# Use Case Details - Structured

Final versions of the requirements documents used for walk-throughs with the main stakeholders.

IDs can be anything from consecutive numbers to meaningful acronyms about what kind of use case this is and what system it is a part of.

## UC1 Withdraw Cash

### General info

#### Description: Customer will use their card, ask for cash, receive cash and receipt.

#### Actors: Customer

#### Supporting roles/systems: Bank

#### Type: System

#### Pre-conditions: Insufficient cash in ATM - ?? Must be maximum of withdrawal RULE

### Scope info

#### Level: Goal

#### Includes:

The use cases that are extracted out of this use case and given a special name, so they can be reused. They are required to be a part of this use case. This use case can be considered a grouped use case if it includes one of a group of partial goal use cases.

#### Included in:

The use case(s) that uses this one as a necessary part of it.

#### Use cases grouped by this ID:

If it doesn’t have an included group above, then it will be a category for several use cases.

#### Grouped by:

The group that has others like this one.

* GG1 Do transaction

### Tracking info

#### Author: BA class of 1/11/19

#### Date created: 1/11/19

#### Date revised:

### Project info

#### Design constraints: ATM chassis Thiefproof SDYS9549595959

#### Priority: 7

#### Value to sponsor: Reduces overhead of employees and utilities….

#### Sponsor: CEO of bank

### Course of Events

The sequence of tasks in conversation format between actor and system. For best linking to other steps, start each number with a system task except for the trigger. Combine actor responses to system events when well. Rules are placed under the task unless they can be reused and then they are referenced and placed in a separate file.

The number of tasks per number is usually small and starts with the system or the role. Tasks are individually stated so the system/role can do multiple things but in separate sentences. The last task will prepare the state of the system so that this use case can be performed again. There will be no condition statements to branch into two separate use cases. There may be a section that is removed to a named partial use case and called an <<include>> to shorten the detailed use case.

References that can be used here to document anything other than a functional requirement are:

* T# - Text file item number – used for error messages and small prompts
* D# - Design file item number – used for web pages, full screen menus, etc.
* R# - Report file item number – used for printed or on-screen report formats
* \* - a Data Dictionary item – used to refer to data description and validation so that the detail doesn’t have to be specified here. Also bolded and colored is good.
* Rule# - Rule file item number – used to refer to process rules. Generally, this will follow one path only and another use case will pick up any other options. Some data validation rules find their way here but should be collected under the Data Dictionary. Unnumbered rules are not reusable and will just be defined below their functional requirement.

1. The use case starts when actor inserts card.
2. The system reads the card and displays screen (D#1) for entering PIN. The actor enters PIN.
3. The system request the bank to validate account. The bank returns the account data. The system logs the communication.
4. The system displays screen (D#2) for selecting transaction type. The actor selects withdraw cash.
5. The system displays a withdrawal amount screen (D#3). The actor enters amount to withdraw.
   1. RULE – Increments of $20.
   2. RULE – Maximum withdrawal amount = $200
   3. RULE – Sufficient funds in account
6. The system requests the bank to record the withdrawal. The bank confirms valid transaction. The system logs the communication.
7. The system dispenses cash. The system displays screen to take cash (D#5). The system logs the cash dispensing. The system deducts amount from ATM cash on hand.The actor takes the cash.
8. The system displays a screen (D#4) to ask for another transaction. The actor declines.
9. The system prints the receipt (D#6). The system displays a screen (D#7) to ask for another transaction. The actor takes the receipt.
10. The system ejects the card. The system displays a screen (D#8) to get card. The actor takes the card.
11. The system displays idle screen (D#9).

### Alternate flows (errors, exceptions)

The error flows are where a rule is broken, or something interrupts the normal “happy path” of the course of events. This often is during communication or other type of I/O.

* Invalid account (#3) – detail the steps for getting a valid account.
* Invalid PIN (#2) – try three times and keep card after that…
* System times out (#2, 4) - ??
  + RULE – Time out– 30 seconds
* System times out (#7) - ??
  + RULE – Reclaim cash – after timeout, the machine takes the cash back.

### Alternate flows (extension points)

An exception to branching is when there is an optional <<extends>> of a partial use case. But the use case returns to where the option was taken.

* <Name> (<number(s) in course of events where this could occur>) – <description of what to do and where to return in the course of events>
* <Name> (<number(s) in course of events where this could occur>) – <description of what to do and where to return in the course of events>

### Post-conditions

What are your tests that tell you that this is a successful completion of a use case? It may be a repetition of one of the tasks or a file or document that has been completed. But there are minimal ways to complete the goal and there are very excellent ways to complete it. Put both down. Some people use MoSCow – must have, should have, could have instead of min and max conditions.

* <Name of goal>
  + <minimum set of qualifications>
  + <maximum set of what could be>

### Notes/ Special Requirements

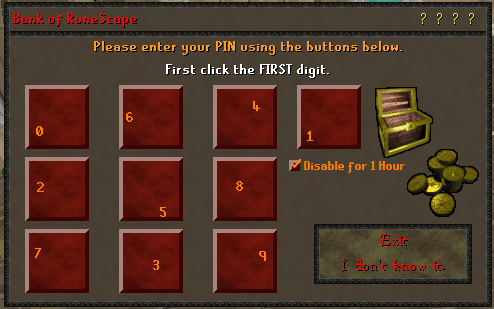
Any kind of quality, capacity, security, availability, disaster recovery information that is because of this use case. Maybe you also have ideas about design, or people who need to be checked with, etc.

* PIN must be not visible to anyone near ATM and keyboard keeps entries private.

# Design

(Separate file)

### D#1 – Enter PIN



# Data dictionary

* Authorization info
  + PIN – never visible on screen, four digits only.