# Assignment 2 in3240/in4240

Group members:

# Task 1

The V-model structures testing into both the relationship between development and testing, and into different testing levels. Every development activity has a corresponding testing activity, and each level has specific objectives. These levels are somewhat flexible and may overlap depending on your project, but the model is clear that analysis and design of tests should start during the corresponding development activity, even if some flexibility is applied.

The four test levels are Unit, Integration, System, and Acceptance, each representing different goals. Unit tests aim to validate that a single component works the way it is intended. Integration tests verify that components interact properly with each other and external systems. System tests assess whether the entire product behaves as intended and meets project goals. Acceptance testing evaluates the system from a user perspective to see if it fulfills its purpose and is acceptable for release.

For the Ticket Machine, a valid unit test could be checking if the method for retrieving expiration time returns correct data. For example, a test could assert that a 7-day card activated now correctly shows an expiration seven days later. Following ISTQB principles, such a test can only confirm that a specific defect is absent; other defects could still exist. Since testing every possible case is impossible, risk based prioritization becomes essential, aligning with the principle that exhaustive testing cannot be achieved.

For integration testing a good example could be inserting an invalid card and verifying that the error screen displays properly. This tests the flow of data, input reading, error handling, and error display together. While broad in scope, integration tests must be precise about what is being verified, since many points of failure exist across the components. Failures early in the chain can lead to unexpected behavior later, creating values or states that haven't even been considered as possible.

This naturally connects to the pesticide paradox; running the same test repeatedly won’t find new bugs. New inputs and variations must be introduced to create different faults. Once a specific error is addressed, that test might always pass unless the system or parameters change. Defect clustering is also likely during integration testing; typically, a small number of modules contain the majority of defects. Recognizing this in planning and implementing early focused testing of individual units helps reduce the risk of overwhelming defect clusters.

System testing takes the complete, integrated product and checks if it fulfills the specified requirements. For the Ticket Machine, this could involve simulating a full customer journey: buying a ticket, recharging a card, and checking balances. This is mostly done in a test environment, simulating real conditions without yet being live. During system testing it is crucial to remember the absence-of-errors fallacy, a system can be free of bugs but still fail if

it does not meet user or business needs. System testing checks not just for correctness but for overall usefulness, stability and more.

Context dependency becomes even more apparent at the system and acceptance levels. While unit and integration testing mainly focus on technical correctness, system testing must consider aspects like security, usability, complexity, and performance depending on the system's intended use. For the Ticket Machine, usability and accessibility become critical because the end users are the general public, who vary widely in technical, cognitive and physical abilities.

Acceptance testing shifts the focus fully to real-world use. Here, the system is tested by actual users or business stakeholders to validate that it fulfills its intended purpose. An example for the Ticket Machine would be having a first-time user attempt to buy and activate a ticket without external help. This tests whether the system is intuitive, if the displayed information is understandable, and whether the business requirements, such as reliable transaction handling, are met. In the context of the Ticket Machine, accessibility testing would also be a major focus, ensuring that differently abled users can successfully interact with the system without discrimination.

# Task 2, Test Conditions and Strategy

In this task, we outline the test conditions based on provided system specifications, prioritize them by severity, and clarify the testing strategy chosen for their identification.

## Ticket Machine Module:

| Use Case | State Flow |
|---|---|
| **Buy a rechargeable travel card** | 1 (Start) → 2 (Choose Travel Card) → 6 (Confirm Purchase) → 7 (Payment) → *1* |
| **Buy a 7-day ticket** | 1 (Start) → 2 (Choose 7-day Ticket) → 6 (Confirm Purchase) → 7 (Payment) → *(after payment)* → *1* |
| **Buy a 30-day ticket** | 1 (Start) → 2 (Choose 30-day Ticket) → 6 (Confirm Purchase) → 7 (Payment) → *(after payment)* → *1* |
| **Recharge existing travel card** | 1 (Start) → 3 (Read Travel Card) → 5 (Deposit Amount) → 6 (Confirm Deposit) → 7 (Payment) → *1* |
| **Check travel card balance** | 1 (Start) → 3 (Read Travel Card) → *(Display balance info)* → *1* |

## Control Module:

| Use Case | State Flow |
|---|---|
| **Register a trip (≥1 hour since last trip)** | 1 (Read Card) → 2 (Check Expiry & Last Registration) → *(≥1hr since last)* → 3 (Deduct Balance) → 4 (Show Info) → *(return after display)* → *1* |
| **Multiple trips within 1 hour** | 1 (Read Card) → 2 (Check Expiry & Last Registration) → *(<1hr since last)* → 4 (Show Info, no deduction) → *1* |
| **Expiration info (valid ticket/card)** | 1 (Read Card/Ticket) → 2 (Check Expiry) → *(Valid)* → 4 (Show Expiration Info) → *1* |
| **Ticket validity (invalid ticket/card)** | 1 (Read Card/Ticket) → 2 (Check Expiry/Validity) → *(Invalid/Expired)* → 5 (Show Error Message) → *1* |

We assume the system properly handles:

- Damaged, partially damaged, or corrupted travel cards
- Payment failures e.g declined cards
- Cancel/abort actions at any state

## Strategy/Approach

The strategy used to identify test conditions primarily combines analytical approaches such as Risk-Based and Requirement-Based testing. We assess potential risks and user actions from the system specifications, we prioritize areas with high potential impact. Additionally, we incorporate Error-Guessing as a complement, trying to predict likely faults based on existing knowledge and user scenarios. This forms a sort of "grey-box" testing scenario, where we have comprehensive information from the specifications but no direct access to the implemented system. We then evaluate our test conditions in relation to stakeholders and the impact on these entities.

## Test Conditions Prioritized by Severity

- **Financial risk**
- **Operational disruption**
- **User experience impact**

### High-Severity Conditions

These could lead to serious consequences like financial loss, system unreliability, or user frustration:

- **Adherence to laws and regulations regarding universal design and accessibility**
  - Huge fines
- **Expired tickets being accepted**
  - Allows fare evasion, directly impacting revenue.
- **Cards with insufficient balance being accepted**
  - Impacts revenue.
- **Trip registration errors**
  - If the correct fare isn't deducted, it results in billing inconsistencies
- **Overcharging users for multiple trips in a short period**
  - Affects pricing policy and affects user trust.
- **Credit card processing failures**
  - Prevent users from completing purchases,weakens system reliability.
- **Inability to purchase or recharge tickets**
  - Blocks essential functionality and prevents users from traveling.
- **Incorrect pricing or discount errors**
  - Can lead to disputes and complaints.
- **Wrong ticket expiry settings**
  - May let users travel longer than allowed or deny access unfairly.

### Medium-Severity Conditions

These don't cause direct financial loss but still degrade usability or reliability:

- **Non-functional cancel button**
  - Interrupts navigation and frustrates users.
- **Valid cards being rejected or unknown cards not handled properly**
  - Causes confusion and degrades user trust.
- **Incorrect confirmation or summary screen info**
  - Leaves users uncertain about what they purchased or registered.
- **UI inconsistencies or unclear layout**
  - Affects flow and can lead to user errors, especially in time-sensitive situations.

## Low-Severity Conditions

These conditions represent minor usability inconveniences:

- **Minor UI layout inconsistencies**
  - Slightly affects usability but no major disruption.

# Task 3

In this exercise, we use Blackbox (specification-based) design techniques to design the most important test cases to test some of the test conditions determined in task 2. Our choice of test cases is actually based on the second principle, P2: We can't test everything in the ISTQB's testing principles. The selected test cases thus constitute a subset of many available tests, but ones with a reasonable probability of finding most of the bugs in the system. The test cases are chosen from the most important events that are possible to occur when the user interacts with the system.

**Objective:**
Verify that the system processes ticket buying, travel card refilling, balance inquiry, trip registration, and error conditions appropriately.

**Pre-conditions:**
The traveler is in front of an operational ticket machine or control module.
User interaction happens on the home screen.

**Inputs:**

- Select user category
- Select ticket type
- Confirmation
- Payment

**Description of Use-case:**

The use case illustrates user and ticket system interaction for purchasing a transport ticket. Any exceptions and exception handling are described below.

**State 1 - Start window**
**Expected user action:** Starts from the home screen and selects "buy ticket"
**Expected outcome:** The system shows available ticket types
**Transition:** To state 2

**State 2 - Select ticket**
**Expected action:** Selects the desired ticket type (Travel Card, 7-Day Ticket, 30-Day Ticket)
**Expected outcome:** Confirmation screen is displayed
**Transition:** To state 6

## State 6 - Confirmation window

**Expected action:** User confirms or cancels

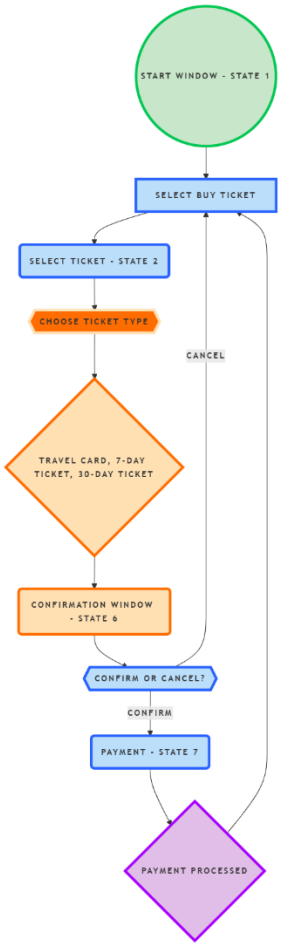
**Expected outcome:** If confirmed, to payment, else back to start

**Transition:** To state 7 or to state 1

## State 7 – Payment

**Expected action:** Pays with payment card

**Expected outcome:** The payment is successful, and a ticket is printed. System goes back to (“Select Buy Tickey”)




This flowchart illustrates the interactions between the different states.

## Equivalent Partition

We apply equivalence partitioning to the input related to user categories eligible for discounts.

**Partitions identified:**
**Valid partitions:** Rechargeable Travel Card, 7-Day Ticket, 30-Day Ticket
**Invalid partitions:** Any other input (ex: blank input)

**Test Cases EP:**

| | | | |
|---|---|---|---|
| TC-1 | Rechargeable Travel Card | Goes to deposit window | Valid |
| TC-2 | 7-Day Ticket | Goes to confirmation window | Valid |
| TC-3 | 30-Day Ticket | Goes to confirmation window | Valid |
| TC-4 | Invalid input (e.g., blank) | Error message and back to start | Invalid |

**Explanation of how EP was applied:** We identified collections (partitions) of inputs that are expected to behave the same and picked one representative of each.
**Coverage:**
**Total partitions**: 4
**Covered:** 4
**EP Coverage:** 100%

All the identified equivalence partitions (EPs) were tested. The following partitions: Rechargeable Travel Card, 7-Day Ticket, 30-Day Ticket, and Invalid Input. Each has one representative test case. As all 4 partitions are covered by TC-1 to TC-4, the EP coverage achieved is 100%

**Traceability:**

- TC-1 → REQ-01 Requirement 1: Purchase Travel Card
- TC-EP-2 → REQ-02 Requirement 2 : Purchase 7-Day Ticket
- TC-EP-3 → REQ-03 Requirement 3: Purchase 30-Day Ticket
- TC-EP-4 → REQ-04: Input validation

## Boundary Value Analysis (BVA)

**BVA Usage:** When money is deposited into a travel card, we apply BVA.
assumed that the minimum deposit is 1 NOK, and the maximum is 2000 NOK.

**Test Cases (BVA):**

| Test Case | Deposit Amount | Expected Result |
|---|---|---|
| TC-1 | 0 NOK | Error: Below minimum |
| TC-2 | 1 NOK | Accept deposit |
| TC-3 | 1999 NOK | Accept deposit |
| TC-4 | 2000 NOK | Accept deposit |
| TC -5 | 2001 NOK | Error: Above maximum |

**Coverage:**
**Boundary values:** 0, 1, 1999, 2000, 2001
**BVA Coverage:** 100%

All boundary values within and outside the valid range of deposit values were tested. Each boundary value was thoroughly examined to ensure the system responds appropriately to both valid and invalid inputs. Achieving full BVA coverage ensures confidence that the system effectively manages edge cases.

**Traceability:**
TC-1 to TC-5 → REQ 05: Recharge Travel Card

## Decision table

**Test case and preconditions:** We apply for Decision Table Testing for ticket discounts based on traveler type and day of the week.

| **Condition** | **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** | **Rule 5** | **Rule 6** |
|---|---|---|---|---|---|---|
| **Weekday** | **True** | **False** | **True** | **False** | **True** | **False** |
| **Weekend** | **False** | **True** | **False** | **True** | **False** | **True** |
| **Adult** | **True** | **True** | **False** | **False** | **False** | **False** |
| **Child** | **False** | **False** | **True** | **True** | **False** | **False** |
| **Student** | **False** | **False** | **False** | **False** | **True** | **True** |
| **Action** | | | | | | |
| **Discount Applied** | **0%** | **30%** | **0%** | **100%** | **0%** | **50%** |

**Coverage:**
**Number of rules:** 6
**Covered:** 6
**Coverage in Decision table:** 100%

We tested all six decision rules, making sure each one was represented by at least one test case. This gave us full decision table coverage, ensuring that every combination of traveler type and day of the week that impacts discounts has been carefully checked and validated.

**Traceability:**
**TC-1 to TC-6 → REQ-06:** Discount Calculation Logic
**(Test case 1 to case 6 covers the sixth requirement)**

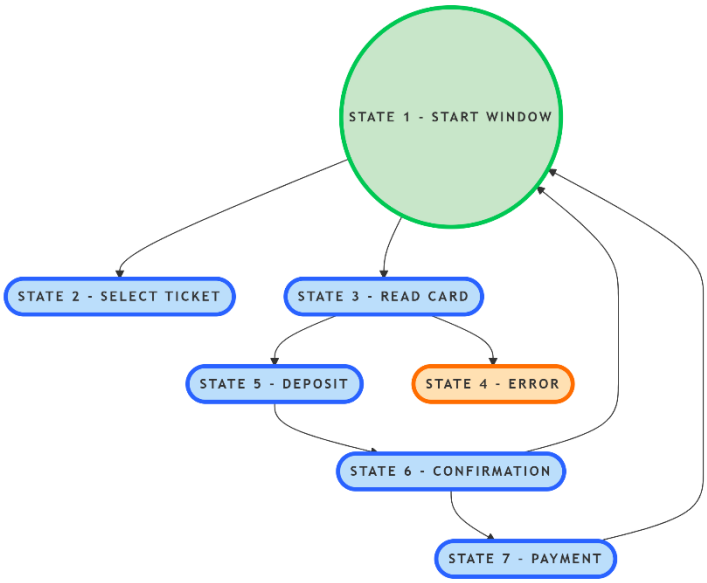
## State Transition Testing

**Goal:**
Verify that the system appropriately handles travel card refills, from option selection to payment.

**Pre-condition:**
The user is at a working ticket machine, and the start window (state 1) is where the interaction begins.

**Test inputs:**

1. **Start Window**
2. **Select Ticket/Card Window**
3. **Read Card**
4. **Error Message (Unknown Card)**
5. **Deposit to Travel Card**
6. **Confirmation Window**
7. **Payment Window**




**Coverage:**
**State coverage :** 100%
**Transition coverage :** 100%

By achieving both 100% transition coverage and state coverage, the test ensures that the entire state machine is thoroughly examined without leaving any trace behind for testing its functional flow.

# Task 4

## Priority List of Test Cases by Severity

| Severity | Test Case ID | Description |
|---|---|---|
| **High** | UI/Touch-screen Checks (implicit) | Ensure consistent layout and responsive input throughout, adherence to accessibility regulations |
| | TC-ST-5 | Ticket Validity (Expired/Invalid Ticket)<br>Prevent fare evasion and revenue loss |
| | TC-ST-1 | Register Trip (≥1hr)<br>Ensure accurate fare deduction |
| | TC-EP-1, TC-EP-2, TC-EP-3 | Ticket Purchase Flows<br>Ensure travel card and period ticket purchase functionality |
| **Medium** | TC-BVA-1, TC-BVA-5 | Boundary Deposit Errors<br>Reject below min and above max deposit values |
| | TC-ST-4 | Multiple Trips <1hr<br>Avoid double-charging |
| | TC-ST-3 | Expiration Info (Valid Ticket)<br>Ensure accurate display |
| **Low** | TC-DT-1 to TC-DT-6 | Discount Calculation<br>Correct discounts for each user type and day |
| | TC-BVA-1, TC-BVA-5 | Boundary Deposit Errors<br>Reject below min and above max deposit values |
| | TC-EP-4 | Invalid Input Handling<br>Handle blank/invalid input gracefully |

# Task 5

The ticket system must be usable by everyone, since public transportation must be accessible for everyone, in Norway this is required by law. Programs meant for the public also needs to follow the relevant legislation like the Convention on the Rights of Persons with Disabilities from the UN, EU directives, and national directives/laws. The public is made up of individuals with a wide range of circumstances, abilities, and needs, all equally deserving of consideration.

When designing the ticket system we need make personas for the different user groups. We have to include the commuter in a rush, the blind, the elderly, the tech illiterate, and children and design a system that accommodates their needs. Having these personas in mind when design the system will make it easier to create a system that works for everyone.

Some of these groups will have design interest that align, and others will be conflicting. For instance, a fast and simple interface with minimal buttons may benefit commuters in a rush and tech-illiterate users, who might find it frustrating if the ticket machine begins to spout directions at them, is contrasted by the need for a screen reader for the blind commuter. Finding the balance here will require testing with the relevant user groups and incorporating feedback from the users to find a good that covers the needs of all customers.

There is a long list of physical and technological requirements that goes into making a universally designed ticket system, and on top of that the user interface needs to be fast and intuitive. Thankfully there exists standards for the technological solutions and guidelines for the design, and adherence to these will make the decisions surrounding the design simpler, and the implementation of these solutions easier.

Just to touch on some of the requirements for the ticket system, it needs to have the interfaces needed for the severely visually impaired to use their technological aids. Screen reader, possibility to zoom, good colour schemes and a good intuitive interface. There also needs to be a physical button interface with tactile markings. The combination of these factors combined with a well-designed interface should make the ticket system usable for the blind. There is also some physical demands to the ticket machine, it must be possible for short people/children/wheelchair users to interact the machine.

The design should prioritize simplicity for children and new users, ensuring the menu is intuitive. The ticket system should cater to the general public which means everyone. Given that users are often in a hurry, the home screen should feature a shortcut for purchasing the “one zone, one hour” adult ticket, along with a more detailed purchase menu.

User interface needs to be fast and responsive so there is no annoyance in buying a ticket. Furthermore, people are often in a hurry when using public transportation, so there should be a button to buy the “one zone one hour” adult on the home screen. This makes buying a ticket less of a hassle and not annoy the consumer group who needs to buy a ticket and run

and catch their transport. Buying one single ticket should be fast and easy, however buying the correct monthly pass, with the right discount, for the right districts, is only something you need to do 12 times a year and costs considerably more than a single ticket. For this function the speed is less important and the precision more important.

While the more specialized demands for certain user groups is best tested with user interviews, and user observation, there can also be some insight into the new machines and their usability for the general public. Some interesting metrics to gather for the machine/system can be to measure of average time spent buying a single ticket, and the amount of complaints to customer support about wrong type of ticket ordered.