**CMPT 276 Group 20**

# Macrohard Corp.

# TrackMaster

## Release 1

Release Date: 2024/06/07

**Approved By:**

| | |
|---|---|
| **Sit Kwan Wai,** *Project Manager* | ______________________ |
| **James Isaac,** *CEO* | ______________________ |
| **Jin Bowen,** *CFO* | ______________________ |
| **Saadati Aryan,** *Chief of Staff* | ______________________ |

## Release History Page

| | |
|---|---|
| **Rel. 1** | - 2024/06/07 |
| | - Was composed of just the preliminary Requirements Spec. |

## Release Table Page

| Configuration Item: | Version | Number of Each Configuration Item in: |
|---|---|---|
| | Rel. 1 | |
| <u>Documents</u> | | |
| Requirements Spec | 1 | |
| User Manual | | |
| Architectural Design Document | | |
| Detailed Design Document | | |
| Integration Report | | |
| <u>Modules</u> | | |
| Main.cpp | | |
| Change.h | | |
| Change.cpp | | |
| InternalRequester.h | | |
| InternalRequester.cpp | | |
| Request.h | | |
| Request.cpp | | |
| Requester.h | | |
| Requester.cpp | | |
| Product.h | | |
| Product.cpp | | |
| ProductRelease.h | | |
| ProductRelease.cpp | | |

**CMPT 276 Group 20**

# Macrohard Corp.

# TrackMaster

**Requirements Specification**

**Version 1**

Written by:

**James Isaac** ____________________________

**Jin Bowen** ____________________________

**Saadati Aryan** ____________________________

**Sit Kwan Wai** ____________________________

Approved by:

**Quality Assurance Dept.** ____________________________

## Document Version History

**Version 1** - Original 2024/06/07 by James Isaac, Jin Bowen, Saadati Aryan, and Sit Kwan Wai

# Table of Contents

# 1.0) Introduction

## 1.1) System Overview

TrackMaster, the issue-tracking software created by Macrohard Corp. is used to manage and maintain requests for bug fixes and features of a project. This software applies to both customer support and project management teams by allowing for effective communication between clients and developers. This system helps customer support to efficiently keep track of incoming customer requests, as well as previous bug and feature requests submitted. Ultimately, this will help ensure prompt resolution schedules by helping development teams identify and document any issues regarding the project.

Our system is designed to allow staff personnel to log issues and customers to submit requests via customer support. The system will maintain data integrity by ensuring proper data entry upon use. This is done through validating customer and employee details, department names, phone numbers, and product release IDs. Furthermore, users are able to track the status of each issue, which are updated daily by the project's software engineers.

Systems such as TrackMaster are critical for streamlining the process of recording and organizing the multitude of issue-requests that can be submitted. Furthermore, this can often be an extremely chaotic operation in smaller to middle-sized companies. By implementing this system, the client can expect significant improvements in project organization. Ultimately, this automated system will prove to be exponentially more reliable and effectively replace manual tracking methods.

## 1.2) Business Objectives

We are contracted to design and develop TrackMaster solely for Super Fun Unlimited (SFU). Our client is a medium-sized game development company, competing against other established members in the market such as EA and Epic Games. The marketing department aims to complete TrackMaster's development within the next four months. This timeline will align with future market demand projections and upcoming product launch opportunities. Furthermore, it will provide us with ample time for thorough testing to ensure TrackMaster meets the high standards expected by Super Fun Unlimited and its potential customers.

## 1.3) Software Project Outline

The development of the issue-tracking software will commence on 2024/06/07. As we need to ensure that the software requirements are thoroughly understood and documented, we are able to begin as soon as we are finished clarifying the system's functionality, user interactions, and data management.

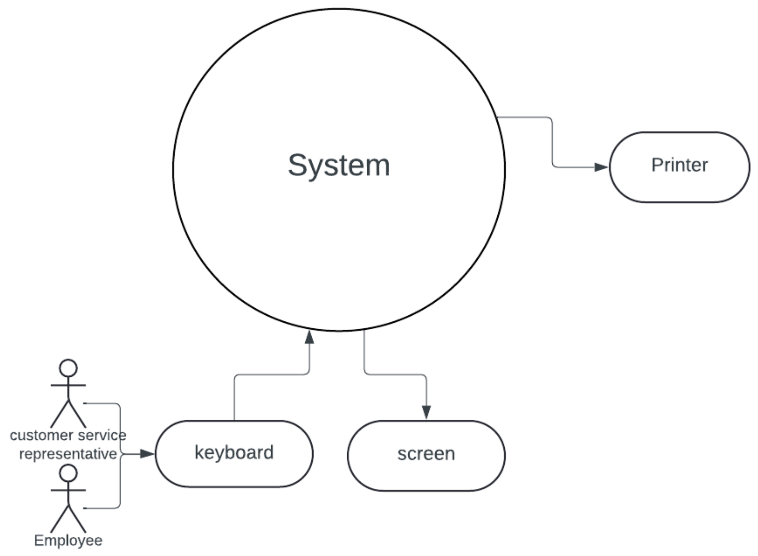
To execute this project in an efficient manner, we have estimated a base requirement of 4 software engineers, 4 programmers, and 4 quality assurance testers on a work schedule of 40 hours a week. In addition, resources such as servers for hosting the software and databases for storing data should be considered. Other costs may include external consulting services and ongoing maintenance fees.

Engineering suggests that the project can be completed by 2024/08/02, accounting for any unforeseen technical challenges. Major milestones, defined by the release of deliverables such the requirements specification, are set every 2 weeks starting 2024/06/07.

Given the relatively short deadline, an agile development method such as Scrum that emphasizes iterative development, would be the most suitable development life cycle model. Consequently, this will accommodate any changing requirements and priorities, allowing us to adjust to new requests based on user feedback. Furthermore, we will be developing this in C++, as it will allow for an easier integration into Super Fun Unlimited’s Legacy Systems. This will also provide protection over sensitive information such as bug reports and feature requests.

# 2.0) Environment

## 2.1) Context Object Flow Diagram



Our software system is the main entity within our context diagram. The system takes in inputs from both the customer service representative and the employees through the keyboard, which is also the main and only source of input. The system provides two output streams. The main output stream is an external screen that will display the information to the user. The secondary source of output is an external printer which can be used to provide hard copies of the information from the system.

## 2.2) Hardware

The suggested hardware specifications are as followed: **

| Component | Minimum Specification | Recommended Specification |
|---|---|---|
| Processor | Dual-core 2.8GHz | Quad-core 3.0GHz |
| Memory | 2GB | 4GB |
| Disk Space | 4GB | 4GB |
| Printer | Dot matrix capable of printing on A4 standard size papers | Not applicable |

**The minimum hardware specification is also counting for complying with the linux based operating system.

## 2.3) Operating System

Our software will be developed and tested using the linux Ubuntu 22.04 (Jammy Jellyfish) operating system. However, this can still be run on various linux based platforms.

## 2.4) Database

Our database will not be implemented using any external database management software, rather implemented and maintained by our own team. Our team will be taking advantage of fast accessibility of fixed length binary tables to store information.

## 2.5) Development Environment

The development environment for this software will be Visual Studio Code v1.89.0 on a linux platform.

## 2.6) Maintenance Environment

Maintenance will be handled by our own team of software engineers. The maintenance procedure can be done remotely by our staff who must be given administrative access to the local operating systems. The system will need to be shut down and rebooted during the procedure.

The goal of our team is to provide the longest lasting software solution to our customers, therefore all of the customer data management will be done in a versatile manner which can allow us to avoid data translation as we move to new versions towards the future.

The data required for the maintenance will be transmitted through the internet to the customers operating system. In the event of a customer's inability to provide remote internet access to our servers, we are ready to provide traveling in person services, only in circumstances deemed plausible at a higher cost.

# 3.0) User Interface and Operations

## 3.1) User Interface Sophistication

The user interface for this system is a character-based command line interface with a fixed-height display without the option to scroll. Users can select items by entering the character associated with the desired action, displayed to the left of each item. If the information exceeds the screen's display capacity, options to navigate to the next or previous set of items will be available. These options are accessed using commands for the next and previous sets, indicated at the bottom left corner of the screen. The user will be able to enter input at any time, with the exception of when the system is loading. Furthermore, the user may choose to opt out of the program and exit to the main menu by entering a command indicated at the bottom right of the screen. If the user exits to the main menu before finishing an operation such as creating a new request, the incomplete data will not be recorded.

## 3.2) Use Case Scenarios

**General System Use and Exceptions:**

All user input will be via keyboard.

When a user provides invalid input, the system will display a prompt explaining why the input was invalid and what input is expected. The system will then re-prompt the user to enter valid input. This applies to cases such as invalid syntax, attempting to enter a duplicate value for a field that must be unique, or empty input for required fields.

For more information on exception handling for invalid input related to retained data, please refer to section 4.2.3.

**Use Cases:**

**Use Case 1: Create Request**

**Actor:** Customer service representative, software engineer, or quality assurance engineer.
**Pre-conditions:**
There exists a relevant release for the product mentioned in the request and the user must be in the main menu.

**Interaction steps:**
1. User selects to create a new request.
2. System asks the user whether the request is coming from an external or internal source such as a customer or company employee.
3. User enters a command selecting if it is an external or internal request.
4. System prints out a list of existing external (or internal) requesters.

5. User selects a requester.
    a. If no matching requester exists, the user can select the final option in the list to create a new requester. Proceed to the "Use Case 2: Create Requester”. Once the new requester has been created, continue with the next steps of this use case.
6. System asks for the date of the request.
7. User enters the date of the request (YYYY-MM-DD).
8. System prints out a list of existing products.
9. User selects a product.
    a. If no matching product exists, the user can select the final option in the list to create a new product. Proceed to “Use Case 3: Create Product”. Once the new requester has been created, continue with the next steps of this use case.
10. System prints out a list of the product’s release IDs .
11. User selects a release ID.
12. System prints out a list of priorities that the request can take on.
13. User selects a priority.
14. System displays a list of existing change descriptions that pertain to the relevant product.
15. User selects the change description relevant to their request.
    a. If no change items exist that match what the request is about, a new change item must be created.
    b. Otherwise, skip to step 20 of this current use case.
16. System asks for a description of the change.
17. User enters a description (up to 30 characters).
18. System prints out a list of the product release IDs.
19. User selects an anticipated release ID (can be left blank).
20. System signals complete and backs out to the main menu, or error.

**Results:**
The request is created. The change item is created if no relevant change items already exist.

**Notes and Variants:**
The process may result in new product and requester objects created.

**Use Case 2: Create Requester**

**Actor:** Customer service representative, software engineer, or quality assurance engineer.
**Pre-conditions:**
User is in the main menu or referring from Step 5 of the “Create Request” use case.

**Interaction Steps:**
1. User selects to create a new requester.
2. Systems asks for the requester’s email.
3. User enters email (alpha numeric up to 30 characters)
4. System asks for the requester's phone number.
5. User enters phone number (+ area code xxx-xxx-xxxx)

6. System asks for the requester's first and last name.
7. User enters first and last name (last, first).
8. System asks for the requester’s department.
9. User enters the department (blank or up to 15 characters).
    a. If this field is left blank by the user, the system will assume that the requester is a customer and not an internal employee of the company.
10. System signals complete, or error.

**Result:**
User created.

**Notes and Variants:**
This process may be initiated during the “Create Request” use case. If so, proceed to Step 6 of the “Create Request” use case in order to complete it. Otherwise, the user will be directed back to the main menu.

**Use Case 3: Create Product**

**Actor:** Software engineer, or quality assurance engineer.
**Pre-conditions:**
User is in the main menu or referring from Step 9 of the “Create Request” use case.

**Interaction steps:**
1. User selects to create a new product.
2. System asks for the product name.
3. User enters the product name (alpha numeric up to 10 characters).
4. System signals complete, or error.

**Result:**
Product is created and the initial release for the product is created.

**Notes and Variants:**
This process may be initiated during the “Create Request” use case. If so, proceed to Step 10 of the “Create Request” use case to complete it. Otherwise, the user will be directed back to the main menu.

**Use Case 4: Assess New Change Items**

**Actor:** Software engineer, or quality assurance engineer.
**Pre-conditions:**
Use must be in the main menu, as well as requiring at least one change item to exist

**Interaction steps:**

1. User selects to assess new change items.
2. System prints out a list of change item descriptions that have the “open” status.
3. User selects a particular change item to assess.
4. System prints out a list of statuses the change can take on.
5. User selects a status for the change (“assessed” or “cancelled”).
6. System asks the user for an updated description of the change.
7. User enters an updated description of the change (blank indicating no new description, otherwise up to 30 characters).
8. System prints out a list of the product’s release IDs.
9. User selects an updated anticipated release ID. If this option is left blank, it will indicate no change.
10. System signals complete and backs out to the main menu, or error.

**Result:**
Updated change item.

**Use Case 5: Update Change item**

**Actor:** Software engineer, or quality assurance engineer.
**Pre-conditions:**
The user must be in the main menu. At least one relevant change item must exist.

**Interaction steps:**

1. User selects to update a change item.
2. System prints out a list of products.
3. User selects a product.
4. System prints out a list of change item descriptions that have the “assessed” status, and that pertain to the selected product.
5. User selects a particular change item to update.
6. System asks for a status of the change item (blank indicating no change, “cancelled,” “in progress,” or “done.”)
7. System prints out a list of the product’s release IDs.
8. User selects an updated anticipated release ID. The user can choose to leave this option blank, indicating no change.
9. System signals complete and backs out to the main menu, or error.

**Result:**
Updated change item

**Use Case 6: Inquire a Change Item**

**Actor:** Customer service representative, software engineer, or quality assurance engineer.
**Pre-conditions:**
Users must be in the main menu. At least one relevant change item must exist.

**Interaction steps:**
1. User selects to inquire about a change item.
2. System prints out a list of products.
3. User selects a product.
4. System prints out a list of change item descriptions pertaining to the selected product.
5. User selects a change item.
6. System prints out relevant change item information.
7. User exits.
8. System backs out to the main menu.

**Use Case 7: Print Report**

**Actor:** Customer service representative, software engineer, or quality assurance engineer.
**Pre-conditions:**
User must be in the main menu and requires the change items, related requests, and related requesters to exist.

**Interaction Steps:**
1. System prints a list of report options.
2. User selects one report option out of the following:
    a. A list of all assessed changes for a specific product that are anticipated to be included in an upcoming release and have not been completed or canceled.
    b. A list of all customers and staff who need to be notified about the completion of a specific change and the release in which the change is or will be available.
3. System connects to the user's printer and physically prints out the selected report.
4. System signals complete and backs out to the main menu, or error.

**Result:**
Paper report printed out for the user.

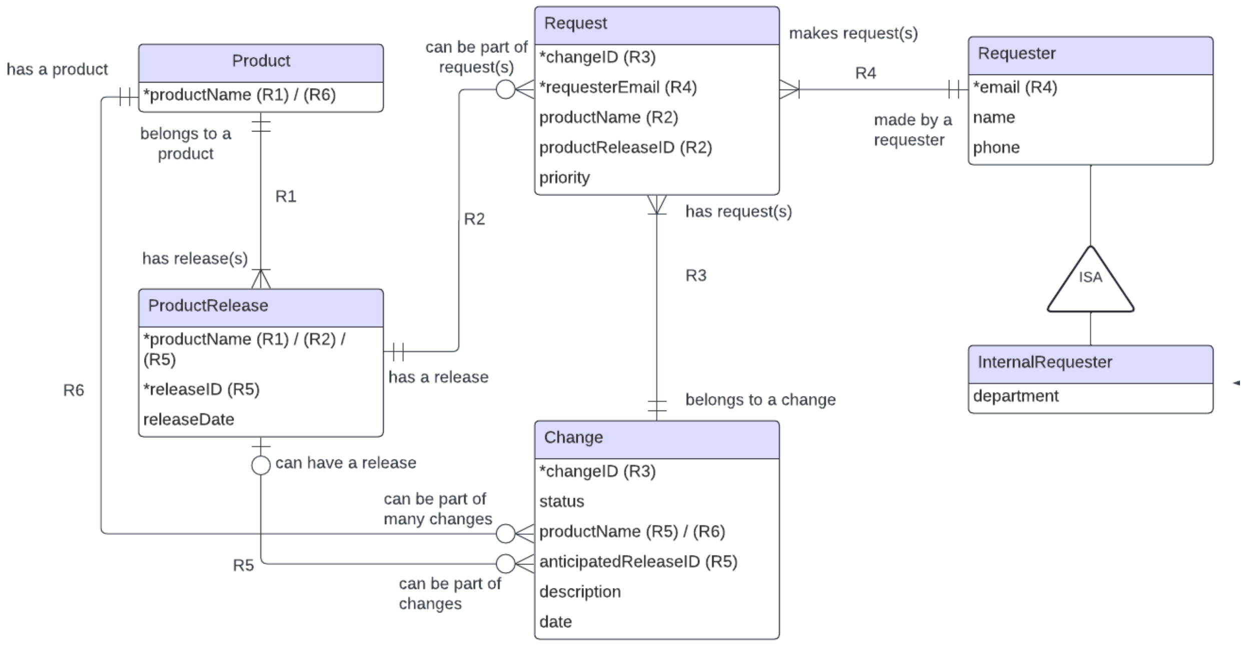
**Notes and variants:**
There may be some intermediary steps between steps 2 and 3 where the user needs to enter printer information.

### 3.3) Scenarios started by other event sources

All events are started via keyboard input.

# 4.0) Retained Data Model

## 4.1) Object Relationship Diagram (ORD)



## 4.2) Discussion of Object Relationship Diagram

**<u>Cardinality</u>**

- **(R1) A product has one or many releases.**
  - **Justification for rejecting optionality of zero:** A product without any release would make it impossible to manage requests associated with specific versions.
  - **Justification for rejecting multiplicity of one:** Products can be updated to new versions, implying that a product can have multiple releases.

- **(R1) A product release belongs to exactly one product.**
  - **Justification for rejecting optionality of zero:** A product release cannot exist independently of a product.
  - **Justification for rejecting multiplicity of "many":** A product release is linked to the specific product that generated it.

- **(R2) A request has exactly one product release.**
  - **Justification for rejecting optionality of zero:** A request must specify the product release it pertains to.
  - **Justification for rejecting multiplicity of "many":** Each request pertains to only to a specific product release.

- **(R2) A product release can be part of zero or many requests.**
    - **Justification for rejecting optionality of one:** There may be instances where no requests are made for a particular product release.
    - **Justification for rejecting multiplicity of one:** Multiple requests can be made for a single product's release, such as multiple customer complaints.

- **(R3) A request belongs to exactly one change.**
    - **Justification for rejecting optionality of zero:** A request must always be associated with a change item to receive a change ID.
    - **Justification for rejecting multiplicity of "many":** Each request corresponds uniquely to a specific change item and its change ID.

- **(R3) A change has one or many requests.**
    - **Justification for rejecting optionality of zero:** A change item is created based on a request. Without any request, a change item would not exist.
    - **Justification for rejecting multiplicity of one:** A single change can address similar requests from different requesters.

- **(R4) A requester makes one or many requests.**
    - **Justification for rejecting optionality of zero:** A requester must have made at least one request, otherwise storing their data is unnecessary.
    - **Justification for rejecting multiplicity of one:** A requester can make multiple requests, such as complaints about different products.

- **(R4) A request is made by exactly one requester.**
    - **Justification for rejecting optionality of zero:** A request cannot exist without a requester who made it. Therefore, a request must be associated with one requester.
    - **Justification for rejecting multiplicity of "many":** Each request is uniquely tied to the requester who submitted it.

- **(R5) A change can have a product release.**
    - **Justification for rejecting optionality of one:** It is not required that a change has an anticipated release ID when it is created.
    - **Justification for rejecting multiplicity of "many":** A change only references a single anticipated product release.

- **(R5) A product release can be part of changes.**
    - **Justification for rejecting optionality of one:** There may be no changes pertaining to a specific product's release.
    - **Justification for rejecting multiplicity of one:** There may be many changes to be made for an anticipated release of the product.

- **(R6) A change has exactly one product.**
    - **Justification for rejecting optionality of zero:** A change must be about a product.
    - **Justification for rejecting multiplicity of "many":** A single change item only references one product.

- **(R6) A product can be part of many changes.**
    - **Justification for rejecting optionality of one:** There may be no changes pertaining to a specific product.
    - **Justification for rejecting multiplicity of one:** There may be many changes for a specific product.

**Key Uniqueness**

- Products must have a unique name to ensure distinct identification.

- Product releases must be uniquely identifiable by the combination of its product name and the release ID. This ensures that even if multiple products have the same release ID, each release can be correctly associated with its specific product. Similarly, a product can have multiple releases, each uniquely identified by different release IDs.

- Requests must be uniquely identifiable by the combination of its change ID and the requester. While a requester can submit multiple requests, each request must pertain to a distinct change item to avoid redundancy.

- Change items must have a unique change ID to allow for clear differentiation between change items.

- Requesters are uniquely identified by their email address. Storing multiple emails for a single requester is unnecessary and would create confusion. A unique email ensures clear identification of each requester.

**Exception Handling**

- For all user inputs, the system will prompt the user to fill in any required field left empty. Furthermore, any field not satisfying its constraints (e.g. change description has too many characters, attempting to create duplicate objects, etc.) will prompt the user to re-enter the field with a valid value.

- In the request form, all fields are required except for priority. If the user enters a non-existent requester or product release, they will be prompted to provide valid information.

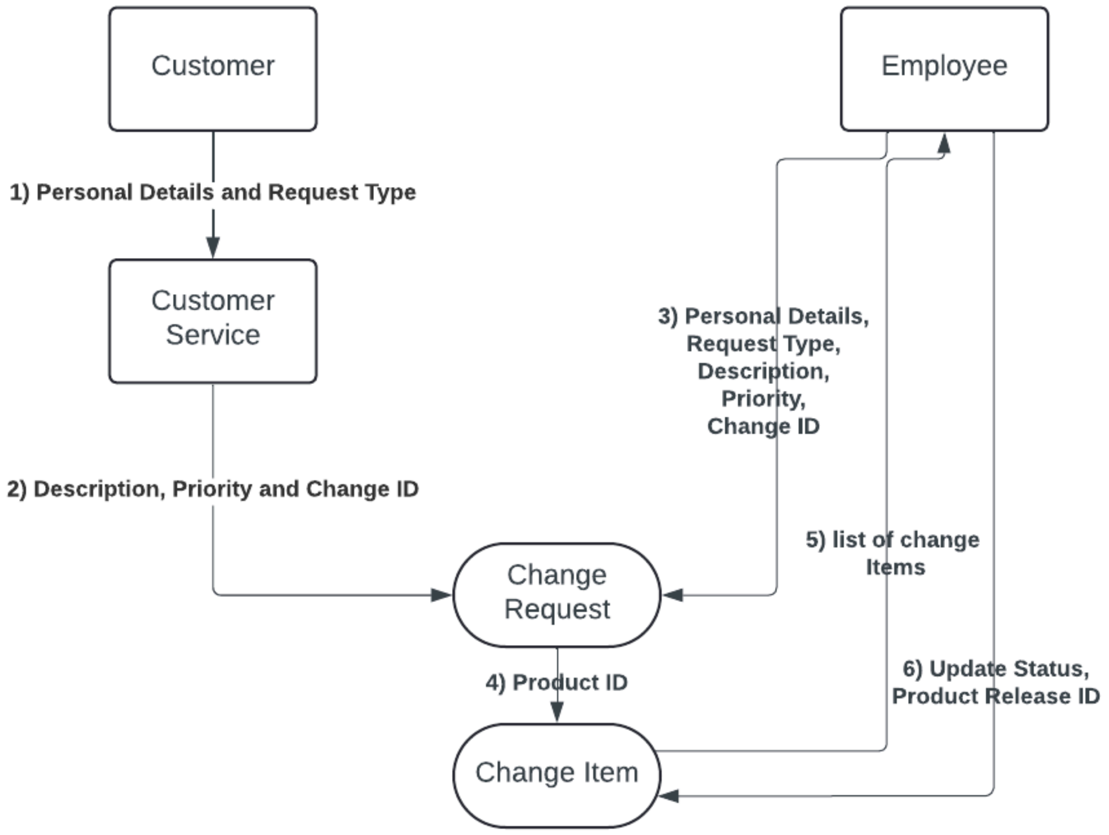
- All fields in a change item are required.

- If a newly auto generated change ID is not unique, the system will make multiple attempts to create a unique one. If generating a unique change ID fails despite multiple attempts, the system will notify the user of this critical error and return them to the main menu without creating the request and change items.

- The fields in the requester and internal requester objects are required

# 5.0) Functional Description

## 5.1) Data Flow Diagram

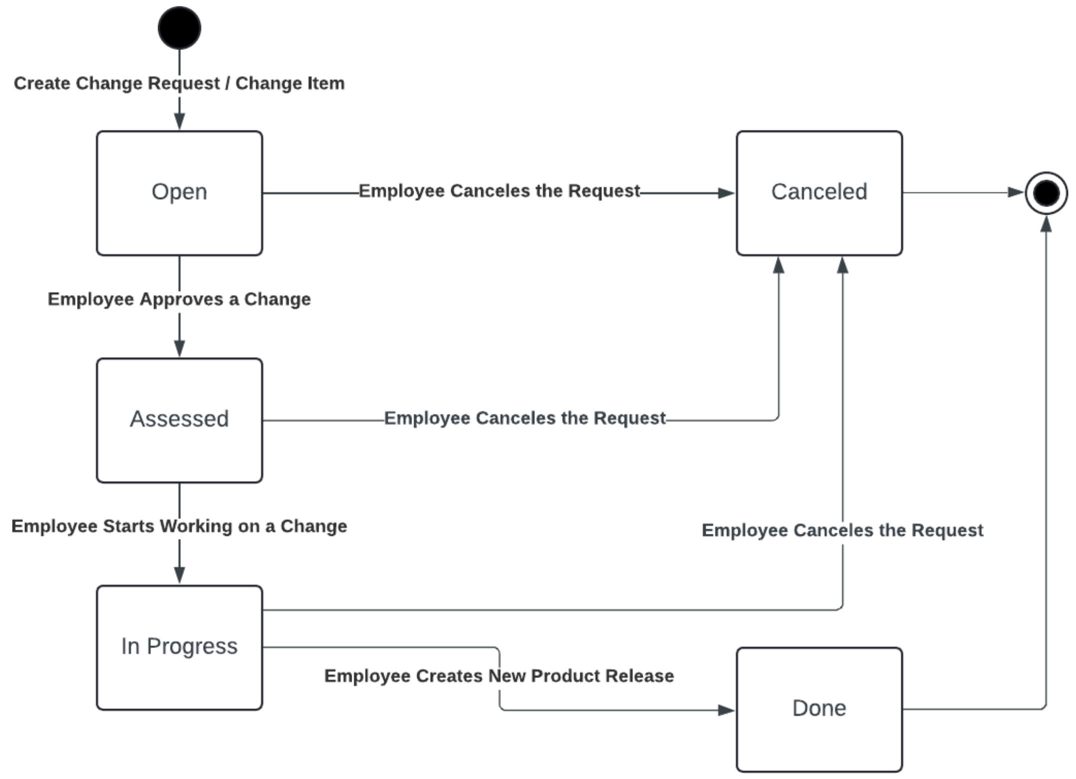

Customer
Employee
1) Personal Details and Request Type
Customer Service
3) Personal Details, Request Type, Description, Priority, Change ID
2) Description, Priority and Change ID
5) list of change Items
Change Request
4) Product ID
6) Update Status, Product Release ID
Change Item

## 5.2) Process Description

When a customer finds an issue, the customer provides personal details and the request through a call to customer service, a customer service representative will then create a description, priority and a change ID for the request. An employee is also allowed to create change requests, they need to provide all personal details, request type, description, priority and create change ID to the change request. The change request will pass the product ID to the change item, a list of change items will be displayed to the employee. After the decision from the employee, the employee will update the status and include a product release ID if applicable.

# 6.0) State/Control Model

## 6.1) State Diagram


Create Change Request / Change Item
Open
Employee Canceles the Request
Canceled
Employee Approves a Change
Assessed
Employee Canceles the Request
Employee Starts Working on a Change
Employee Canceles the Request
In Progress
Employee Creates New Product Release
Done

## 6.2) State Discussion

The state model consists of 5 states for an issue for the tracking system. These are defined as Opened, Assessed, In Progress, Done and Canceled. The “Open” state refers to when a change request and change item is created. “Assessed” refers to the issue being approved by the employee. “In Progress” refers to an issue currently being added/fixed by an employee. “Done” refers to an issue being successfully added/fixed and a new product release is available. “Canceled” refers to an issue decided as worthless and the issue is canceled by a customer service representative or an employee.

When an issue is first brought to the system, from a customer call to a customer service representative or from an employee request, a change request and change item will be created and the system will stay in open state. The employee will then make a decision, they can bring it to the “Canceled” state if it is an invalid issue, or bring it to the “Assigned” state after approval.

After a change is assessed, the employee can decide the issue is invalid and bring it to Canceled state, or bring it to In Progress state. While the system is in “In Progress” state, the employee will start working on adding or fixing the change. After the employee is finished with the work, they can bring it to the Done state and create a new product with a new product release ID.

## 7.0) Dictionary

**Characters -** (“A” | “B” | “C” | … | “Z” | “a” | “b” | “c” | … | “z”)

**Integer digits -** (“0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”)

**Assessed State -** An issue has been approved by the employee.

**Canceled State -** An issue is decided as worthless and no further action is required.

**Change -** An object created by employees, which addresses 1 or more request items.

**Change ID -** A unique combination of integers to identify a change request and change item.

**Customer -** A user who owns a product from **Macrohard**.

**Customer Service -** A customer service representative employee that helps a customer submit a request.

**Date -** In format YYYY/MM/DD, stored as 8 digit integer.

**Department -** An attribute of an employee, consists of a string of only characters with limit 12. See employee for details.

**Description** - Description of the request, a string that includes integers and characters limited to 30.

**Done State -** A change has been finished and a new product release ID is created.

**Email address -** A string that includes integers and characters, with a string size limit of 30.

**Employee -** A user who is employed under **Macrohard**.

**Internal Requester -** An object that represents an employee who makes requests.

**In Progress State -** An employee is working on the change.

**Name -** The user’s name. Consists of a string of only characters with limit 15.

**Open State -** A change request and change item is created pending approval by the employee.

**Personal Details -** Includes name, email address and phone number of the user. See specific attributes for details.

**Phone Number -** 10 digit integer representing phone numbers from BC only.

**Priority -** An integer from 1 to 5. 5 as the highest, 1 as the lowest.

**Product -** An object representing a product of **Macrohard** with a unique product name.

**Product Name -** The product's name. Consists of a string of only characters of limit 15.

**Product Release -** An object holding data on a specific version of a product.

**Release ID -** A unique combination of integers to identify a product release.

**Request -** An object created by either a customer service representative or a software/QA engineer for review, requesting a fix for an issue or the addition of a feature.

**Requester -** An object representing a customer or employee who submits requests.

**Status -** A string describing the status including: "Open", "Assessed", "In Progress", "Done" and "Canceled". See specific status for details.

# 8.0) Performance/Capacity

## 8.1) Response Times

The response time of our system should be under 1 second. For example, displaying a list of change requests, submitting a change request, updating a status, should get a response within a second.

## 8.2) Throughput

Assuming there are 50 customer change requests every day, and 50 employee change requests every day. Also assume there are 365 days in a year, the maximum number of change requests in the system per year is as follows:

= (# of customer issues + employee issues) * numbers of days in a year
= (50 + 50) * 365
= 36500

Assuming for every 2 change requests, 1 change item will be created. Therefore the maximum number of change items in the system per year is as follows:

= ( # of change requests per year) / 2
= 18250

The system is designed to handle receiving 36500 change requests and 18250 change items per year.

## 8.3) File Capacity

- An integer digit combination is 4 bytes.

- Each independent character is 1 byte, each integer digits are considered 1 byte if it is part of a character-integer combination.

**Request Item Size** = changeID + email + productReleaseID + priority
= 4 + 21 + 4 + 4
= 33 bytes

**Change Item Size** = changeID + status + product Name + anticipatedReleaseID + description + date
= 4 + 11 + 15 + 4 + 30 + 4
= 68 bytes

**Requester Item Size** = email + name + phone
= 21 + 15 + 4
= 40 bytes

**Internal Requester Item Size** = department
= 15 bytes

**Product Item Size** = productName
= 15 bytes

**ProductRelease Item Size** = productName + releaseID + releaseDate
= 15 + 4 + 4
= 23 bytes

**Number of Change Requests** = 36500 per year

Assuming the system will have 100 product releases, with ratios of 50 change items per release, 2 change requests per change item, 1.5 requests per requester. There are:

# of change items = 50 * 100 = 5000
# of change requests = 2 * 5000 = 10000
# of requesters = 10000 / 1.5 = 6666

Total data size = (10000 * 33) + (5000 * 68) + (6666 * 40) + (23 * 100)
= 938940 bytes
= 939 kilobytes (kB)

Assuming there to be 2000 lines of code, worth 100 kilobytes:
Total system size = 939 kilobytes + 100 kilobytes= 1.039 megabytes

# 9.0) Prototype and Future Releases

## 9.1) Prototype

The following features will not be implemented in the prototype since they do not have an effect in the core functionality of the software:

- The ability to delete a specific product release.
- The feature to change the priority on a certain request.
- The feature to change the statues on a certain request.

## 9.2) Future release

The following features will be added to feature releases:

- The ability for the customer service representative to track the requests for a specific customer in order to give updates on the progress made on their requests.
- The feature to track what software and specific release does each customer own.
- The addition of a secure login portal for employees. This feature will bring security in for our software and also disallow access to certain features for employees of different rank.
- The feature to identify and disallow inappropriate phone number and email inputs.
- The addition of an estimated cost in dollar amount for a requested feature for a customer.
- Ability to distinguish between a bug and a feature report.
- Assignment of a unique ID for employees and customers for easier access to their information.
- Addition of a description section for each product release to highlight the changes.

# 10.0) Acceptance Criterion

## 10.1) Hard Requirements

The hard requirements for the program are specified in Section 8, which specifies the performance/capacity of the software. This means that all inputs prompted by the user will return an output within a second.

## 10.2) Goals

The system prioritizes user-friendliness to ensure a promising experience for both customers that are reporting issues and employees that are entering requests. Furthermore, we hope to reach a soft goal of increased productivity by simplifying the process of submitting and tracking issues. This can be quantified as a percentage increase in the number of issues resolved within a given timeframe.

## 10.3) Acceptance Test Requirements

The completed program will be subjected to testing within the designed environment specified in Section 2 of this document. Before deployment, Macrohard's quality assurance engineers will test all functionalities related to creating requests, managing change items, issue status updates, and all potential use-cases. Following the installation on customer-owned hardware, identical tests will be run to ensure optimal performance. Customers are encouraged to provide feedback on the program or report any bugs detected within the inital month of the software's installation.

# Appendix A

## Glossary

**Agile development -** A methodology focused on iterative progress, collaboration, and flexibility in responding to changes throughout the software development lifecycle.

**Database -** An organized collection of structured information or data, accessible for efficient retrieval, management, and updating.

**Deliverable -** A tangible output produced as a result of this project, such as this requirements specification document.

**Field -** A specific type of information, like a name or email address, that a user enters into the system and which is recorded in the database records.

**Legacy System -** An old software system used by a company.

**Object -** A specific entity, like a request or product, which has as its own unique properties or characteristics, and can be connected to other objects to show how they relate to each other or interact within the system..

**Scrum -** An Agile framework that facilitates team collaboration and iterative development through defined roles, events, and deliverables to manage and complete complex projects.

**Server -** A computer that provides services, resources, or data to other computers, known as clients, over a network.