors, not only by experience.

5.3.3 Reasoning source

The reasoning thoughts and behaviors can be classified into seven reasoning activities as interpreted in the former sections. But, one question becomes kind of interesting – "how do they determine the created UML class diagrams", which means what is the reasoning source of the participants. Reasoning source is the foundations on their designs, as well as the source of getting inspired. By asking the subjects questions after they created UML class diagrams in the interviews, there is one form presenting relevant collected answers on reasoning sources.

Behavior	**Reasoning source**	**Participant**		**Frequency**
		Novice	**Experienced**	
Determine class	*Nouns in the user requirements*	***#3, #4, #5, #6, #7, #8, #11***	***#10***	**8**
	Domain knowledge	***#4***	*Null*	**1**
	Own experience on UML modeling	*Null*	***#9, #10, #13,#14***	**4**
	Use case model	***#11***	***#12***	**2**
Determine methods of the class	*Verbs in the user requirements*	***#4, #5***	***#9, #10, #12, #13***	**6**
	Use case model	***#3, #4, #7, #11***	***#12***	**5**
	Daily life/Common sense	***#6***	*Null*	**1**
	Own experience on UML modeling	***#8***	***#14***	**2**
Determine the attributes of the class	*User requirements descriptions*	***#3, #4, #5, #7, #11***	***#9, #10, #12, #13***	**9**
	Daily life/Common sense	***#3, #6***	*Null*	**2**
	Domain knowledge	***#7, #8***	***#10***	**3**
	Own experience on UML modeling	***#7, #8***	***#14***	**3**
	Use case model	*Null*	***#12***	**1**
Determine associations between classes	*User requirements descriptions*	***#3, #6***	***#10***	**3**
	Use case model	***#3, #5***	***#10***	**3**
	Daily life/Common sense	***#4, #5***	*Null*	**2**
	Refer to the created classes	*Null*	***#9***	**1**
	Own experience on UML modeling	***#3, #4, #7, #8, #11***	***#12, #13, #14***	**8**
	Domain knowledge	***#11***	***#13***	**2**

Table 5-21 Statistical data on reasoning sources of participants

It explicitly shows the reasoning sources of the participants on four aspects of which are class, attributes of the class and methods of the class as well as the associations between the classes. The table (see Table 5-21) lists all the reasoning sources mentioned by the participants. For determining class, "nouns in the user requirements" is the most frequently referenced reasoning source for the participants. For determining the methods and attributes of the class, "verbs in the user requirements" and "user requirements descriptions" are the most used reasoning sources. For determining associations between classes, "own experience on UML modeling" is the first reasoning source for reference.

As is shown to all, one behavior can be affected and inspired by multiple reasoning sources, such as determining classes of participant #4, his reasoning sources of it come from nouns in the user requirements, as well as the domain knowledge. Also, except for user requirement descriptions and use case model, the other reasoning sources are not written on the paper, but in the brain. These reasoning sources are both based on one word – "experience". The "experience" can come from daily life or common sense and also can be built from own experience on UML modeling as well as the domain knowledge. In brief, the reasoning sources on creating UML class diagrams can be from either one is written on the paper like user requirements, or one is in the brain like domain knowledge.

5.3.4 Reasoning principles

Reasoning principles are rules, guiding software designers to make a better solution on problem solving as well as to abide by. In the study, some participants discussed their reasoning principles on creating UML class diagrams at question time but the others did not consider any reasoning principles. Although the reasoning principles were mentioned by participants, many of them were still too general and not detailed such that it was hard to clarify how they worked while the participants created UML class diagrams.

Participant	**Reasoning principles on creating UML class diagrams**
#3	·Abstract class
#4	·Reusability ·Cohesiveness ·Object-oriented rules
#6	·Implementation (programming) ·Readability
#7	·MVC – model view controller
#8	·High cohesion ·Low coupling
#10	·High cohesion ·Low coupling ·Façade for actors ·Controller for handling large data
#11	·Low coupling ·Avoid inheritance
#14	·Handler

Table 5-22 Reasoning principles on creating UML class diagrams of participants

Reasoning principles on creating UML class diagrams	**Participant**		**Frequency**
	Novice	**Experienced**	
Low coupling	***#8, #11***	***#10***	**3**
High cohesion / Cohesiveness	***#4, #8***	***#10***	**3**
Abstract class/ Reusability	***#3, #4***	*Null*	**2**
Controller for handling large data / Handler	*Null*	***#10, #14***	**2**
Object-oriented rules	***#4***	*Null*	**1**
Implementation (programming)	***#6***	*Null*	**1**
Readability	***#6***	*Null*	**1**
MVC – model view controller	***#7***	*Null*	**1**
Façade for actors	*Null*	***#10***	**1**
Avoid inheritance	***#11***	*Null*	**1**

Table 5-23 Statistical data on reasoning principles of participants

The tables list all of the reasoning principles mentioned by participants and the mentioned reasoning principles are ordered. (see Table 5-22 and 5-23) The reasoning principles are ordinary in software design. The mentioned principles are diversity such that the considerations of different participants on class diagrams are quite different. High cohesion and low coupling were frequently mentioned by the participants. However, some reasoning principles were too general to clarify how they worked on creating UML class diagrams, such as readability. Also, some of reasoning principles were just brain thoughts rather than practical behaviors, meaning that they were not easy to verify during the process of creating UML class diagrams, such as controller for handling large data. Certainly, owing to the limitations of this study, reasoning principles, such as implementation (programming) were really difficult to reflect.

5.4 Quality of the design

After the completion of the data collection, there are four professional experts involved to evaluate the quality of the UML class diagrams created by the participants. These experts come from SE area with quite a bit experience on UML modeling.

Here are the five rules they comply with to make evaluations on the designs.

Rule A: All relevant classes/components have been identified/drawn
Rule B: All relevant relations/dependencies have been identified/drawn
Rule C: Classes and relations have suitable names
Rule D: Syntax is reasonably to UML/understandable
Rule E: Quality of the layout of the design

The scale of scores is from 1 to 10 and it stands for that 1 (very poor), 6(passable) and 10 (perfect).

Participant	Avg. A	Avg. B	Avg. C	Avg. D	Avg. E	Avg. T	Category
#1	4.25	4.25	3.75	6.75	5.75	4.95	N
#2	6.25	6.75	7.25	8.50	7.75	7.30	N
#3	7.25	7.25	7.25	8.00	7.25	7.40	N
#4	5.25	4.00	5.75	4.75	2.75	4.50	N
#5	4.00	3.25	4.75	5.50	3.75	4.25	N
#6	2.00	2.00	3.00	6.50	5.75	3.85	N
#7	5.50	4.75	5.75	5.00	3.75	4.95	N
#8	6.25	6.75	6.50	6.75	6.25	6.50	N
#9	7.00	5.25	6.25	6.00	6.00	6.10	E
#10	6.50	6.25	6.25	6.50	4.50	6.00	E
#11	6.00	4.75	6.25	6.00	6.00	5.80	N
#12	6.75	6.50	6.75	6.50	7.25	6.75	E
#13	5.50	5.25	6.50	6.75	6.00	6.00	E
#14	5.50	5.25	5.50	6.50	6.50	5.85	E

Table 5-24 The average score of rule A,B,C,D,E and total rules for the participants

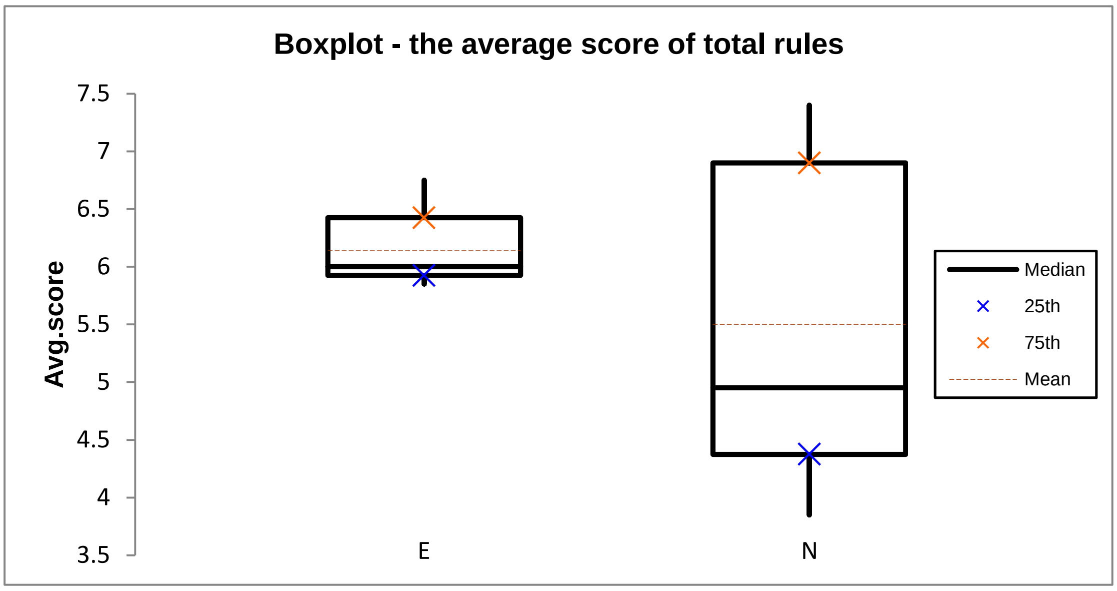
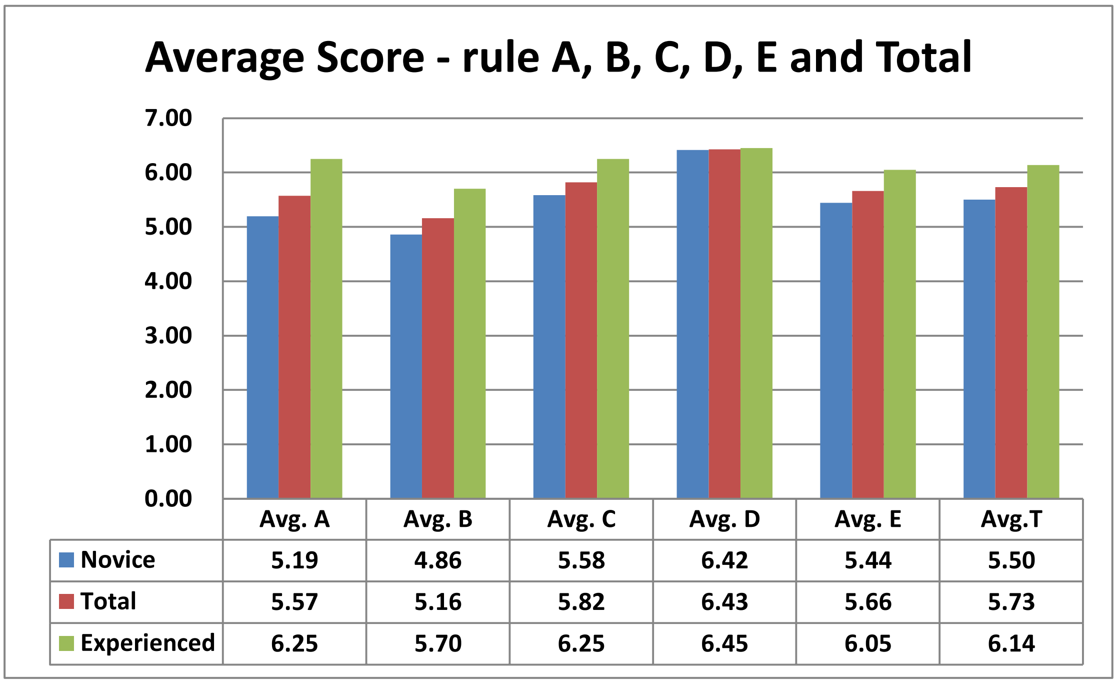


Figure 5-25 Boxplot - the average score of total rules



	Avg. A	Avg. B	Avg. C	Avg. D	Avg. E	Avg.T
🟦 Novice	5.19	4.86	5.58	6.42	5.44	5.50
🟥 Total	5.57	5.16	5.82	6.43	5.66	5.73
🟩 Experienced	6.25	5.70	6.25	6.45	6.05	6.14

Figure 5-26 Comparison – the average score of rule A,B,C,D,E and total rules

As is shown in the table and figures (see Table 5-24, Figure 5-25 and 5-26), the average scores of total rules on experienced participants are close for each other in order that its data distribution is more concentrated, as well as its interquartile rang is smaller. On the contrary, even though the novices acquire the maximum value in the design, the difference of their average scores is big so that its interquartile range is larger than experienced ones, Meanwhile, for the average scores on each evaluation rule and total rules, the situations are the same that the experienced ones are better than average level but the novices are not good as the average level.

Participants	**Pre-interview**	**Interview**	**Post-interview**	**Rank**
No/Category	*Self evaluation*	*Confidence*	*Score of the design*	
#3(N)	**3**	**Be not confident**	**7.40**	**1**
#2(N)	**1**	**Be not confident**	**7.30**	**2**
#12(E)	**4**	**Be not confident**	**6.75**	**3**
#8(N)	**2**	**Be not confident**	**6.50**	**4**
#9(E)	**5**	**Be confident**	**6.10**	**5**
#10(E)	**5**	**Not mentioned**	**6.00**	**6**
#13(E)	**3**	**Be not confident**	**6.00**	**7**
#14(E)	**3**	**Be not confident**	**5.85**	**8**
#11(N)	**4**	**Not mentioned**	**5.80**	**9**
#1(N)	**1**	**Be confident**	**4.95**	**10**
#7(N)	**3**	**Be not confident**	**4.95**	**11**
#4(N)	**3**	**Not mentioned**	**4.50**	**12**
#5(N)	**2**	**Not mentioned**	**4.25**	**13**
#6(N)	**2**	**Not mentioned**	**3.85**	**14**
N: novice participants E: experienced participants				

Table 5-27 Rank list based on the quality of the design for the participants

The table (see Table 5-27) lists all participants involved in this study ranked by their scores on the created class diagrams. Also, it is clearly presenting the confidence situations of each participant pre- and in the interviews. Pre-interviews, the participants were asked to make self evaluations on how confident they felt in creating UML class diagrams from a scale of 1 (not confident) to 5 (very confident). In the interview process, as discussed in the reasoning confidence, the participants showed their confidence or no confidence, as well as not mentioned their confidence for the designs. As a result, it is easy to compare the confidence situations of pre-/in/post interviews in such that participants without confidence can get high scores unexpectedly, contrarily, the participants who have confidence only can acquire the ordinary scores, which is not the same as their confidence. In brief, the confidence is not implying high scores on the quality of the designs.

6 Discussion

6.1 General reasoning process

The primary goal of this study is trying to understand the reasoning of novice and software designers on creating UML class diagrams. As explained in the result section, all the reasoning thoughts and behaviors of the participants have been summarize by seven reasoning activities of which are RS, DC, DA, DM, DAM, CD and NO. These seven reasoning activities describe the reasoning process on how the participants creating UML class diagrams in order that their reasoning can be clearly and easily understood as well as traced by others. The first activity of the participants created UML class diagrams is both RS which is the reasoning starting point and foundations of creating class diagrams. DC, DA, DM and DAM reasoning activities are corresponding to the compositions of UML class diagrams as their names hints, including determining class, attributes of the class, methods of the class and associations. CD is a verification reasoning activity, in which the created UML class diagram will be checked and considered carefully. Except for these six reasoning activities mentioned above, NO is a different one because not every participant performed it in their own reasoning process.

In a word, these seven reasoning activities are separated but unified entirety, explicitly as well as systematically representing the reasoning of novice and experienced participants on creating UML class diagrams.

6.2 Differences between novice and experienced software designers

Another important goal of this study is going to find the differences between novice and experienced software designers of their reasoning on creating UML class diagrams. The results about their differences are concluded from two aspects, which are the amount of reasoning activity and the time spent on reasoning activity.

For the amount of reasoning activity, to create one UML class diagram, the average total amount of the participants as experienced software designers is 21.80 comparing to 19.00 as novice ones, which means the difference is almost three reasoning activity. It seems that the experienced participants indeed did more reasoning activities for their designs in general. Meanwhile, on each reasoning activity, the situations are kind of diversity. For RS, DC, DM and CD, there is no big difference between the types but for DA, DAM and No, novice ones performed DA activities more but experience ones did DAM and NO more.

For the time spent on reasoning activity, experienced participants spent around ten minutes more than novice ones on average total time for creating one UML class diagram. It is said that they actually used more time to consider as well as to decide for one UML class diagram. Simultaneously, on each reasoning activity, the big differences between the types are DAM and NO. The experienced participants took much more time on considering the associations between classes and speaking out their design opinion during the design process.

For statistical significant difference, on the amount of reasoning activity, there is no statistical significant difference on the amount of each reasoning activity category and the total amount of reasoning activity they performed in the reasoning process between the novices and experienced participants. Although there are some differences exiting between different participants on performing reasoning activities, it is not such different on statistics. On the time spent of reasoning activity, the result found is that there is also no statistical significant difference between these two samples on the time spent of reasoning activity RS, DC, DA, DM, CD and the total time they spent for creating class diagrams. But, there is statistical significant difference on the time spent of reasoning activity DAM and NO such that the experienced software designers indeed spent more time on thinking about the relationships between the classes and speaking out their own ideas related to the design as well as improving their designs to make a better solution of the problem domain than novices.

6.3 Other findings

Reasoning structure

There are two reasoning structures found for the participants – “Depth First” and “Breadth First”. “Depth First” is the structure that determines one class completely and then goes on with next class until the end. Contrarily, “Breadth First” is the structure that firstly determines all the needed classes and then continues putting their contents until the end. There is no better or worse to compare the two structures, but for “Breadth First”, the participants who used this structure are more likely to treat the software system or application as a whole. Moreover, there is no proof that the experienced participants will adopt “Breadth First” more and novice ones just use “Depth First”. In the study, top to 60% of the experienced participants used “Depth First” and still 33.3% as novice ones adopt “Breadth First”. In a word, applying which reasoning structure is determined by the habit of the participants.

Furthermore, there are some small differences between the structures on theory and in practice. The real performing way of the participants was not the total same as the figures showing. (see Figure 5-15 and 5-16) For example on “Depth First”, in reality determining one class completely is not meaning that the participants need to determine attributes or methods of the class as showed at the same time. For instance, participants thought that one class only with its name was enough in some situations.

Reasoning confidence

During the process of creating UML class diagrams, different participants showed different confidences on their designs. For instance, “sure” is the key word for the participants to express their reasoning confidences in order that, “not sure” and “quite sure” become the symbols of measuring no confidence. As mentioned in the result section, most of the subjects (85.7%) were not confident enough or not mentioned; and only few (14.3%) were not. What’s more, experience is not equal to confidence in the study, some participants as experience software designers were not confident on their designs so as to use “not sure” many times, such as participant #12, #13 and #14. In a word, the factors determining reasoning confidence are multiple, and experience is important but not the most.

Reasoning source

To determine one UML class diagram, software designers will take advantage of the information related to the problem domain. In other words, the “information” is the reasoning source from which the software designers can get inspired and encouraged. In the study, the participants pointed out their own reasoning sources for deciding one class diagrams. It is no doubt that the requirement specifications and uses case model are obvious and basic reasoning sources for the participants. Moreover, there are some implicit reasoning sources in brain, which are hidden so as not to be easily captured. The reasoning sources in brain are accumulated by experience day and day, of which can be shaped from common sense in daily life, UML modeling as well as domain knowledge. In summary, the reasoning sources can be visible, such as the user requirements descriptions, and certainly be invisible, such as the experience form previous UML modeling. To make a better class diagram, both of them need to be treated and considered importantly.

Reasoning principles

In this study, a few reasoning principles were discussed by the participants. High cohesion and low coupling were the most frequently discussed principles. As just explained earlier, some reasoning principles were not really behaved on their creating UML class diagrams but they were just thoughts rather than behaviors. Meanwhile, some were difficult to reflect on the design because of limitations of the study, even for the most mentioned ones. For instance, owing to a small task related to this study, it is hard for the participants to show high cohesion and low coupling between different systems. Briefly, this research just presents some reasoning principles mentioned by the participants on creating UML class diagrams but not to elaborate how the reasoning principles worked such that further studies should be carried out.

6.4 Quality of the design

All the designs in this study were evaluated and marked by four professional experts in SE with deep UML modeling backgrounds. Considering on five evaluation rules, the professional experts made an objective and impartial judgment. From the evaluation results, it is clear that experienced participants get higher scores at the average level and its data distribution is more concentrated than novices. But, the experience does not imply the higher ranks in the score rank list because of that the two best scores in the rank list are the novice participants rather than experienced ones. Similarly, confidence is also not meaning higher scores. The participants, who showed their confidence before or in the interview process, are not necessarily making a better design as their confidence presents. In other word, there is no direct correlation between the high scores and the experience, as well as the confidence.

7 Threats to Validity

The threats of validity related to this study will be discussed in this section on internal validity, external validity, methodology validity and construct validity.

7.1 Internal validity

This study is conducted to investigate the reasoning of how novice and experienced software designers on creating UML class diagrams. Although all participants have previous experience on UML modeling and are selected carefully, the fact is that some participants still feel unfamiliar and difficult to the problem. As a result, there are 16 interviews held totally but 2 interviews are invalid. To avoid wrong categories on selecting the participants so as to guarantee the interview data efficient in the future, one suggestion is that the subjects could be taking a small test in advance to be decided whether is suitable for the study or not.

Meanwhile, it is known that different people have different reasoning ways such that reasoning way is diversity. To clarify the reasoning on creating UML classes deeper, it is satisfied to increase the number of the participants to improve the result confidences.

7.2 External validity

The results or findings of this study are based on the interview transcripts of the participants. All interview transcripts are analyzed and summarized strictly in order that the result and findings are convinced. However, the conclusions of the study still lack of generalizability. On one hand, as just mentioned in the internal validity, the number of the participants is limit. On the other hand, because of very litter researches related to this study direction carried out before, the results and findings are kind of fresh and need carefully verified in the other studies.

7.3 Construct validity

This study is conducted by a master student in software engineering. Thus, the research lacks of professional considerations or suggestions. Even though the supervisor offers many valuable advices, there are actually no experts on reasoning area of SE taking part in directly. For improvement, some professional experts should get involved in the future study.

7.4 Methodology validity

Methodology validity in this study is standing for measuring the used experiment method. This research is conducted by TAP. TAP is a good approach which can easily understand the whole process of the participants' thinking, doing and feeing as going on the task. However, this methodology still has some weakness, such as unnatural experimental environment, disturbing the interviewees' operations. For instance, during the interview, the participant will be disturbed while the phone suddenly rings so that the reasoning thoughts of the participant may disappear. In one word, TAP is reliable but not omnipotent. To eliminate the shortcomings of TAP is impossible and in the future, what can be improved is trying to decrease the influence of them on the results.

8 Further research

Potential further researches based on the results or findings are suggested as below.

- Choose more participants to expand the study. This study concludes the seven reasoning activities for describing the reasoning of novice and experienced software designers on creating UML class diagram. However, the number of the participant is limit. Therefore, more participants are beneficial to clarify the reasoning better.
- Verify the results and findings of the study. As discussed earlier, there are little studies on this topic before, the results and findings need to be verified by the other studies.
- Consider the situation by using computer for creating UML class diagram. In this study, the participants draw their class diagrams on the paper. In fact, the software designers frequently make their designs on the computer by using software tools, such as Enterprise Architect. Therefore, further researches should involve the situation that software designers creating UML class diagrams on the computer so as to compare the outcomes with this study.

9 Conclusion

The reasoning of novice and experienced software designers on creating UML class diagrams can be described by seven reasoning activities as RS, DC, DA, DM, DAM, CD and NO in general. These reasoning activities vividly explain the whole reasoning process of them creating class diagrams, which is also dividing the reasoning thoughts and behaviors into their own corresponding reasoning activities. In terms of differences between different types of software designers, the experienced ones performed more reasoning activities and spent more time on their designs than novice ones generally.

Furthermore, there are still some other beneficial findings related to the reasoning process which can be concluded that a) two reasoning structures were found – "Depth First" and "Breadth First". Most participants used "Depth First" to create class diagrams; b) experience does not imply that having enough reasoning confidence. Some experienced participants unexpectedly showed their no confidence on their designs; c) reasoning sources of creating UML class diagrams were mainly in two parts – on the paper, such as requirement specifications and in the brain, such as the daily life experience; d) there were some general reasoning principles discovered but not specific; e) experience and confidence do not imply higher quality of the designs.

I believe the results and findings of this study are a good start to clarify the reasoning of software designers creating UML class diagrams thoroughly as well as to establish the foundations for further relevant researches.

Acknowledgement

I would like to thank all participants who took part in the interviews and Professor Michel Chaudron for his support, suggestions and feedbacks during the study.

Reference

[1] Schenk K D, Vitalari N P, Davis K S. Differences between novice and expert systems analysts: What do we know and what do we do?[J]. Journal of Management Information Systems, 1998, 15(1): 9-50.

[2] Bolloju N, Leung F S K. Assisting novice analysts in developing quality conceptual models with UML[J]. Communications of the ACM, 2006, 49(7): 108-112.

[3] Hadar I, Soffer P. Variations in conceptual modeling: classification and ontological analysis[J]. Journal of the Association for Information Systems, 2006, 7(8): 568-592.

[4] Soffer P, Hadar I. Applying ontology-based rules to conceptual modeling: a reflection on modeling decision making[J]. European Journal of Information Systems, 2007, 16(5): 599-611.

[5] Jalloul G. UML by Example[M]. Cambridge University Press, 2004.

[6] Brugali D, Torchiano M. Software development: case studies in Java[M]. Pearson Education, 2005.

[7] Tang A, Aleti A, Human Reasoning and Software Design: An Analysis, *Proceedings Workshop Studying Professional Software Design, University of California, Irvine,* 2010.

[8] Zannier C, Chiasson M, Maurer F. A model of design decision making based on empirical results of interviews with software designers[J]. Information and Software Technology, 2007, 49(6): 637-653.

[9] Gero J S, Mc Neill T. An approach to the analysis of design protocols[J]. Design studies, 1998, 19(1): 21-61.

[10] Atman C J, Adams R S, Cardella M E, et al. Engineering design processes: A comparison of students and expert practitioners[J]. Journal of Engineering Education, 2007, 96(4): 359-379.

[11] Reasoning, available at *http://www.thefreedictionary.com/reasoning.*

[12] Think-aloud Protocol, available at
http://www.hu.mtu.edu/~njcarpen/hu3120/pdfs/thinkaloud.pdf.

[13] Socha D, Tenenberg J. Sketching software in the wild[C]//Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013: 1237-1240.

[14] Cherubini M, Venolia G, DeLine R, et al. Let's go to the whiteboard: how and why software developers use drawings[C]//Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 2007: 557-566.

[15] Petre M, Blackwell A F. Mental imagery in program design and visual programming[J]. International Journal of Human-Computer Studies, 1999, 51(1): 7-30.

[16] Purchase H C, Colpoys L, McGill M, et al. UML class diagram syntax: an empirical study of comprehension[C]//Proceedings of the 2001 Asia-Pacific symposium on Information visualization-Volume 9. Australian Computer Society, Inc.,

2001: 113-120.
[17] Van Heesch U, Avgeriou P. Naive Architecting-Understanding the Reasoning Process of Students[M]//Software Architecture. Springer Berlin Heidelberg, 2010: 24-37.
[18] Tang A, Aleti A, Burge J, et al. What makes software design effective?[J]. Design Studies, 2010, 31(6): 614-640.
[19] Tang A, van Vliet H. Software Architecture Design Reasoning[M]//Software Architecture Knowledge Management. Springer Berlin Heidelberg, 2009: 155-174.
[20] Van Someren M W, Barnard Y F, Sandberg J A C. The think aloud method: A practical guide to modelling cognitive processes[M]. London: Academic Press, 1994.
[21] Anderson R E, Helstrup T. Visual discovery in mind and on paper[J]. Memory & cognition, 1993, 21(3): 283-293.
[22] Ahmed S, Wallace K M, Blessing L T. Understanding the differences between how novice and experienced designers approach design tasks[J]. Research in Engineering Design, 2003, 14(1): 1-11.
[23] Cross N, Dorst K, Roozenburg N. Research in design thinking[J]. 1992.
[24] Dorst, K., Christiaans, H., and Cross, N. Analysing design activity[M]. Chichester: Wiley, 1996.
[25] Tang A, Tran M H, Han J, et al. Design reasoning improves software design quality[M]//Quality of Software Architectures. Models and Architectures. Springer Berlin Heidelberg, 2008: 28-42.
[26] Jaaskelainen R. Think-aloud protocol[J]. Handbook of Translation Studies. Amsterdam: John Benjamins Publishing, 2010: 371-3.
[27] Ericsson, K.A. and Simon, H.A. Protocol Analysis; Verbal Reports as Data – Revised Edition, MIT Press, Cambridge, Massachusetts, 1993.
[28] Atman C J, Chimka J R, Bursic K M, et al. A comparison of freshman and senior engineering design processes[J]. Design Studies, 1999, 20(2): 131-152.
[29] Kavakli M, Gero J S. The structure of concurrent cognitive actions: a case study on novice and expert designers[J]. Design Studies, 2002, 23(1): 25-40.
[30] Cheesman J, Daniels J. UML components[M]. Reading: Addison-Wesley, 2001.
[31] Schoonewille H H, Heijstek W, Chaudron M R V, et al. A cognitive perspective on developer comprehension of software design documentation[C]//Proceedings of the 29th ACM international conference on Design of communication. ACM, 2011: 211-218.
[32] Moody D, van Hillegersberg J. Evaluating the visual syntax of UML: An analysis of the cognitive effectiveness of the UML family of diagrams[M]//Software Language Engineering. Springer Berlin Heidelberg, 2009: 16-34.
[33] Nikiforova O, Sejans J, Cernickins A. Role of UML Class Diagram in Object-Oriented Software Development[J]. Scientific Journal of Riga Technical University. Computer Sciences, 2011, 44(1): 65-74.

Appendices

Appendix A – interview guide

Interview guide

Purpose of the interview:
Investigating the reasoning of how the software designers creating models during the software design process
Structure
1 .Information to the interviewee **(1-3 min)**
2. Distribute the background questionnaire and answer it **(4-6 min)**
3. Distribute the task specification to the interviewee **(1 min)**
4. The interviewee reads the specifications of what need to do **(5-7 min)**
5. The interviewee finished the task specification (during this step, the interviewees should keep talking how they do the task)**(30-40min)**
6. Done **(MAX 60 min)**
7. Change to interview next interviewee

Information

- This study is part of the master thesis project to understand the reasoning of how software designers creating model as well as improving the future teaching in software design industry

- The interview is held individually

- The whole procedure will last around 1 hour

- The whole procedure is consisting of two parts. First, you need to answer the background questionnaire. Secondly, you will get the task specification and then accomplish the requirements in a limit time.

- All your answers are just used for the research and will be kept confidentially.

- When you doing the task in the second part, the procedure will be recorded by digital equipment. And the record will be deleted while the study is completed.

- **This study is conducted by Think-aloud Protocol method (TAP).** TAP method is a good approach which can understand the whole process of interviewees' thinking, doing and feeling as going on the task. In briefly, while you taking the task of the study in the second part, please keep talking to reflect your reasoning thoughts of how you creating your own models.

- In a word, thank you for your contribution to this study!

Ⅰ Background Questionnaire

Here list some questions about your background and experience in software design, especially for UML modeling in software design process. The answers you provide are used for the master thesis project research.

Please read and answer all of the questions. Feel free to write your own answers and also thanks for your comments.

The information that you provide will be held in strict confidence.

Finally, thank you for your help.

Your Name:

Date:

Study Background

1. Are you a student now? Yes / No

A) What's your highest diploma level?

A. Bachelor Level
B. Master Level
C. Doctor Level
D. Other Level

B) What's your major?

A. Software Engineering or relevant
B. Computer Science or relevant
C. Others

C) If your major is others. Please mention below:

Your major:

Work Experience

2. How many years of experience do you have in industrial software development?

A. 1-3 years
B. 4-6 years
C. 7-9 years
D. >=10 years

A) What type of work do you do currently?

A. Manager
B. Project leader
C. Member
D. Others

B) Which kind of job have you ever taken? (You can choose multiple)

A. Requirement analyst
B. Software Architect
C. Software Designer
D. Software Analyst
E. Programmer
F. Tester

UML Experience

3. Have you ever studied UML when you were a student? Yes / No

A) How many projects have you experienced using UML modeling in practice?

A. 1-5 projects
B. 6-10 projects
C. >= 10 projects

B) How confident are you in creating UML designs? (From 1 to 5 and 1 means not confident, 5 means very confident)

A. 1
B. 2
C.3
D.4
E.5

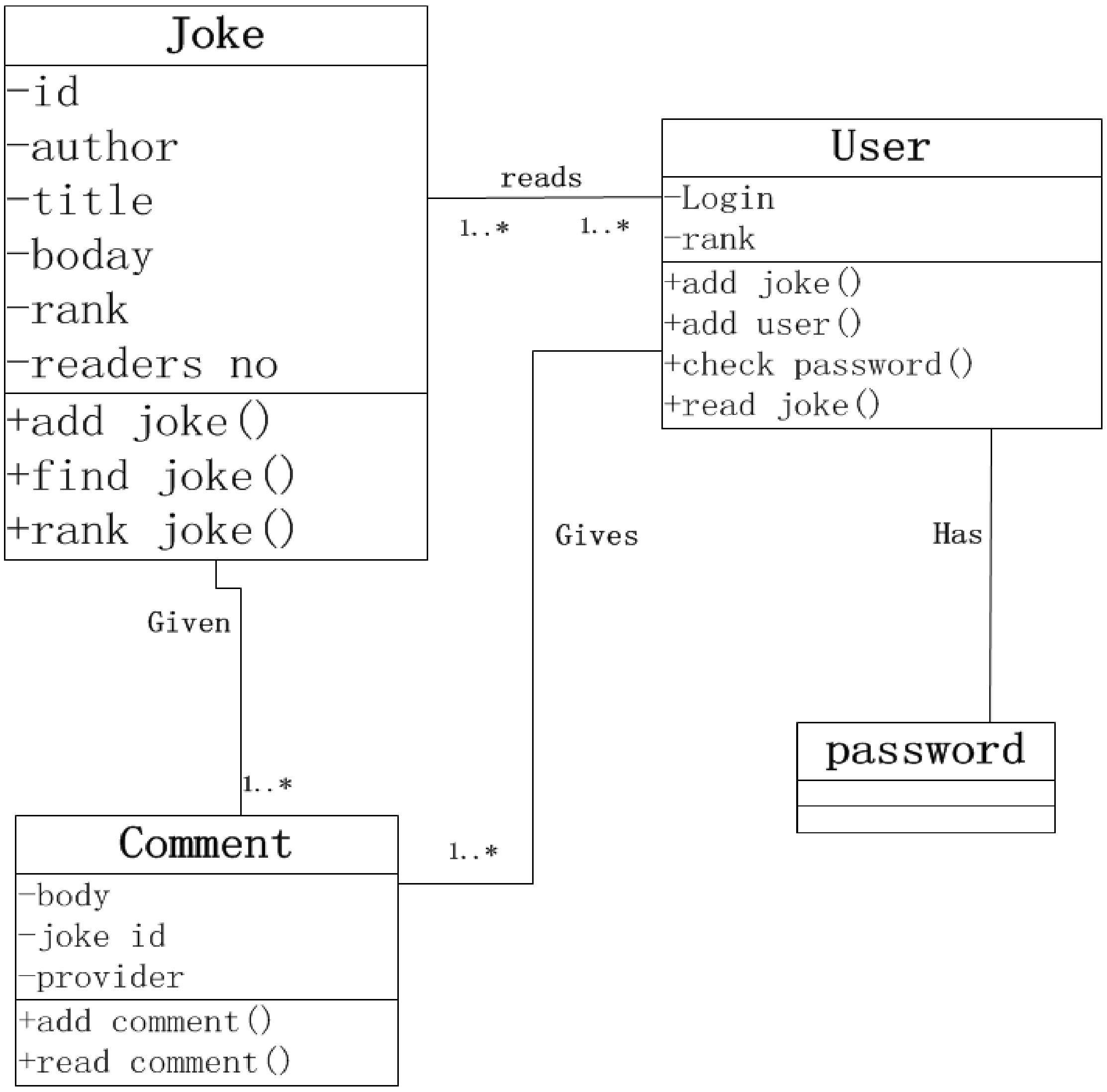
Comment:

Please make any additional comments that you want relevant to your former background.

Ⅱ Task Specification

Specification: *In this part, your task is going to design an initial UML class model of a reservation/tours online system based on the user requirements and use case model presented below. Please read the information of the user requirements and the use case model carefully and then draw the initial class diagram on the paper. When you create the class model, please keep talking out loud continuously about what you are thinking. The procedure will be recorded individually for research and be deleted once the study is done. All your information will be kept confidentially.*

First, this is an example of class diagram of Joke-teller system, please take a reference.

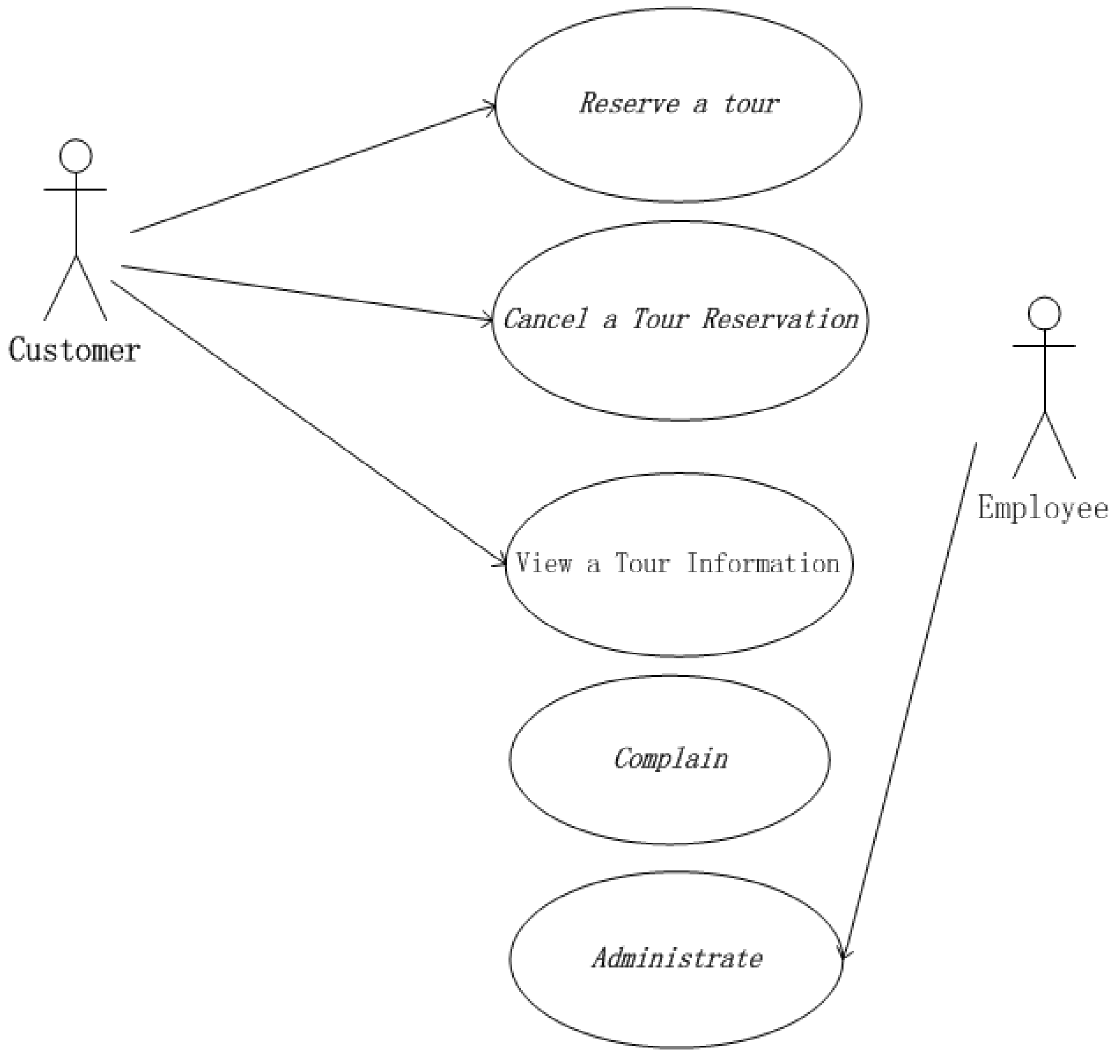


Joke-teller class diagram

1. Tours Online user requirements

Software for a travel agency provides reservation facilities for the people who wish to travel on tours by the travel bureau. The application software stores information about tours. Users can access the system to make a reservation on a tour and to view information about the tours available without having to go through the trouble of asking the employees at the agency. The third option is to cancel a reservation that was made. Any complaints or suggestions that the client may have could be sent by email to the agency or stored in a complaint database. Finally, the employees of the corresponding agency could use the application to administrate the system's operations. Employees could add, delete, and update the information on the customers and the tours. For security purposes, employees should be provided a login ID and password by the manager to be able to access the database of the travel agency

2. Use case model of Tours Online:



Please draw your own initial class diagram below:
(Please remember to speak out your reasoning thoughts)

Please summarize the main steps on how you created the initial class diagram of this case.

Appendix B – interview data

Interview data

-Collected data of each participant

1. Interview 1
Interview dialogue

RS: [Read the user requirements]

DC: At beginning, it should be an object called as the website. …. I hope so. And second one is the user.

DAM: Then the users have access to the website, so the multiplicity from user to website is 1…*. OK. Many users can access to one website and one website can have many users. Yes, 1…*. Then the users apply to login the website.

RS: [Read the user requirements]

DM: So, making reservations is a method. Cancel the reservations and view information is also the method of the website. Meanwhile, the user can access to the system to make complains, so it is the method and it should also have a method called administration.

DC: The next object is employee.

DA: It has the attributes of login ID and password. The login ID and the password should be provided by the manager who has the access to the database.

RS: [Read the user requirements]

DM: The object user should have method of adding, updating and deleting. And the employees can add, update, and delete users, the information of customers and tours. So, the website should also have the methods which are add tours, update and delete.

CD: [Have a look at the example; change the object name website to system; check the object system]

DA: The user can access to the system to do these methods and the employee also can access to the system to do them. And the system should have the attributes of user information, including id, name and tours selected. It also contains employee id and tours id.

RS: [Read the user requirements;]

CD: [Check the created class diagram]

DA: Mm…, the system should have the attributes of complain id.

DC: Yes, the name should be system rather than website. The employees can administrate operations of the system.

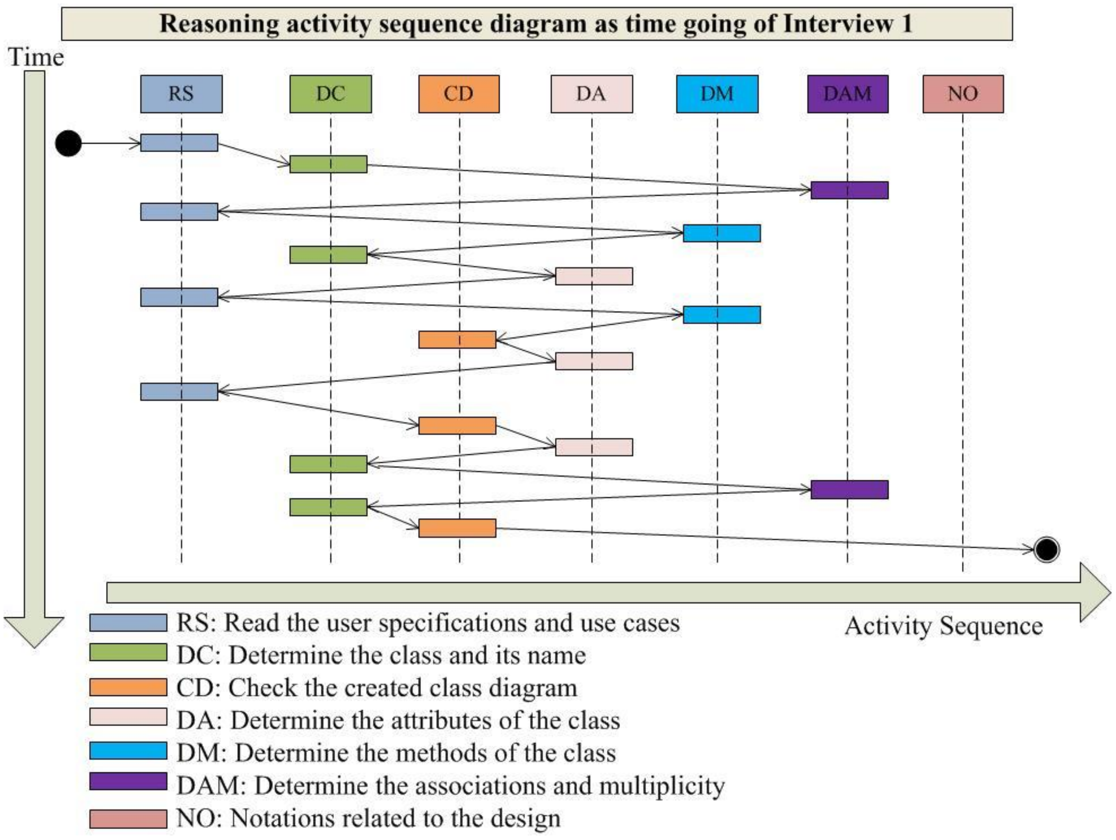
DAM: The system can have many employees such that the multiplicity is 1…* and the employees can only administrate one system. (Consider the association between the employee and the user, and then delete the connection between them)

DC: The employees should be managed by manager. The manager provides login ID and password to the employees.

CD: The employees can administrate the information of the system and they can access the complaints which sent by emails from customers. The user can view information, complain, make and cancel a reservation. Mm…, good! And the employees can add, update and delete the information of the user. (Reconnect the association of employee and user; inspect the created class diagram again)

Mm ….. I think it is done.

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview 1
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	*Start Time*	*Finish Time*	*Duration*
RS	**Before starting, always reading requirements**		**5m**
DC	[00:00:00]	[00:00:29]	**29s**
DAM	[00:00:29]	[00:01:05]	**36s**
RS	[00:01:05]	[00:01:17]	**12s**
DM	[00:01:17]	[00:02:40]	**1m23s**
DC	[00:02:40]	[00:02:50]	**10s**
DA	[00:02:50]	[00:03:29]	**39s**
RS	[00:03:29]	[00:03:58]	**29s**
DM	[00:03:58]	[00:05:07]	**1m9s**
CD	[00:05:07]	[00:05:47]	**40s**
DA	[00:05:47]	[00:07:17]	**1m30s**
RS	[00:07:17]	[00:07:27]	**10s**
CD	[00:07:27]	[00:07:52]	**25s**
DA	[00:07:52]	[00:08:04]	**12s**
DC	[00:08:04]	[00:08:49]	**45s**
DAM	[00:08:49]	[00:09:36]	**47s**
DC	[00:09:36]	[00:10:00]	**24s**
CD	[00:10:00]	[00:11:05]	**1m5s**
	RS	**5m 51s**	
	DC	**1m 48s**	
	CD	**2m 10s**	
	DA	**2m 21s**	
	DM	**2m 32s**	
	DAM	**1m 23s**	
	Total Time	**16m5s**	

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:

-Comparing the user requirements and use case model, which one is more useful when drawing the class diagram? Why?

I think the user requirement is more useful because it contains more details, such as the attributes of the objects. The use case model only gives me the picture of the system, nothing more information provided. At first, the use case model is useful and it is a good start to understand what the system should be like but later, to better understand the details of the system, the user requirements can provide more and be more useful. And the decisions of what objects the system should have, what attributes, methods, associations and multiplicity should contain, mostly are based on the user requirements.

-How do you decide the objects of the class diagram? What will you add?

It should be more objects, like the database as well. It should have one object database connected to the system which storing all the information about tours, users and employees, the database should be managed by manager. The user in my class diagram is the customer in the use case model and I add the actor of the manager because the employees should be provided a login ID and password by the manager from the user requirements. Certainly, the employee could be the manager.

-Common steps on how to create the class diagram

- Read the user requirements
- Look at the use case model
- Try to identify the object
- Try to identify the attributes of the object
- Try to identify the relationship between objects
- Try to think about the methods by understanding what actions they perform
- Read again to verify whether you miss some important points

2. Interview 2

Interview dialogue

RS: [Read the user requirements]

DC: First of all, I think it should be an object customer who can use the system and it has the same meaning of the user in the text.

RS: [Read the user requirements]

DC: OK, let me take some notes from the text. Reservations, travel agency bureau, tours and the users can access to the same system, so I guess the system should be in the diagram…How can I name it? Reservation? [Have a look at the example] Mm… If I look at the Joke-teller system, there is no system self. Reservation? Tour? I think the tour should be in the middle.

DM: Let's see the customer again. The customer can reserve tour, cancel tour and view information.

RS: [Read the user requirements and take notes on important points]

DC: Mm… It should be another class called complaints.

RS: [Read the user requirements]

DC: Mm… There is another class named administration. I just name the administration but I don't think it's a good name.

DA: And it should have the attribute of customer name. Oh, no, the customer name is in the object customer self, so it should be the customer ID. Yes, the customer ID should be in the customer as well as in the administration. Is it possible? (Check the example) Yes. All the information of customers is in the class customer self and to be able to manage the customer, the customer ID should be in the class administration. And the class tour has the attribute of tour ID and the tour ID is also in the class administration.

RS: [Read the user requirements and thinking]

DC: Actually, I think it could be nice the class administration can be used by employees. It should be a class employee. And the employee has login ID and password.

DM: To only have the method for deleting, adding and updating everything in one class, so it could add the method of adding, deleting and updating to the class administration and just refer to the employee, maybe.

DA: I just do that so I have attribute of employee ID in the administration class as well. It doesn't say anything about employee self, but I think it could be nice to have and I also need a tour. Mm…. add, delete and update, the administration class is finished. And employee, it has login ID and password. Since there is a difference between normal employee and manager, the employee class could have something like rights or roles. Maybe the role includes rights such that only the manager can actually delete or give the access. I don't know. I just add role.

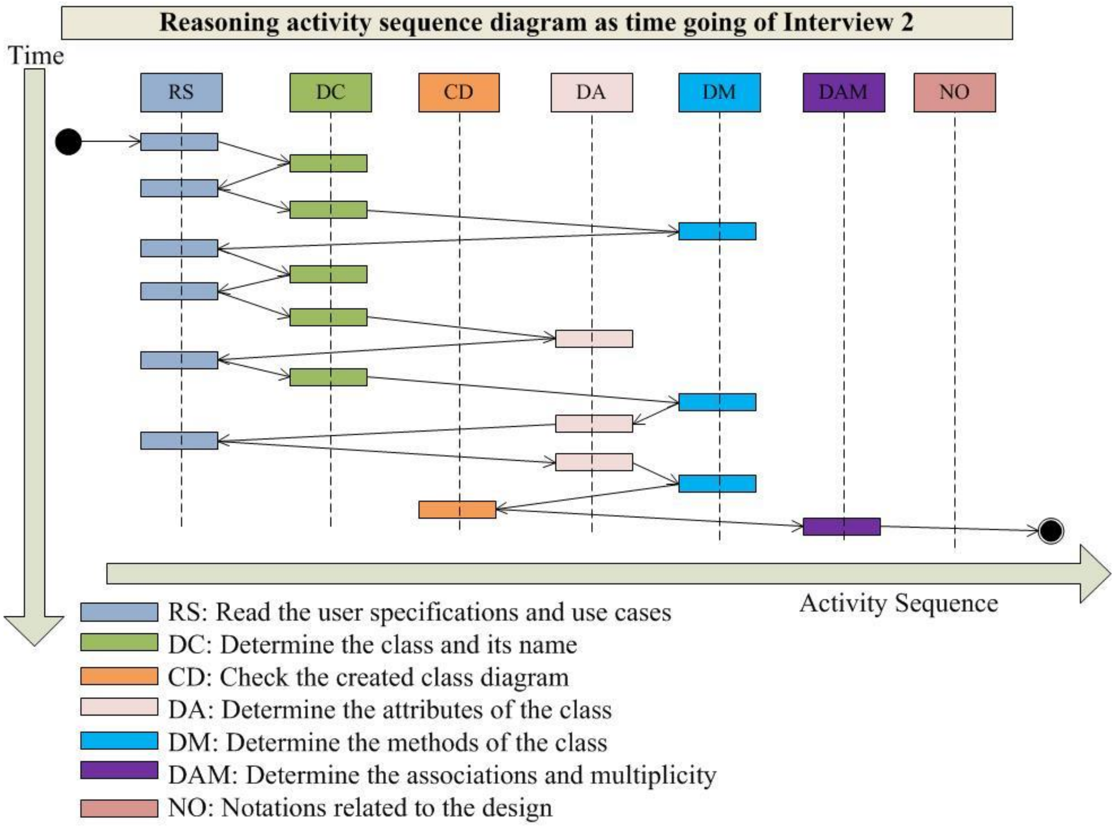
RS: [Read the user requirements]

DA: Mm… Can I summarize it as the information? I add attribute of information to the tour and the information about tours, such as the destination.

DM: (Look at the example) the example is totally different. Mm… I think the class employee should have methods at least so I add the view information to it. Ok, the

complaints. Any further information about complaints? I have complaints and let's say that it has ID and information which containing the complaints and suggestions. I don't know what methods I should have. Add complaints? Read complaints? OK. I think customer can add the complaints and the employees have to read the complaints.
CD: [Check the created class diagram]
DAM: Mm… Now, I have to make everything. (Have a look at the example and thinking of the association's name) One customer can reserve tour. Oh, no. I want to use a word for summarizing the all the actions. Let's say handle. And employee also handles the tour. Mm…it should be a good name to describe what employee can do on tour but I don't know how to name it. OK. Administration class can administrate one or several employees as well as customers. It should be one-to-many association. The customer can make or send complaints or suggestions. It also should be one-to-many association. (Forget the multiplicity of the other end of the association) One tour can be booked by one or several customers and the tour information can be viewed by one or several employees. One administration can administrate one or several employees and customers. And since the complaints are individual, the complaints only can be sent or made by one customer. ….
OK, I think it's done.

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview 2
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	*Start Time*	*Finish Time*	*Duration*
RS	**Before starting, always reading requirements**		**5min**
DC	[00:00:00]	[00:00:25]	**25s**
RS	[00:00:25]	[00:00:50]	**25s**
DC	[00:00:50]	[00:02:51]	**2m1s**
DM	[00:02:51]	[00:04:10]	**1m19s**
RS	[00:04:10]	[00:05:34]	**1m24s**
DC	[00:05:34]	[00:05:50]	**16s**
RS	[00:05:50]	[00:06:27]	**37s**
DC	[00:06:27]	[00:07:12]	**45s**
DA	[00:07:12]	[00:08:51]	**1m39s**
RS	[00:08:51]	[00:09:26]	**35s**
DC	[00:09:26]	[00:10:14]	**48s**
DM	[00:10:14]	[00:10:35]	**21s**
DA	[00:10:35]	[00:12:39]	**2m4s**
RS	[00:12:39]	[00:12:58]	**19s**
DA	[00:12:58]	[00:13:23]	**25s**
DM	[00:13:23]	[00:16:42]	**3m19s**
CD	[00:16:42]	[00:18:30]	**1m48s**
DAM	[00:18:30]	[00:23:20]	**4m50s**
RS		**8m20s**	
DC		**4m15s**	
CD		**1m48s**	
DA		**4m8s**	
DM		**4m59s**	
DAM		**4m50s**	
Total Time		**28m20s**	

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time

-How do you think of the example class diagram, use case model and the user requirement?

To come up with the class diagram of this case, of course the use case model and the text is more important. But for me, since I kind of forget, it is easy to draw something when you have examples and you can check how they do in any similarity between what I suppose to do and what they do.

-Comparing the user requirements and use case model, which one is more useful when drawing the class diagram? Why?

The use case model gives you some ideas. For example, when you see the circle of reserve a tour, you know that should be a method. But for more detail information, it exists in the text. The use case model can offer some hints that the class diagram should be like. I need both of them but probably, the use case model is not enough. It would be possible to draw the class diagram when only based on the text. The use case model is a nice addition.

-Do you think your former experience affect your decisions on your created class diagram?

Of course, it is the only experience I have about UML class diagram.

-Common steps on how to create the class diagram

- read the requirements
- check the use case model
- check the example to have an idea how UML class diagram should look like
- identify the entities involved in the system from requirements and use case diagram
- identify the actions the entities can perform
- identify their attributes
- identify the relations between them
- last 4 steps are not dome necessarily in this order but iteratively until the class diagram is done

3. Interview 3

Interview dialogue

RS: [Read the user requirements]

DC: First, we need to make sure that how many entities we should have in this class diagram. Of course, the first is the user and then I define the user as an abstract class. For this abstract class, it has two subclasses which are the customer and the employee. Next, the entity is the reservation.

RS: [Read the user requirements]

DC: Mm… I guess there should be another entity called tour.

RS: [Read the user requirements]

DC: I guess it should be an entity named as complain. Also, let's see the manager. I think it is the type of the user and I add it called administrator.

CD: [Find and speak out the evidence from the user requirements to support the existing of the created entities]

DM, DAM: OK. The customer can add reservation and one customer has several reservations. The tour and the reservation should be also connected. One reservation could be only to one tour but one tour, you can make several reservations depending on your information about the tour.

DA: (Ask whether the attributes can be defined freely) OK. As the user, I want to have the name, gender and perhaps the userID. As the customer, I don't think I need to add more attributes. Let's go back the tour. It should contain the duration, destination and how many reservations could make about the tour. How should I name it? Amount? Ok, that's it.

CD: [Check the created class diagram]

DM: And the customer could also view tour and cancel reservation.

DA: Oh, I forgot something important. The tour should have TourID and also the reservationID is in the reservation class. And in the reservation, it should contain the total amount of the reservations and the attribute is the amount.

RS: [Read the user requirements]

DM: OK. The customer could also add complaints.

DAM, DA: One customer can make several complaints and for the class of complain, it should have the category, like I want to complain your website or complain your service in different areas. And also it should have the attribute of the comment.

CD: [Check the created class diagram]

DM: The employee could add/delete/update both users and tours. So, I add these methods to the class of employee.

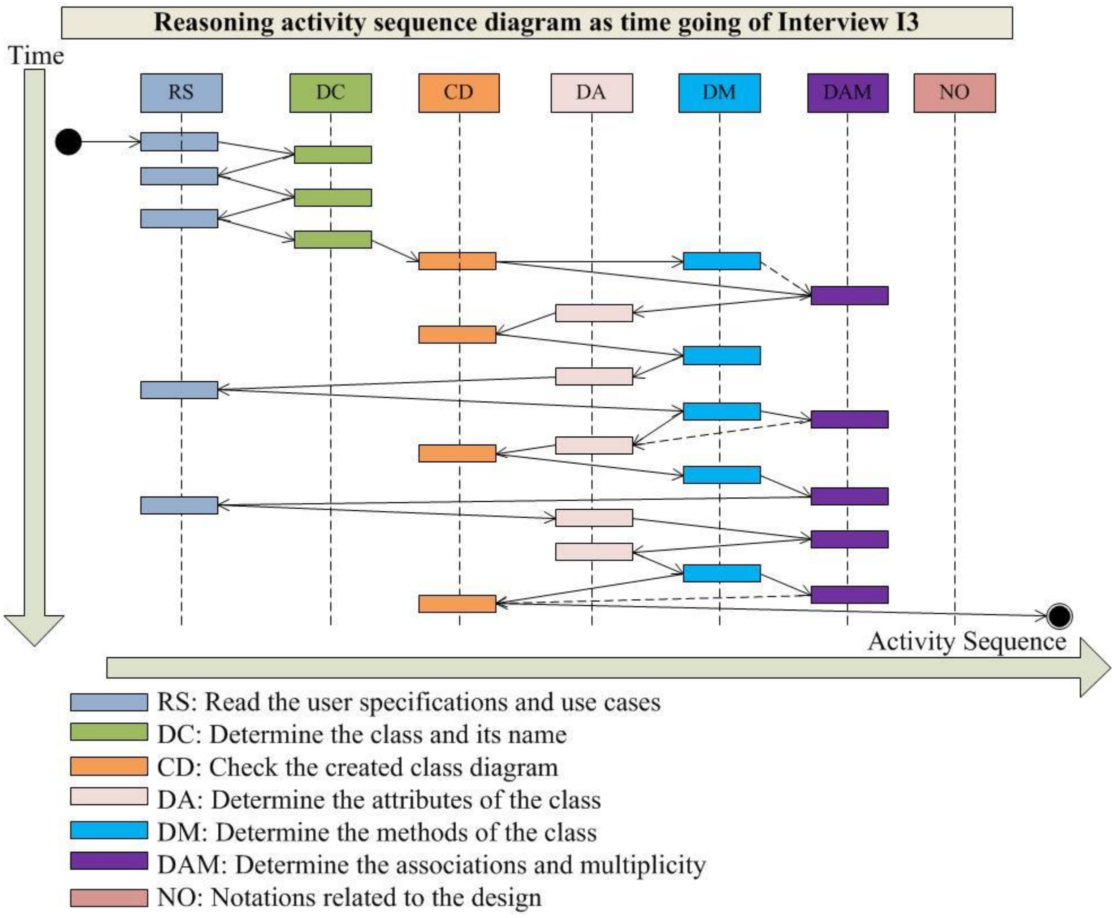
DAM: And the employee would be connected customer and the tour. I think one employee could manage several customers and one customer could be managed by several employees. Also, one employee can manage several tours and one tour could be managed by several employees.

RS: [Read the user requirements]

DA: The employee could have the attributes of login ID and password. Because the employees should be provided a login ID and password by the administrator so the entity of administrator should also have the same attributes of login ID and password.

The difference is that the administrator can add/update/delete the employees.
DAM: And only one administrator could manage several employees. The customer? The customer can view tour information such that there should be an association between them.
DA: (Think for a while) Maybe, the tour should have the attribute of status, like the tour is available or is closed.
DM: And the tour should have the action of viewing available tour which only returning the available tours, but I am not quite sure.
DAM, CD: (Think for a while and finish the left association's name) [Check the created class diagram]
Mm…OK, I think everything is done.

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview I3
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	*Start Time*	*Finish Time*	*Duration*
RS	**Before starting, always reading requirements**		**5min**
DC	[00:00:00]	[00:01:50]	**1m50s**
RS	[00:01:50]	[00:02:30]	**40s**
DC	[00:02:30]	[00:02:50]	**20s**
RS	[00:02:50]	[00:03:18]	**28s**
DC	[00:03:18]	[00:05:06]	**1m48s**
CD	[00:05:06]	[00:05:27]	**21s**
DM, DAM	[00:05:27]	[00:07:23]	**1m56s**
DA	[00:07:23]	[00:08:59]	**1m36s**
CD	[00:08:59]	[00:09:12]	**13s**
DM	[00:09:12]	[00:10:11]	**59s**
DA	[00:10:11]	[00:11:11]	**1m**
RS	[00:11:11]	[00:11:31]	**20s**
DM	[00:11:31]	[00:12:07]	**36s**
DAM, DA	[00:12:07]	[00:12:57]	**50s**
CD	[00:12:57]	[00:13:13]	**16s**
DM	[00:13:13]	[00:14:25]	**1m12s**
DAM	[00:14:25]	[00:16:00]	**1m35s**
RS	[00:16:00]	[00:16:13]	**13s**
DA	[00:16:13]	[00:18:13]	**2m**
DAM	[00:18:13]	[00:21:00]	**2m47s**
DA	[00:21:00]	[00:21:37]	**37s**
DM	[00:21:37]	[00:22:20]	**43s**
DAM, CD	[00:22:20]	[00:24:20]	**2m**
	RS	**6m41s**	
	DC	**3m58s**	
	CD	**2m40s**	
	DA	**5m58s**	
	DM	**3m36s**	
	DAM	**6m27s**	
	Total Time	**29m20s**	

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question Time:

-What reasoning did you use for deciding what should be classes in your diagram?

Usually, except complain and reservation class, in my opinion that the entity which containing actions should be the class. Then, I think the things can be stored as documents should also be the classes, such as complain and reservation. Moreover, according to the user requirements, the nouns shown up in it should either be classes or be attributes.

-What reason did you use for determining attributes and methods of the classes?

The methods I created are based on the actions and the actions are based on the use case model. And as is known to all, the class diagram has different abstract levels and actually the methods are also from the user requirements as well as use case model. If going inside of the classes, it is not enough just based on the use case model but, in terms of the initial class diagram, it works.

For the attributes, some are based on the common sense. For example, the attributes of user class, i.e. name, gender, user ID, whenever we register on this kind of system, we need to provide this information to it. And the other attributes come from the user requirements. At the same time, I derive some attributes, like category of complain class. I think the employees may feel interested about which category this complain is in and how many complains in this category. So, I add this attribute.

-What reasoning did you use for deciding associations between classes?

I think the associations are based on the actions. Basing on the actions means the source is use case model. It is known that the customer can make a reservation such that they should be connected for each other. It is the initial class diagram so I don't need to consider too many details, however, for deciding the associations in detailed class diagram, certainly I will take advantage of the use case model as well as the user requirements and my experience.

-Did you think of any design principles while designing the class diagram?

Mm… I think I use the abstract class in my design. I have a abstract class-user which made of three concrete class-- customer, employee and administrator. Basically, they will share the same attributes but they can have different behaviors. As mentioned in the requirements, only the employees will be provided the login ID and password. This is my consideration about the design principles.

-What's your naming strategy? i.e. how did you think of the names of your class diagram?

Actually, I am not good at this naming thing and I use manage as the association' name but I don't think it is a good name to present the clear connection between the related classes. Mm…I name it because of the text book and we should avoid this kind of names, it is too general. For attributes' name, it is the same with the user requirements and the actions are based on the use case model as well as the requirements. In addition, some names not mentioned in the materials are based on my experience as well as the common sense in our daily life.

-Common steps on how to create the class diagram

- Draw all the classes
- Write actions while building associations and also write attributes maybe needed in each action (follow the user requirements to define actions)
- Add more classes/attributes/actions based on my experience

4. Interview 4

Interview dialogue

RS: [Read the user requirements]

DC: The first thing I will do is trying to identify all the concepts in this case. The first thing I start is the tour.

NO: Before I put all the information about the tour should contain, I just draw a big box like this. After I have all the elements with the boxes, I will figure out the relationship between them and the multiplicity. Finally, I probably put all the attributes and their types into the classes. (Ask the consist of the initial class diagram) OK, method also.

CD: [Make some notes and sketches as a base for the initial class diagram, check it]

RS: [Read the user requirements] (After a while and start to draw the class diagram)

DC: Mm… I try to avoid the system, database and these kinds of things and try to capture the most important things in the domain. I am going by the descriptions and finding the important concepts in the description. Basically, the reservation is important as well as the tour. And we have users – customer and employee. They can give the feedbacks about the tour, I will draw it now. First, I just put the entities of reservation, tour and the user. The user has two subclasses of customer and employee.

DA: The user also has the attributes of login ID and password and the subclasses can reuse it.

DAM: I just make a connection between them.

CD: [Check the created class diagram]

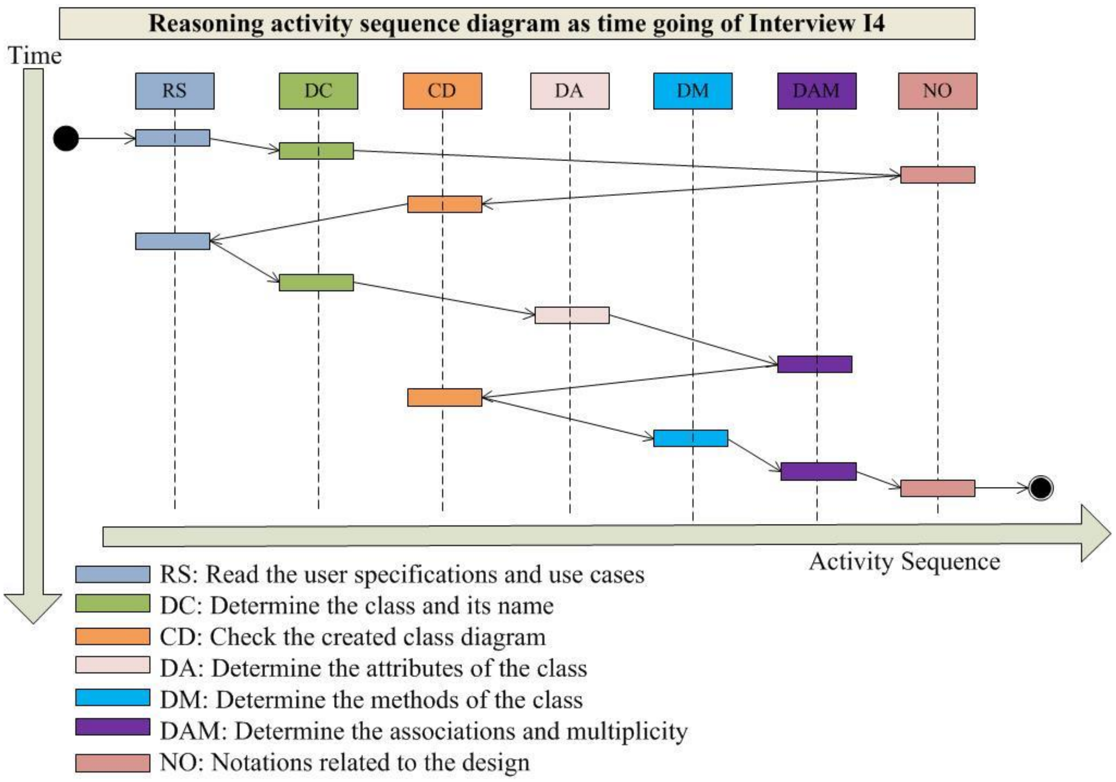
DM: Mm… I put the operations. Make reservation, cancel reservation, view tour information and update.

DAM: Mm...I figure out the association name and its multiplicity. (Finish the names and multiplicity)

NO: Actually when I am drawing the initial class diagram, I don’t want to think more operations and attributes because it is too early.

Mm… I think it’s OK.

Reasoning activity diagram



Reasoning activity sequence diagram as time going of Interview I4
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	***Start Time***	***Finish Time***	***Duration***
RS	**Before starting, always reading requirements**		**5min**
DC	[00:00:00]	[00:00:33]	**33s**
NO	[00:00:33]	[00:01:30]	**57s**
CD	[00:01:30]	[00:10:12]	**8m42s**
RS	[00:10:12]	[00:10:52]	**40s**
DC	[00:10:52]	[00:16:32]	**5m40s**
DA	[00:16:32]	[00:17:56]	**1m24s**
DAM	[00:17:56]	[00:18:18]	**22s**
CD	[00:18:18]	[00:20:18]	**2m**
DM	[00:20:18]	[00:23:35]	**3m17s**
DAM	[00:23:35]	[00:26:30]	**2m55s**
NO	[00:26:30]	[00:27:51]	**1m21s**
RS		**5m40s**	
DC		**6m13s**	
CD		**10m42s**	
DA		**1m24s**	
DM		**3m17s**	
DAM		**3m17s**	
NO		**2m18s**	
Total Time		**32m51s**	

Addition:
1. The timing distributes is based on the interview video record.
2. The video records from the beginning of making design of the interview.
3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:
-What reasoning did you use for deciding what should be classes in your diagram?
I try to think about the domain and read the descriptions to find what the most important stuff is in them. In the description, I suppose to do a tour reservation system. So, I guess the biggest things included of course are the tour and the reservation. It is said that I try to find key words in descriptions what is important. Then, I could find about the classes and I am also thinking about the things that could be stand alone. If the things are too primitive, I would treat them as attributes. If the things are too stand alone, I would put them as classes. Unfortunately, there is no examples related to my views in this class diagram but it is my considerations about attributes and classes.

-What reason did you use for determining attributes and methods of the classes?
When come to the methods, I just take a look at the use case model of which the users want to do in the system. As far as I'm concerned, people can identify the actions by identifying the verbs, like we have the verb of reserve and then it should be an action. I think this is the first step to identify the actions by verbs but later, in terms of details, I need to consider more.
When come to the attributes, as mentioned before, if the name is too primitive, it should be attributes. Because I think the name of attributes cannot contain more information, the name is just a name. But, like the customer, it can contain names, person number and … As a result, we know the customer is a class or entity rather than the simple attribute's name.
-What reasoning did you use for deciding associations between classes?
When come to the associations, I am trying to figure out how would look like in the real life. Such as the reservation, I know the reservation should be connected to the customer. It is easy to know that the customers make reservations such that there should be a connection between them. I think the things are based on the domain model that actually the knowledge about how it works in the real life or in the situations.
My modeling right now is both based on the descriptions and on my own experience. If you ask me tomorrow, it must be totally different in the class diagram. I think that can also explain why people have lots of different designs because of your own interpretations about the domain. This is what I think about the domain, so it should be like this.
-Did you think of any design principles while designing the class diagram?
Of course, I did. As presented in my class diagram, I did user and inherent things. I don't want to duplicate the information for both customer and employee because both of them contain the same attributes. I am going to use the object-oriented design principles. In these principles, you want to avoid the duplications and you want to have cohesiveness which means that in class diagram, the one just is responsible to do one things and noting else.
-What's your naming strategy? i.e. how did you think of the names of your class diagram?
When come to the names, of course it should be based on the domain and in this case, the domain knowledge is in the description so I take the names from the descriptions. And one other thing, in the description, there is a complaint or suggestion. I was thinking just to use a name of review to cover both of them. Certainly, the name of review is also based on the domain. In the real life, you can review the information at the travel agency system. In the situation, of course you can advice to the agency or you can complain the service of the agency.   In a word, the name strategy is mixed from my own understanding of the domain plus the user descriptions.

-Common steps on how to create the class diagram

1. Identify main entities based on description and own knowledge about the domain
2. Identify and specify relationships such as associations between entities
3. Identify attributes and actions on classes
4. Repeat 1-3 steps and refine

- Always think about principles such as
    - Reusability
    - Cohesiveness
    - Other OO (object-oriented) principles
- Not think about actions that early

5. Interview 5

Interview dialogue

RS: [Read the user requirements and make some sketches before drawing the class diagram on the paper]

DC: As I understand this user requirement, I will draw three different classes. One is customer, one is the employee and one is the system.

DAM: (Connect the customer and the system, employee and the system)

DM: The customer can use the system to view information, cancel the reservation and make reservations.

DA: The customer should have the id as well as the employee.

RS: [Read the user requirements and have a look at the example]

DC: It should have the class of password connected to the employee.

DM: The employees also can add/delete/update.

RS: [Read the user requirements]

DA: Mm… The system also has the tour id and displays the information.

CD: [Check the created class diagram with the user requirements] Yes, the customer can access to the system and the employees can give to the system.

DC, DM: Mm… I just create a class of administrate which containing the method of adding, deleting and updating.

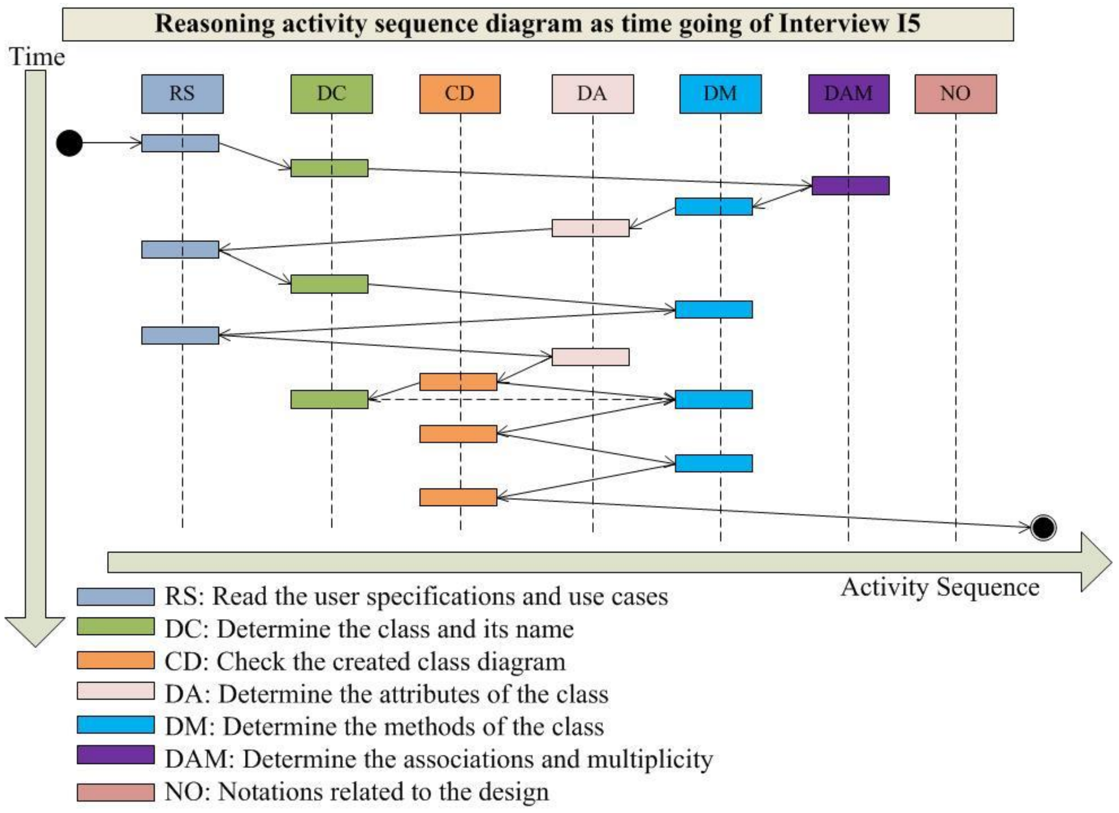
CD: Mm… The employee can do the administration for affecting the system. Yes, that's it. [Check the created class diagram]

DM: Ok, I add the method to the system called as complain db.

CD: [Finally check the created class diagram]

Mm… I think it's done.

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview I5
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	*Start Time*	*Finish Time*	*Duration*
RS	**Before starting, always reading requirements**		**5min**
DC	[00:00:00]	[00:00:32]	**32s**
DAM	[00:00:32]	[00:00:52]	**20s**
DM	[00:00:52]	[00:01:34]	**42s**
DA	[00:01:34]	[00:01:40]	**6s**
RS	[00:01:40]	[00:04:35]	**2m55s**
DC	[00:04:35]	[00:04:54]	**19s**
DM	[00:04:54]	[00:05:34]	**40s**
RS	[00:05:34]	[00:06:03]	**29s**
DA	[00:06:03]	[00:07:50]	**1m47s**
CD	[00:07:50]	[00:09:24]	**1m34s**
DC, DM	[00:09:24]	[00:11:00]	**1m36s**
CD	[00:11:00]	[00:12:39]	**1m39s**
DM	[00:12:39]	[00:13:39]	**1m**
CD	[00:13:39]	[00:15:09]	**1m30s**
	RS	**8m24s**	
	DC	**57s**	
	CD	**4m43s**	
	DA	**1m53s**	
	DM	**3m52s**	
	DAM	**20s**	
	Total Time	**20m9s**	

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:

-What reasoning did you use for deciding what should be classes in your diagram?

To come up with the classes, I read the user requirements and then decide the three classes – customer, employee and system. Additionally, by viewing the use case model, it shows the clear interactions between the actors and the system. I get the information from both of them. Later, I come up with the additional classes via reading the specifications again to make the class diagram visible to everyone.

-What reason did you use for determining attributes and methods of the classes?

I get the hints from the user requirements to decide what they (the classes) should have and what they should do.

-What reasoning did you use for deciding associations between classes?

To decide the relationship in my class diagram, it mainly based on the use case model. In the use case model, we know the actors – employee and customer can do different actions and as a result, I will decide the links between the different classes. In addition, I am also inspired from my own experience of daily life.

-Did you think of any design principles while designing the class diagram?

<u>No, I don’t have any design principles.</u>

-What’s your naming strategy? i.e. how did you think of the names of your class diagram?

OK, In terms of the names, some names in my class diagram, I take them directly from the user requirements. And the other names are based on the use case model and my general thoughts from my experience.

-***Common steps on how to create the class diagram***

1. Read the specifications or use case model
2. Understand them
3. Put into blocks (create classes)
4. Read the specifications or use case model again
5. Add methods/attributes/associations
6. Finalize the class diagram

6. Interview 6

Interview dialogue

RS: [Read the user requirements]

CD: (Make the detailed sketches on the paper as the foundation to draw the initial class diagram; take a reference of the example many times during making sketches) [Check the created sketches]

DC: Let's start. From the requirements of the introduction, we can get a lot of information about this travel agency system. First, to design the class diagram, I will decide the classes of this software application. I decide the three classes which are customer, software application and employee.

DA: I just start with the customer and set the son attributes like id, name and age.

DM: For the method, I just create the methods related to the attributes, such as set id/name/age and get id/name/age.

DAM: Next class is the software application. For the relationship between the customer and the software application, I think several customers can only use one software application.

DA: In terms of the attributes of the software application, I just set the path name. I think in the software application, there are lots of paths between destinations and the path can be chosen by the customers. Certainly, I also set the departure city, arrival city and departure time, arrival time as the attributes.

DM: Since the customer can make a reservation, cancel the reservation and view tour information, I add the corresponding methods in the class. I forgot one thing which the customer can make complaints to the travel agency for letting it get some feedbacks from the customers. So, I also add the method called complains.

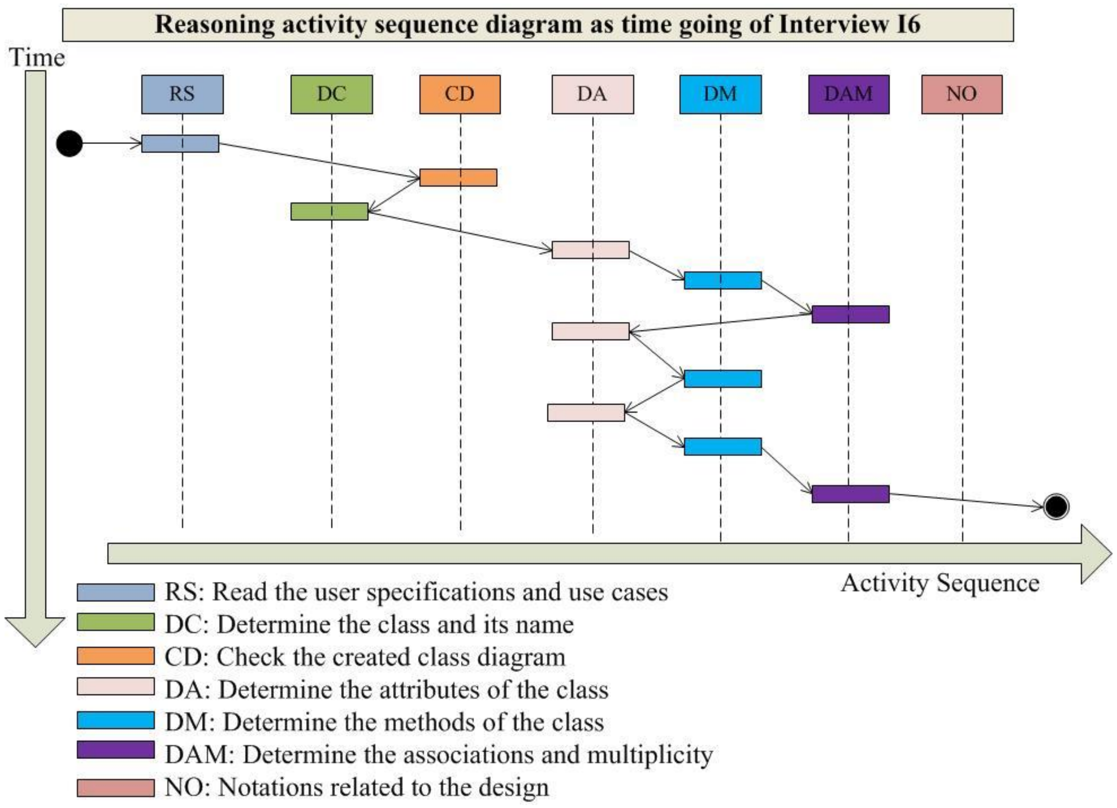
DA: The third class is the employee. For considering the security concerns, I need to set the attributes of login ID and password.

DM: For the methods, I just create the corresponding actions related to the attributes, like set login ID/password and get login ID/password.

DAM: And I think several or one employee can administrate one software application and one software application can be used by more than one employee.

Mm… that's pretty much and I think it is done.

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview I6
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	***Start Time***	***Finish Time***	***Duration***
RS	**Before starting, always reading requirements**		**5min**
CD	**The time is not recorded in the video**		**6min**
DC	[00:00:00]	[00:00:55]	**55s**
DA	[00:00:55]	[00:01:30]	**35s**
DM	[00:01:30]	[00:02:18]	**48s**
DAM	[00:02:18]	[00:03:46]	**1m28s**
DA	[00:03:46]	[00:04:55]	**1m9s**
DM	[00:04:55]	[00:06:23]	**1m28s**
DA	[00:06:23]	[00:07:27]	**1m4s**
DM	[00:07:27]	[00:08:02]	**35s**
DAM	[00:08:02]	[00:09:40]	**1m38s**
RS		**5m**	
DC		**55s**	
CD		**6m**	
DA		**2m48s**	
DM		**2m51s**	
DAM		**3m6s**	
Total Time		**20m40s**	

Addition:
1. The timing distributes is based on the interview video record.
2. The video records from the beginning of making design of the interview.
3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:
-What reasoning did you use for deciding what should be classes in your diagram?
OK. Apparently getting information from the user requirements, I think the class diagram should be consisting of three parts. The first one is customer. The second one is the software application and the third one is the employee. It is easily known that the application should be used by the customers and be administrated by the employees. As a result, I put these three classes into the class diagrams. This is my reasoning to decide the classes during making the design.
Since this initial class diagram is the sketch of UML modeling design, I think the three classed is fairly enough for the first implementation of the software. I would like to add database but I am not sure where and when to add because of lacking some experience of UML design. This is why I just decide only three classes.

-What reason did you use for determining attributes and methods of the classes?
OK. Let's take an example of employee. I set the login ID and password as the attributes because of the safety concerns. Not each one has the authority of using the application in the travel agency, only the particular staffs have the right to administrate the application. For each attribute, I can come up with the corresponding or related methods. This is my thought.

-What reasoning did you use for deciding associations between classes?
As the information I can get from the user requirements, I can say that there is only one software application running in the travel agency. But lots of customers want to use this application. That is why I connected the customer and the software application. The customers can use the software application to make reservation and something needed. It is the similar to the relationship between the employee and the software application. This is what I thought.

-Did you think of any design principles while designing the class diagram?
To think of the design principles, in my opinion, the most important point at beginning is to make the class diagram reasonable. I think the class diagram is the sketch of software design, we should guarantee the software which can be implemented. And we also need to make it as simple as possible to let people who have experience about software design or not both understand the class diagram. This is what I can come up with.

-What's your naming strategy? i.e. how did you think of the names of your class diagram?
OK. First we need to make the names as easy as possible and also try to make it reasonable. For example, the name of customer is easily identified by the software developer and we cannot make the names in the class diagram too complicated. If complicated, it will bring to many troubles in the further software development.

-Common steps on how to create the class diagram
- First what I can get from the user requirements at the first sight are customers, software application itself and the employee.
- Set the attributes and methods to each class in the diagram
- Set the multiplicity and association between the classes as well as the association name

7. Interview 7

Interview dialogue

RS: [Read the user requirements]

DC, DA: I would like to start with the deployment diagram. First of all, I have the user which containing the attributes of login ID and password. <u>(The user equals the customer)</u>

DC: And I should have the reservation.

DM: In the reservation, it could make reservations and cancel the reservations as said in the description.

DA: For attributes, maybe it has information about reservations and extras.

DAM: So, one customer can make zero or several reservations.

RS: [Read the user requirements]

DC, DA: Then, I just create the tour which including the attributes of date/time and place, maybe.

DM, DAM: It has the method of getting tour information and one tour has zero or several reservations. OK.

RS: [Read the user requirements]

DC: Complaint.

DM: The user can make complaints.

DA: It may have the attributes of the text which presenting the descriptions of this complaints and the subject.

DAM: One user can also make zero or several complaints.

CD: [Check the created class diagram]

NO: Mm… As inspired from doing android projects currently, I think we should have databases about the complaints as well as the tour and the user.

RS: [Read the user requirements]

DC: Employee.

DA: For employee, it has the login id and password as well.

DM: And the employees can add/delete/update to administrate the users and tours.

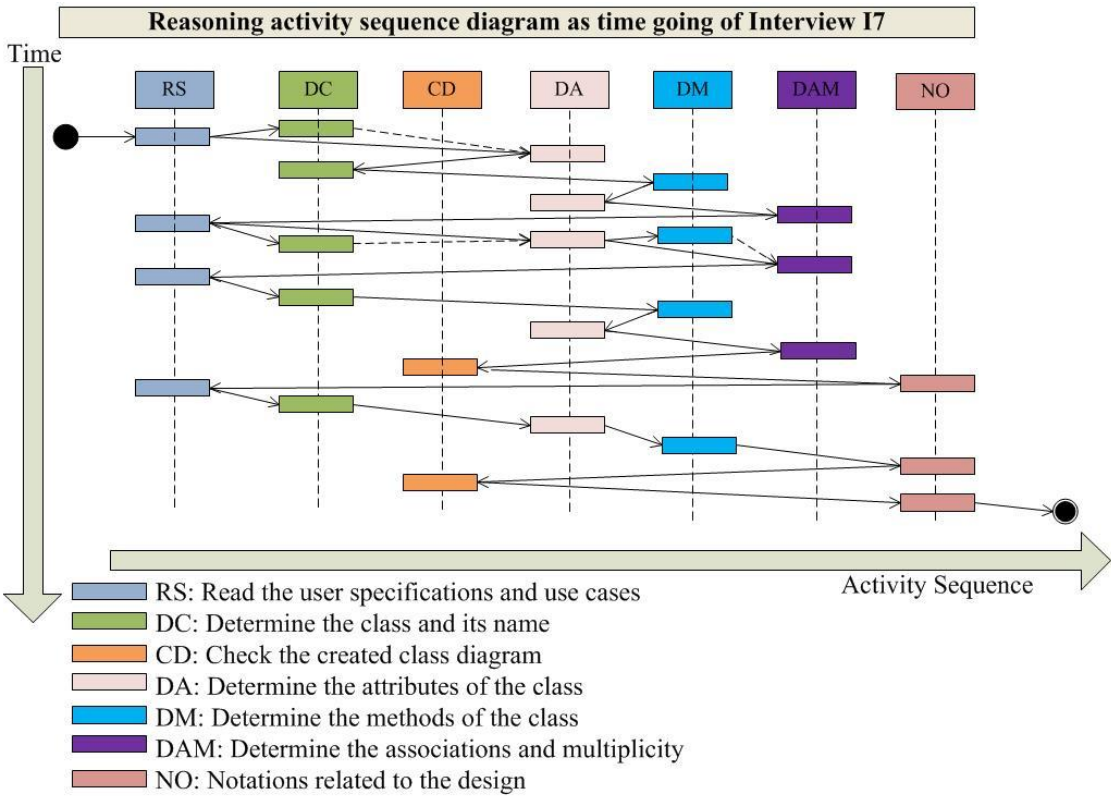
NO: I think the employee can administrate the relevant databases directly via like database adapter. They can manage the tours, complaints and users' databases. Yes, I draw it in this way.

CD: [Check the created class diagram and think for a while]

NO: I am not sure whether I have done or not. I am doing android currently and that's why I doing like this way. (Check the user requirements and be asked whether to finish the initial class diagram) Yeah…I am not very happy with this design and as said, I am doing this way instead. (Draw a new complementary class diagram with MVC view) OK. I just create tour-controller and the user interface connected the controller named tour-view. And I have tour class which containing tour information also connected controller. Of course, the tour class need to be connected the database which storing the tour data. The user has the user-controller and the user interface properly. Maybe it's not UI but it is kind of contact between them. The user-controller needs to be connected to the tour-controller such as the users want to view the tour information. The user wants to make complaints such that the user-controller will be

connected with the complain-controller, for example. The complain-controller is connected to the complain class and the complain class is connected to the database. Mm… I am thinking like this because of the android development. (Explain the reasons why drawing the class diagram like this with the android knowledge and thoughts)
OK. I think I am done.

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview I7
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

There is no reasoning activity timing distributes on interview 7 because the digital camera was kind of broken at interview time.

Question time:

-What reasoning did you use for deciding what should be classes in your diagram?

Well. The simple way to do that is taking the nouns. I mean the key words in the user descriptions, such as the tour, reservation, user, employee, complain and something like that. You can see that the customer should be a class and the tour should be a class, for example.

-What reason did you use for determining attributes and methods of the classes?

OK. The attributes should be… [Check the user requirements] Le’s what we have in the descriptions. It doesn’t provide much information but if we see the use case model, for example, the tour should be reserved and it should be the method of the class tour but there should be a different class for reservation as well. Cancel the reservation and view tour information also should be the methods. Let’s back to the descriptions to see the verbs like add/delete/update and they should be methods as well. For attributes, [Check the user requirements] it doesn’t mention too much about that. The attributes are also nouns connected to the classes. Right?! For example, login ID and password connected to the class employee are attributes. [How to derive the attributes not mentioned clearly in the user requirements] Mm… For example, I set date/time and place as attributes for the class tour. I am thinking what information you should get from the tour and the tour like the journey, you need to know the place, you need to know the flight and you need to know the price properly. So, if I don’t find the clear information about attributes in the user requirements, the others are derived based on my domain knowledge and experience.

-What reasoning did you use for deciding associations between classes?

My reasoning was properly here and it was thinking about how the user interface would work and how the data would be stored. I was thinking from the android perspective. Let’s see here. For example, the tour connects to the reservation and one tour can have many reservations because many people can book the single tour, like the tour to Dubai. And one user can make many reservations but the reservation is only to be identified by one user. It is similar to the association between user and complaint. One-to-many association works in my design.

-Did you think of any design principles while designing the class diagram?

Not particular. I was just thinking about the stand-alone android application. Also, I was thinking about the MVC and the way of implementing on the android.

-What’s your naming strategy? i.e. how did you think of the names of your class diagram?

Well. The name should properly be simple and I was thinking the names should be one-single meaning. For example, the tour is the tour. We need to use exact and simple names and not to confuse people. The class names are always nouns and the attributes name are mostly nouns, or maybe adjective. The method names are verbs and nouns, like get information. This is what I was thinking and they should be as simple as possible, but exact.

-Common steps on how to create the class diagram

- Read the user requirements
- Highlight keywords, i.e. nouns for class names and attribute names, verbs for methods.
- Draw initial classes and relationships
- Draw the class diagram the way it could be implemented using MVC (Model-View-Controller) design

8. Interview 8

Interview dialogue

RS: [Read the user requirements]

DC, DA: OK. I first create class customer and put the attributes, properly like username and password.

DM: And then we can have functions. I saw in the description he (customer) can make a reservation of tour, view information of tour. So, the class (Customer) may have the functions – register to tour, get information, properly. Mm… it also need a function to resister the username and password, as a result, I just add the method – register account.

DC: OK. Then we have another person class employee.

DA: It also has the attributes of username and password. Mm…And I think the customer should have the attribute of email, as well. (Let's back the class employee) All right. The user name, password. But mostly, I think it also should have the attribute like the administration right. I just put the administration level to this class (Employee).

DM: And it has the functions of administrating users. OK. Add/remove/update user is the methods. Probably employee has the rights of adding tour information. So, add/update/remove tour is also method of this class (Employee).

DAM: One to several employees can administrate on to many customers.

DC: OK. Let's come to the tricky part with the software. So, we have the database. This database for me I think is the only database storing all the relevant information. It has the customer table, employee table and tour table.

CD: [Think for a while and check the created class diagram]

DC: And I just create the class tour.

DA: The tour maybe has a lot of information, such as id, description.

DM: I just forgot one point that the customer also has the method – cancel register.

RS: [Read the user requirements]

DA: I think the tour also has the count-customer which means counting the number of the customers registered on the tour and properly price. Yes, I am not sure… I am just making a super simple one (class diagram).

DAM: (Finish the association including association name and multiplicity between database and other created classes including employee, customer and tour)

RS: Mm… I am not sure that I am gone or in the right direction. [Check the use case model]

DC: Yeah, I need complain. Add complain table to the database. Also, I properly need a class of complain.

DA: This class may have attributes of id, description and tag.

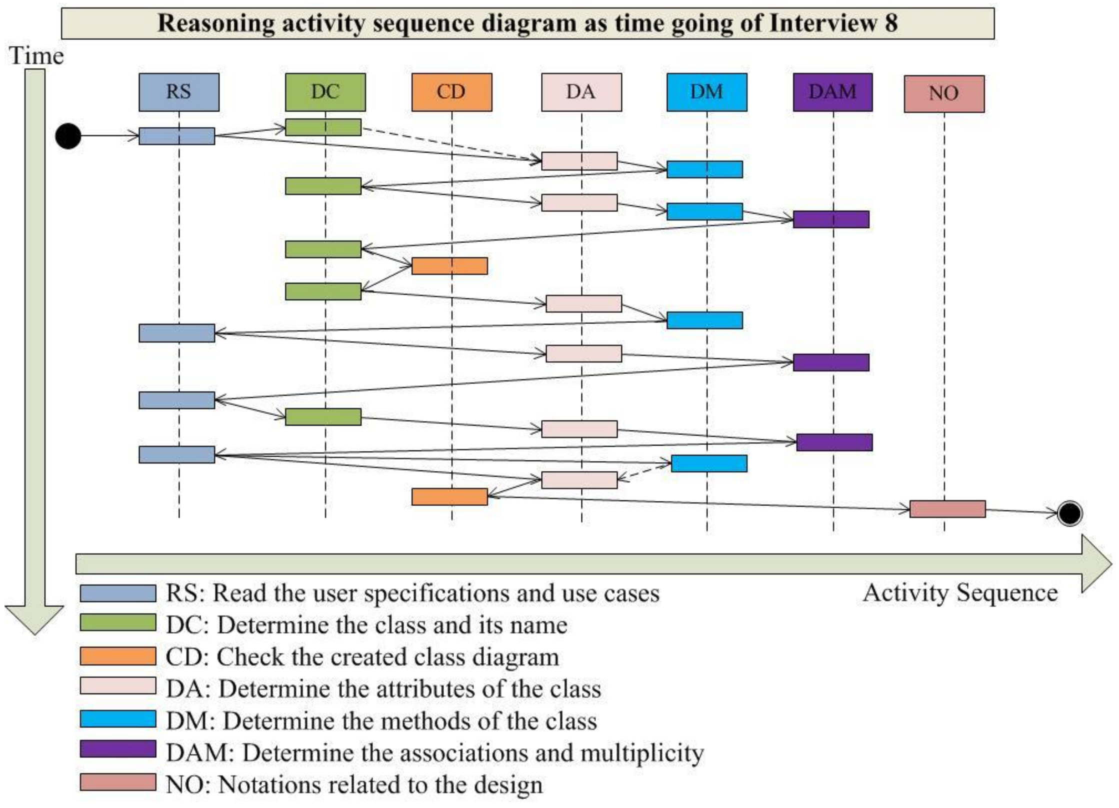
DAM: So, one customer can create from one to many complaints and the complaints can be stored in the database.

RS: [Check the user requirements and use case model] I think I get all the use cases here…

DM, DA: And I add one function to the customer for making complaints… Properly, the customer can view tour information and reserve the tour such that the class tour

should have the attribute of customer id, I am not sure…
CD: [Check the crated class diagram with the requirements and use cases]
Notation: I think I am done but I don't know. I think I have covered all the use cases but… Mm… OK. I just make a simple one, maybe it is wrong.
But, I think it is done.

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview 8
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	*Start Time*	*Finish Time*	*Duration*
RS	**Before starting, always reading requirements**		**5min**
DC， DA	[00:00:00]	[00:00:34]	**34s**
DM	[00:00:34]	[00:01:27]	**53s**
DC	[00:01:27]	[00:01:40]	**13s**
DA	[00:01:40]	[00:02:27]	**47s**
DM	[00:02:27]	[00:03:46]	**1m19s**
DAM	[00:03:46]	[00:04:10]	**24s**
DC	[00:04:10]	[00:05:33]	**1m23s**
CD	[00:05:33]	[00:05:58]	**25s**
DC	[00:05:58]	[00:06:15]	**17s**
DA	[00:06:15]	[00:06:34]	**19s**
DM	[00:06:34]	[00:06:48]	**14s**
RS	[00:06:48]	[00:07:11]	**23s**
DA	[00:07:11]	[00:08:05]	**54s**
DAM	[00:08:05]	[00:10:21]	**2m16s**
RS	[00:10:21]	[00:10:29]	**8s**
DC	[00:10:29]	[00:11:13]	**44s**
DA	[00:11:13]	[00:11:21]	**8s**
DAM	[00:11:21]	[00:12:02]	**41s**
RS	[00:12:02]	[00:12:43]	**41s**
DM， DA	[00:12:43]	[00:14:03]	**1m20s**
CD	[00:14:03]	[00:17:40]	**3m37s**
NO	[00:17:40]	[00:18:23]	**43s**
	RS		**6m12s**
	DC		**2m41s**
	CD		**4m2s**
	DA		**3m8s**
	DM		**3m16s**
	DAM		**3m21s**
	NO		**43s**
	Total Time		**23m23s**

Addition:
1. The timing distributes is based on the interview video record.
2. The video records from the beginning of making design of the interview.
3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:

-What reasoning did you use for deciding what should be classes in your diagram?

Mm… that's actually kind of tricky. I think everything that need to store some information maybe need to become class because it wants to set data and also log, receive data. The action should happen as well because it needs the functions to perform. This is why I created. For example, the Tour class doesn't have the access to the database but it need to store some information like description, price. So, it should be a class. Moreover, I think every table in the database needs a class.

-What reason did you use for determining attributes and methods of the classes?

I don't know maybe just for feeling. I try to put every action the classes need to perform as the methods. I also try to have the relevant data as the attributes and the methods are just like storing, retrieving data. [How to derive attributes not mentioned in the user requirements] I don't know and it just based on my brain reasoning. I think I should have these attributes so I just add. It is kind of my reasoning how I solve the problems and the derived attributes are mostly based on my domain knowledge according to my personal daily life experience.

-What reasoning did you use for deciding associations between classes?

Ok. It is the tricky part. I just think that how does the class connect? Ok. For instance, we have the customer and the employees. How is the relationship between them? The employees can administrate the customers so I create the relationship between them. I just think how the different classes connect for each other and then I create the relation. The name is standing for the relation meaning.

-Did you think of any design principles while designing the class diagram?

I try to do high cohesion, maybe. I try to connect every method to the central class. And also do low coupling. Everything in my class diagram is connected to the database, but it's acceptable. I have experience about database using UML design and I think database is extremely important for me.

-What's your naming strategy? i.e. how did you think of the names of your class diagram?

Actually, I try to go though by the description but I think it is kind of wrong, like the add-user in the employee class. In fact, it should be add-customer instead of user and it is more accurate. I use the names directly from the requirements but for the names not mentioned, I just name it based on my experience in logic.

-***Common steps on how to create the class diagram***

1. Created user classes (Customer + Employee) and their methods and relations
2. Added the database and its tables
3. Created the Tour class and updated other classes
4. Created the complain class and updated other classes

9. Interview 9

Interview dialogue

RS: [Read the user requirements]

NO: Mm… this is a tricky task because there are lots of freedom things here such that I think I have to make quite many design choices. The first question I would like to discuss with the stakeholders is how the customers actually access to the software would be built though web-service or something like that. We could skip that part and just to say it doesn't matter because it's belonging to the interface. We just focus on the core software, I guess. The other thing is that do we have any ideas about the reservation of the tour they like? What information are we going to request to the user? Normally, in the use case, you write down the general sequence things and very verb description of different steps. We have a lot of choices to make of each use case. Normally, I know what people usually do is to read the descriptions and pick out all the objects you find. Without thinking about the database, we want to identify the entities which the software needs to be related to. So, we can make a reservation, cancel a tour, view tour information, add complaints and the employees can administrate it. Mm… the reason I am quiet is that I am thinking do I want to model actual software piece and the data structure which the information that software handle? For me, what I normally do is to make problem-domain model.

DC: So, I have customer first to take care of. Then, I have the tour.

DAM: One customer can have many tours and one tour can be associated to many customers as well.

RS: [Read the user requirements]

DC: OK. Perhaps it is the reservation rather than tour. This is the tour. (Change the tour as the reservation and create the new tour)

DAM: The reservation is belonging to one tour and one tour has many reservations. That could be one way to look at it. For the travel agency, it has several and many tours. One tour can be related many reservations and many reservations can be done by one customer. I could connect the customer and tour directly and the reservation can be treated as an association class. It depends on how you want to do it and it would be the same thing.

RS: [Read the user requirements]

DC: Of course, I need to have complaint object taken care of.

DAM: One customer can make several complaints and in the description, it doesn't say anything about the complaints related to the reservations or tours. But, we properly would like to the complaint related to the reservation, I guess. The association between them should be an option thing so I just put the multiplicity 0…1.

RS: [Read the user requirements]

NO: What I do now actually is not the software design and I am trying to identify the things the software manages.

DA: The customer has public attributes id and password. I am not adding the employee here right now because the employee is not the actual system, who will use the system through the software parts and the employee here is just self.

DC: Mm… OK, I understand it. Perhaps we should do this way. I just create the user as abstract class which the customer as well as the employee is both the user here and this is clear.

CD: [Check the created class diagram] I need to draw this again.

DAM: Mm…I think the customer should be directly associated to the tour in order to view tour information.

DM: The tour should also have the method of creating reservation which returns reservation, of course. The reservation should be cancelled.

RS: [Read the user requirements] I really need to read the text over and over again.

DC: Mm… I add the email object and the customer creates it. The email object can create complaints.

NO: All right and I could re-draw it to make it nicer.

DC: In the description, it talks about the complaint database such that I could add this object – complaint database. The complaint database consists of all the complaints made by customers and also I need to have tour database containing all the tours information.

DAM: And the association related to the databases is the composition instead of aggregation. The aggregation means kind of logic ownership but for the composition, it means like physical ownership.

NO: OK. We have nine classes now. I will change some relationship as my understanding of the problems going on.

DAM: Mm… the customer could access the tour database directly to view tour information and the association between the customer and the tour is kind of wrong such that I remove it. I think the tour is related to the reservation and the reservation is related to the customer. The association should be like this way, not like the tour self connected to the customer, I am quite sure about it.

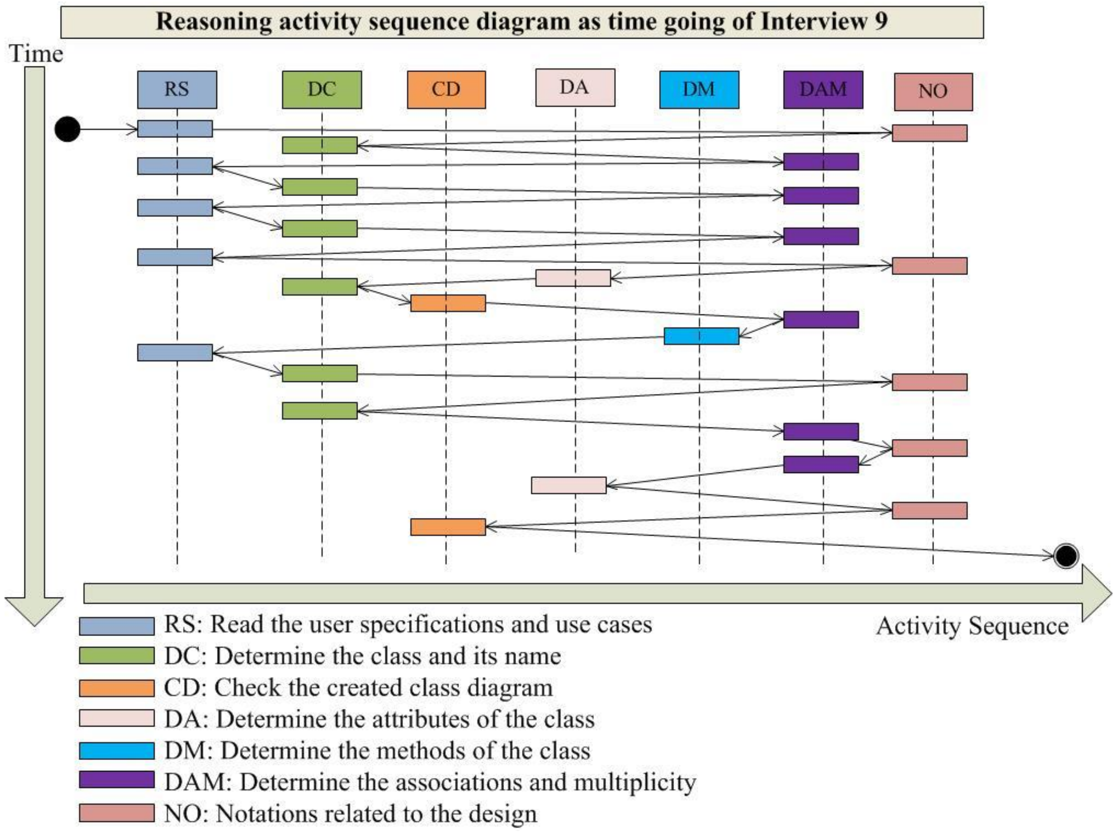
DA: The reservation class should have the negotiation-price and advanced-payment as the attributes.

NO: Mm… the implementation of this system should be web-based interface in order that the customer can access to the information about tours via it. Mm… I mean this is the initial class diagram for me to understand the actual problems. This is my sketch. The next step is to add software components to identify how we do now. The description doesn't say anything about how to do. I cannot do the software design now because I am lack of constraints and non-functional requirements. What I can get now is just functional requirements and the actual implementation of the system should be based on the non-functional requirements. From the user requirements and the use cases, I just can do this.

CD: [Check the created class diagram]

All right, I think this is my initial one. (Re-draw the initial class diagram)

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview 9
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	***Start Time***	***Finish Time***	***Duration***
RS	**Before starting, always reading requirements**		**5min**
NO	[00:00:00]	[00:04:20]	**4m20s**
DC	[00:04:20]	[00:04:55]	**35s**
DAM	[00:04:55]	[00:05:25]	**30s**
RS	[00:05:25]	[00:06:50]	**1m25s**
DC	[00:06:50]	[00:07:07]	**17s**
DAM	[00:07:07]	[00:09:00]	**1m53s**
RS	[00:09:00]	[00:09:19]	**19s**
DC	[00:09:19]	[00:09:27]	**8s**
DAM	[00:09:27]	[00:10:42]	**1m15s**
RS	[00:10:42]	[00:11:54]	**1m12s**
NO	[00:11:54]	[00:12:14]	**20s**
DA	[00:12:14]	[00:12:54]	**1m**
DC	[00:12:54]	[00:13:36]	**42s**
CD	[00:13:36]	[00:14:40]	**1m4s**
DAM	[00:14:40]	[00:15:23]	**43s**
DM	[00:15:23]	[00:16:12]	**49s**
RS	[00:16:12]	[00:16:49]	**37s**
DC	[00:16:49]	[00:17:39]	**50s**
NO	[00:17:39]	[00:17:47]	**8s**
DC	[00:17:47]	[00:18:21]	**42s**
DAM	[00:18:21]	[00:19:06]	**45s**
NO	[00:19:06]	[00:19:38]	**32s**
DAM	[00:19:38]	[00:21:03]	**1m57s**
DA	[00:21:03]	[00:21:48]	**45s**
NO	[00:21:48]	[00:25:04]	**3m16s**
CD	[00:25:04]	[00:28:04]	**3m**
	RS		**8m33s**
	DC		**3m6s**
	CD		**4m4s**
	DA		**1m25s**
	DM		**49s**
	DAM		**6m31s**
	NO		**8m36s**
	Total Time		**33m4s**

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user

requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:

-What reasoning did you use for deciding what should be classes in your diagram?

Well. I just try to think about what 'things' the software system will need to take care about and what 'things' we have to take care about. The 'things' are not the things we can always touch, such as the reservation is not the physical object but the user is. However, the reservation is the information that is separated from the user and the tour and it is the important and different thing the system needs to take care about.

-What reason did you use for determining attributes and methods of the classes?

The attributes and the methods are directly from reading the description of the problem. When we work on this example in the real world, of course we will add more. Let's say the tour for instance, the tour properly has location, price and date perhaps, I think that. That is the information about the tour. For the reservation, you have other information, like negotiated-price, payment and something similar. Perhaps you also should have payment tool to realize the payment but the description doesn't mention it. Maybe there also need to have transaction. The attributes will be added more while we are realizing the system. For the methods, I think it's kind of tricky when just speaking about the class diagram here. It is easy to talk about the methods when talking about the software entities handling something. I mean what we haven't done here is to state the software modules actually executes the objects. You properly want to have methods for them and that could be normally easy. For this case, I couldn't find many methods right now but I am pretty sure I will add lots of methods around the class diagram if I can spend more time to think about it. That would be the design for next week when we realize the system but not for now.

-What reasoning did you use for deciding associations between classes?

Actually, that should be together. When you think about the 'things', you also need to think about the relations at the same time. It is not different problem things and they go together. For example, if I change a set of classes here, of course I will change the corresponding associations. So, they go hand by hand. When you do software design, my experience is that where you need to change the definition of the classes, because you have unwanted associations. You can have a pattern for instance. You want to reduce the complexity and you don't want to have the things knowing too much about the other things, also you want to reduce the dependency for the other classes. That will affect which class you defined and which dependency you have between them. That is hard work and not easy to do well. That is essence of system engineering part. In a word, the classes and the associations go hand by hand.

-Did you think of any design principles while designing the class diagram?

No. Because I actually didn't design software system here and from the functional requirements here, I just want to identify what things the software should handle and how those things relate for each other. The next design different form this initial class diagram, that would be the software design which depends on the other constraints I need. What kind of software system do you want? What platform is going to build? This (the created class diagram) thing is the first thing I will do. Without this one, there is no ides of talking about the software design.

-What's your naming strategy? i.e. how did you think of the names of your class diagram?

OK. That's interesting. The class should be the name what is about and the method is to say what is doing. The attribute is the name what data or property is about. I am not sure but I am thinking about that. The naming is important… [How to name derived names not mentioned in the descriptions] Do I have these names? I don't think I have derived names, really… It is very important. If you look at the names, you should never confuse the intention of this class. When presenting the information about your design, you seldom present the whole picture but just pieces of. The name should be as good as possible. When I say the user, everyone doesn't just only think about the customers but also can think of the employees as well. From my experience, the naming will be refactoring, refactoring, refactoring. I will replace the names if understanding the wrong names and also try to have the most suitable names in the design.

-Common steps on how to create the class diagram

1. Read the specification thoroughly
2. Re-read the specification
3. Identify 'things' and their relations based on the description
4. Benchmark outcome of 3 against the description
5. Update class diagram

10. Interview 10

Interview dialogue

RS: [Read the user requirements]

NO: I try to go through the text to see how the use case related to the text self. Also, I go through the text to see any implicitly information mentioned I suppose to do. In the example class diagram, the security domain and joke domain are into the same class but for me, I will put the security domain, like the login id and password, as a separated class.

DC: So, I start the class diagram – travel agency. I have got customer and employee. I have tour as a class. I have reservation as a class. The complain database? The complain database is neither for reservation nor tour. It is not clear who suppose to do complain in the use case model. Mm… it is difficult to say any information that can be shared between customers and employees. OK. I just create them differently instead of inheritance because they don't have information in common. (Focus on the reservation class)

DA: So, I am not going to put any attributes inside because of not mentioned in the descriptions. I could guess some attributes, such as the date, the price but I will leave out, as well as the tour class.

DM: I am going to put the public operation called view reservation. It is not mentioned how to fetch the reservation, maybe reservation id or customer id, such that I just use dot at the moment.

DAM: View reservation can be accessed to both the customer and employees. The customer can access to the reservation. The reservation is not clear, (Not mention too much details) in my case, I would put a limit and every reservation owns to only one customer who is responsible to it. I just guess it because of not mentioned in the description. I make this assumption.

DA: The employee has the attributes of password, login ID and access level.

DM: For the methods, it can set password and so on. I am just combining the employee domain and the security domain. The employee could add/delete/update the information on the customer and the tour but I really don't know what information they have.

DAM: As a result, I just put a generic association named manage. Several customers can be managed by several employees but I am not sure the multiplicity, I would discuss with the requirement engineers to know the details. The employee could also manage tour and the tour should be managed by at least one employee. The multiplicity could be one to several. The tour should belong to the reservation and several reservations can include one to several tours. And also there can be an association class between them to handle the date of the reservation and tour, but I just leave out.

CD: [Check the created class diagram]

DAM: (Change the association name from manage to administrate)

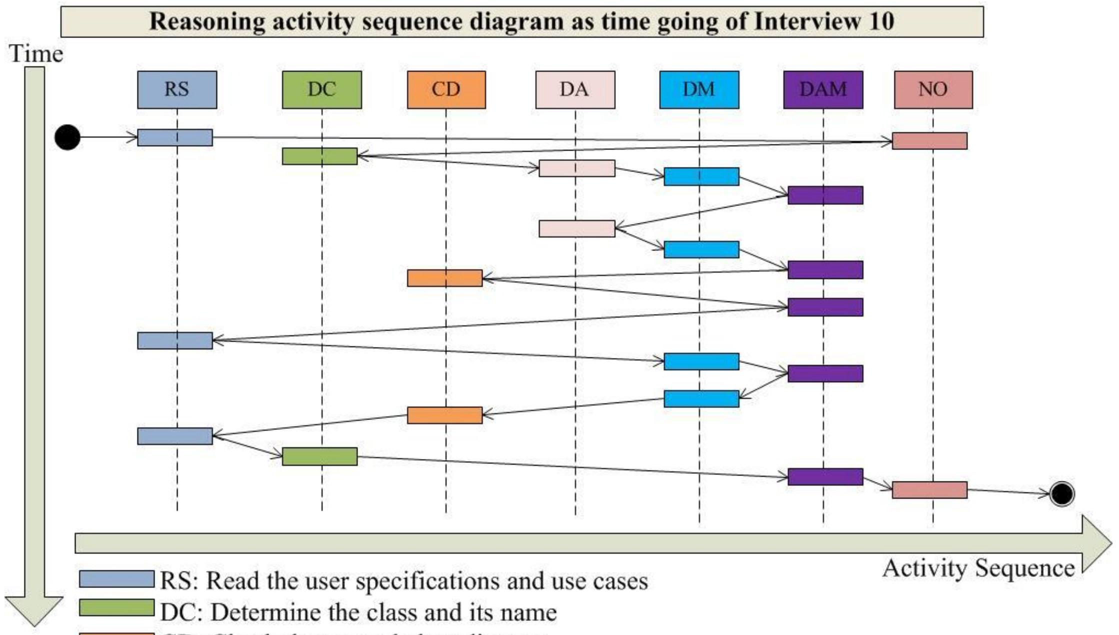
RS: [Read the user requirements]

DM: (Add operations of making reservation to the reservation class)

DAM: (Connect the employee and reservation class, association name: access |

multiplicity: many to many) (Connect the employee and reservation class second association, role name: responsible | multiplicity: one to many) (Draw the responsibility part on another paper)
DM: (Add operation of canceling reservation to the reservation class)
CD: [Check the created class diagram]
RS: [Read the user requirements]
DC: I would like to have the customer facade and several customer facades could manage at most one customer. And there should have a class of complaint.
DAM: One customer can make several complaints. (Remove three associations)
NO: OK. I think that's I can go with the information I have so far. I would like to discuss with the people who are responsible for the text and use case model more and more before I can do something more.

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview 10
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence

🟦 RS: Read the user specifications and use cases
🟩 DC: Determine the class and its name
🟧 CD: Check the created class diagram
⬜ DA: Determine the attributes of the class
🟦 DM: Determine the methods of the class
🟪 DAM: Determine the associations and multiplicity
🟥 NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	***Start Time***	***Finish Time***	***Duration***
RS	**Before starting, always reading requirements**		**5min**
NO	[00:00:00]	[00:02:41]	**2m41s**
DC	[00:02:41]	[00:05:02]	**2m21s**
DA	[00:05:02]	[00:05:26]	**24s**
DM	[00:05:26]	[00:06:18]	**52s**
DAM	[00:06:18]	[00:08:40]	**2m22s**
DA	[00:08:40]	[00:10:21]	**1m41s**
DM	[00:10:21]	[00:11:28]	**1m7s**
DAM	[00:11:28]	[00:14:55]	**3m27s**
CD	[00:14:55]	[00:16:40]	**1m45s**
DAM	[00:16:40]	[00:16:58]	**18s**
RS	[00:16:58]	[00:18:28]	**1m30s**
DM	[00:18:28]	[00:19:12]	**44s**
DAM	[00:19:12]	[00:25:12]	**6m**
DM	[00:25:12]	[00:25:22]	**10s**
CD	[00:25:22]	[00:27:42]	**2m20s**
RS	[00:27:42]	[00:28:12]	**30s**
DC	[00:28:12]	[00:29:00]	**48s**
DAM	[00:29:00]	[00:30:09]	**1m9s**
NO	[00:30:09]	[00:32:20]	**2m11s**
	RS	**7m**	
	DC	**3m9s**	
	CD	**4m5s**	
	DA	**2m5s**	
	DM	**2m53s**	
	DAM	**13m16s**	
	NO	**4m52s**	
	Total Time	**37m20s**	

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:

-What reasoning did you use for deciding what should be classes in your diagram?

Mm… for this case, everything going to be complex is the class. The customer is the class because it will store lots of information, even though I leave it out at the moment. It can have lots of attributes and that' why it is a class. It is the same as the tour, employee, compliant and the reservation. The customer façade is the class because it is going to be mapping to the interface. In a word, facade becomes the class just because of the interfaces and the others are for storing large amount of data.

-What reason did you use for determining attributes and methods of the classes?

I just take it from the text. In the descriptions, it mentions somewhere… (Add the operation of viewing tour to the tour class; Connect the customer and tour class) Anyway, I am looking into verbs in the text which describing different nouns do. View tour information, make reservation and cancel reservation is easy to be known as the methods in the relevant classes. For attributes, I don't find enough information in the text except for the employees. The employee should have login id and password as mentioned in the text but I also add the access level based on my domain knowledge.

-What reasoning did you use for deciding associations between classes?

Mm…I would say that the association would be the verb which taking subject and object in general. Sometime, go through the text look at the verb taking the subject and object. For example, the customer can access to the reservations such that the access is the typical association. On the other hand, the role name sometimes is more interesting than the association name in order that I skip the association name just focusing on the role name. (Look at the responsible as the role name in the class diagram)

-Did you think of any design principles while designing the class diagram?

Not so much. Actually, low coupling and high cohesion are desirable. But in this class diagram, there are only a few classes and I put little data into them such that high cohesion becomes shaky. I would say that the other principle I follow is the users of the system want to access to it by different interfaces. I want different interfaces, so I want facades to handle them. In this case, the customer and the employee should have own role facade to access the system. And also, I would like to have the controllers to handle different classed which storing quite lots of data.

-What's your naming strategy? i.e. how did you think of the names of your class diagram?

Mm… In general, I want the classes to be nouns, the associations to be verbs, the attributes to be nouns and the operation to be verb phrases, such as set password. Some names I can take from the text directly but some names need to be mixed with the identification of the concepts and the responsibility of the concepts, like customer facade or employee controller. It also means that some names are mixed by the descriptions of the requirements and the design pattern I would use. I use facade and controller such that I would have the similar names.

-Common steps on how to create the class diagram

I iteratively go through the analysis to identify concepts and associations. These are then structured and given multiplicities. I add information about attributes, operations and roles if they are present. Depending on the size of the system and the possibility of introducing new classes or associations, I refactor according to a few design rules:

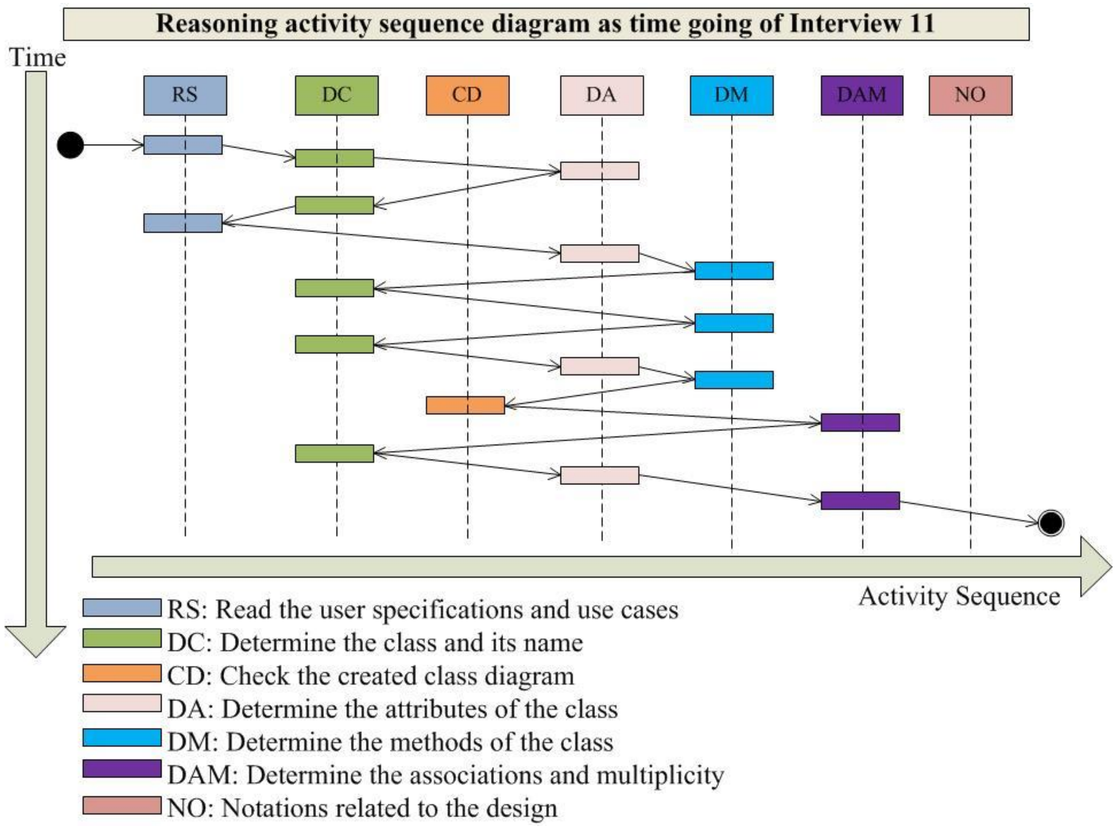
- **LOW COUPLING**
- **HIGH COHESION**
- **FACADES FOR ACTORS**
- **CONTROLLERS FOR LARGE DATA**

11. Interview 11

Interview dialogue

RS: [Read the user requirements]
DC: The first class I think is the admin,
DA: which containing the attributes of user name and password.
DC: Then, it should have the class of the tour.
RS: [Read the specification]
DA: The class of tour has the attributes of reservation id or something like that and tour information.
DM: In terms of the method, it has getting information.
DC: The next class is the reservation.
DM: It has the operations of reserve and cancel.
DC: And another class is customer-info,
DA: which containing attributes of name and reserve-id.
DM: For method, it has the operation of add and update.
CD: [Check the created class diagram and consider the associations between the created classes]
DAM: (Connect the admin and customer-info, customer-info and tour, tour and reservation) (Be asked whether finish it or not) I think I am done. (Not fill in the association name and multiplicity) (The interviewee just names the association and adds the needed multiplicity)
DC: Mm… The admin is my employee. OK. I can just create separate class employee,
DA: which containing name and id as attributes.
DAM: Connect the employee class and admin and change the admin to user as the class name.
(Finish the class diagram)

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview 11
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	***Start Time***	***Finish Time***	***Duration***
RS	**Before starting, always reading requirements**		**5min**
DC	[00:00:00]	[00:00:42]	**42s**
DA	[00:00:42]	[00:01:00]	**18s**
DC	[00:01:00]	[00:01:26]	**26s**
RS	[00:01:26]	[00:01:46]	**20s**
DA	[00:01:46]	[00:02:36]	**50s**
DM	[00:02:36]	[00:03:10]	**34s**
DC	[00:03:10]	[00:03:55]	**45s**
DM	[00:03:55]	[00:04:32]	**37s**
DC	[00:04:32]	[00:04:49]	**17s**
DA	[00:04:49]	[00:05:09]	**20s**
DM	[00:05:09]	[00:05:34]	**25s**
CD	[00:05:34]	[00:06:53]	**1m19s**
DAM	[00:06:53]	[00:12:00]	**5m7s**
DC	[00:12:00]	[00:13:15]	**1m15s**
DA	[00:13:15]	[00:13:21]	**6s**
DAM	[00:13:21]	[00:13:45]	**24s**
	RS	**5m20s**	
	DC	**3m25s**	
	CD	**1m19s**	
	DA	**1m34s**	
	DM	**1m36s**	
	DAM	**5m31s**	
	Total Time	**18m45s**	

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:

-What reasoning did you use for deciding what should be classes in your diagram?

OK. I look at the use case model and I should have customer and employee. What the customer has to do is to reserve a tour, cancel a tour and view tour information. As a result, they can be combined with one class. Because of the behaviors both related to the tour, there is a tour class. The employee could add, delete and update the information on the customers and the tours. For security purpose also mentioned in the description, I have the class of user containing user name and password both for customer and employee. I have the class for tour and to make reservations, I also have the class. The reservation class is the controller class such that customer can reserve and the reservation can get tour information. (Mention all five created classes) That's it.

-What reason did you use for determining attributes and methods of the classes?

It depends on what actions available in the use case model. Like the customer, he can reserve a tour such that I should have the tour-id to reserve, it's available. Cancel a tour means that I should have the method of canceling tours. For viewing tour information, I have the method of get-info to receive the tours' information. For administration, it should have name and password as mentioned in the description. For the customer-info, it should have name and id as attributes at least. Also, it has the operations of add and update. (Mention the attributes and methods of the classes)

-What reasoning did you use for deciding associations between classes?

It depends on the experience of software designers and the domain. For my case, I think the employee is associated with the class user. Each employee has one user name and password as well as the customer, not more. The customer can read the tour and also make reservations. The reservation gets information from tour.

-Did you think of any design principles while designing the class diagram?

Generally, the good design should be low coupling. It is also good to avoid inheritance. I think the most important issue is the low coupling.

-What's your naming strategy? i.e. how did you think of the names of your class diagram?

(Illustrate the reasons why have the class, attributes, methods not for the question) Some names are taken directly from the user requirements and use case model. The others are based on my domain experience of software design.

-***Common steps on how to create the class diagram***

1 – How many classes the diagram should have
2 – What each class contains (attributes and operations)
3 – The relation between classes
4 – Take care about the principles of the design (inheritance and coupling)

12. Interview 12

Interview dialogue

RS: [Read the user requirements]

NO: Well, well, well… this is not very easy because we can reserve a tour, cancel a tour, make reservations and view tour information… The tour can have several reservations, I guess. Several tours can be reserved as the same tour, right? If the tour is full, no reservation can be available for that tour. So, I need an extra paper for sketch.

DC: OK. The first class is the tour.

DAM: And one tour can have several reservations.

DC: The second class is the reservation.

RS: [Read the user requirements; take a look at the example]

DC: It might be user or customer for the next class. The user, right, he can read several tours.

NO: Mm… reserve, cancel, view, this is complicated…

DM: OK. In the tour class, it can reserve a tour, view tour information.

DA: And the tour has a limit number in order that I add the attributes of free seats, maybe.

DAM: The tour has several reservations and the reservation could be canceled by the user.

CD: [Check the created class diagram]

DM: OK, the user can also create complaints.

DAM: One user can make several complaints. (Modify some associations and its names)

DC: Then, there should be employee class.

DAM: The employee could access to the tour for adding/deleting/updating tours. The employee also could read the complaints and have the password like the example.

DA: Employees should be provided a login ID and password by the manager such that it maybe should have login information containing ID and password. And one employee has only one login information.

CD: [Check the created class diagram]

DAM: Maybe the employee could respond to the complaints. Probably, the user can access to tour for doing relevant services but employee can do everything about the tour. The employee could also administrate the reservations as well.

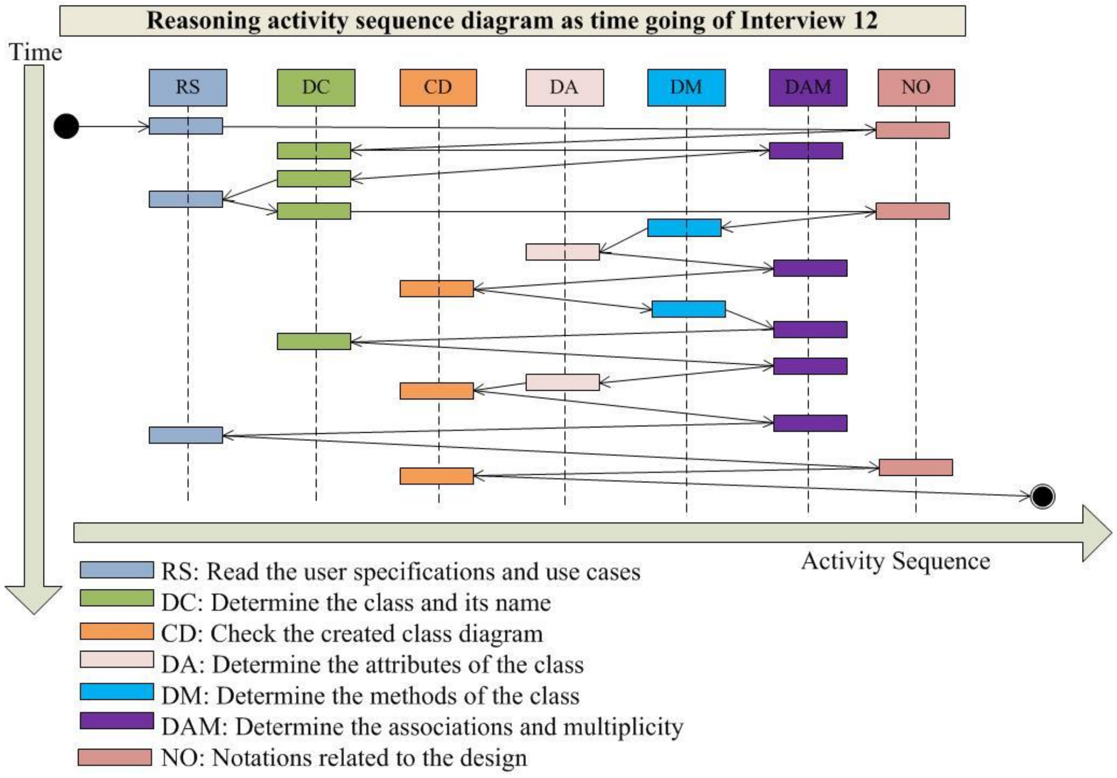
RS: [Read the user requirements]

NO: Maybe the user need to access to the system by login functions and he can create new account to get in by own id and password. But, I am not sure because it is not clear. Maybe it is an open system and it is tricky…

CD: [Check the created class diagram]

I just make this design simple. (Redraw the class diagram based on the sketch)

Reasoning activity diagram



Reasoning activity sequence diagram as time going of Interview 12
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	***Start Time***	***Finish Time***	***Duration***
RS	**Before starting, always reading requirements**		**5min**
NO	[00:00:00]	[00:02:30]	**2m30s**
DC	[00:02:30]	[00:02:54]	**24s**
DAM	[00:02:54]	[00:03:04]	**10s**
DC	[00:03:04]	[00:03:22]	**18s**
RS	[00:03:22]	[00:04:15]	**53s**
DC	[00:04:15]	[00:04:47]	**32s**
NO	[00:04:47]	[00:06:00]	**1m13s**
DM	[00:06:00]	[00:06:23]	**23s**
DA	[00:06:23]	[00:06:50]	**27s**
DAM	[00:06:50]	[00:07:29]	**39s**
CD	[00:07:29]	[00:09:59]	**2m30s**
DM	[00:09:59]	[00:10:15]	**16s**
DAM	[00:10:15]	[00:12:00]	**1m45s**
DC	[00:12:00]	[00:12:14]	**14s**
DAM	[00:12:14]	[00:13:19]	**1m5s**
DA	[00:13:19]	[00:15:19]	**2m**
CD	[00:15:19]	[00:16:31]	**1m12s**
DAM	[00:16:31]	[00:19:10]	**2m39s**
RS	[00:19:10]	[00:21:00]	**1m50s**
NO	[00:21:00]	[00:24:20]	**3m20s**
CD	[00:24:20]	[00:28:20]	**4m**
	RS	**7m43s**	
	DC	**1m28s**	
	CD	**7m42s**	
	DA	**2m27s**	
	DM	**39s**	
	DAM	**6m18s**	
	NO	**7m3s**	
	Total Time	**33m20s**	

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:

-What reasoning did you use for deciding what should be classes in your diagram?

Mm…I look at the use case model and the customer could reserve a tour, so I think it is reasonable that the tour is the class as well as the complaint and reservation. Mm…actually, nothing special for me to decide the classes and I just identify the important objects in the system as the class.

-What reason did you use for determining attributes and methods of the classes?

Yeah, I follow the description and the use case model as well to determine the attributes and methods of the class.

-What reasoning did you use for deciding associations between classes?

In briefly, I just try to see that the connections between different objects which can operate and which can be operated on different object.

-Did you think of any design principles while designing the class diagram?

No. I didn't think of.

-What's your naming strategy? i.e. how did you think of the names of your class diagram?

I name the names according to the description and it is easy to be understood. For derived names, such as the attribute of free seats in the tour class, I think it is necessary and I come up with the names based on my domain knowledge.

-***Common steps on how to create the class diagram***

- Read the requirements
- Check use cases
- Identify classes and multiplicities
- Identify operations and attributes
- Check the requirements again

13. Interview 13

Interview dialogue

RS: [Read the user requirements]

DC: First, it is the user. The employee and the customer are both the users and I use the inheritance that the employee and the customer are the subclass of the user class.

DM: In the user class, it should have public operations which are reserving tour, viewing tour and canceling tour.

RS: [Read the user requirements]

DC: There should be another class tour.

DM: The employee could add/delete/update tour, also.

CD: [Check the created class diagram; take a look at the example]

DA: OK. The user has the password.

RS: [Read the user requirements]

DC: Mm… I just add the complaint class.

DM: The employee could add/delete complaints and the customer could make complaints.

DAM: The customer could make one or several complaints and the employee could administrate one or several complaints.

DA: Mm…it is kind of complicated now. I guess we need to know how many reservations we have now. I add the attributes that number of seats and number of free seats to the tour class.

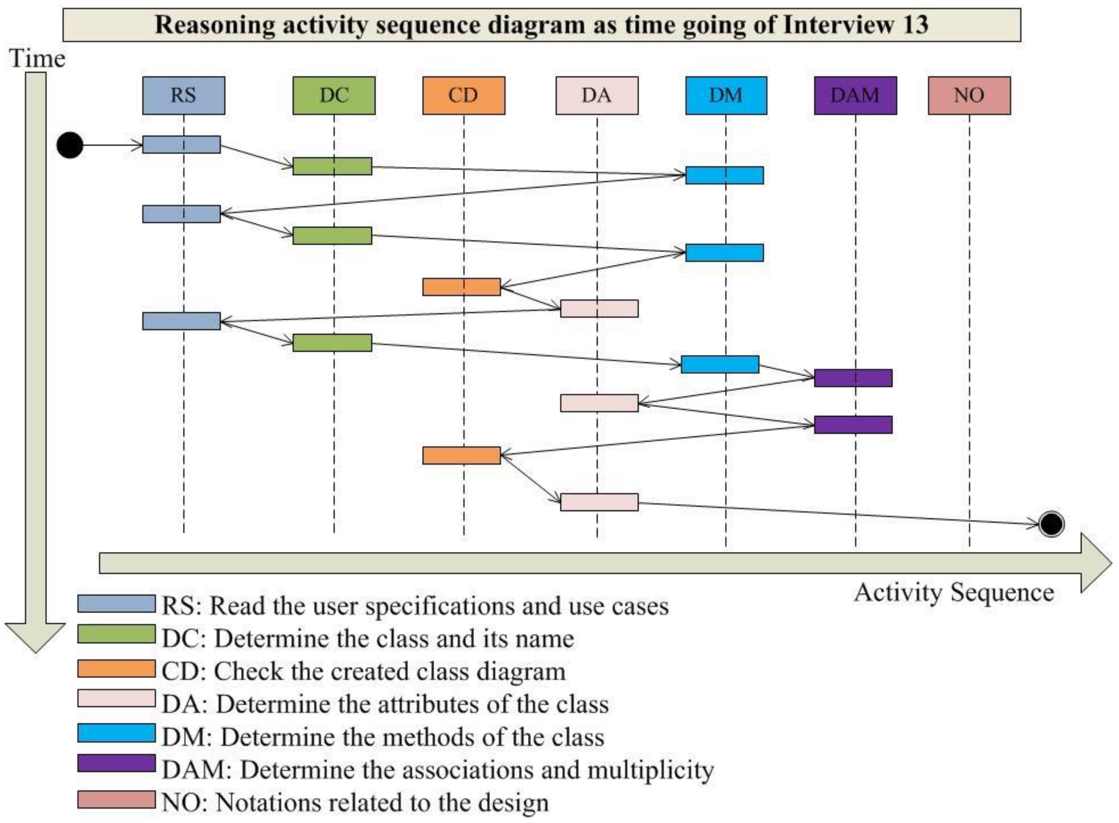
DAM: (Set the multiplicity one-to-many to the association between customer and tour; association name is traveler)

CD: [Check the created class diagram]

DA: The user also needs the login ID.

Mm…at least I have something to start with. (Finish the class diagram)

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview 13
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	***Start Time***	***Finish Time***	***Duration***
RS	**Before starting, always reading requirements**		**5min**
DC	[00:00:00]	[00:01:23]	**1m23s**
DM	[00:01:23]	[00:01:56]	**33s**
RS	[00:01:56]	[00:02:30]	**34s**
DC	[00:02:30]	[00:02:50]	**20s**
DM	[00:02:50]	[00:03:12]	**22s**
CD	[00:03:12]	[00:04:42]	**1m30s**
DA	[00:04:42]	[00:04:51]	**9s**
RS	[00:04:51]	[00:05:25]	**34s**
DC	[00:05:25]	[00:05:50]	**25s**
DM	[00:05:50]	[00:06:34]	**44s**
DAM	[00:06:34]	[00:07:34]	**1m**
DA	[00:07:34]	[00:08:16]	**42s**
DAM	[00:08:16]	[00:10:53]	**2m37s**
CD	[00:10:53]	[00:13:10]	**2m17s**
DA	[00:13:10]	[00:13:32]	**22s**
	RS		**6m8s**
	DC		**2m8s**
	CD		**3m47s**
	DA		**1m13s**
	DM		**1m39s**
	DAM		**3m37s**
	Total Time		**18m32s**

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:

-What reasoning did you use for deciding what should be classes in your diagram?

I think it is kind of based on my feeling and experience. When I have finished reading the description, for example, I think the tour, the user containing employee and customer should be classes and I didn't think too much on it.

-What reason did you use for determining attributes and methods of the classes?

I think the methods are clearly stated in the text and for the user, it is easy and clear that they can do like reserving, viewing and canceling. I just make the methods as public. [Why do you have the attributes in the tour class?] I think they are relevant to the tours and we can know how many tours have been booked and how many are available to the user. The attributes can be treated and considered as the reservations in the system.

-What reasoning did you use for deciding associations between classes?

It is obvious that the user need to have the login id and password for security purpose. And for the composition, I think no user certainly means no login id and password because they will disappear as the father class (the user class is the father class of the login and password class). It is similar to the customer and compliant because complaint could be made only by customers and the employee just manages and deals with it. [Change the association between the user and tour class; add the tour database class; Reconnect the user, tour database and tour class] As a result, the user can access to the tour database to do different service based on different purposes.

-Did you think of any design principles while designing the class diagram?

Mm…not now and I didn't think of.

-What's your naming strategy? i.e. how did you think of the names of your class diagram?

I think it is a good naming strategy to use the capital letter when deciding the class names. The method names should clearly present what should do and the attributes name should be close connected to the programming language. For derived names, they are based on my experience and domain knowledge.

-***Common steps on how to create the class diagram***

- Understand the task
- Identify classes
- Identify dependencies between classes
- Add methods to classes
- Add attributes to classes
- Iterate

14. Interview 14

Interview dialogue

RS: [Read the user requirements]

NO: I am thinking like this the system can offer a number of services, such as reserve, cancel, view, complain and administration. The employee could use the functions of administration and the customer could reserve, cancel, view information and complain. So, I will see I have.

DC: The first is the system.

DM: It has the operations – reserve, cancel, view, complain and administration. Maybe the complain is not correct in the system because the complaints are sent by emails and stored outside of the system, so I am not sure whether I should include it or not. (Think about the complain and delete the operations of complain in the system)

DC: Then, there should be class customer and employee.

DM: The customer could make reservations, cancel the tour and view information. The employee could do administration.

DAM: (Set the association from the system to the customer and the employee respectively; the multiplicities are both 1 to 1…*)

CD: [Check the created class diagram]

DA: The customer should have user id to log in the system such that I add the attribute of customer id. Also, it has the reservation id. For the employee, it has the attributes of tour id and employee id. The system also has the tour id as well.

RS: [Read the user requirements]

DM: OK. The employee could add/delete/update the information of the tour and customer. I add the corresponding operations to the employee class.

RS: [Read the user requirements and take a look at the example many times]

DC: I am changing the customer to the customer handler and the employee to the employee handler. Mm…I need to change it and rewrite it.

NO: (Make notes and think how to redesign the class diagram) I am thinking like this way of using sequence diagram to understand how the users access to the system. (Think about the system using the sequence diagram)

DC: The first one is the tours. And I have the customer hander to deal with the customer.

DM: The customer handler could add customers, delete customers, login customers. And also, it could reserve tour and cancel tour.

DA: The attributes of the customer handler are the password and the customer id.

DM: Back to the tours, it could have the methods of available tours, viewing tours, adding/deleing tours.

NO: Mm… this is a simple case but I have troubles of finding a good solution.

DC: Then, I have the employee handler to handle the employees.

DM: It has the operations of adding/deleting employees and offering employee access.

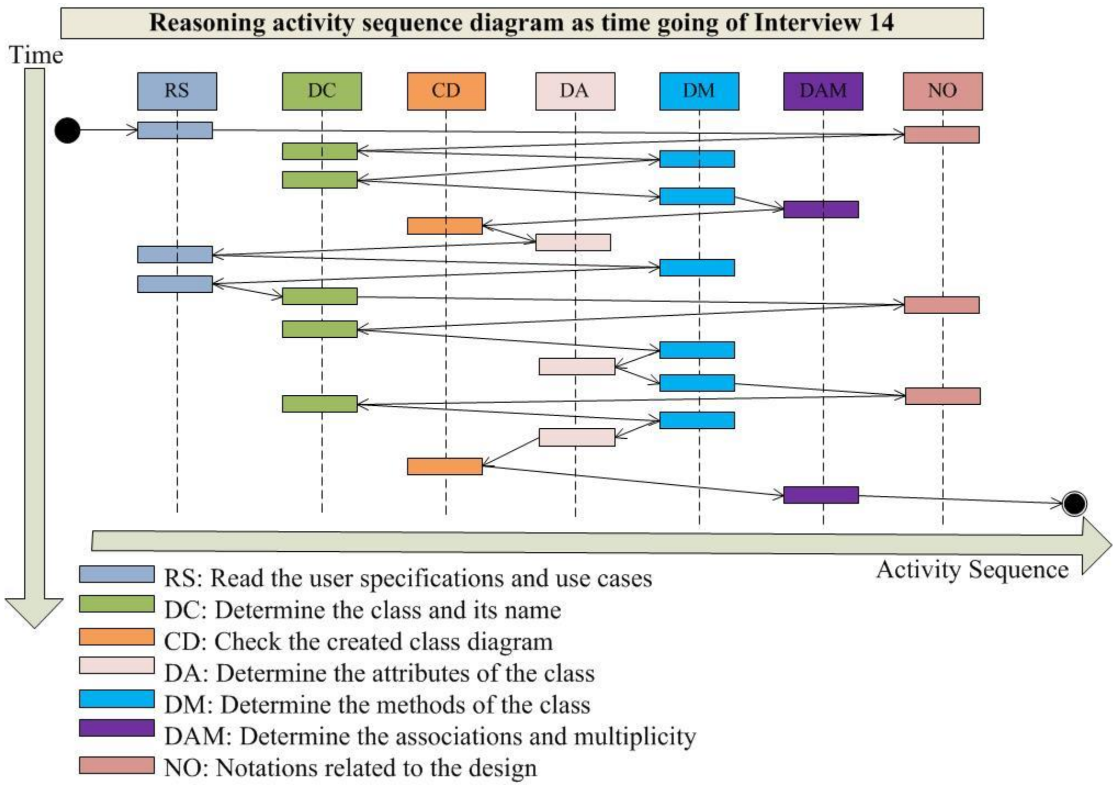
DA: (Add the same attributes to customer and employee – name, id, and password; change the tours to the tour handler and add class tour)

CD: [Check the created class diagram]

DAM: (Determine the associations between different classes) I have one employee

handler to handle one or several employee and it is similar to the customer handler as well as the tour handler. For handlers, they are one-to-one associations.
Mm…OK. This is my solution about this case.

Reasoning activity diagram


Reasoning activity sequence diagram as time going of Interview 14
Time
RS
DC
CD
DA
DM
DAM
NO
Activity Sequence
RS: Read the user specifications and use cases
DC: Determine the class and its name
CD: Check the created class diagram
DA: Determine the attributes of the class
DM: Determine the methods of the class
DAM: Determine the associations and multiplicity
NO: Notations related to the design

Reasoning activity timing distributes

Reasoning Activity	*Start Time*	*Finish Time*	*Duration*
RS	**Before starting, always reading requirements**		**5min**
NO	[00:00:00]	[00:02:30]	**2m30s**
DC	[00:02:30]	[00:02:50]	**20s**
DM	[00:02:50]	[00:04:15]	**1m25s**
DC	[00:04:15]	[00:04:43]	**28s**
DM	[00:04:43]	[00:05:23]	**40s**
DAM	[00:05:23]	[00:06:23]	**1m**
CD	[00:06:23]	[00:08:05]	**1m42s**
DA	[00:08:05]	[00:09:43]	**1m38s**
RS	[00:09:43]	[00:10:33]	**50s**
DM	[00:10:33]	[00:11:45]	**1m12s**
RS	[00:11:45]	[00:13:25]	**1m40s**
DC	[00:13:25]	[00:15:16]	**1m51s**
NO	[00:15:16]	[00:18:30]	**3m14s**
DC	[00:18:30]	[00:19:04]	**34s**
DM	[00:19:04]	[00:19:55]	**51s**
DA	[00:19:55]	[00:20:23]	**28s**
DM	[00:20:23]	[00:21:11]	**48s**
NO	[00:21:11]	[00:21:21]	**10s**
DC	[00:21:21]	[00:21:56]	**35s**
DM	[00:21:56]	[00:22:50]	**54s**
DA	[00:22:50]	[00:24:10]	**1m20s**
CD	[00:24:10]	[00:29:55]	**5m45s**
DAM	[00:29:55]	[00:34:04]	**4m9s**
RS		**7m30s**	
DC		**3m48s**	
CD		**7m27s**	
DA		**3m26s**	
DM		**5m50s**	
DAM		**5m9s**	
NO		**5m54s**	
Total Time		**39m4s**	

Addition:

1. The timing distributes is based on the interview video record.

2. The video records from the beginning of making design of the interview.

3. Before the interviewee starting to make design, he/she needs to read the user requirements and use case model to understand the domain task. There is no exact calculation on it such that I set the default duration time of each interview's first RS activity as five minutes.

Question time:

-What reasoning did you use for deciding what should be classes in your diagram?

I was thinking about the sequence diagram and in the sequence diagram, it can be easy to see the interaction between different objects and how they communicate for each other. I identified the classes according to it and also I made the design using the handler thoughts. I have the employee and corresponding handler to deal with the operations they need to do. It is similar to the customer and tour. Certainly, I need to have the handlers to process the relevant information and services about them.

-What reason did you use for determining attributes and methods of the classes?

My thought is that the user in the system would like to do. For customers, they would like to access to the system and make reservations, for example, so I think they should be methods. For attributes, it is easily known that each customer should have own name, id and password.

-What reasoning did you use for deciding associations between classes?

Mm…for example, I think that the customer class store customer information and the customer handler would process the information in order that they would be associated. And the customer handler can reserve tours; as a result, it should be connected with the tour handler. I just make the association simple and if the association too complicated, it is different to make a good software design, I think.

-Did you think of any design principles while designing the class diagram?

I probably should have but I didn't. (Handler)

-What's your naming strategy? i.e. how did you think of the names of your class diagram?

I try to make the names understood and keep them simple. We can easily know the responsibility of the classes and what operations the classes would do.

-***Common steps on how to create the class diagram***

1. Make an initial block diagram to find the objects
2. Identify the responsibility for each class
3. Decide on the interaction between the objects
4. Modify the solution when I understand the problem domain better
5. Add attributes that each object needed
6. Add associations

Appendix C – Statistical significant difference document

Ⅰ Statistical significant difference on the amount of reasoning activity

1. The amount of reasoning activity RS

2 Sample Mann-Whitney – The amount of RS

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	**E**	**N**
Count	5	9
Median	3	4
Mann-Whitney Statistic	36.50	
p-value (2-sided, adjusted for ties)	0.9453	

2. The amount of reasoning activity DC

2 Sample Mann-Whitney – The amount of DC

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	**E**	**N**
Count	5	9
Median	4	4
Mann-Whitney Statistic	40.00	
p-value (2-sided, adjusted for ties)	0.7836	

3. The amount of reasoning activity DA

2 Sample Mann-Whitney – The amount of DA

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	**E**	**N**
Count	5	9
Median	2	3
Mann-Whitney Statistic	26.50	
p-value (2-sided, adjusted for ties)	0.1470	

4. The amount of reasoning activity DM

2 Sample Mann-Whitney – The amount of DM

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	E	N
Count	5	9
Median	3	3

Mann-Whitney Statistic	36.00
p-value (2-sided, adjusted for ties)	0.8913

5. The amount of reasoning activity DAM

2 Sample Mann-Whitney – The amount of DAM

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	E	N
Count	5	9
Median	5	2

Mann-Whitney Statistic	49.00
p-value (2-sided, adjusted for ties)	0.1242

6. The amount of reasoning activity CD

2 Sample Mann-Whitney – The amount of CD

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	E	N
Count	5	9
Median	2	2

Mann-Whitney Statistic	40.00
p-value (2-sided, adjusted for ties)	0.7736

7. The amount of reasoning activity NO

2 Sample Mann-Whitney – The amount of NO

Test Information		
H_0: Median Difference = 0		
H_a: Median Difference ≠ 0		
Category	**E**	**N**
Count	5	9
Median	3	0
Mann-Whitney Statistic	51.50	
p-value (2-sided, adjusted for ties)	0.0531	

8. The total amount of reasoning activity

2 Sample Mann-Whitney – The total amount of Reasoning activity

Test Information		
H_0: Median Difference = 0		
H_a: Median Difference ≠ 0		
Category	**E**	**N**
Count	5	9
Median	22	18
Mann-Whitney Statistic	44.00	
p-value (2-sided, adjusted for ties)	0.4227	

Ⅱ Statistical significant difference on the time spent of reasoning activity

1. The time spent of reasoning activity RS

2 Sample Mann-Whitney - Time of RS

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	E	N
Count	5	8
Median	0.005208333	0.004184028
Mann-Whitney Statistic	45.00	
p-value (2-sided, adjusted for ties)	0.1643	

2. The time spent of reasoning activity DC

2 Sample Mann-Whitney - Time of DC

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	E	N
Count	5	8
Median	0.002152778	0.002118056
Mann-Whitney Statistic	33.00	
p-value (2-sided, adjusted for ties)	0.8262	

3. The time spent of reasoning activity DA

2 Sample Mann-Whitney - Time of DA

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	E	N
Count	5	8
Median	0.001446759	0.001788194
Mann-Whitney Statistic	29.00	
p-value (2-sided, adjusted for ties)	0.4208	

4. The time spent of reasoning activity DM

2 Sample Mann-Whitney - Time of DM

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	**E**	**N**
Count	5	8
Median	0.001145833	0.002274306
Mann-Whitney Statistic	27.00	
p-value (2-sided, adjusted for ties)	0.2723	

5. The time spent of reasoning activity DAM

2 Sample Mann-Whitney - Time of DAM

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	**E**	**N**
Count	5	8
Median	0.004375	0.002303241
Mann-Whitney Statistic	49.00	
p-value (2-sided, adjusted for ties)	0.0481	

6. The time spent of reasoning activity CD

2 Sample Mann-Whitney - Time of CD

Test Information

H_0: Median Difference = 0

H_a: Median Difference ≠ 0

Category	**E**	**N**
Count	5	8
Median	0.002835648	0.002326389
Mann-Whitney Statistic	43.00	
p-value (2-sided, adjusted for ties)	0.2723	

7. The time spent of reasoning activity NO

2 Sample Mann-Whitney - Time of NO

Test Information

H₀: Median Difference = 0

Hₐ: Median Difference ≠ 0

Category	E	N
Count	5	8
Median	0.004097222	0
Mann-Whitney Statistic	50.00	
p-value (2-sided, adjusted for ties)	0.0210	

8. The total time spent of reasoning activity

2 Sample Mann-Whitney - The time spent of total reasoning activity

Test Information

H₀: Median Difference = 0

Hₐ: Median Difference ≠ 0

Category	E	N
Count	5	8
Median	0.023148148	0.015295139
Mann-Whitney Statistic	48.00	
p-value (2-sided, adjusted for ties)	0.0673	