Solver: Priscilla Teo
Email Address: PTEO008@e.ntu.edu.sg

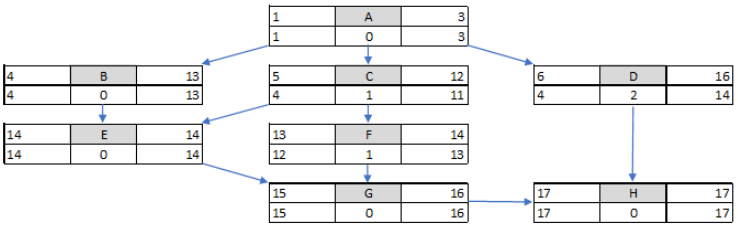
1. (a)

(i)

Legend:

Latest Start	Task	Latest End
Earliest Start	Slack Time	Earliest End

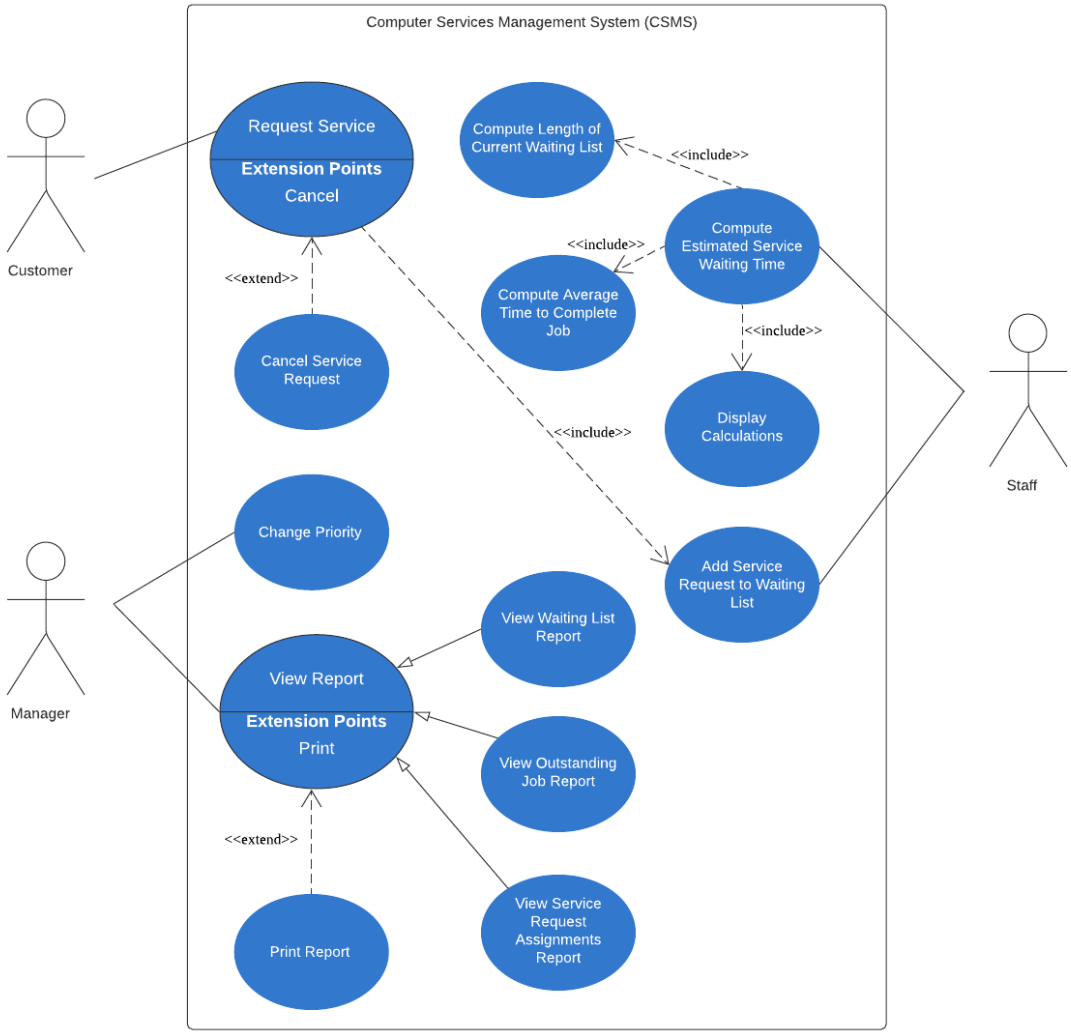
Note:
Latest/Earliest Start - start of week (eg. Start of Week 1)
Latest/Earliest End - end of week (eg. End of Week 17)
Slack Time = Latest Start - Earliest Start



(ii) Critical Path: ABEGH
Shortest Time to Completion = 17 weeks

(iii) Slack Time (Task C) = Latest Start – Earliest Start = 5 – 4 = 1 week

(b)

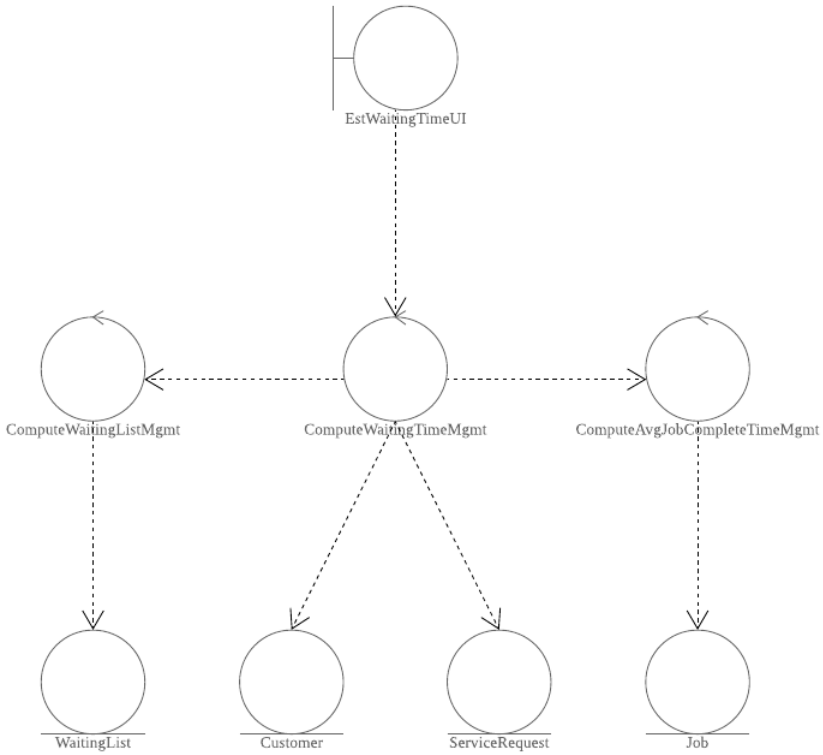


(c)

Use Case ID:	UC02
Use Case Name:	Compute Estimated Service Waiting Time

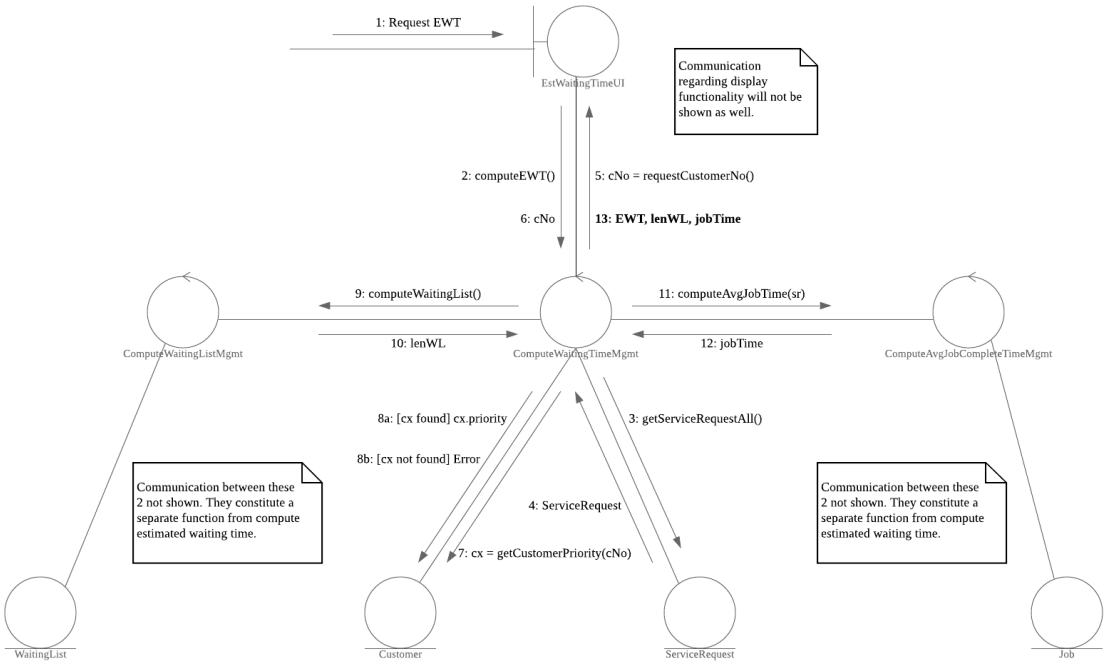
Actor:	Staff
Description:	This use case allows the technical staff to determine the estimated service waiting time for the customer's new service request.
Preconditions:	- The customer has made a service request - The staff has the five0digit customer number
Postconditions:	- The staff receives the estimated service waiting time, along with other information such as
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The staff selects "Compute Estimated Service Waiting Time". 2. The system retrieves all service request information from the service request file. 3. The system requests for the customer number. 4. The staff enters the customer number. 5. The system verifies the validity of the customer number. 6. The system retrieves the customer's priority number from the customer file. 7. The system mathematically computes the estimated service waiting time. This will invoke UC03 (Compute Length of Waiting List) and UC04 (Compute Average Time to Complete Job). 8. The system invokes UC05 (Display Calculations) to display all the relevant information.
Alternative Flows:	AF1: Customer number is invalid
Exceptions:	NA
Includes:	UC03 – Compute Length of Waiting List UC04 – Compute Average Time to Complete Job UC05 – Display Calculations

2. (a) Note: For both parts, please only draw the respective diagrams for the *compute estimated waiting time* functionality. Do not spend time drawing other irrelevant classes.



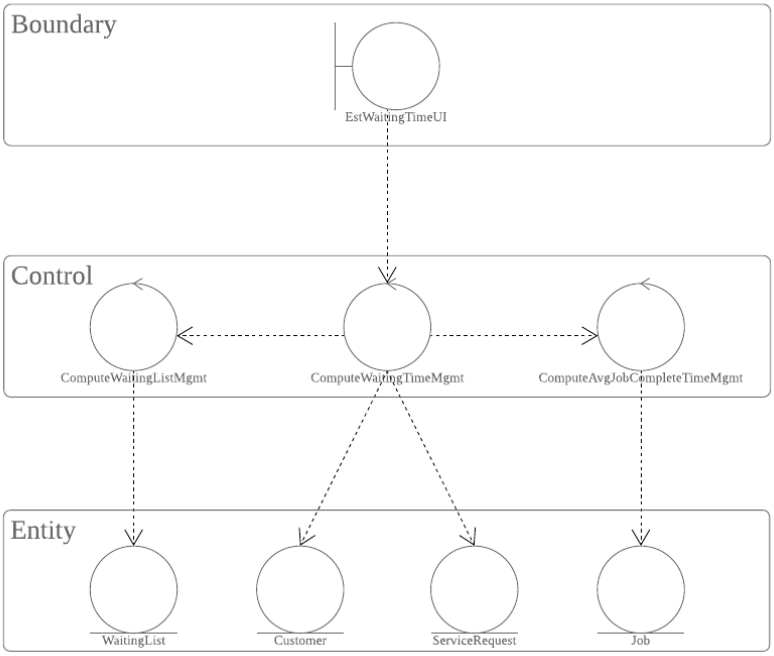
Explanation (not necessary in the paper, just my interpretation):
- One main boundary (EstWaitingTImeUI) for this function, staff wants to compute estimated waiting time, and will only explicitly call ComputeWaitingTimeMgmt. Even though EstWaitingTimeUI needs 3 values, it will get all 3 values only from ComputeWaitingTimeMgmt.
- In terms of getting the other 2 controls (ComputeWaitingListMgmt, ComputeAvgJobCompleteTimeMgmt) to compute, it will be called by ComputeWaitingTimeMgmt and the respective values will be returned to the latter.
- Onto the entities:
    - ComputeWaitingTimeMgmt will pull ServiceRequest information (explicitly stated) and Customer (mentioned as they need staff to input customer number).
    - ComputeWaitingListMgmt needs to check their current WaitingList record to compute. I assume that they don't need ServiceRequest as the latter are not valid entries in the waiting list yet.
    - I am not 100% sure about ComputeAvgJobCompleteTimeMgmt but I had a Job entity, considering how later in the project description they mentioned an Outstanding Job Report. Not that reporting should be included into this class diagram, but that gave me the idea to add in Job entity.

(b)


1: Request EWT
EstWaitingTimeUI
Communication regarding display functionality will not be shown as well.
2: computeEWT()
5: cNo = requestCustomerNo()
6: cNo
13: EWT, lenWL, jobTime
9: computeWaitingList()
10: lenWL
11: computeAvgJobTime(sr)
12: jobTime
ComputeWaitingListMgmt
ComputeWaitingTimeMgmt
ComputeAvgJobCompleteTimeMgmt
8a: [cx found] cx.priority
8b: [cx not found] Error
3: getServiceRequestAll()
4: ServiceRequest
7: cx = getCustomerPriority(cNo)
Communication between these 2 not shown. They constitute a separate function from compute estimated waiting time.
Communication between these 2 not shown. They constitute a separate function from compute estimated waiting time.
WaitingList
Customer
ServiceRequest
Job

3. (a)

(i) Layered Architecture



(ii) 3 benefits:

- Separation of concern: This allows us to focus on a smaller scope of problems in each layer, as opposed to looking at the problem across different layers
- Isolation: Each layer is decoupled, thus modifications in one layer will not affect downstream layers
- Changeability: Can easily replace one whole layer with another, while interface is still maintained

1 drawback:

- Performance overhead: more pronounced when more layers are added

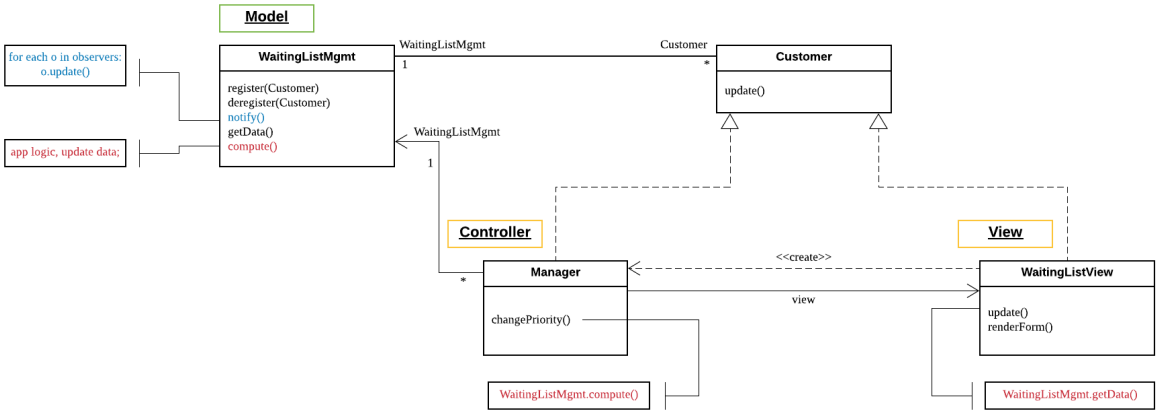
(b)

(i) Push update is a one-way communication from the subject. The subject will push the details of their announcement to all observers, regardless of whether each observer is interested or not.

Pull update is a two-way communication between the subject and the object. The subject can send a simple ping (Observer.update()), to which interested observers can call back (Subject.getData()) for details.

(ii) The manager can alter the customer's priority in the waiting list. This will affect not only the customer in question, but other customers in the waiting list (as their order will be recomputed with a singular change). There is currently no implementation of a notification system to inform customers of their change in waiting list position and consequently time.

(iii) <Observer Pattern>



- Roles:
  a) Model (WaitingListMgmt) - contains the processing (operations) and data involved
  b) View (WaitingListView) - presents the output
     - Defines and manages how data is presented to user
     - Each View provide specific presentation of same Model
     - Each View "observes" the Model - Whenever the data Model changes, all Views are immediately notified -> so they can update their graphical presentation
  c) Controller (Manager) - manages user interaction
     - Captures user input (events can be mouse clicks, key presses etc.) and passes these interactions to View and Model
     - Each View is associated to a Controller that captures and processes user input + modifies the data Model
     - User interacts with system solely through Controllers
- Observer Pattern (View-Model relationship):
  - When Model (Subject) data is changed, it updates all the Views (Observers, via notification mechanism); allowing multiple Views to same Model

    Contextually: WaitingListMgmt will perform its recomputation, updating to all WaitingListViews.

4. (a)

(i) Equivalence Class

Inputs	Valid	Invalid
Customer number (i1)	10001 <= i1 <= 89999	i1 <= 10000
Customer Priority (i2)	i2 ∈ {"Priority 1","Priority 2","Priority 3"}	i2 ∉ {"Priority 1","Priority 2","Priority 3"}
Average Time to Complete a job (i3)	10 <= i3 <= 60	i3 <= 9 i3 >= 61

(ii) Boundary Value

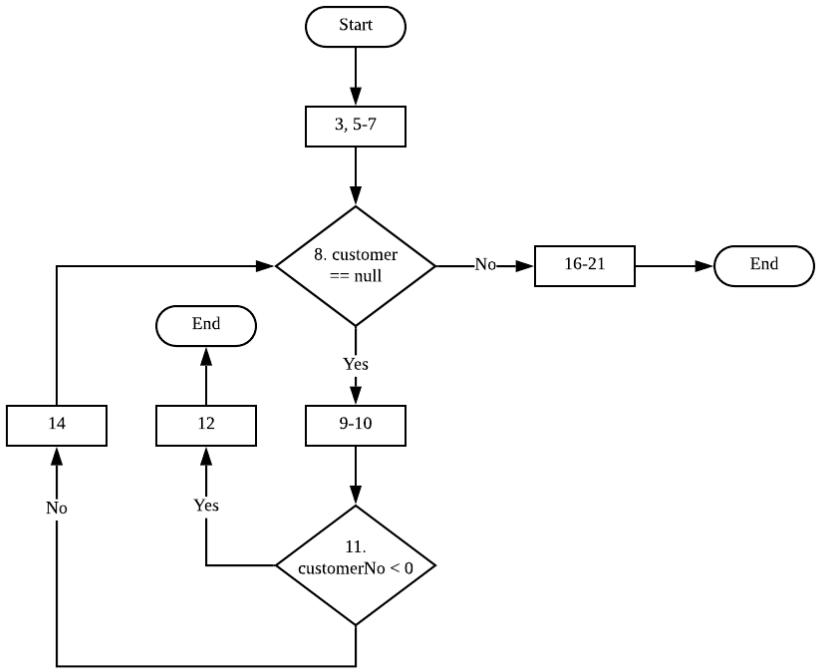
Inputs	Just-Below	On	Just-Above
Customer number (i1)	10000	10001 89999	90000
Customer Priority (i2)	NA		
Average Time to Complete a job (i3)	9	10 60	61

(iii) Defensive Testing

Customer Number	Customer Priority	Average Time to Complete a Job	Expected Result
10001	**Priority 1**	**10**	Pass
10001	**Priority 2**	10	Pass
10001	**Priority 3**	10	Pass
10001	**Priority 0**	10	Fail
89999	Priority 1	10	Pass
10000	Priority 1	10	Fail
90000	Priority 1	10	Fail
10001	Priority 1	**60**	Pass
10001	Priority 1	**9**	Fail
10001	Priority 1	**61**	Fail

(b)

(i)



(ii) Since both decision points are binary,
Cyclomatic complexity = |decision points| + 1 = 2 + 1 = **3**

(iii) **

#	Path	customerNo	findCustomerRecord()
1	3-7, 8, 16-21	10001	1st time: customer != null
2	3-7, 8, 9-10, 11, 12	-1	1st time: customer == null
3	3-7, 8, 9-10, 11, 14, 8, 16-21	-1	1st time: customer == null 2nd time: customer != null

#	getNewCustomerNo()	Expected Result
1	*NA*	result
2	-1	Exception("invalid customer!")
3	10001	result

All the best for your exams!