

The Department of Computer Science

**CIS2158**

**Software Engineering**

Level 5

Coursework 1

2020/2021

|  |  |  |
| --- | --- | --- |
| **Group 2C** | **Student Name** | **Student ID** |
| Sean Carlin | 24609650 |
| Lewis Patterson | 24625418 |
| Jack Fearon | 24248134 |

Table of Contents

[Introduction 3](#_Toc68239257)

[Requirements Analysis and System Design 4](#_Toc68239258)

[Context Diagram 5](#_Toc68239259)

[Vending Machine Context Diagram 5](#_Toc68239260)

[Use Case Diagrams 7](#_Toc68239261)

[Vending Machine Use Case Diagrams 8](#_Toc68239262)

[List of candidate objects and responsibilities 11](#_Toc68239263)

[Object Sequence Diagram 12](#_Toc68239264)

[Class Diagram 16](#_Toc68239269)

[State Transition Diagram 17](#_Toc68239270)

[Vending Machine State Transition Diagrams 17](#_Toc68239271)

[Object Collaboration Diagram 21](#_Toc68239272)

[Activity Diagram 24](#_Toc68239278)

[Vending Machine Activity Diagram 26](#_Toc68239279)

[Implementation 29](#_Toc68239280)

[Demo Design Features 29](#_Toc68239281)

[Testing 29](#_Toc68239282)

[Conclusion 29](#_Toc68239283)

[References 30](#_Toc68239284)

[Work Plan 32](#_Toc68239285)

## Introduction

This report contains the information relevant to the design and development of the given project, which is to create the software behind a drink vending machine for ‘Edgehill University Café’ that will allow their customers to purchase and vend a chosen drink from the machine. While designing and developing the system those involved will have shown their ability to effectively evaluate an unfamiliar software engineering problem by outlining system requirements and any key considerations that need to be taken into account to be latter be able to create a range of UML diagrams that detail how the system will interact with the customers, staff and even how the different parts of the system will interact within itself and how the processes those parts hold will be used and when. As well as fully developing said system and producing relevant testing documentation to catch any issues that might arise.

The structure of this report starts with an analysis of requirements of the system that explains why an analysis of requirements is important to the development and creation of a software solution. Following this will be several sections that outline the design of the system itself using industry standard UML(Unified Modelling Language) diagrams to create an accurate detailed picture for the developer that is tasked that is tasked with the implementation of this system, were in the section after this, that developer will go into detail about the process undertaken in implantation, a record of features that have been added and what those features are responsible for, also listing in detail through the testing process to make sure no problems are missed while the solution is still within development.

## Requirements Analysis and System Design

A requirements analysis is the process of defining user expectation for a system (Grady, 2006), such as what the user will be able to use the system for, how the user will interact with the system and how the system will meet these expectations. The requirements of the users in this case would be the functioning of a vending machine which allows the users to purchase drinks, either through cash or loyalty card.

In our system, different diagrams illustrate how the user interacts with the system, these are mostly high-level diagrams such as the use case diagram and context diagram which show direct user interaction. However, there are other in-depth diagrams such as the activity diagram which highlights the processes a user takes while also detailing how the system is operating.

Alternatively, a systems analysis or design is the process of defining how the system will operate to meet the needs of the users. (Maciaszek, 2007) System design details how the system works and is of more use to the developers of the system, than the users as it allows them to understand what to implement and how to implement it.

Like requirements analysis, there are diagrams in our project which are more system design orientated such as the state transition and object collaboration diagram which are focused more on the components of the system and how they change, rather than how the user interacts. Although, activity diagrams can also give insight to a system design as they describe the processes a system must take to complete its processes with validation and choice implemented.

## Context Diagram

A context diagram is a high-level UML diagram which illustrates how different sections of a system interact with each other, as well as with outside entities connected to the system.

Within a context diagram, the system is contained within a rectangle. However, there can be smaller rectangles within the system to group certain processes, such as a front end, back end, database, controller, etc. Each of these sections can have multiple processes within them and can connect to other sections or external actors as well. (Eriksson, Penker and Lyons, 2004)

External actors in a context diagram are entities which are not part of the system being developed. (Pilone and Pitman, 2005) However, they can utilise or be utilised within the system to complete system processes.

Context diagrams can help to define the scope of a project, allowing developers to have a comprehensive understanding on how long the system will take to develop, the functionality, validation and possible software the system will need to function, as well as allowing developers to understand if a system is feasible. (Mcdonald, 2018)

Although context diagrams do provide the developers with a greater understanding of the system from a high level, (Chonoles and Schardt, 2003) there are other diagrams which can provide developers with a more in detail look to the individual processes that happen in the system (Schmuller, 2004), allowing them to understand how to develop it better, such as an activity diagram.

The context diagram for the vending machine system, is show below in figure 1:

### Vending Machine Context Diagram

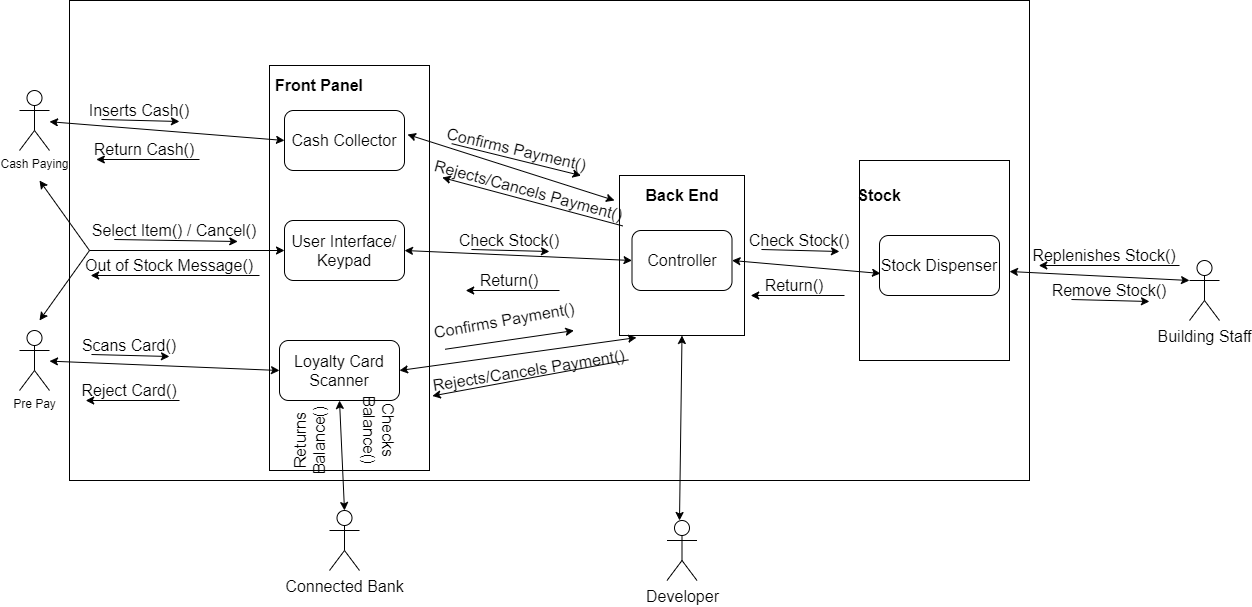


Figure 1 Context Diagram

Our context diagram depicts how the system is interacted with and operates with external entities. The first of these being the customers, where pre-pay customers scan their card which is then checked via the Loyalty Card Scanner to the Connected bank to validate credit. Alternatively, cash paying customers have their cash validated through the Cash Collector.

Either customer can then interact with the UI/keypad to choose a drink or cancel the purchase of a drink. When choosing a drink, the controller checks the stock of the selected drink and returns an out-of-stock message, if the drink is out of stock.

On the other side of the processes is the restocking and removal of any drinks, completed by the building staff, this is done in case of low stock or an error when vending. Finally, the developer is connected to the back end, implementing functionality for the system.

## Use Case Diagrams

A Use Case diagram is a behavioural UML diagram which allows designers to illustrate how users interact with a system. Use case diagrams denote users as external entities of the system called ‘Actors’. (Stevens and Pooley, 2008) Although users are displayed in use case diagrams, the main point is to illustrate the internal processes that they interact with. (Jürjens, 2005)

Use Cases are the functions of the system which the user can complete or interact with, such as choosing a drink in a vending machine. They can also account for scenarios where the system cannot proceed, such as a customer trying to pay for an item at a shop but having insufficient funds. Use cases which can check information like this, are usually extended onto the initial process through the ‘<<Extends>>’ annotation and can also be optional in a process.

Like the extension of use cases, there is also the inclusion of use cases, annotated by ‘<<Includes>>’. The inclusion is used to separate a large use case function into separate use cases which would occur inside the large use case. (Pilone and Pitman, 2005)

Use case diagrams can be created for a system, as well as specific sections of a system to understand individual processes better. (Fowler, 2004) This gives developers more insight into how the system functions. As these diagrams are a high-level design, most people can read them without experience in software development. This can allow designers in industry to illustrate to boards how a system will work.

However, in these diagrams many aspects for development are not considered such as the creation of objects, encapsulation, states of objects during certain processes, etc. (Donald G, 2021)

The Use Case diagrams for our vending machine are shown in Figure 2, 3 and 4.

### Vending Machine Use Case Diagrams

*1 – Drink Selection*

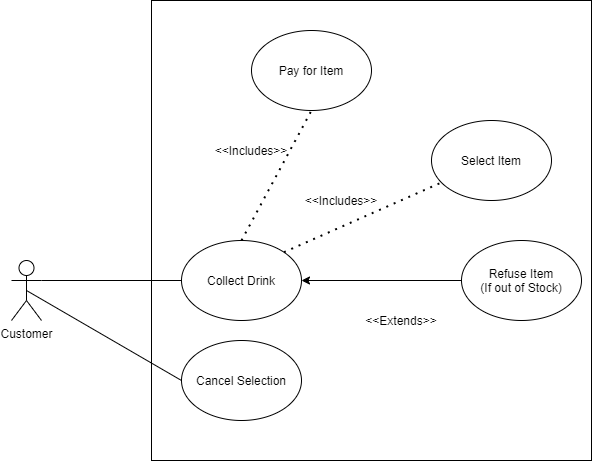


Figure 2 Use Case 1 – Drink Selection

This use case illustrates how a customer selects and collects a drink. The customer collects their drink by selecting an item and paying for it. Customers are refused their selected drink if it is out of stock, they can also cancel the selection of a drink at any time.

#### Selecting a Drink

Select an Item

If the item is out of stock

Refuse the item

Pay for the item

Collect the drink

The selection can be cancelled at any time

*2 – Cash Payment*

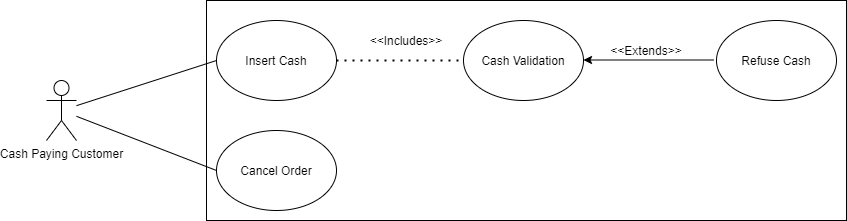


Figure 3 Use Case 2 – Cash Payment Validation

This use case illustrates how payment from cash paying customers is processed. Cash paying customers enter an amount of cash in the Insert Cash use case, this cash is then checked in the Cash Validation use case, if the cash is insufficient or counterfeit, it’s refused.

Cash Paying customers can cancel the payment at any time.

#### Paying For a Drink with Cash

Cash paying customers insert cash

Validate the cash payment

If the payment is insufficient

Refuse it

Payment can be cancelled at any time

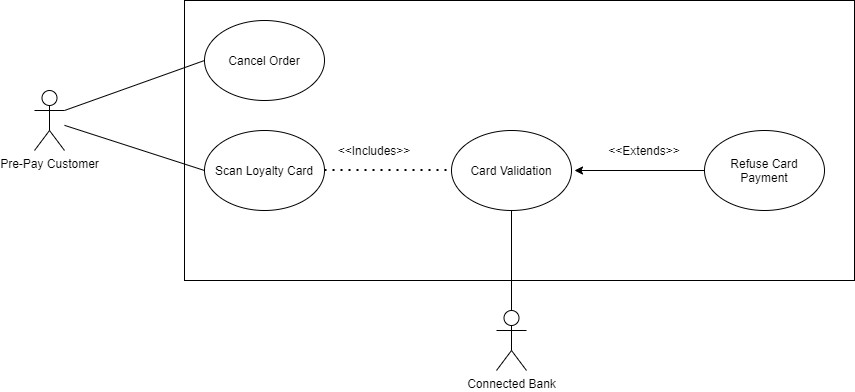
*3 – Card Payment* 

Figure 4 Use Case 3 – Card Payment Validation

The Card Payment validation is like the Cash Payment validation. However, includes another external actor, the bank of the connected loyalty card. Loyalty card holders scan their cards through the Scan Loyalty Card use case.

The cards are then validated in the Card Validation use case. This checks the Connected Bank to see if there is enough credit, if not, the card is refused.

Pre-Pay customers can cancel the payment at any time.

#### Paying For a Drink with Loyalty Card

Pre-pay customers scan their loyalty card

Validate the card through the bank account connected

After 20 seconds, of if there is not enough credit

Refuse the card payment

Payment can be cancelled at any time

## List of candidate objects and responsibilities (Lewis/Sean)

The candidate objects and their responsibilities are presented in Table 1.

|  |  |
| --- | --- |
| **Candidate Objects** | **Responsibility** |
| Cash Scanner | Uses light to determine the size of a coin and electromagnets to detect the metal in a coin. |
| Loyalty Card Scanner | Check the bank connected to the loyalty card, to confirm the connected account has enough credit. |
| Stock Sensor | A slot sorted stock sensor which counts how many slots have been moved forward (Slots are moved forward after a drink is dispensed). |
| User Interface / Keypad | Allows a customer to select a drink, view the prices, cancel the order and complete their payment. |
| Bank | Returns the credit of the bank account linked to the loyalty card of pre-pay customers. |

Table 1: Candidate objects identified and their responsibility

## Object Sequence Diagram

Representing object interactions through a flow of timed events required to accomplish system functionality. Interactions are presented in a step-by-step appearance to ensure the user understands each process. (Greeff, 2004). These diagrams are also referred to as Interaction diagrams, event diagrams and event scenarios.

### Diagram Notation

Sequence diagrams are made up of various components that help define the overall system scenario. The basic components are as follows: (Examples presented in Figure 5)

|  |  |
| --- | --- |
| * Actor | * Class (Object) |
| * Activation | * Lifeline |
| * Messages (*Method Invocation*) |  |

#### Actor

An entity that represents a role that would interact with the subject. In other words, an instance of an actor is not part of the corresponding subject. The actor is often seen as the *User* of the subject.

#### Class (Object)

The class represents the roles which the *Object* plays. In others words the objects are the top of the diagram (which represent the lifeline) represent the *class roles.* (Bidgoli, 2003)

#### Activation

Sequence diagrams have an advantage to represent the passage of time graphically (Britton, 2005). To facilitate these timed intervals, activation blocks are required to represent the operations taking place within said intervals.

#### Lifeline

The interactions within a sequence diagram are represented via *Lifelines*, which are used to capture system interactions as they occur across several parts of the system (Friedenthal, 2015). Throughout lifelines, messages are exchanged with other lifelines and represent an invocation of an operation (*Method Invocation).*

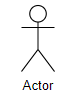
**Activation**

**Class**

**Lifeline**

**Messages**

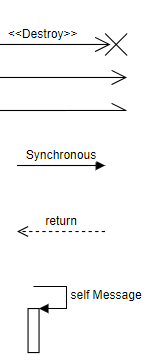
Figure 5 - Notation Examples



#### Message Patterns

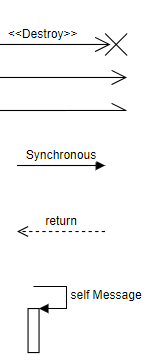
As previously mentioned, *Method Invocation* plays a key role when communication between objects occurs. (Bidgoli, 2003) The messages communicating between objects are linked via a variety of different arrows. Each arrow represents a different meaning and function:

##### Synchronous Message



**1.**

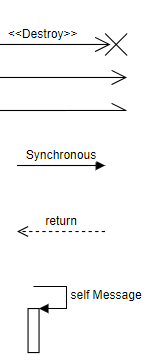
Requires a response before the interaction can continue. Defines a communication between lifelines.



**2.**

##### Asynchronous Message

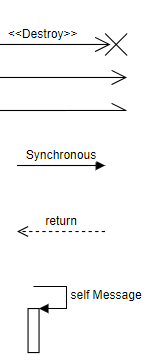
Does not need a reply for an interaction to continue. It would continue without an immediate response from the corresponding lifeline.



**3.**

##### Reply/Return Message

Replying lifeline from a corresponding interaction.

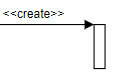


**4.**

##### Self-Message/Recursive

Invocation of a message from the same lifeline.

##### Create Message



**5.**

This message creates a new object, the arrow points to the new object created.

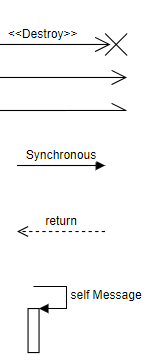
##### Found Message

**6.**



A message received from an unknown recipient.

##### Delete Message



**7.**

This type of message *destroys* an object.

##### Lost Message

**8.**



A message sent to an unknown recipient.

### Benefits within the Software Engineering Industry

These diagrams have proved to be beneficial towards its users, specifically within big projects. Sequence diagrams are great tools in a project as it shows the users what needs to happen in a step-by-step manner (Greeff, 2004). These diagrams are good at presenting what is happening, for extracting requirements and for interacting with customers.

Within UML, there are many diagrams that are presentable to a user and can be overwhelming, as they contain notation and information which can be confusing. Thus, making the simplicity of the sequence diagrams more valuable to the software engineering industry, they are easy to generate and maintain, making changes within the sequence diagram for a system design can be achieved easily. Sequence diagrams decompose the lifeline into many levels of abstraction to form a presentable and eligible diagram for the user, instead of everything being in one huge diagram. (Douglass, 2015)

Sequence diagrams help the software engineering industry through its use as a collaboration tool and for documentation. Documentation aids the dynamic view of the system design, as mentioned previously, *in many levels of abstraction.* As a collaboration tool, sequence diagrams enable the team to discuss the interactions between entities. This enhances productivity. Furthermore, the diagram assists the development teams break down each step, breaking down each step reveals potential architectural, interface and logical problems early before serious development commences.

### Drawbacks of Sequence Diagrams

The sequence diagram enhances production within the system development, however there are some minor drawbacks. Some system sequence diagrams can become too complex if there are too many *lifelines* involved, which can be overwhelming for the users. Other minor drawbacks include the user of message notations, each need to be different to represent each sequence, which again can become complex.

### Vending Machine Diagrams

#### Selecting a Drink

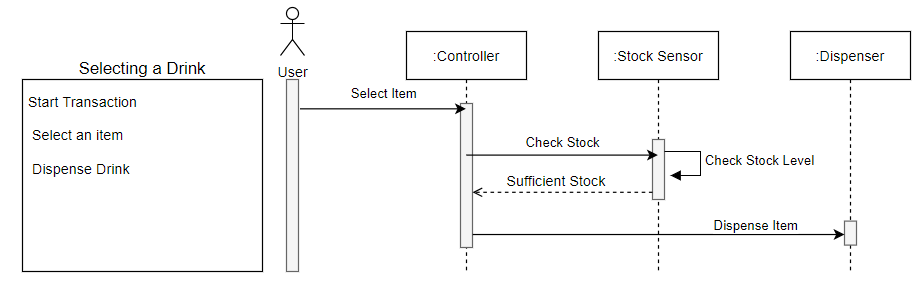


Figure 6 – Object Sequence Diagrams - Selecting A Drink

As presented in figure 6, the user will select an item via the :controller(keypad), the controller will proceed to tell the stock sensor to check the stock level of the item selected. Once this is confirmed with the stock sensor, the vending :controller will dispense the item.

#### Successful Cash Payment

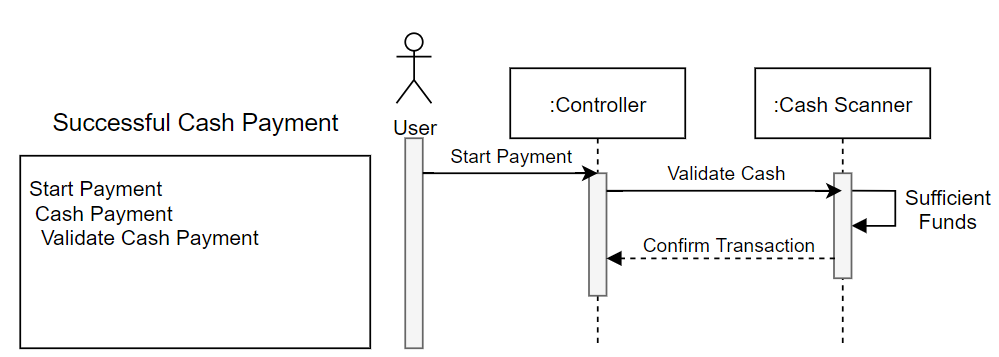


Figure 7 – Object Sequence Diagrams – Successful Cash Payment

As presented in figure 7, the user will select their payment method via the controller, in this case Cash has been selected. The controller will request the :CashScanner to confirm sufficient funds for the item selected for purchase and will confirm the transaction, resulting the drink being dispensed.

#### Successful Card Payment

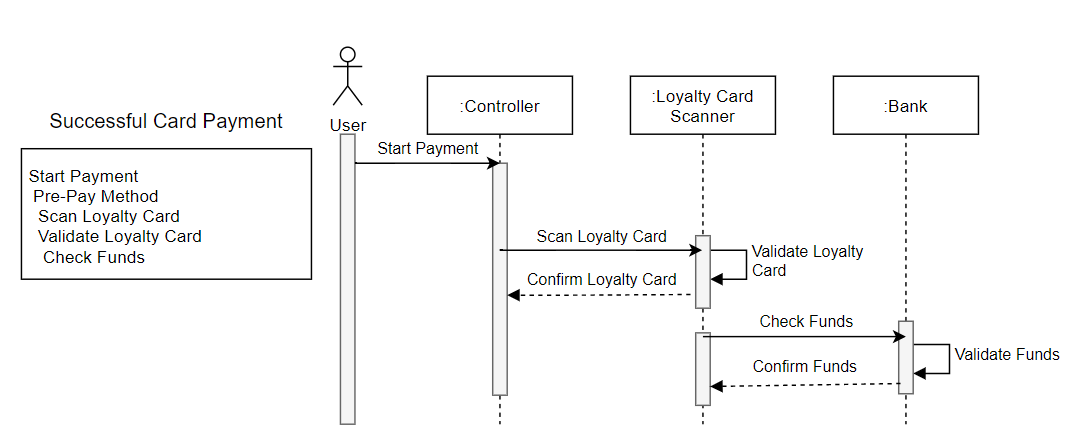


Figure 8 – Object Sequence Diagrams – Successful Card Payment

As presented within Figure 8, the user has selected the card payment option via their loyalty card, which has their bank account linked for quicker purchases. The user will scan the loyalty card which is validated and then confirmed by the :LoyaltyCardScanner. At the same time, the loyalty card scanner has the :Bank validate the sufficient funds within the linked bank account to ensure a successful transaction.

## Class Diagrams

A main artefact within the *Unified Modelling Language (UML)*, the class diagram is a static object-oriented structural diagram that presents the system by showing the systems classes, attributes, methods, and interrelationships with other classes (Ambler, 2005). Defining the structural data and behaviour within each class presented. In more detail, the class diagram is a logical entity in UML showing the instances or objects executed at run time. They are also used for general *conceptual modelling* and translating the detailed models into programming code, as well as *Data Modelling* (Sparks, 2008)*.*

Class diagrams involve numerous principles that make it unique:

* Compartments
* Behaviour
* Relationships

#### Compartments

Class diagrams are made up of 3 compartments. The top box is the name of the *class* it represents, which should be in bold centred text, with the first letter being capitalized. The middle compartment contains all the *attributes* within that class, which are left aligned and first letter being lowercase. The bottom compartment contains the *methods* that class can execute, these are also left aligned, and the first letter is lowercase, as presented in Figure 9.

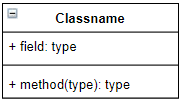


Figure 9 – Class Example

#### Members

Mentioned previously, the class diagrams are broken into compartments. Each member within the compartments is represented via *UML Mechanisms.*

##### Visibility

When specifying class member visibilities, for example, an attribute or method/function. Notations must be placed before the members name to present its visibility (Holub, 2017). The notations are as follows.

|  |  |  |  |
| --- | --- | --- | --- |
| + | Public | # | Protected |
| - | Private | ~ | Package |

#### Behaviour

The behaviour of the class is defined via the methods being used within. As mentioned with *Visibility,* these classes can be externally visible, via public. Visible to children, which for example is a master class with sub-classes within object-oriented programming – inheritance, which is protected. Or it can be fully hidden using the protected notation. With this approach, highly maintainable structural units can be achieved (Sparks, 2008).

#### Interfaces

Interfaces are like classes but have nothing but pure virtual functions. The primary icon for an interface is like a class but with special denotation called a stereotype. Which would look like <<type>>. This element is used to extend the UML Language. When you see this stereotype used above a class, it represents that class is special and conforms to a rather rigid specification (Martin, 2009).

#### Relationships

Class diagrams are used to present the relationships between numerous classes in a system. Relationships within classes begin with inheritance.

##### Dependency

Rarely the relationship between 2 classes is very weak and are not implemented with member variables at all. Instead, they might be implemented with member function arguments.

##### Association

A relationship between 2 classes indicating that at least one side of the relationship inherits the other by using its methods or manipulating data. (SourceMaking, 2007) It could either be a functional or structural relationship. A functional relationship requires Class B to do something for Class A. A structural relationship requires Class B to be something for Class A.

##### Aggregation

A form of association that implies the collection of one class of objects within another. It represents a part-whole or part-of relationship. It may not involve more than 2 classes. Furthermore, as aggregation is a form of association there is not much difference when it comes to implementation (Martin, 2009).

### Benefits withing Software Engineering Industry

Classes within software engineering involve *Data Hiding* using *Encapsulation* and *Inheritance,* making projects much easier to manage. These diagrams are simple and quick to read, they are also much easier to create. Class diagrams in the software engineering industry are the foundation to creating a system, whether it is big or small.

Within system development, class diagrams give the developers a sense of orientation, providing a detailed insight to the overall structure of the systems being developed. The detailed insight to the system presents the developers with an overview of the collaborations occurring among the different system elements, including their properties and relationships. (microTOOL, 2021)

### Drawbacks of Class Diagrams

Within a production environment, class diagrams are firstly considered, although, it comes with some cons. Sometimes when a large system is under development, the class diagrams could be overwhelming towards the developers. This resulting in frustration towards the team due to the overall structure of the diagram (Pedamkar, 2020).

### Vending Machine Class Diagram

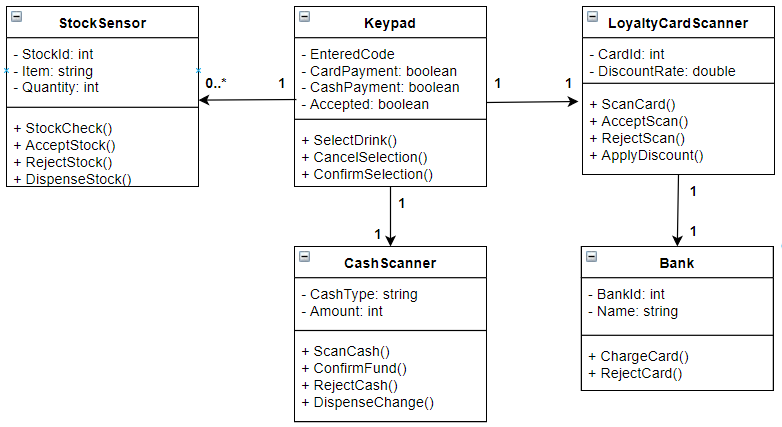


Figure 10 – Class Example

As presented within Figure 10, the overall system layout consists of 5 classes. Some inheriting others for the methods and data. Each class stores the appropriate data types with their attributes for components within the system. The Keypad class is the heart of the operation as it sends and receives the commands throughout the system. Each class has their own methods that can be used via inherited classes.

## State Transition Diagram

A state transition diagram highlights the different state an object can be in during system processes. (Holt, 2007) The diagram also details how and when an objects state will change, this can be in the case of events and actions.

The transition from one state to another in a state transition diagram is depicted by an arrow from the previous state to the new state with the event that caused this change highlighted above the arrow. (Chonoles and Schardt, 2003)

State transition diagrams can further depict how a state can change by implementing guards into a state. A guard is essentially a Boolean which needs to have its conditions met to continue to the next state. (Alhir, 1999) Once an event is triggered which would typically result in the transition to the next state, the guard is first checked and if true, allows the transition to occur, otherwise exits the transition, allowing developers to understand where validation is needed in development.

Furthermore, conditional connectors can be placed within states to check which state an object will transition to based on a condition. Essentially, a conditional connecter depicts an if statement with the different states of the object as the results.

State transition diagrams can be effective in industry as they provide an in depth look into each object and how it will change throughout the use of the system. Furthermore, it allows developers to understand the events they need to be aware of to activate changes in state, as well as the state that will become of the transition.

The state transition diagrams for the vending machine objects are show below in figures 11, 12 and 13.

### Vending Machine State Transition Diagrams

*1 – Cash Scanner*

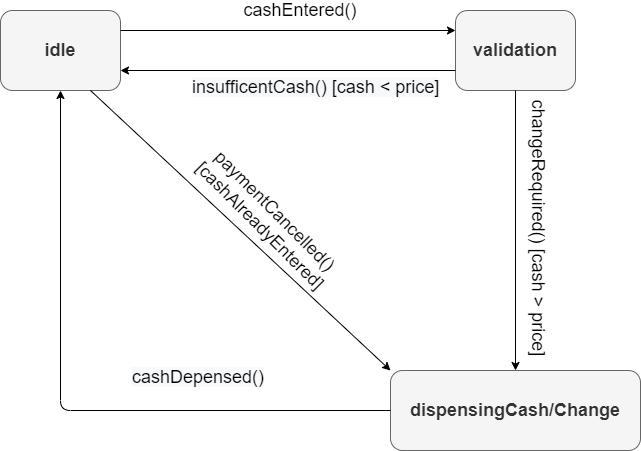


Figure 11 State Transition 1 – Cash Scanner

This diagram begins with the cash scanner in its idle state, once cash is entered the scanner moves to the validation state, where it checks the cash entered compared to the price of the drink.

If there is not enough cash then the object is returned to the idle state, otherwise if more cash is entered than the price of the drink, the scanner moves to the dispensingCash/Change state. Once the cash is dispensed, the object returns to the idle state.

Users can also cancel their payment from the idle state, which will also invoke the dispensing state as any money they had entered will be returned to them.

*2 – Loyalty Card Scanner*

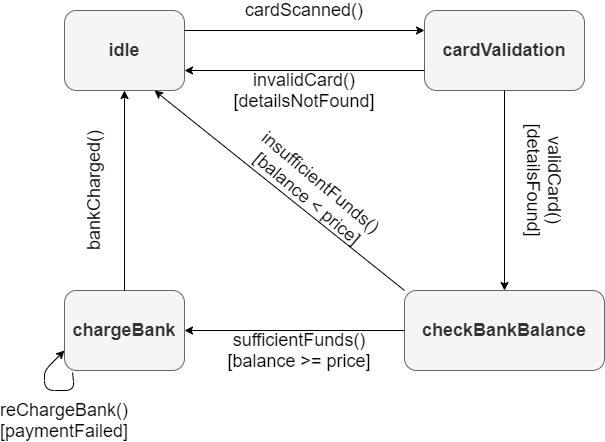


Figure 12 State Transition 2 – Loyalty Card Scanner

This diagram begins in the idle state and moves to the validation state once a loyalty card is scanned. If the card is invalid, it returns to the idle state. However, if the card is valid the object moves to the checkBankBalance state instead.

If the bank’s funds will are insufficient, the user is returned to the idle state, otherwise the object moves to the chargeBank state. A guard is implemented in this state in the case a payment would time out. If the payment doesn't fail, the scanner returns to the idle state.

3 – Stock Sensor

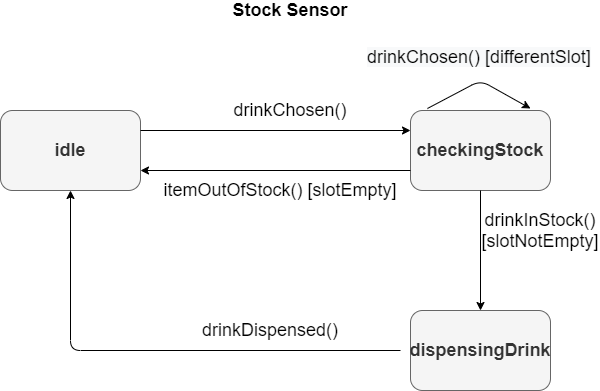


Figure 13 State Transition 3 – Cash Scanner

The stock sensor diagram begins in the idle state and moves when a drink has been chosen into the checkingStock state where the stock of the selected drink is chosen. If a different drink were to be chosen during this process, that drink is checked instead.

If the chosen slot has no drinks, the sensor returns to idle. Otherwise, the sensor allows for the stock to be dispensed by moving to the dispensingDrink state. Once this process has been completed, the sensor returns to idle.

## Object Collaboration Diagram

An object collaboration diagram depicts the different relationships between the objects that reside within the system. Unlike an Object Sequence Diagram which shows the flow and timings of messages, an Object Collaboration Diagram shows the architecture of the objects within the system. (Javapoint, n.d.)

These diagrams are useful as they clearly depict the relationship the objects within a system have, as well as how the system is interacted. This is especially useful as it can be used by the programmer that is implementing the system to better understand it, and when paired with an Object sequence diagram, even though they do display similar information, they will be able to understand the entire architecture of the system and the relationship the objects hold within it, as well as the process that the system is interacted with including the timings and specific steps those processes take.

The object collaboration diagrams for our vending machine are shown in figure 14, 15 and 16.

### Vending Machine Object Collaboration Diagrams

*Selecting a Drink*

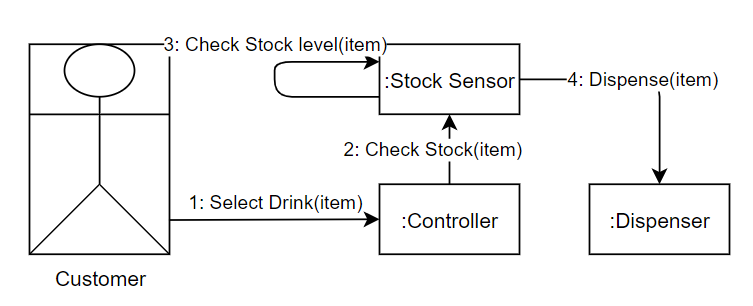


Figure 14 Object Collaboration 1 –Selecting a Drink

*In the diagram above, the customer interacts with the :User Interface class and selects a drink, which then sends a message to the :Vending machine class to send a message to the stock sensor to check if the drink the user selected is available, and then will send a message to the :Vending Arm class to vend the specific drink the user selected, once done a message will be sent to the :Dispenser class to dispense the drink, then another message to open the cover and eventually to then close the cover.*

*Payment with Loyalty card*

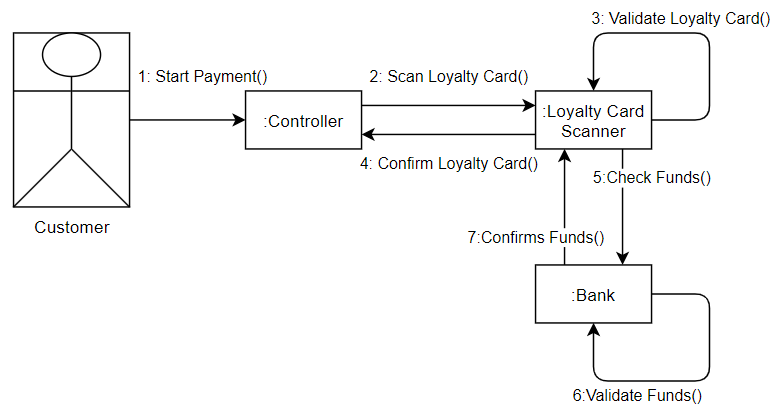


Figure 15 Object Collaboration 2 – Payment with Loyalty Card

*Payment with Cash*

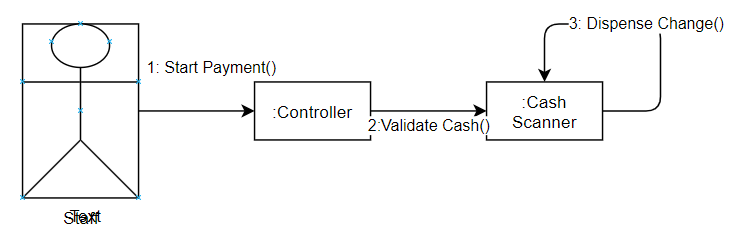


Figure 16 Object Collaboration 3 – Payment with Cash

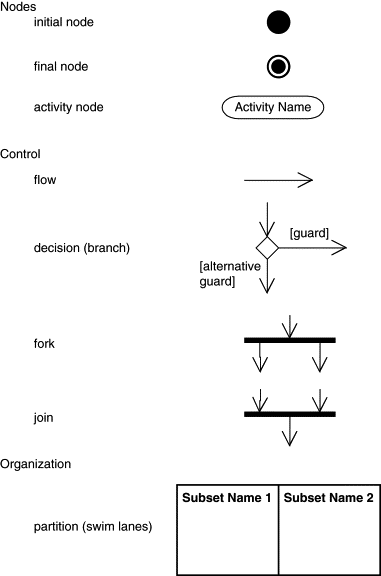
## 

## Activity Diagram

An activity diagram is a diagram which illustrates a start to end process of the system, by highlighting the activities that occur throughout and the flow of control from one activity to the next. (Halpin and Morgan, 2008) Activity diagrams help to show the dynamic aspects of a system and how the system reaches the end point, sometimes through different routes. (Osis and Donins, 2017) Furthermore, an activity diagram can show validation in a system.

Activity diagrams begin with an initial node and end with a final node. However, multiple final nodes can be implemented in an activity diagram to show that the process can end in different ways. (Halpin and Morgan, 2008) Activities in the system are depicted by activity nodes, each activity node represents a process that is occurring within the system.

These nodes are show below:

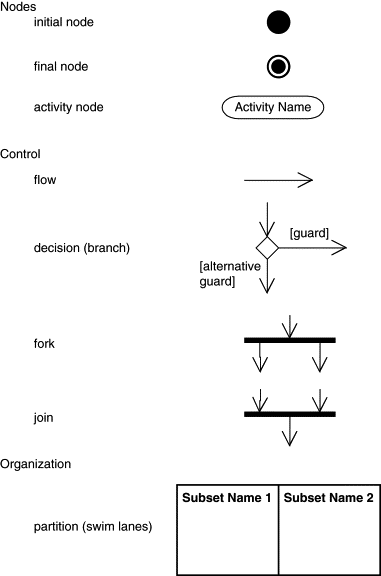


(Teorey, 2011 [Activity Diagram Notation]). In: Teorey, T., 2011. *Database modeling & design* San Francisco, Calif.: Morgan Kaufmann.

Activity Diagrams can also include decision branches which be used as a point of validation, (Banos, Lang and Marilleau, 2015) similarly, the node can be used for choice where a system may go in different routes dependant on a condition.

To depict which route a system will follow, guards are used although, some lines at a decision branch may not have guards, this is usually only if all others do, meaning that if the conditions for all other guards are not met, to follow the line with no guard. (Wazlawick, 2014)

The notation for a decision branch and it’s guards is show below:



(Teorey, 2011 [Activity Diagram Notation]). In: Teorey, T., 2011. *Database modeling & design* San Francisco, Calif.: Morgan Kaufmann.

Activity diagrams can also accommodate for different activities happening at the same time as other activities using forks and joins.

Forks are when the flow of processes is split into separate lines with different activities happening on each at the same time. (Friedenthal et al. 2015) A fork can be used for validation in a system, an example of this would be timing out when completing a transaction.

Joins are when different incoming flow lines become one outgoing flow line, that can be used to reconnect the lines separated at a fork. (Douglass, 2014) However, unlike forks, joins can have different reasons for occurring. These being ‘{AND}’, ‘{OR}’ and ‘{XOR}.

{AND} – This type of join occurs when all incoming flows reach the join.

{OR} – This type of join occurs when one of the incoming lines has reached the join.

{XOR} – This type of join only occurs when one of the incoming lines has met the join, but the other has not. This is useful for validation, as a process can either fail or succeed. (Gunter, 2011)

Activity diagrams can highlight the processes between the system and outside entities. This is known as a swim lane and is used to separate the different entities from the system. (Bastos and Ruiz, 2002) The processes performed by the outside entities are shown in their own respective lanes and flow lines can cross these lanes to show how they connect to the system.

Activity diagrams provide developers with the understanding of the conditions which need to be met for the system to move forward (Britton and Doake, 2005). Furthermore, it can give developers an understanding of exactly when to begin and end processes, along with where to implement cancellation or time out options.

Knowing this before development, can save developers a lot of time when creating the system as well as ensure that errors do not occur later in the system.

The activity diagram for our system is show below in figure 17:

### Vending Machine Activity Diagram

*Purchasing an Item*

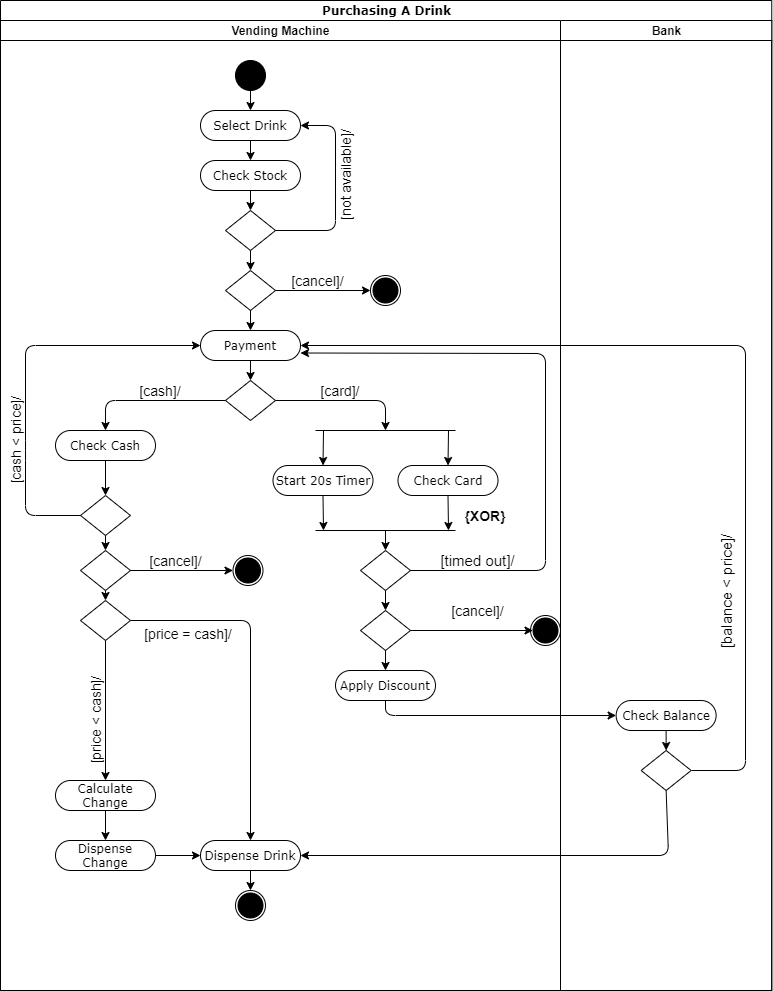


Figure 17 Activity Diagram – Purchasing an Item

The first activity is to select a drink, the system then checks the stock and if the item is out of stock, the system returns to the select a drink phase. Otherwise, the system proceeds. At this stage the customer can cancel. If they proceed, then the payment section begins.

If the customer chooses to pay by cash, the system checks the cash. If insufficient the customer is returned to payment. However, if the cash is sufficient, they are then offered the choice to cancel before finishing, otherwise the system will either calculate and dispense their change provided and then dispense the drink, or else simply dispense the drink if they have entered the exact cash.

The alternative payment method is card. Once the card is scanned, a fork splits the flow line and activates a timer of twenty seconds while checking the card. In the case that the timer has elapsed before the card has returned valid, the customer will be returned to the payment option. Alternatively, if the card is validated, they are given the option to cancel.

If they continue, the discount is applied as they are a loyalty card holder, and then the balance of the bank is checked through a swim line. If the balance is insufficient, they are returned to payment, otherwise their drink is dispensed.

# Implementation

The vending machine demo will be created using the C# language, specifically the Windows Form App (.NET Framework). This allows us to utilise the toolbox of C# for much quicker GUI design and more efficient event triggering, than alternatives such as Java where a GUI would have to be built in the code itself or using NetBeans IDE.

The development model used in the implementation of the system was the prototyping. Which is the process of creating a workable model of the system and evaluating this against the requirements of the system and from. (Budde, et al.) The prototype can then be reworked and reassessed or accepted. We began creating a functional prototype, then implemented more user-friendly GUI using the original functionality and finally implemented the additional features.

We have implemented some additional features, mainly the admin panel where a staff member can edit the stock of a drink in a vending slot, as well as the price of the item itself.

## Demo Design Features

The design of our Vending Machine Demo is shown below in figure 18 & 19, highlighting the customer panel which is where the customer would purchase a drink and the administration panel where drinks can be restocked or edited. Details of these features in the screenshots are shown in table 2 The video of this demo is shown in the files within the zipped folder.

*Panel 1 – Customer Panel*

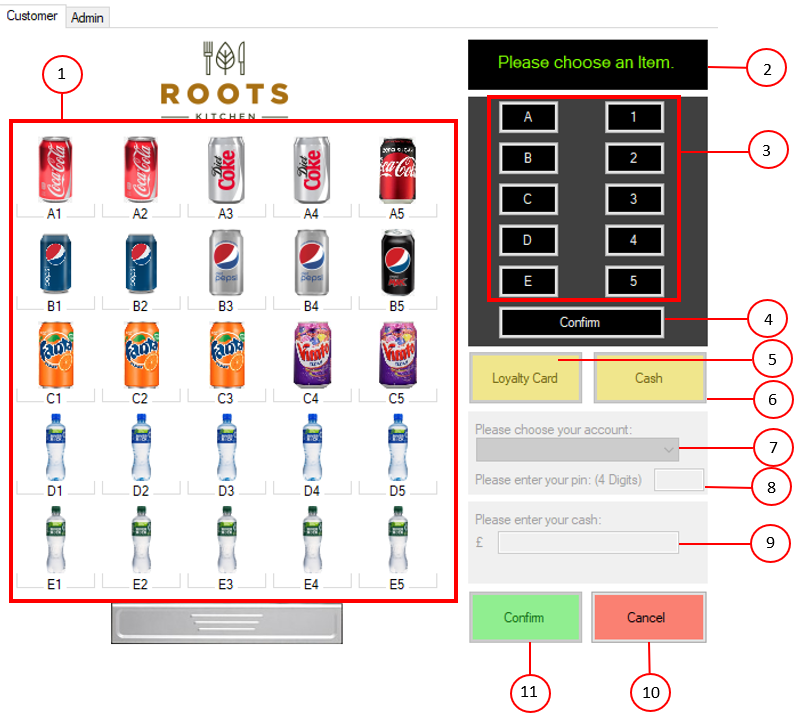


Figure 18 Panel 1 – Customer Panel

|  |  |
| --- | --- |
| **Feature** | **Responsibility** |
| 1 | Images of all items in slots, which can be chosen when selecting a drink. |
| 2 | A text box which updates through the drink purchase. Telling customers to choose an item, if an item is out of stock, the price and stock of an item, error messages with payment, change dispensed amount and a confirmation of the dispense. |
| 3 | Buttons which can be used to select a drink, numbers can only be chosen after a letter already has been chosen. |
| 4 | Confirms the drink selected, disabling feature 3 and 4, then moves to the payment section enabling feature 5 & 6. |
| 5 | Button to enable the loyalty card panel with features 7 & 8. |
| 6 | Button to enable the cash panel with feature 9. |
| 7 | A combo box with all loyalty card holders' names, allowing the customer to choose their account. |
| 8 | A text box which allows customers to enter the pin connected to their loyalty card to confirm that it is theirs and purchase the drink. |
| 9 | A text box which allows customers to enter the amount of cash they want, confirms the value is cash and checks if it is sufficient or not, if so, it allows the purchase of a drink. |
| 10 | Allows the customer to cancel at any stage, returning the system to its initial state which is the state in the screenshot in Figure 18. |
| 11 | Confirms the purchase of a drink, after the payment validation and calculates the change. This starts a 3 second timer which resets the timer after it elapses. |

Table 2: Features of the drink vending machine and their responsibilities

*Panel 2 – Customer Panel*

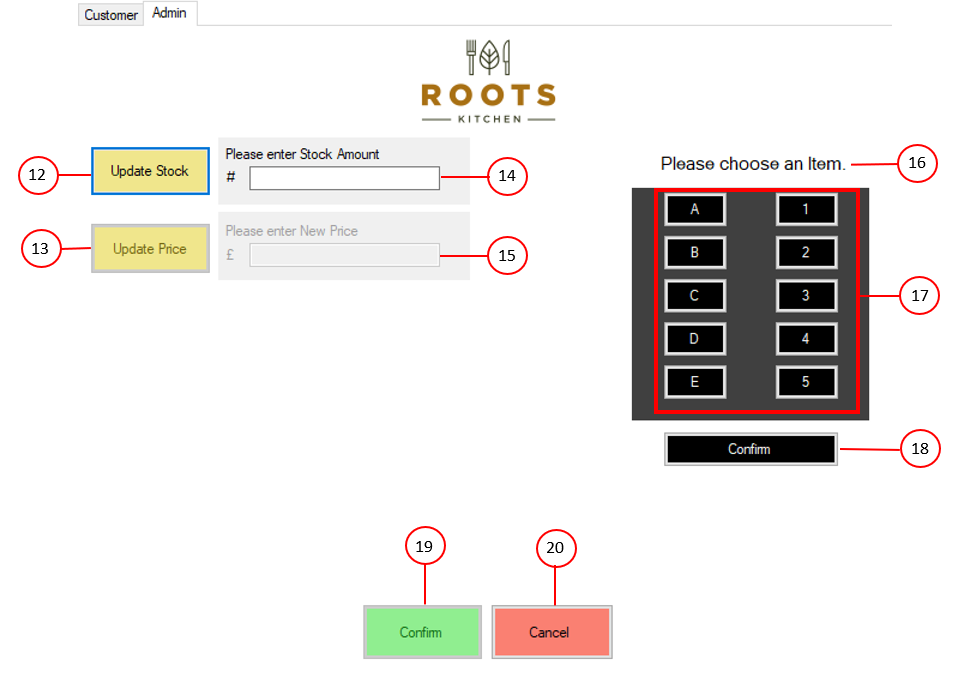


Figure 19 Panel 2 – Customer Panel

|  |  |
| --- | --- |
| **Feature** | **Responsibility** |
| 12 | Enables the stock panel and feature 14, disables price panel and feature 15. |
| 13 | Enables the price panel and feature 15, disables stock panel and feature 14. |
| 14 | Textbox where an admin can amend the amount of stock of a drink. |
| 15 | Textbox where an admin can amend the price of a drink. |
| 16 | Similar to feature 2 for panel 1, a text box which provides the admin with messages throughout the processes such as choose an item, enter a value, invalid value, too much stock entered, and change confirmed. |
| 17 | Buttons to choose an item, numbers cannot be chosen until a letter is chosen. |
| 18 | Confirms the selected drink for the edit of price or stock. |
| 19 | Confirms the edit of price or stock and resets the form to its initial state which is what is shown in Figure 19. |
| 20 | Cancels the editing process and returns the form to its initial state. |

## Testing

Rigorous testing was carried out to ensure the vending machine system works correspondingly with the users’ actions. Within the table below you can see the various tests carried out, with the system having built in validation, there were no errors.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Description | Outcome | Evidence |
|  | Customer tab loads | As Expected. |  |
|  | Admin tab loads | As Expected. |  |
|  | Character Buttons work correctly | As Expected. |  |
|  | Numerical buttons work correctly | As Expected. |  |
|  | Confirm button disables keypad. | As Expected. |  |
|  | Loyalty Card Clicked – Enables panel Disables Cash Panel | As Expected. |  |
|  | Cash Option Clicked, Enables panel and disables loyalty card option | As Expected. |  |
|  | Click confirm without choosing item. – Refuses further action | As Expected. |  |
|  | Non numerical input – rejects amount | As Expected. |  |
|  | Enter less than required payment amount – rejects sale. | As Expected. |  |
|  | Enter more than required – accepts sale and gives change accurately. | As Expected. |  |
|  | Deducted item stock after sale. | As Expected. |  |
|  | Confirm sale without selecting account – rejects sale. | As Expected. |  |
|  | Confirm sale without entering pin for selected account. – rejects sale. | As Expected. |  |
|  | Confirm sale without entering correct pin for selected account. – rejects sale. | As Expected. |  |
|  | Confirm sale entering correct pin for selected account. – Accepts sale | As Expected. |  |
|  | Confirm sale with insufficient funds within bank account. – Rejects sale | As Expected. |  |
|  | Confirm sale with sufficient funds within bank account. – Accepts sale | As Expected. |  |
|  | Click Update stock option, enables stock panel, keypad and disables price panel | As Expected. |  |
|  | Click update price option, enables price panel and keypad whilst disabling stock panel. | As Expected. |  |
|  | Enter more quantity than limit, rejects. | As Expected. |  |
|  | Enter invalid currency amount, rejects. | As Expected. |  |
|  | Change stock level, input correct level and confirm. Accepts stock change. | As Expected. |  |
|  | Change item price, accepts new price and updates item | As Expected. |  |
|  | Attempt to select item out of stock, rejects. | As Expected. |  |
|  | Cancel button resets admin panel. | As Expected. |  |

# Conclusion

To conclude, this report contains all the information to evidence the analysis of requirements and the design of the system, including multiple different diagrams that depict how the program will interact with itself, the real world and the process that commonly will be used while using the vending machine. Furthermore, this report also includes details on how this design has been implemented, what methods of development was used and what tools were used to create the implementation. Included in the implementation documentation is detailing on core demo design features and their responsibility as well as sufficient testing to ensure no errors or issues with the build.

During the creation of this report detail was achieved when creating the various diagrams and understanding of how and why those diagrams are used within system design, as well as clearly demonstrated ability and understanding to create sufficient and relevant documentation for the implementation process and exhaustive testing of those implementation results. Although each team member has developed clear understanding of the software design process it is evident in word count that while we may have included a lot of detail and information that we believe to be relevant, it is clear that there is some development to be done on furthering our ability to work within restrictions and not go over the top when not necessary to.

# References

1. Alhir, S., 1999. *UML in a nutshell*. Beijing [etc.]: O'Reilly.
2. Ambler, S. W. (2005). UML Class Diagrams,. *The Elements of UML™ 2.0 Style*, pp. 47–72.
3. Banos, A., Lang, C. and Marilleau, N., 2015. *Agent-based spatial simulation with NetLogo*. Oxford, UK: Elsevier Science.
4. Bastos, R. and Ruiz, D., 2002. Extending UML activity diagram for workflow modeling in production systems. *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, [online] Available at: <https://ieeexplore.ieee.org/abstract/document/994510> [Accessed 26 March 2021].
5. Bidgoli, H. (2003). *Encyclopedia of information systems.* Amsterdam: Academic Press.
6. Britton, C. and Doake, J., 2005. *A student guide to object-oriented development*. Amsterdam: Elsevier Butterworth-Heinemann.
7. Budde, R., Kautz, K., Kuhlenkamp, K. and Züllighoven, H., 1982. *Prototyping: An Approach to Evolutionary System Development*. Heidelberg, Berlin: Springer.
8. Chonoles, M. and Schardt, J., 2003. *UML 2 for dummies*. New York: John Wiley.
9. Donald G, F., 2021. *Use Cases: the Pros and Cons*. [online] KSC. Available at: <https://www.cs.hmc.edu/~mike/courses/mike121/readings/reqsModeling/firesmith.htm#:~:text=Use%20cases%20are%20a%20powerful,communicating%20with%20customers%20and%20users.> [Accessed 3 March 2021].
10. Douglass, B. P. (2015). *Agile systems engineering.*
11. Eriksson, H., Penker, M. and Lyons, B., 2004. *UML 2 toolkit*. Indianapolis, Ind.: Wiley.
12. Fowler, M., 2004. *UML Distilled*. 2nd ed. Boston: Addison-Wesley, p.104.
13. Friedenthal, S. (2015). *A practical guide to sysml.* Waltham, Massachusetts: Morgan Kaufmann.
14. Grady, J., 2006. *System Requirements Analysis*. Massachusetts: Academic Press.
15. Greeff, G. (2004). *Practical guide to E-manufacturing and supply chain management.* Amsterdam: Newnes.
16. Gunter, S., 2011. *Opposing views on the UML {xor} constraint*. [online] Stuart Gunter. Available at: <https://stuartgunter.wordpress.com/2011/04/03/opposing-views-on-the-uml-xor-constraint/> [Accessed 26 March 2021].
17. Halpin, T. and Morgan, T., 2008. *Information modeling and relational database*. Amsterdam: Morgan Kaufmann.
18. Holt, J., 2007. *UML for systems engineering*. London: The Institution of Engineering and Technology.
19. Holub, A. I. (2017). Allen Holub’s UML. *Allen Holub*, 15-20.
20. Javapoint. (n.d.). *UML Collaboration Diagram*. Retrieved March 31, 2021, from https://www.javatpoint.com/uml-collaboration-diagram
21. Jürjens, J., 2005. *Secure systems development with UML*. Berlin: Springer.
22. Maciaszek, L., 2007. *REQUIREMENTS ANALYSIS AND SYSTEM DESIGN*. 3rd ed. Harlow, Essex: Pearson.
23. Martin, R. C. (2009). UML - Class Diagrams. In R. C. Martin, *Clean Code* (pp. 4-7). Upper Saddle River, N.J.: Prentice Hall.
24. Mcdonald, K., 2018. *Context Diagram*. [online] kbpmedia. Available at: <https://www.kbp.media/context-diagram/#:~:text=The%20context%20diagram%20helps%20you,level%200%20data%20flow%20diagram.> [Accessed 1 April 2021].
25. microTOOL. (2021, 04 04). *The class diagram. A structured path towards systems.* Retrieved from microTOOL: https://www.microtool.de/en/knowledge-base/what-is-a-class-diagram/#:~:text=The%20advantages%20of%20the%20class,as%20their%20properties%20and%20relationships.
26. Osis, J. and Donins, U., 2017. *Topological UML Modeling*. Amsterdam: Elsevier.
27. Pedamkar, P. (2020). Class Diagram. *EDUCBA*, 1-5.
28. Pilone, D. and Pitman, N., 2005. *UML 2.0 in a nutshell*. Beijing: O'Reilly & Associates.
29. Schmuller, J., 2004. *Sams teach yourself UML in 24 hours*. Indianapolis, Ind.: Sams.
30. SourceMaking. (2007). Class Diagrams. *UML Modeling Systems*, 4-7.
31. Sparks, G. (2008). Database modelling UML. *Methodsandtools*, 1-5.
32. Stevens, P. and Pooley, R., 2008. *Using UML*. 2nd ed. Harlow: Addison-Wesley.
33. Teorey, T., 2011. *Database modeling & design*. 5th ed. San Francisco, Calif.: Morgan Kaufmann.
34. Wazlawick, R., 2014. *Object-Oriented Analysis and Design for Information Systems*. Waltham, MA: Elsevier.

# Work Plan

