Requirements Engineering

Computer Repair System

Submitted By: Gerard O Sullivan (T00196432)

Computing with Software Development

Date Submitted: 08/05/2021

**Table of Contents**

[1. Introduction/overview 4](#_Toc59482360)

[2. Functional Components 5](#_Toc59482361)

[3. User Requirements 6](#_Toc59482362)

[3.1. ComputerRepairSYS will manage parts 6](#_Toc59482363)

[3.2. ComputerRepairSYSwillprocessrepairs 6](#_Toc59482364)

[3.3. ComputerRepairSYSwillperform administrative reporting 6](#_Toc59482365)

[4. System Requirements 7](#_Toc59482366)

[4.1. System Level Use Case Diagram 7](#_Toc59482367)

[4.2. Manage Parts 8](#_Toc59482368)

[4.1.1. Add Part 8](#_Toc59482369)

[4.1.2. Update Part 10](#_Toc59482370)

[4.1.3. Remove Part 12](#_Toc59482371)

[4.1.4. List Part 14](#_Toc59482372)

[4.2. Process Repairs 15](#_Toc59482373)

[4.2.1. Log Repair 15](#_Toc59482374)

[4.2.2. Estimate Repair 17](#_Toc59482375)

[4.2.3. Commence Repair 21](#_Toc59482376)

[4.2.4. Finalise Repair 23](#_Toc59482377)

[4.2.5. Dispatch Repair 25](#_Toc59482378)

[4.3. Perform Admin 27](#_Toc59482379)

[4.3.1. Analyse Yearly Revenue 27](#_Toc59482380)

[4.3.2. Analyse Yearly Stock 28](#_Toc59482381)

[5. System Model 29](#_Toc59482382)

[5.1. Level-0 DFD 30](#_Toc59482383)

[5.2. Level-1 DFD 31](#_Toc59482384)

[5.3. Level-2 DFD (Process P1: Manage Parts) 32](#_Toc59482385)

[5.4. Level-2 DFD (Process P2: Manage Repairs) 33](#_Toc59482386)

[5.5. Level-2 DFD (Process P3: Perform admin) 35](#_Toc59482387)

[6. Data Model (Class Diagram) 36](#_Toc59482388)

[6.1. Class Diagram 36](#_Toc59482389)

[6.2. Relational Schema 36](#_Toc59482390)

[6.3. Database Schema 36](#_Toc59482391)

[7. Conclusion 38](#_Toc59482392)

# Introduction/overview

This is a system designed to manage computer repairs for a computer repair shop. The goal of this specification is to specify requirements and produce a working prototype.

The system will be used to store data on the parts required to repair computers as well as process repairs and display data analysis on revenue.

# Functional Components

The functional components to be delivered in the Computer Repair System are shown below. These functions include a range of functions to track a repair from the time it is logged through various stages (cost estimation, commencement, completionand collection).

# User Requirements

## ComputerRepairSYS will manage parts

* + 1. ComputerRepairSYS will add a part type
    2. ComputerRepairSYS will update a part type
    3. ComputerRepairSYS will remove a part type
    4. ComputerRepairSYS will query a part in inventory

## ComputerRepairSYSwillprocessrepairs

* + 1. ComputerRepairSYS will log a repair
    2. ComputerRepairSYS will estimate the cost of repair
    3. ComputerRepairSYS will commence a repair
    4. ComputerRepairSYS will finalise a repair
    5. ComputerRepairSYS will dispatch a repair

## ComputerRepairSYSwillperform administrative reporting

* + 1. ComputerRepairSYS will produce a yearly revenue analysis
    2. ComputerRepairSYS will produce a yearlystockanalysis

# System Requirements

This section of the document presents the system requirements of the proposed system.

## System Level Use Case Diagram

The following system level use case diagram illustrates the high-level system requirements.

Manager

Customer

Repair Staff

## Manage Parts

This module includes functions to add, amend and remove an inventory part. A function to list inventory parts in stock is also provided.

### Add Part

This function adds the details of a repair part to the system. Each part is uniquely identified by a PartID.

<<includes>>

Repair Staff

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Add part | |
| **Use Case Id** | MP001 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Repair Staff | |
| **Other Participating Actors** |  | |
| **Description** | This function adds a part to the system. | |
| **Preconditions** |  | |
| **Trigger** |  | |
| **Expected Scenario** | **Repair Staff** | **System** |
|  | **Step 1:** Repair staff invokes add Part Function.  **Step 4:** The repair staff enters the required data:   * PartID * Name * Cost * QtyInStock | **Step 2:** Display UI  **Step 3:** The System assigns a new part ID.  **Step 5:** The system validates the data.   * All fields must be entered. * Part Name must be type String (20) * Part Cost must be numeric and greater than zero. * QtyInStock must be greater than zero.   **Step 6:** Set part status to ‘Available’ (A)  **Step 7:** The System saves the data in the **Parts file:**   * PartID * Name * Cost * QtyInStock * Status   **Step 8:** System displays a confirmation message:  *“Part NNNNN has been added”.*  **Step 9:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **Invalid data detected** |  | **Step 5:** Invalid data detected.  **Step 6:** Display an appropriate error message.  **Step 7:** Go back to **Step 3.** |
| **Conclusions** | A Part has been added to the system | |
| **Post conditions** | That Part can now be assigned to a repair. | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

### Update Part

This function updates the details of a part in the system.

<<includes>>

Repair Staff

<<extends>>

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Update Part | |
| **Use Case Id** | MP002 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Repair Staff | |
| **Other Participating Actors** |  | |
| **Description** | This function updates a part’s details in the system. | |
| **Preconditions** |  | |
| **Trigger** |  | |
| **Expected Scenario** | **Repair Staff** | **System** |
|  | **Step 1:** Repair staff invokes update Part Function.  **Step 3:** The repair staff enters the required data:   * Part Name   **Step 5:** The repair staff selects the part to be updated from the list.  **Step 7:** The repair staff enters the required data to be updated:   * Name * Cost * QtyInStock | **Step 2:** Display UI  **Step 4:** The system retrieves all available parts, from the **Parts File,** whose name matches what the user has entered and loads the list on the UI.  **Step 6:** The system retrieves the full details of the selected part to be updated from the **Parts File** and loads the part on the UI.  **Step 8:** The system validates the data.   * All fields must be entered. * Part name must be type String (20) * Part Cost must be numeric and greater than zero. * QtyInStock must be greater than zero.   **Step 9:** The System updates the data in the **Parts file:**   * Name * Cost * QtyInStock   **Step 10:** System displays a confirmation message:  *“Part NNNNN has been updated”.*  **Step 11:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **Invalid part in the parts table detected** |  | **Step 4:** Invalid data detected.  **Step 5:** Display an appropriate error message:  *“The part NNNNN is not in the* ***Parts File***”.  **Step 6:** Return to **Step 3.** |
| **Invalid data detected** |  | **Step 8:** Invalid data detected.  **Step 9:** Display an appropriate error message.  **Step 10:** Go back to **Step 7.** |
| **Conclusions** | A part has been updated on the system | |
| **Post conditions** |  | |
| **Business Rules** | Only parts with status ‘Available’ may be updated | |
| **Implementation Constraints** |  | |

### Remove Part

This function removes a part by changing its status on the system. This is to prevent parts that are discontinued from being used in a repair. When a part is removed its status is set to Not Available (“N”).

Repair Staff

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Remove Part | |
| **Use Case Id** | MP003 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Repair Staff | |
| **Other Participating Actors** |  | |
| **Description** | This function removes a part’s details in the system. | |
| **Preconditions** |  | |
| **Trigger** |  | |
| **Expected Scenario** | **Repair Staff** | **System** |
|  | **Step 1:** Repair staff invokes remove Part Function.  **Step 3:** The repair staff enters the required data:   * Part Name   **Step 5:** The repair staff selects the part to be removed from the list  **Step 7:**The repair staff confirms that the part selected is to be removed from the **Parts File.** | **Step 2:** Display UI  **Step 4:** The system retrieves all available parts, from the **Parts File,** whose name matches what the user has entered and loads the list on the UI.  **Step 6:** The system retrieves the full details of the selected part to be removed from the **Parts File** and loads the part on the UI.  **Step 8:** The System sets the parts status to **Not Available(‘N’) in the Parts File**  **Step 9:** The System updates the data in the **Parts file:**   * Status   **Step 10:** System displays a confirmation message:  *“Part NNNNN has been removed”.*  **Step 11:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **Invalid part in the parts table detected** |  | **Step 4:** Invalid data detected.  **Step 5:** Display an appropriate error message:  *“The part NNNNN is not in the* ***Parts File***”.  **Step 6:** Return to **Step 3.** |
| **Conclusions** | A part has been removed from the system | |
| **Post conditions** |  | |
| **Business Rules** | Only parts with status ‘Available’ may be removed | |
| **Implementation Constraints** |  | |

### List Part

This function lists the full details of a part on the system.

Repair Staff

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | List Part | |
| **Use Case Id** | MP004 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Repair Staff | |
| **Other Participating Actors** |  | |
| **Description** | This function lists a part’s details in the system | |
| **Preconditions** |  | |
| **Trigger** |  | |
| **Expected Scenario** | **Repair Staff** | **System** |
|  | **Step 1:** Repair staff invokes the list part function.  **Step 3:** The repair staff enters the required data:   * Part Name   **Step 5:** The repair staff selects the part to be viewed from the list.  **Step 7:** The repair staff reviews the details of the part that is being displayed. | **Step 2:** Display UI  **Step 4:** The system retrieves parts, from the **Parts File,** whose name matches what the user has entered.  **Step 6:** The system retrieves the full details of the selected part to be viewed from the **Parts File** and loads the part on the UI.  **Step 8:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **Invalid Part in the Parts table Detected** |  | **Step 4:** Invalid data detected.  **Step 5:** Display an appropriate error message:  *“The part NNNNN is not in the* ***Parts File***”.  **Step 6:** Return to **Step 3.** |
| **List all parts button** | **Step 3:** Staff member selects to list all parts.  **Step 5:** The repair staff selects the part to be viewed from the list.  **Step 7:** The repair staff reviews the details of the part that is being displayed. | **Step 4:** The system retrieves all parts from the **Parts File.**  **Step 6:** The system retrieves the full details of the selected part to be viewed from the **Parts File** and loads the part on the UI.  **Step 8:** Reset UI |
| **Conclusions** | The details of a single part are displayed to the user | |
| **Post conditions** |  | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

## Process Repairs

### Log Repair

This function logs a repair to the system.

Customer

Repair Staff

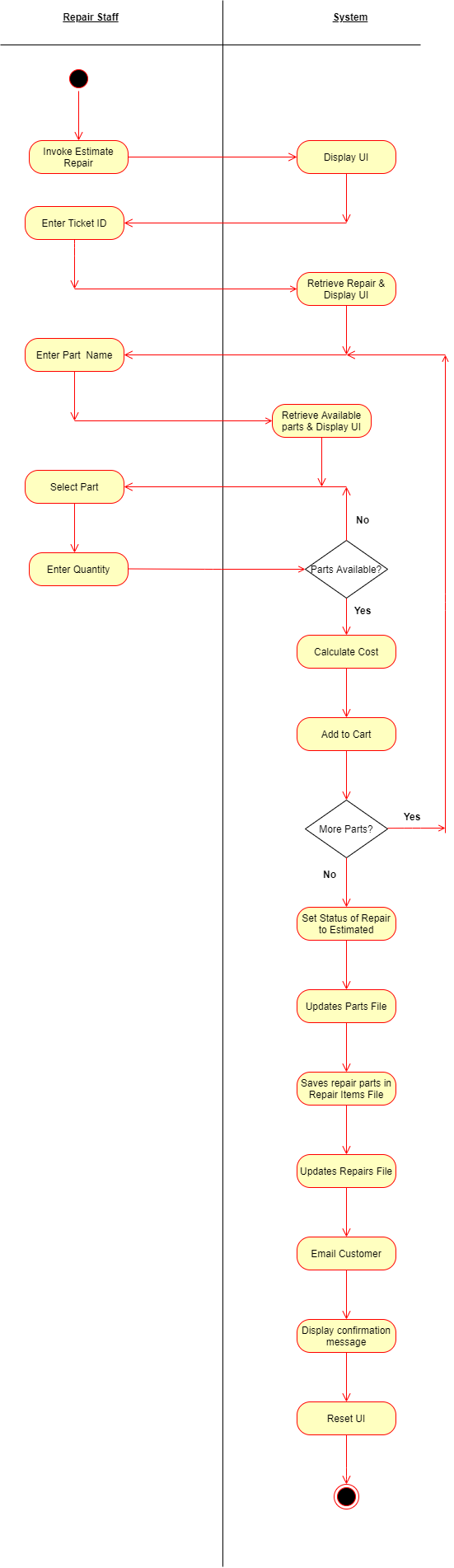
|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Log Repair | |
| **Use Case Id** | PR001 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Repair Staff | |
| **Other Participating Actors** | Customer | |
| **Description** | This function logs a repair to the system | |
| **Preconditions** |  | |
| **Trigger** |  | |
| **Expected Scenario** | **Repair Staff** | **System** |
|  | **Step 1:** Repair staff invokes Log Repair Function.  **Step 4:** The repair staff enters the required data:   * First Name * Last Name * Address * Eircode * Email * Phone * Description (OfProblem) | **Step 2:** The system assigns next Repair Ticket ID in NNNNN format  **Step 3:** Display UI  **Step 5:** The system validates the data.   * All fields must be entered. * First Name must be type String (20) * Last name must be type String (20) * Customer Address must be type String (50) * Customer Eircode must be type String (8) * Customer email must be a valid format. * Customer Phone must be an Integer and be a minimum of 8 characters long and a maximum of 15. * Description must be type String (70)   **Step 6:** Set Repair status to ‘Logged’ (L)  **Step 7:** Set total cost to **€30**(as a standard charge for assessing repair)  **Step 8:** The system emails the customer the Repair details.  **Step 9:** The system prints a label containing the Repair Ticket ID to be placed on the repaired item.  **Step 10:** The System saves the data in the **Repairs file:**   * RepairTicketID * First Name * Last Name * Address * Eircode * Email * Phone * Description * Date Logged * Status * Total Cost * Date Paid   **Step 11:** System displays a confirmation message:  *“The Repair* NNNNN *has been Logged”.*  **Step 12:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **Invalid data detected** |  | **Step 5:** Invalid data detected.  **Step 6:** Display an appropriate error message.  **Step 7:** Go back to **Step 4.** |
| **No Internet Connection**  **(Try to connect Again)** | **Step 10:** User choses to try again | **Step 8:** Unable to connect to the internet.  **Step 9:** Ask user to continue without internet connection.  **Step 11:** Go back to **Step 4.** |
| **No Internet Connection**  **(Continue no internet)** | **Step 10:** User choses to continue without internet | **Step 8:** Unable to connect to the internet.  **Step 9:** Ask user to continue without internet connection.  **Step 11:** The system prints a label containing the Repair Ticket ID to be placed on the repaired item.  **Step 12:** The System saves the data in the **Repairs file:**   * RepairTicketID * First Name * Last Name * Address * Eircode * Email * Phone * Description * Date Logged * Status * Total Cost * Date Paid   **Step 13:** System displays a confirmation message:  *“The Repair* NNNNN *has been Logged”.*  **Step 12:** Reset UI |
| **Conclusions** | A Repair has been logged on the system | |
| **Post conditions** | The repair is ready to commence estimating cost of repair | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

### Estimate Repair

This function estimates the cost of a repair and emails the customer of the estimated cost.

Customer

Repair Staff



|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Estimate Repair | |
| **Use Case Id** | PR002 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Repair Staff | |
| **Other Participating Actors** | Customer | |
| **Description** | This function estimates a repair and informs the customer of the cost. | |
| **Preconditions** | The Repair must be logged in the Log Repair function before the Estimate Repair function can begin | |
| **Trigger** |  | |
| **Expected Scenario** | **Repair Staff** | **System** |
|  | **Step 1:** Repair staff invokes the Estimate Repair Function.  **Step 4:** The repair staff enters the required data:   * Repair Ticket ID   **Step 6:** The repair staff enters the name of a part that is to be used in the repair.  **Step 8:** The repair staff selects the part required.  **Step 9:** The repair staff selects the quantity required.  **Step 13:** The repair staff confirms if all parts have been added | **Step 2:** The system retrieves all logged repairs from the **Repairs File.**  **Step 3:** Display UI  **Step 5:** A variable “basket Balance” is instantiated and is set to **€30**.  **Step 7:** The system retrieves all Available parts **(‘A’)**, from the **Parts File,** whose name matches what the user has entered and loads the list on the UI.  **Step 10:** The system validates the data.   * There is sufficient stock in store.   **Step 11:** The cost of the repair is updated.  **Step 12:** Ask the staff if more parts required.  **Step 14:** If more parts required.  Return to step 6.  **Step 15:** The system emails the customer details of the estimated repair.  **Step 16: For each item in the cart:**   * The System updates the QtyInStockin the **Parts file** * The system saves the part details in the **Repair Items File**:   + TicketID   + PartID   + Qty   + Cost   **Step 17:** The system updates the total cost of the repair in the Repair File.  **Step 18:** The system sets Repair status to **‘Estimated’ (E)**  **Step 19:** System displays a confirmation message:  *“The Repair* NNNNN *has been Estimated”.*  **Step 20:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **The Repair status is NOT equal to Logged(‘L’)** |  | **Step 5:** Invalid data detected.  **Step 6:** Display an appropriate error message.  **Step 7:** Go back to **Step 4.** |
| **Invalid RepairTicketID** |  | **Step 5:** Invalid data detected.  **Step 6:** Display an appropriate error message.  **Step 7:** Go back to **Step 4.** |
| **Invalid QtyInStock in the Parts File** |  | **Step 10:** Invalid data detected.  **Step 11:** Display an appropriate error message.  **Step 12:** Go back to **Step 9.** |
| **Conclusions** | A Repair has been estimated and an email has been sent to the customer | |
| **Post conditions** | An email has been sent to the customer and the repair may not continue until payment has been received | |
| **Business Rules** | Only repairs with status ‘Logged’ may be estimated and only parts with status ‘Available’ may be used in estimating the repair. | |
| **Implementation Constraints** |  | |

### Commence Repair

This function allows the repair to be paid for so staff can begin working on the product.

Customer

Repair Staff

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Commence Repair | |
| **Use Case Id** | PR003 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Repair Staff | |
| **Other Participating Actors** | Customer | |
| **Description** | This function allows a repair to be commenced once payment has been received. | |
| **Preconditions** | The Repair must be estimated in the Estimate Repair function before the Commence Repair function can begin | |
| **Trigger** |  | |
| **Expected Scenario** | **Repair Staff** | **System** |
|  | **Step 1:** Repair staff invokes the Commence Repair Function.  **Step 4:** The repair staff enters the required data:   * Repair Ticket ID   **Step 9:** The user confirms repair is to be commenced | **Step 2:** The system retrieves all estimated repairs from the **Repairs File.**  **Step 3:** Display UI  **Step 5:** The system retrieves repair details, from the **Repairs File.**  **Step 6:** The system checks the status of the repair as it must be **Estimated(‘E’)**  **Step 7:** The system retrieves part details, from the **RepairItems File** where the RepairTicketID matches the chosen RepairTicketID**.**  **Step 8:** The System displays the Repair details.  **Step 10:** The system prompts the user to confirm repair to be commenced.  **Step 11:** The system emails the customer a receipt of payment.  **Step 12:** The system sets the repair **status equal to Paid(‘P’)** in the **Repairs file.**  **Step 13:** The System updates the **Date Paid** in the **Repairs file** and sets it to the current date.  **Step 14:** System displays a confirmation message:  *“The Repair* NNNNN *has been paid for you may commence repair”.*  **Step 15:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **The user selects not to Commence repair** |  | **Step 10:** Go back to **Step 4.** |
| **There are no parts in the repair items file (If the repair is a software issue)** | **Step 9:** The user confirms repair is to be commenced | **Step 7:** System displays a message confirming that there are no parts for this repair:  *“*No parts were recorded for this repair*”.*  **Step 8:** The System displays the Repair details.  **Step 10:** The system prompts the user to confirm repair to be commenced.  **Step 11:** The system emails the customer a receipt of payment.  **Step 12:** The system sets the repair **status equal to Paid(‘P’)** in the **Repairs file.**  **Step 13:** The System updates the **Date Paid** in the **Repairs file** and sets it to the current date.  **Step 14:** System displays a confirmation message:  *“The Repair* NNNNN *has been paid for you may commence repair”.*  **Step 15:** Reset UI |
| **Conclusions** | A repair has been paid for and recorded in the system. | |
| **Post conditions** | The item is ready to commence physical repair | |
| **Business Rules** | Only repairs whose status is equal to Estimated (‘E’) may be selected for Commence Repair. Repair may not commence until full payment has been received. | |
| **Implementation Constraints** |  | |

### Finalise Repair

This function allows the repair to be paid for so staff can begin working on the product.

Customer

Repair Staff

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Finalise Repair | |
| **Use Case Id** | PR004 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Repair Staff | |
| **Other Participating Actors** | Customer | |
| **Description** | This function finalises a repair and marks it as finished once the repair is complete. | |
| **Preconditions** | The Repair must be paid for in the Commence Repair function before the Finalise Repair function can begin | |
| **Trigger** |  | |
| **Expected Scenario** | **Repair Staff** | **System** |
|  | **Step 1:** Repair staff invokes the Finalise Repair Function.  **Step 4:** The repair staff enters the required data:   * Repair Ticket ID   **Step 9:** The user confirms repair finalisation. | **Step 2:** The system retrieves all estimated repairs from the **Repairs File.**  **Step 3:** Display UI  **Step 5:** The system retrieves repair details, from the **Repairs File.**  **Step 6:** The system checks the status of the repair as it must be **Paid(‘P’)**  **Step 7:** The System displays the Repair details.  **Step8:** The system prompts the user to confirm repair finalisation.  **Step 10:** The system emails the customer and informs them that the repair has completed and is ready for collection.  **Step 11:** The system sets the repair **status equal to Finalised (‘F’)** in the **Repairs file.**  **Step 12:** System displays a confirmation message:  *“The Repair* NNNNNN *has been completed and is ready for collection. The customer has been emailed”.*  **Step 13:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **The user selects not to dispatch repair** |  | **Step 10:** Go back to **Step 4.** |
| **Conclusions** | A repair has been paid for and recorded in the system. | |
| **Post conditions** | The item is ready to be collected | |
| **Business Rules** | Only repairs whose status is equal to Paid (‘P’) may be selected for Finalise repair. | |
| **Implementation Constraints** |  | |

### Dispatch Repair

This function allows the repair to be paid for so staff can begin working on the product.

Customer

Repair Staff

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Dispatch Repair | |
| **Use Case Id** | PR005 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Repair Staff | |
| **Other Participating Actors** | Customer | |
| **Description** | This function dispatches a repair to a customer and marks the repair as complete. | |
| **Preconditions** | The Repair must be finished repair in the Finalise Repair function before the Dispatch Repair function can begin | |
| **Trigger** |  | |
| **Expected Scenario** | **Repair Staff** | **System** |
|  | **Step 1:** Repair staff invokes the Dispatch Repair Function.  **Step 4:** The repair staff enters the required data:   * Repair Ticket ID   **Step 9:** The user confirms repair dispatch. | **Step 2:** The system retrieves all estimated repairs from the **Repairs File.**  **Step 3:** Display UI  **Step 5:** The system retrieves repair details, from the **Repairs File.**  **Step 6:** The system checks the status of the repair as it must be **Finalised(‘F’)**  **Step 7:** The System displays the Repair details.  **Step 8:** The system prompts the user to confirm repair dispatch.  **Step 10:** The system emails the customer to thank them for their service.  **Step 11:** The system sets the repair **status equal to Finalised (‘F’)** in the **Repairs file.**  **Step 12:** System displays a confirmation message:  *“The Repair* NNNNNN *has been completed”.*  **Step 13:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **The user selects not to dispatch repair** |  | **Step 10:** Go back to **Step 4.** |
| **Conclusions** | The repair has been complete | |
| **Post conditions** | The Item has been collected and the repair is complete | |
| **Business Rules** | Only repairs whose status is equal to Finalised (‘F’) may be selected for Dispatch Repair. | |
| **Implementation Constraints** |  | |

## Perform Admin

### Analyse Yearly Revenue

This function displays the total revenue based on the managers chosen date.

Manager

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Analyse Yearly Revenue | |
| **Use Case Id** | PA001 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Manager | |
| **Other Participating Actors** |  | |
| **Description** | This function displays the yearly revenue for the business for a chosen year. | |
| **Preconditions** |  | |
| **Trigger** |  | |
| **Expected Scenario** | **Manager** | **System** |
|  | **Step 1:** The manager invokes the Analyse Yearly Revenue function.  **Step 3:** The manager selects the year to analyse.  **Step 5:** The manager reviews the information that is being displayed. | **Step 2:** load current year on UI  **Step 4:** The system Selects total cost of Repairs from the **repairs file** for the year entered grouped by Month. The system displays on the UI.  **Step 6:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **No data for year selected** |  | **Step 4:** Invalid data detected.  **Step 5:** Display an appropriate error message.  "There is no data for the year currently selected."  **Step 6:** Go back to **Step 3.** |
| **Conclusions** | The system displays the yearly revenue information for the date requested | |
| **Post conditions** |  | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

### Analyse Yearly Stock

This function analyses the total value of each stock item sold in a year.

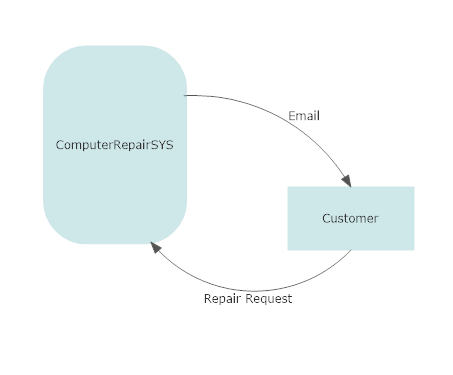
Manager

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Analyse Yearly Stock Sales | |
| **Use Case Id** | PA002 | |
| **Priority** | 1 | |
| **Source** | Manager | |
| **Primary Business Actor** | Manager | |
| **Other Participating Actors** |  | |
| **Description** | This function analyses the total value of each stock item sold in a year. | |
| **Preconditions** |  | |
| **Trigger** |  | |
| **Expected Scenario** | **Manager** | **System** |
|  | **Step 1:** The manager invokes the Analyse Yearly Stock function.  **Step 3:** The manager selects the year to analyse.  **Step 5:** The manager reviews the information that is being displayed. | **Step 2:** load current year on UI  **Step 4:** The system Selects total cost of Parts from the **RepairItems file** for the year entered grouped by Month. The system displays on the UI.  **Step 6:** Reset UI |
| **Alternate Scenarios** | **Actor** | **System** |
| **No data for year selected** |  | **Step 4:** Invalid data detected.  **Step 5:** Display an appropriate error message.  "There is no data for the year currently selected."  **Step 6:** Go back to **Step 3.** |
| **Conclusions** | The system displays the yearly stock information for the date requested | |
| **Post conditions** |  | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

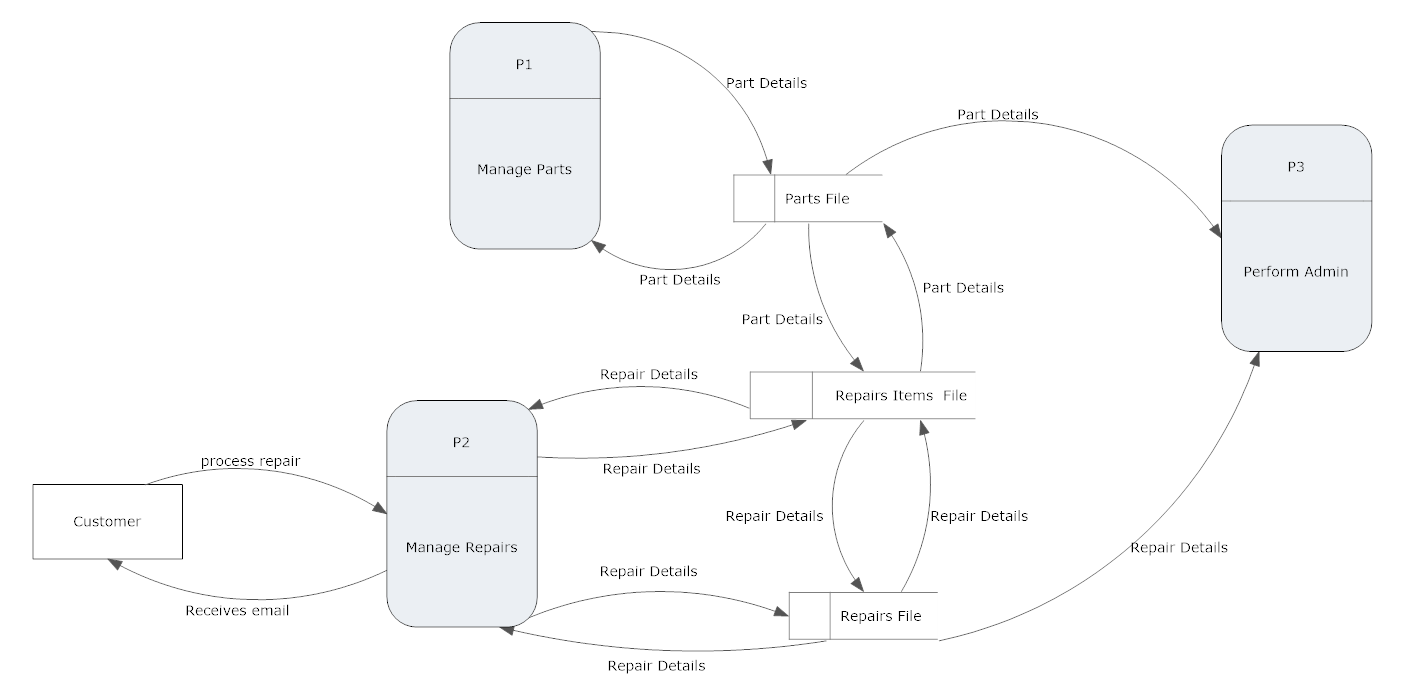
# System Model

The following dataflow diagrams have been produced for the system:

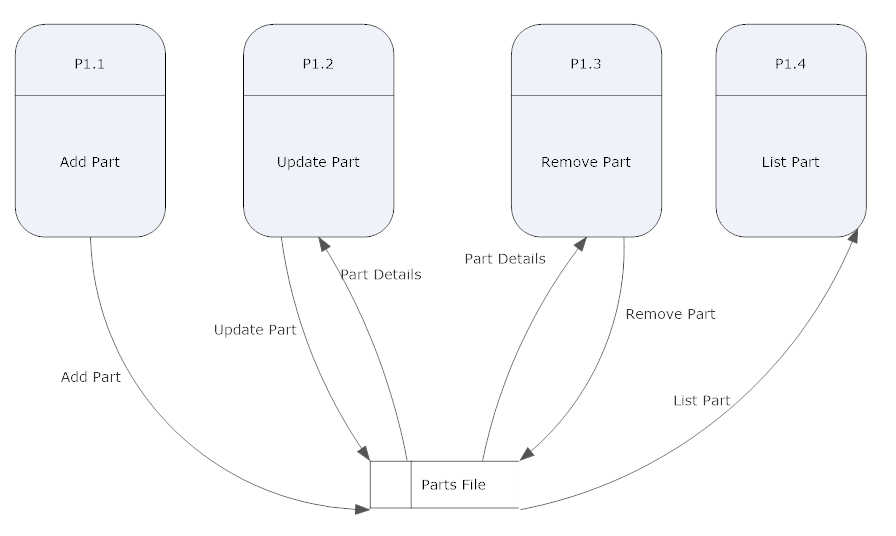
## Level-0 DFD



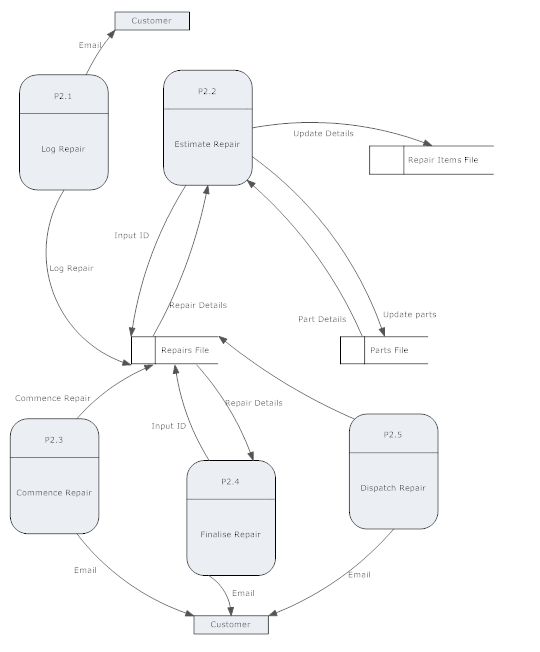
## Level-1 DFD



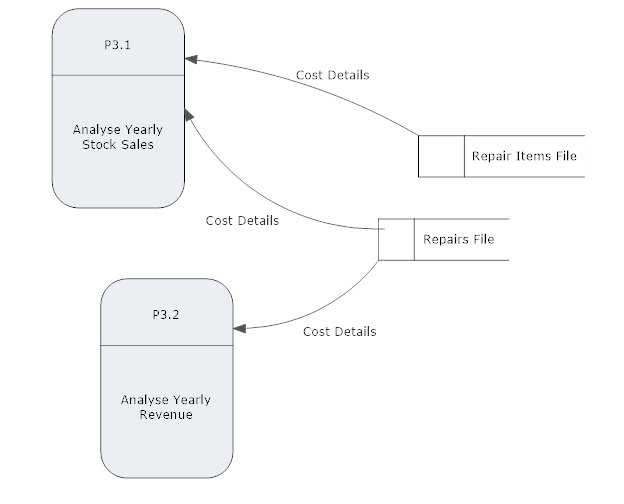
## Level-2 DFD (Process P1: Manage Parts)



## Level-2 DFD (Process P2: Manage Repairs)



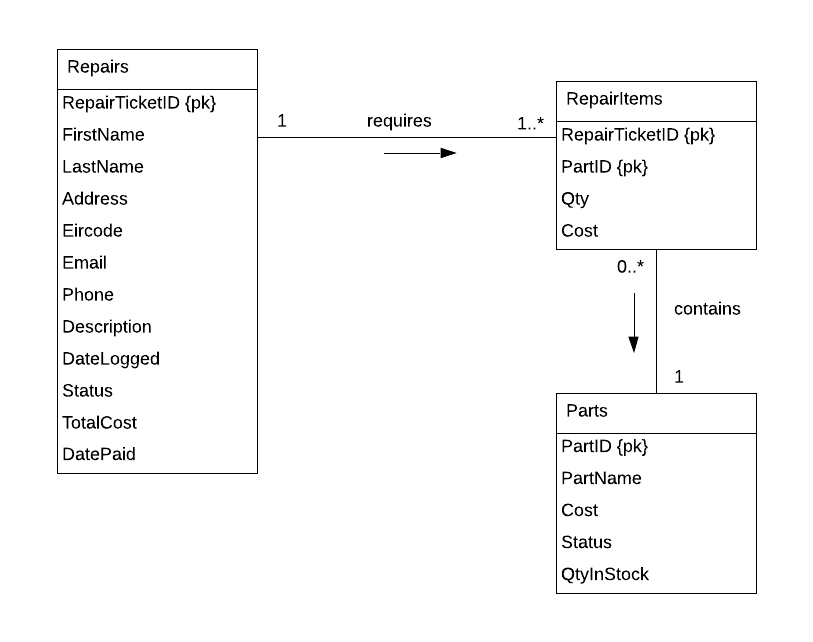
## Level-2 DFD (Process P3: Perform admin)



# Data Model (Class Diagram)

This section of the document will present the proposed database for the system.

## Class Diagram



## Relational Schema

**Repiars (**RepairTicketID,FirstName, LastName, Address, Eircode, Email, Phone, Description, DateLogged, Status, TotalCost, DatePaid)

**Repair Items (**RepairTicketID,PartID, Qty, Cost)

**Parts (**PartID,PartName, Cost, Status, QtyInStock)

## Database Schema

**Relation Parts**

PartID NUMERIC (5) NOT NULL

PartName VARCHAR2 (20) NOT NULL

Cost NUMERIC (6,2) DEFAULT 0 NOT NULL

Status CHAR (1) DEFAULT 'A' NOT NULL

QtyInStock NUMERIC (4) DEFAULT 0 NOT NULL

**Primary key:** PartID

**Relation Repairs**

RepairTicketID NUMERIC (5) NOT NULL,

FirstName VARCHAR2 (20) NOT NULL,

LastName VARCHAR2 (20) NOT NULL,

Address VARCHAR2 (50) NOT NULL,

Eircode VARCHAR2 (8),

Email VARCHAR2 (50) NOT NULL,

Phone VARCHAR2 (15) NOT NULL,

Description VARCHAR2 (70) NOT NULL,

DateLogged DATE NOT NULL,

Status CHAR (1) DEFAULT 'L' NOT NULL,

TotalCost NUMERIC (6,2) DEFAULT 30 NOT NULL,

DatePaid DATE,

**Primary key:** RepairTicketID

**Relation RepairItems**

RepairTicketID NUMERIC (5) NOT NULL,

PartID NUMERIC (5) NOT NULL,

Qty NUMERIC (4) Default 0 NOT NULL,

Cost NUMERIC (6,2) Default 0 NOT NULL,

**Primary key:** RepairTicketID, PartID

**Foreign key:** RepairTicketID, PartID

# Conclusion

All user requirements have been met. A fully functioning program has been developed. My program can process parts data such as adding, removing, updating, or deleting parts as well as processing repairs and displays data for analysis.

I have created a database for the program, and it stores all the necessary data. I have conducted software testing on the program and have removed any bugs that I have found.

In conclusion, this is a fully functioning piece of software that has met all the users’ requirements.