Written by:

jonahhal
hermagst
simonamo
Dafolvel

Task 1

User requirements -> Acceptance test execution:

The user requirements phase uses acceptance testing. This form of testing involves establishing confidence in the system, with focus on the non-functionality. Forms of acceptance testing is validation of the fitness of the system to users and stakeholders. In our case, the representatives of *A Ticket Machine* have to validate if the system meets their needs, and can cooperate with users to better define acceptance criteria based on real-world scenarios. For example, an adult buying a 7-day ticket on a weekday with a lot of server traffic. We then follow the necessary states in the ticket machine module, expecting a confirmed payment from the info screen with a dispensed and valid 7-day ticket. With the user expecting a quick transaction with few delays.
When focusing on the user requirements and meeting their expectations, we can start as early as possible in the development cycle. It also ensures that the system meets user expectations avoiding absence-of-error fallacy. It can also help uncover defects on a high level, in addition to testing with different contexts the system can be dependent on.

System Requirements -> System test execution:

The System requirements phase uses system test execution, with the objective being to test the behavior of the whole system. It is meant to verify that the system meets its requirements when it comes to specification and purpose.
A Ticket Machine could perform system testing using the black box testing technique. With black box testing, the tester would find defects with the system requirements. In this way it ensures that the ISTQB test principle seven absence-of-error fallacy is met. System testing involves testing the required and expected behavior of the system. An example of this is testing for false positives in giving the control module an invalid travel card and to see if it recognizes it as valid or not.

Global design -> Integration test execution:

Integration test execution is used to test the global design phase where the objective is to test the interface and interactions between components and different parts of the system.
In *A Ticket Machine* it's important that the ticket machine module and the control model are correctly integrated with each other. The integration will be done incrementally. Allowing for early

testing with testers doing different tests at different stages of the integration. Testers will not try to exhaustively test the integration, but rather focus on the area with highest risk. An example of this is testing for the updated response between newly written data from *the ticket machine module* to *the control module*. So that under high traffic on the servers, the system performs well with minimal delay when cards are created and balances are updated from *the ticket machine module*, to when cards are read and updated from *the control module*.

Detailed design -> component test execution:

The detailed design phase uses component test execution, with the objective being to verify the software item's functions and search for defects. Software items can be classes, objects, modules and methods. With testing on functionality and non-functionality characteristics regarding robustness, structural and resource-behavior.

Component testing shows the presence of defects when the tests are being executed. It does not aim to test everything, and because it's testing one component at a time it applies to testing ISTQB principle four defect clustering. Each component will have some of the same tests but each component is different and therefore requires some different test cases. An example of component testing in the context of the ticket machine system is testing if the instance variable for the balance on a travel card class is updated correctly after a deposited amount.

Task 2

In descending order of priority the test conditions for this system are:

1. The control module properly manages the registration of trips and tracking expiration time. This includes allowing multiple trips within the same hour of registration without charging the card multiple times. Over-registering the user's trips would be a critical error in the system and probably illegal.
2. The system only swallows cards after some period of time. This is to prevent unwanted access to users' cards or stolen cards. If this were to happen it would be a severe security issue and would also lead to a terrible user experience. Swallowed cards should be retrievable by strictly authorized personnel. Failure in this case implies a security risk and a bad user experience.
3. Verify that purchasing a new travel card succeeds and gives appropriate feedback to the user. This is the primary feature of the system and is therefore a vital condition.
4. Verify that the system can handle cases where the user’s bank account does not have sufficient funds to complete the transaction. This includes giving an appropriate error message before returning the credit card and resetting the system’s state. The system should also not give the user a valid travel card in this case. This condition could be severely exploited if failed so it is of vital importance.
5. The system can identify the validity of any ticket or travel card. The control module should be able to identify expired or spent travel cards and tickets, denying access while also displaying an appropriate error message. This feature is important for repeated usage of the system and could also lead to exploitation if failed.
6. Verify that the transferral of funds to an existing travel card succeeds, and give appropriate feedback to the user. Similar to the previous condition this feature is also very important for repeated usage of the system.
7. The ticket module can properly differentiate between the different travel card options and handles them appropriately. This includes displaying the right information to the user and requesting the correct amount of money from the payment service for each of the options. This feature is important to provide a predictable user experience and could also be exploited if failed.
8. Verify all discounts are properly applied. Since discounts are only on the weekends, the system should check for this condition, and then if applicable prompt the user for what type of ticket they want to purchase. This step is missing from the specification of the ticket module, but we could imagine it between state 1 and 2 as the next step if the user selects ‘Buy a ticket or travel card’. This condition is also important for a predictable user experience and could also be exploited if failed.
9. Verify that the system can handle any user-initiated cancellations at any stage of the transaction. This includes giving feedback of the cancellation, returning the user’s credit card if inserted and then resetting the state of the system. For a good user experience it is necessary to handle any cancellations, however this is not a critical feature of the system.

10. The control module can identify the status of any ticket or travel card and write it back to the user. Information about remaining duration, funds remaining on card, etc. should be easily accessible for users of the system. For a good user experience the user should be able to view basic information about their tickets and travel cards, however this is also not a critical condition.

Because the system involves a payment process there will always be some degree of risk involved. We therefore mostly used an analytical approach to identify the test conditions for the system. For the less important test conditions we used a more methodical approach, doing some amount of error guessing after identifying lesser, but still necessary aspects of the system.

Task 3

Design the test cases. Explain how you have used the technique to derive and design the test case. Ensure traceability to the correct requirement in the requirements specification. All test cases will be numbered with their corresponding test conditions in task 2. E.g the test case for verifying that all discounts are properly applied will have the number 8 from task 2.

8 - Equivalence partitioning:

Test case: Verify all discounts are properly applied for travel card refills on travel cards on the weekend.

Invalid Partition	Valid for 100% discount		Valid for 30% discount		Valid for 50% discount	
Age < 0	Age 0	Age 17	Age 18	Age > 18	Student	Student

Test case: Verify all discounts are properly applied for travel card refills on travel cards on a weekday.

Invalid Partition	Valid for 0% discount		Valid for 0% discount		Valid for 0% discount	
Age < 0	Age 0	Age 17	Age 18	Age > 18	Student	Student

8 - Decision table:

Test case: Verify all discounts are properly applied for travel cards. This gives us 3 different conditions, Adult, Student and Weekday.
We have built and simplified the decision table with the actions that are equivalent.

	Rule 1-4	Rule 5	Rule 6	Rule 7 & 8
Conditions:				
Weekday	T	F	F	F
Adult	X	T	T	F
Student	X	T	F	X
Actions:				
Discount	0%	50%	30%	100%

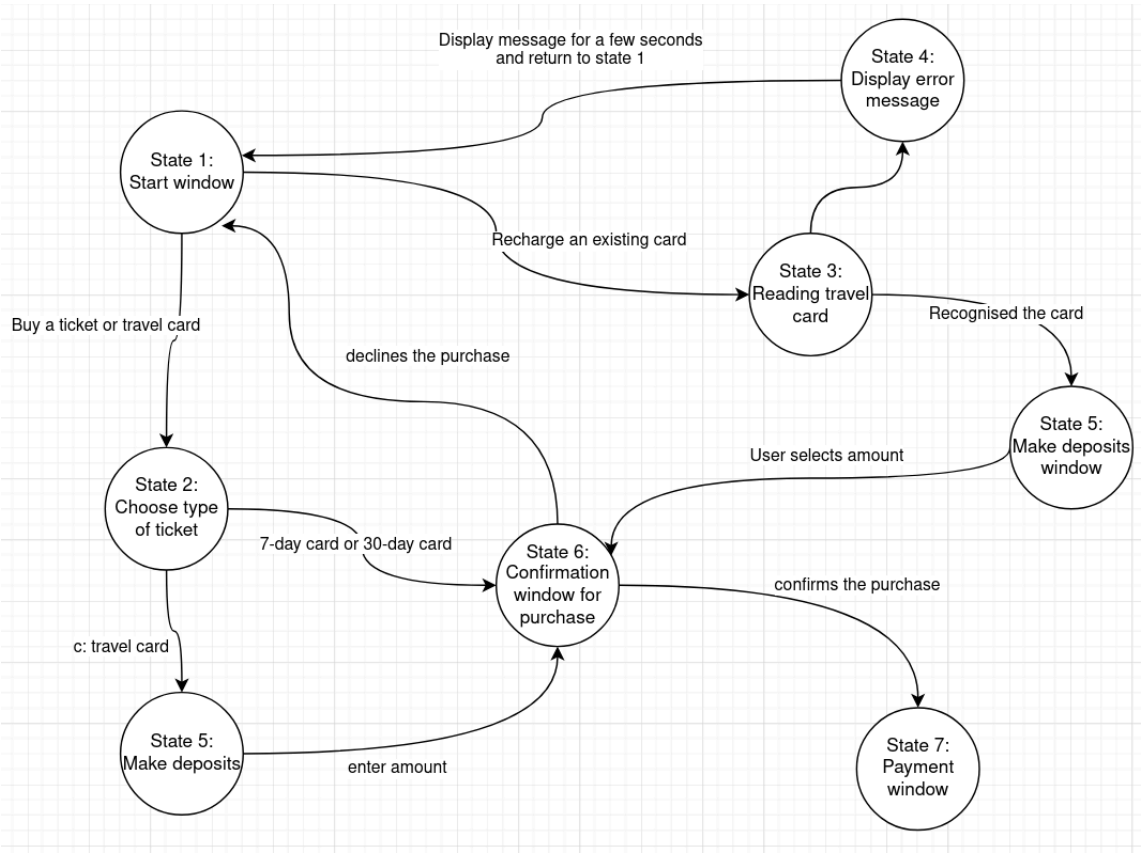
4, 10 - Use case testing:

We used the use case testing to describe the interaction between the user and the system when the user does not have sufficient funds in their bank account to complete the transaction.

Use case name	Insufficient Funds Handling	
Pre-conditions	System is in state 1	
Actor(s)	User: U and System: S	
Post-conditions	Transaction outcome is communicated	
Main Success Scenario	Step	Description
	1	U: selects the type of ticket or travel card
	2	S: displays the total cost for the selected ticket
	3	U: inserts their credit card to pay
	4	S: Contacts bank
	5	S: Receives a response from the bank about insufficient funds
	6	S: Display an error message: "Insufficient funds."
	7	U: Receives the error message and acknowledges by pressing "OK."
	8	S: Eject the credit card and reset state.
Extentions	U.3	User doesn't insert card S: System times out and returns to state 1.
	S.5	bank confirms U has sufficient funds S: Completes transaction, ejects card and displays a confirmation message to the user

3, 4, 6, 7, 9 - State transition

Test case: The system can identify the validity of any ticket or travel card.



NOTE: there is an optional control flow to reset the system to state 1 in every state if the user cancels the transaction.

Task 4

In task 3 we consider both the qualitative and quantitative aspects of the testing process. The test cases developed, through equivalence partitioning, decision tables, and use case testing, were designed to target specific and critical functionalities within the system. Testing the processing of transactions, application of discounts, and error handling during payment failures.

The extent of test coverage can be effectively assessed by how well these test cases cover the outlined functional requirements. For example, the decision table tests ensure that all logical combinations of conditions (like user categories and days of the week) have been examined for correct discount application, while use case testing explores real-world scenarios involving transaction processes and feedback mechanisms. This ensures not only that the system performs as expected under typical usage conditions but also safeguards against potential errors in critical operations.

To assess the test coverage of our test specification we must first specify the requirements we are measuring coverage for. Here we can use the test conditions specified by task 2 as this gives us a general sense of the important functional requirements in the system. Here we can see that we cover cases 3, 4, 6, 7, 8, 9, 10 which gives us a test coverage of 7/10 = 70%. However, it's important to note that if we even had 100% test coverage, it would not mean that all test items are tested. This coverage only represents the percentile of test coverage items executed from task 2.

Task 5

To make the system more user-friendly and accessible, we can establish some guidelines that may help with achieving a wider range of users.

- The functionality should be explicit. That means that the wording is clear.
- There should be a help button to further help the user understand how the interaction will work.
- Images used are required to take up at least 40% of the screen. This does not apply to symbols.
- Instructions provided for understanding and operating content do not rely solely on sensory characteristics of components such as shape, color, size, visual location, orientation, or sound.
- The visual presentation of text and images of text has a contrast ratio of at least 4.5:1
- The content can be presented without the loss of information or functionality and without requiring the user to scroll.
- Make the text of all alternatives bold to emphasize the options.
- Give the user the opportunity to cancel the given choice and return to the previous state.
- Allow the user to change language. English is required, as is any native language
- A speaker shall present the options in a given state, and read out the option selected.
- Any button presented be at minimum 3 cm wide and tall.

Complying with these guidelines will make the system more understandable and easier to use.

Sources

- https://www.w3.org/TR/WCAG/
- Graham, D., Black, R., & Van Veenendaal, E., *Foundations of Software Testing*, 1st ed., Andover, Hampshire, Cengage Learning EMEA, 2019, pp. 39-48, 112-132.
- https://astqb.org/istqb-foundation-level-seven-testing-principles/