# 

# Title Page

**Remarkably Busy Business’s**

**Issue Tracker System**

**REQUIREMENTS SPEC**

**Release 1**

Release Date: 2024/06/07

Approved by:

Project Manager A: \_\_\_\_\_\_\_\_\_\_\_

Project Manager B: \_\_\_\_\_\_\_\_\_\_\_

Project Manager C: \_\_\_\_\_\_\_\_\_\_\_

Project Manager D: \_\_\_\_\_\_\_\_\_\_\_

# Version History Page

Version 1 -24/06/07

-release of the preliminary requirements spec

Version 2 -24/06/

-added title page and release history

-updated context diagram, ORD, State Diagram

-updated use cases

# 

# Table Of Contents

[**Title Page 1**](#_nrds4zj3vaay)

[**Release History Page 2**](#_jkchnfddgf7y)

[**Table Of Contents 3**](#_gxfpb8edmhtb)

[**Section 1: Introduction 4**](#_b37ki1mxa9y)

[1.1: System Overview 4](#_r5icj4gpukmw)

[1.2: Business Objective 4](#_y959a7hgpq23)

[1.3: Software Project Outline 4](#_g96332nvy9y6)

[**Section 2: Environment 5**](#_yja8rygdhy46)

[2.1: Context Diagram 5](#_u21z2b4the6q)

[2.2: Hardware 5](#_7myorilqs598)

[2.3: Operating System 5](#_x3yzq972ldw0)

[2.4: Development Environment 5](#_68ylyghn7eq1)

[2.5: Maintenance Environment 6](#_9ttr53x3m59r)

[**Section 3: User Interface and Operations 6**](#_3cosrncchh03)

[3.1: User Interface Sophistication 6](#_bm4txsflkjbv)

[3.2: Use Case Scenarios 6](#_v2qv92rgt1r8)

[3.3: Scenarios Started by Other Event Sources 10](#_1flf8m72p6fm)

[**Section 4: Retained Data Model 10**](#_770f7q54wz28)

[4.1 Object Relationship Diagram 10](#_p0zg67jfc4r4)

[4.2 Discussion of Cardinalities 10](#_v43cbsgfnxc4)

[**Section 5: State Control Model 13**](#_yyo7b3rv4zb5)

[5.1: State Diagrams 13](#_np4erdsnm16z)

[5.2: State Diagrams Discussion 13](#_gxyx3s2r1ecb)

[**Section 6: Data/Process/State Dictionary 14**](#_j2p64f8xb8vq)

[**Section 7: Performance and Capacity 17**](#_kvqdmpofmnbi)

[7.1: Response Times 17](#_5c1bbu6g8ir3)

[7.2: Throughput 18](#_vxcvtnhirywa)

[7.3: File Capacity 19](#_2m3pxjhqsufl)

[**Section 8: Prototype and Future Releases 20**](#_hg4u5upv50p6)

[8.1: Prototype Considerations 20](#_wu0dk2ae1y6l)

[8.2: Future Release Considerations 21](#_bb5unaj5ki1d)

[**Section 9: Acceptance Criterion 21**](#_9unut1y171jv)

[9.1: Hard Requirements 21](#_6cqg7abdmu1)

[9.2: Goals 22](#_7tzdhaby6pox)

[9.3: Acceptance Test Requirements 22](#_9tlh5o8ccnt1)

[**Appendix 22**](#_vq9tsamrrj9y)

# **Section 1: Introduction**

## **1.1: System Overview**

The system that we will be designing will be used by all kinds of software companies in order to help them track any bug fixes or enhancements made to their products. Products that software companies make are almost never perfect on release or even after a couple of big fixes. In an industry where companies are competing to produce the best software, it is essential for them to constantly improve their products to gain advantages over their competitors.

The purpose of an issue tracking system is to help software companies track bug fixes and enhancements made to their products. This will be done by either receiving a bug report or enhancement request, from a customer or an employee of the company. After being reported, all of the required info about a change item will be entered into the system. The program can also report information about a particular change item, or all change items in a release to the user. Finally, once a change item has been completed, everyone who needs to be, will be informed.

Software companies are in one of the most competitive markets in the modern day with technology taking over society. Due to this, it is crucial for them to get an advantage over other companies by effectively organising their work. By using our software, companies can reduce the time it takes to distribute work among programmers, and reduce the chance of wasting time working on already completed or unimportant tasks. Our program will help our customers focus on improving their products.

## 1.2: Business Objective

We will be designing software for a medium sized company, Capsule Corporation. We are given a budget of $100,000 and are required to finish it in three months. The software is designed for Capsule Corporation and no other companies.

## 1.3: Software Project Outline

Due to the ever evolving nature of the software industry it is essential to start immediately on the software. This project will require a team of four programmers and the software will be completed and delivered within three months. We will need to interview representatives from Capsule Corporation in order to get additional information and insight about how they will use the program.

# Section 2: Environment

## 2.1: Context Diagram

In an issue tracking system, employees and customers request changes for some release of some product. This creates a “change item,” a description of a desired change for that product. This issue tracker reports these change items to software development teams. Those teams then make the changes described by the change items and include them as part of the future releases of that product. Finally, those releases of the product are distributed to customers.

## 2.2: Hardware

The program does not require any specialised hardware. Any workstation meeting some minimum specifications (16 GB RAM, a 9th generation i5 intel processor or above, and 1 TB SSD storage) will be suitable to run the program. We strongly recommend a second disc to backup records as a safeguard against hardware failure. We also recommend that you are able to increase disk space in the case that you use the system with greater throughput or, over a longer period, than you anticipate.

## 2.3: Operating System

The program will be developed for and tested on Ubuntu 22.04.3 LTS, (Jammy Jellyfish) and the program will be explicitly supported only for this OS and release. Perhaps in a further contract the program could be updated to support more platforms or releases.

## 2.4: Development Environment

The program will be developed using GNU developer tools such as the GCC compiler and debugging tools. The program will be written in C17 and C++17. The program will be tested on Ubuntu 22.04.3 LTS, (Jammy Jellyfish) in a workstation with an Intel Core i9-13900, and 32 GB of RAM.

## 2.5: Maintenance Environment

As the program is being developed exclusively for the customer, they will be provided the source code and documentation to maintain, fix, update, and modify the software in any form they feel is appropriate.

If an update changes the amount of information stored in a record the database files will need to be translated to maintain fixed length records and minimise disk access times.

As the record files are separate to the program, as long as records are not modified, updating to a new release will only require restarting the program, and running the executable of the new release.

# Section 3: User Interface and Operations

## 3.1: User Interface Sophistication

We will be using scrolling terminals for our user interface. The terminal will present the user with options that the user can choose to complete their actions.

## 3.2: Use Case Scenarios

**Case 1:**

**Create a Product**

Precondition:

No preconditions for this use case.

Description:

The user wants to register a product in to the system. In order to do this, a new ‘product’ object must be created.

Steps:

1. Through various prompts, the user will enter a ‘adding a product’ menu.
2. The user will be prompted to enter the name of the new product.
3. The user will be prompted to confirm their action.

Expected Results:

A new product with the information that the user entered is successfully created.

Variants & Exceptions:

* If the information given matches an existing product, the system will not allow the user to add the product. (exception thrown)
* The name of the product must be 1-10 chars. (exception thrown)

**Case 2:**

**Create a Product Release**

Preconditon:

The product that the release is for, must already be in the system (see Use Case 1 for creating a product).

Description:

The user wants to create a product release for a product. In order to do this, a new ‘product release’ object must be created.

Sequence:

1. Through various prompts, the user will enter a ‘create a product release’ menu.
2. The user will be prompted to select the product that the release is for.
3. The user will be prompted to enter an ID for the new release.
4. The user will be prompted to enter the date for this new release.
5. The user will be prompted to confirm their actions.

Expected Results:

A new product release with the information that the user entered is successfully created.

Variants & Exceptions:

* If the information given matches an existing product product release, the system will not allow the user to add the product release. (exception thrown)
* The release ID must be 1-8 chars. (exception thrown)
* The date entered must be in the format YYYY-MM-DD. (exception thrown)

**Case 3:**

**Create a Change Request**

Preconditions:

A change request must have at least 1 requester to be made. The product that this request is for, must exist. The release that this request is for, must exist.

Description:

The user wants to create a change request. In order to do this, a new ‘change request’ object must be created.

Steps:

1. Through various prompts, the user will enter a ‘create a change request’ menu.
2. The user is prompted to select a requester from a list of existing ones, for this request. If the requester does not exist, the user must create them first. (see Use Case 4 for creating a customer)
3. The user is prompted to select the product that the change request is for.
4. The user is prompted to select the release that the change request is for.
5. The user is prompted to enter the date of the request.
6. The user is prompted to enter the description of the request.
7. The program lists change items in the system, for the user to check if the description matches an existing change item. If a matching change item does not exist, the user must create one. (see Use Case 5 for creating a change item)

Expected Results:

A new change request with the information that the user entered is successfully created.

Variants & Exceptions:

* The date entered must be in the format YYYY-MM-DD. (exception thrown)
* The description of the change request must be 1-30 chars. (exception thrown)

**Case 4:**

**Create a Requester**

Precondition:

The requester must have made a request (see Use Case 3 for creating a change request). In order to do this, a new ‘requester’ object must be created.

Description:

The user wants to create a requester in the system, in order to create a change request.

Steps:

1. The user enters a ‘create a requester’ menu.
2. The user is prompted to enter the requester’s name.
3. The user is prompted to enter the requester’s phone number.
4. The user is prompted to enter the requester’s email address.
5. The user must select if the requester has a department or not. If yes, the user is prompted to enter the requester’s department.
6. The user will be prompted to confirm their actions.

Expected Results:

A new requester with the information that the user entered is successfully created. The program creates a ‘requester ID’, to reference the requester in the future.

Variants & Exceptions:

* If the information given matches an existing requester, the system will not allow the user to add the requester (exception thrown). The user may choose to either continue making a change request, or return to the main menu.
* The name of the requester must be 1-30 chars. (exception thrown)
* The phone number of the requester must be 10 digits. (exception thrown)
* The email of the requester must be 1-24 chars. (exception thrown)
* If exists, the department of the requester must be 1-12 chars. (exception thrown)

**Case 5:**

**Create a Change Item**

Preconditon:

A change request must exist for this change item.

Description:

The user wants to create a change item. In order to do this, a new ‘change item’ object must be created.

Sequence:

1. The user enters a ‘create a change item’ menu.
2. The user is prompted to enter the priority of the change item.

Expected Results:

A new change item with the information that the user entered is successfully created. The program creates a ‘change ID’, to reference the request in the future.

Variants & Exceptions:

* No exceptions (the user is guided to select a valid priority for the change item).

**Case 6:**

**Update / Cancel a Change Item**

Preconditon:

The change item to be updated must exist. If updating a release, the release must already be in the system (see Use Case 2 for adding a product release).

Description:

The user wants to update a change item or cancel one (update status to ‘Cancelled’).

Sequence:

1. The user enters an ‘update change item’ menu.
2. Through various filters that the user may select to narrow down their selection, user selects the change item that they want to update.
3. The user selects whether they want to update the description, priority, status, or release.
4. The user will be prompted to enter the new description, priority, status, or release.

Expected Results:

A change item is successfully updated.

Variants & Exceptions:

* The new description of the change request must be 1-30 chars. (exception thrown)
* The user cannot update the status of a change item that is in a ‘Done’ or ‘Cancelled’ state. (exception thrown)

**Case 7:**

**Create a Report**

Preconditon:

No preconditions for this use case.

Description:

The user wants to create a report. This report can be either a report of all requesters that need to be informed about a change item, or all change items of a product that are not done or cancelled.

Sequence:

1. The user enters a ‘create a report’ menu.
2. The user is prompted to choose the type of report that they want to create (a report of all requesters that need to be informed about a change item, or all change items of a product that are not done or cancelled).
3. If choosing the first type of report ( all requesters that need to be informed about a change item):
   1. The user is prompted to select the product associated with the change item.
   2. The user is prompted to select the change item.
4. If choosing the second type of report (all change items of a product that are not done or cancelled):
   1. The user is prompted to select the product
5. The user confirms their actions.

Expected Results:

A new change item with the information that the user entered is successfully created. The program creates a ‘change ID’, to reference the request in the future.

Variants & Exceptions:

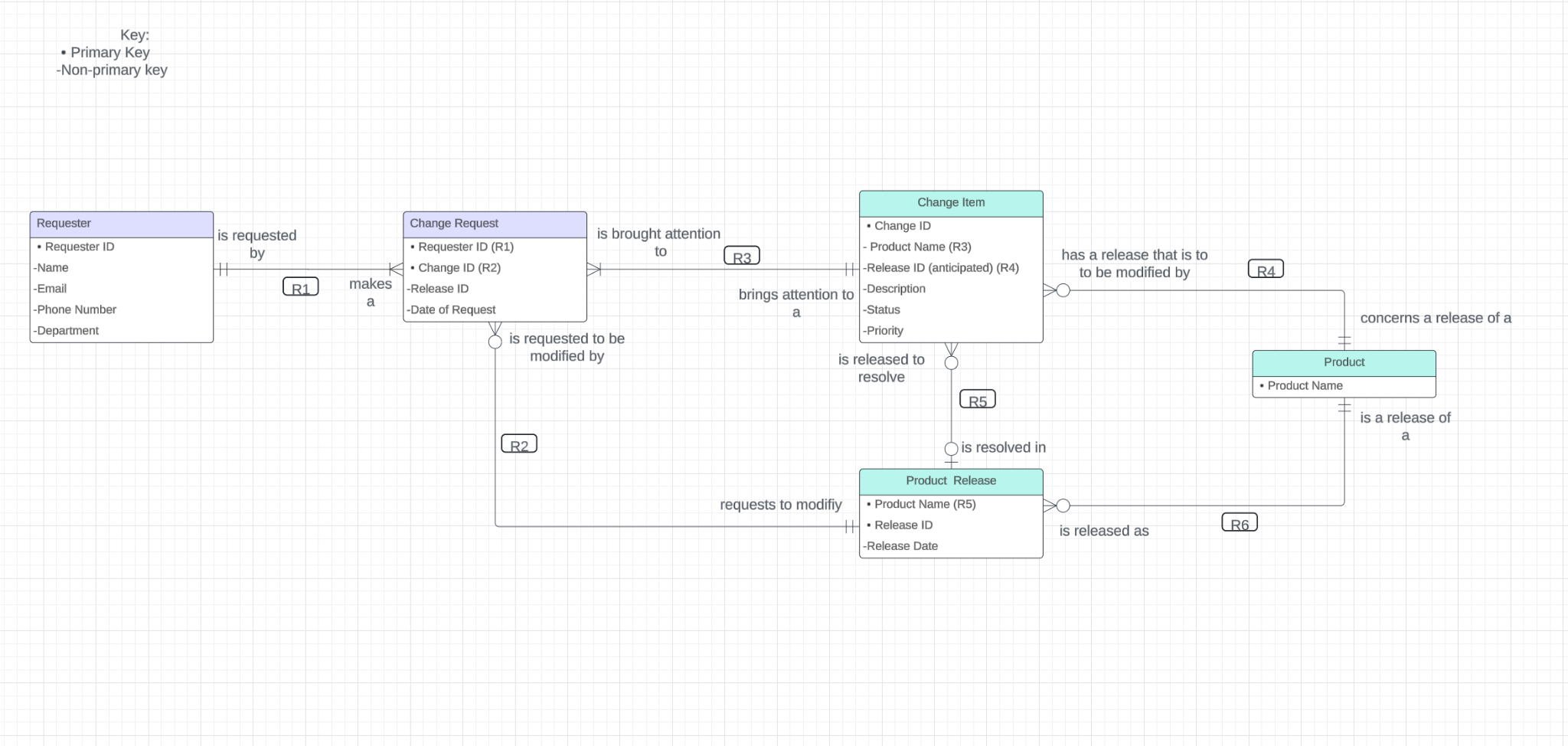
* No exceptions (the user is guided to select a valid priority for the change item).

## 3.3: Scenarios Started by Other Event Sources

There are no scenarios started by other event sources. All use case scenarios are started by the user’s interaction with the system.

# Section 4: Retained Data Model

## 4.1 Object Relationship Diagram

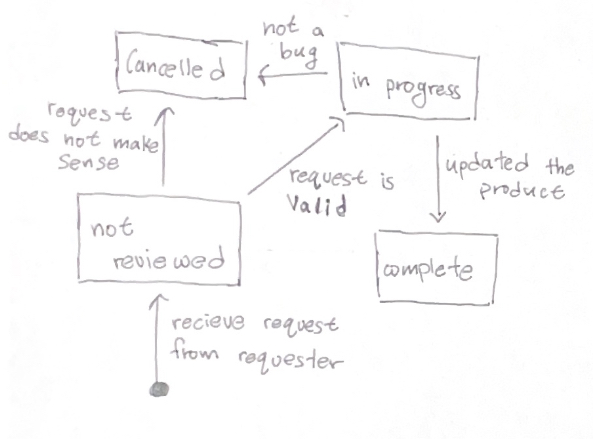
****

## 4.2 Discussion of Cardinalities

1. **Requester makes one or more Change Requests (R1**):
   1. A person is registered as a requester when they make their first Change Request, and not before that. This means the minimum Change Requests a Requester makes must be one. As a result (0..1) and (0..\*) cardinalities are not applicable.
   2. Any requester can make multiple Change Requests, and is not limited to just one. Consequently. The (1..1) cardinality is not applicable.
2. **Change Request is requested by one and only one Requester (R1)** :
   1. A Change Request cannot exist without a Requester requesting it, meaning there must be a minimum of one requester associated with each request. As a result, the cardinalities (0..1), (0..\*) do not apply.
   2. Each new person making a request generates a new Change Request. As a result, no change request can have multiple Requesters. The cardinality (1..\*) is therefore not applicable.
3. **Change Request request to modify one and only one Product Release (R2)**:
   1. A change request will always address a specific product, since requesters must mention which product release they are requesting a change to. 0 can therefore not be an option, and the cardinalities (0..1), (0..\*) do not apply.
   2. A change request cannot be targeting multiple product releases at the same time. The cardinality (1..\*) does not apply as a result.
4. **Product Release is requested to be modified by 0 or more Change Requests (R2)** :
   1. When a product is freshly launched, it is possible for there to be no requests for modifying it. As a result, the cardinalities (1..1), (1..\*) do not apply.
   2. Once a product release is out, there can be numerous changes it requires, and so multiple change requests are likely to be made. As a result, the cardinality (0..1) does not apply.
5. **Change Request brings attention to one and only one Change Item (R3)** :
   1. Each change request created must have a Change Item they are associated with via the Change ID. If the Request is unique, a Change Item is generated in the process of generating the Change Request. As a result, there must be at least one Change Item per Request. The cardinalities (0..1), and (0..\*) thus do not apply.
   2. Each change request is associated with a single change item. If the requester requests multiple changes, each one turns into a separate request. So, since a Change Request cannot bring attention to multiple Change Requests, the cardinality (1..\*) does not apply.
6. **Change Item is brought to attention one or more Change Requests (R3)** :
   1. A Change Item is only created when Requested. Since there cannot be 0 Change Requests for an Item, the cardinalities (0..1) and (0..\*) are not applicable.
   2. There can be multiple Change Requests for the same Item, and therefore, the cardinality (1..1) does not apply.
7. **Change Item concerns a release of one and only one Product (R4)**:
   1. A Change Item by definition reports on modifications to be made on a product. It is therefore impossible for it to be associated with no products, making the cardinalities (0..1), (0..\*) not applicable.
   2. If multiple Products have the same problem/ change, each one gets its own Change Item. A single Change Item only addresses a single Product, meaning the cardinality (1..\*) does not apply.
8. **A product has 0 or more Change Items (R4):**
   1. When a product has just been launched, it is possible for it to have no requested changes for a brief period. Since 0 must be an option, the cardinalities (1..1), and (1..\*) do not apply.
   2. A single product can have multiple issues, and is not limited to one. Thus, the cardinality (1..1) does not apply
9. **Change Item is resolved in 0 or 1 Product Release (R5):**
   1. When the anticipated release is yet to be decided for a Change Item, it is not associated with any releases.. As a result, 0 must be an option, and (1..1) and (1..\*) cardinalities are not applicable.
   2. A single change item must be resolved in a specific release. So, the cardinality (1..\*) does not apply.
10. **Product Release has been modified to resolve 0 or more Change Items (R5):**
    1. The first version of a product ( the first release) may not have been released as a modification of any change items. Since 0 must be an option, (1..1), and (1..\*) do not apply.
    2. A single product release can address multiple change items, and is not restricted to a single one. As such, the cardinality (0..1) does not apply.
11. **Product Release is a release of one and only one Product (R6):**
    1. A product release is always a release of some product, and therefore needs a minimum of 1. The cardinality (0..1), (0..\*) therefore do not apply.
    2. A product release can never be for multiple products at once. As a result, the cardinality (1..\*) is not applicable.
12. **Product has 0 or more Product Release (R6)s:**
    1. When a product is in the process of development, it will exist but not have a product release. Since 0 must be an option, the cardinalities (1..1), and (1..\*) do not apply.
    2. Each product can have multiple releases. In fact, resolved change items are available to the public through new releases. As a result, the cardinality 01..\*) is not applicable.

# Section 5: State Control Model

## 5.1: State Diagrams

****

## 5.2: State Diagrams Discussion

The state diagram has four states, not reviewed, in progress, cancelled, and complete. The not reviewed state is when the request is first received from the requester. In progress is when the bug is being worked on. Cancelled is when the bug doesn’t make sense. Complete is when the bug has been updated.

When a requester wants to request a change, a change request is then entered. This leads to the first state where the request has not been reviewed yet. After this, when a programmer requests to view the request and enters the required information, the state then moves to in progress if the programmer determines that the request is valid. On the other hand, if the programmer decides that the request doesn’t make sense, the state will then be moved to cancelled.

After the programmer has fixed the bug and updated the product, the state will be moved from the in progress state to the completed state. However, when trying to fix the bug, the programmer could determine that the bug is no longer valid, so the state will be moved to the cancelled state.

# Section 6: Data/Process/State Dictionary

Note: terms in bold have separate entries in the dictionary

**Cancelled**

**State *(of Change Item)***

The state of a **change request** set by an engineer in the case of it being deemed unnecessary/ groundless when first reviewed, or deemed false upon further investigation later on in the process (eg.: it was not a bug; it was user error). Irreversible.

**Change Item**

**Object**

An object that stores information about a bug report/feature request that has been registered . It includes a unique **Change ID**, a **request ID** corresponding to the details of the original request submitted, a **priority** value, a status, and the anticipated **release ID** for the product where the request has been fulfilled.

**Change ID**

**Attribute *(of Object: Change Item);***

***Requirements: Must be an integer between 0 and 999999***

A unique identifier for each **Change Item** in the system. All Change Items are sorted by Change ID.

**Change Request**

**Object**

An object that stores information from a caller reporting a bug or requesting a new feature. The object holds a unique **Request ID** for the request, a **requester ID** that corresponds to the details of the individual making the request, a **Bug ID** that corresponds to the details of the actual bug/feature being reported/requested, a **status** that marks the progress of the request, and the specific **release ID** the request mentions.

**Change Requester**

**Object**

A change requester holds the information of an individual who is entered into the Issue Tracking System as a caller reporting a bug/ requesting a feature. The has a unique **Requester ID** for each person, a **name**, a **phone number**, an **email**, and a **department**.

**Date of Request**

**Attribute (*of Object: Change Request);***

***Requirements: YYYY-MM-DD format (10 characters), separated by a dash.***

The date on which a **change request** was made..

**Description**

**Attribute *(of Object: Bug);***

***Requirements: Length must be between 1-128 characters***

A detailed description of the report or request made by the **change requester.** Must be unique for each bug corresponding to the same product.

**Department**

**Attribute *(of Object: Change Requester);***

***Requirements: Length must be between 1-12 characters***

The department of the **change requester.** Must be an internal department of the company if the requester is an employee; if the requester is not an employee, the department is set to “customer”.

**Done**

**State *(of Change Item)***

The status of a **change request** when it has been resolved. Irreversible.

**Email**

**Attribute *(of Object: Change Requester);***

***Requirements: Length must be between 1-24 characters***

The email address of the **Change Requester**.

**In Progress**

**State *(of Change Item)***

The status of a **change request** when the engineer deems is necessary or worthy upon initial review. They can either change it to “**Done**” when it has been addressed, or **Cancel** it if it turns out to be false.

**Issue Tracker System**

***Program***

The Issue Tracker System is the entire program that allows for a company to view all the bug reports and product feature ideas it receives in one place. It allows for the addition of new reports and requests, the modification of existing ones, the rejection of inappropriate ones. In order to keep track of bugs accurately, information about products and people are also stored. The program keeps track of all individuals who have made reports/requests with their details, and allows the users to view , add, and modify them. The program further allows users to view and add, and products and their releases (users will have the ability to delete in a future release).

**Name**

**Attribute *(of Object: Change Requester);***

***Requirements: Length must be between 1-30 characters***

The full name of the person who makes the bug report/ feature request.

**Not Reviewed**

**State *(of Change Item)***

The Change Item has been entered into the system but has not yet been reviewed by an engineer. Once reviewed, the engineer can then decide its relevance and priority. They could either **Cancel it** or mark it as **In Progress**.

**Phone Number**

**Attribute (*of Object: Change Requester);***

***Requirements: Must be a phone number in the US/Canadian style (+1 ### ### ####) - 10 digits***

A US/Canadian 10-digit phone number of the **Change Requester**.

**Priority**

**Attribute (*of Object: Change Item);***

***Requirements: A number between 1 - 5.***

A value assigned to each **Change Item** in order to determine their order or address. Will always be a value between 1 and 5, with 1 being the highest priority and 5 being the lowest..

**Product**

***Object***

An object that holds the Product IDs of each product.

**Product Name**

**Attribute (*of Object: Product);***

***Requirements: Length must be between 1-10 characters; must be unique***

A unique identifier / name for each product in the company.

**Release Date**

**Attribute (*of Object: Product);***

***Requirements: YYYY-MM-DD format (10 characters), separated by a dash.***

The date on which a product’s specific **release ID** was released or made available.

**Product Release**

***Object***

An object that holds details of a product release. This includes the **product ID**, the **release ID**, and the **release date**.

**Product Number**

**Attribute (*of Object: Product);***

***Requirements: A numerical value between 1 - 9999.***

A unique ID used to identify each existing **product** the company offers.

**Release ID**

**Attribute (*of Objects: Product Release, Change Item );***

***Requirements: Length must be between 1-8 characters***

A unique identifier for the version of the product being released. When mentioned inside a change item, it is the anticipated release item. Until an anticipated product release has been determined, the attribute is left blank.

**Request ID**

**Attribute (*of Object: Change Request);***

***Requirements: A numerical value between 1 - 999999.***

A unique ID used to identify each **change request** entered into the system, regardless of its **status**.

**Requester ID**

**Attribute (*of Object: Change Requester);***

***Requirements: A numerical value between 1 - 9999.***

A unique ID used to identify each **change requester** entered into the system, employee or customer.

**Status**

**Attribute (*of Object: Change Request);***

***Requirements: must be set to one of “Not Reviewed,” “In Progress,” “Done,” or “Cancelled”***

An indication of the status of a request. Once reviewed, it may be ‘cancelled,’ (irreversible), “In progress,” or “Done” (irreversible).

# Section 7: Performance and Capacity

## 7.1: Response Times

The response time of our system should take no longer than 5 seconds. This applies to all system operations. One thing we can test with a stopwatch later is requesting the system to display a list of all customers and staff who need to be informed about a particular change.

## 7.2: Throughput

For throughput calculation, we will be making these assumptions:

* 100 product releases
* 50 changeItems per release
* 2 change requests per changeItem
* 50% internal requests
* 1.5 requests per requester
* 15 departments

To calculate the number of ChangeItems the company is creating, we will use this equation:

# of ChangeItems = (# of Product Releases) \* (# of ChangeItems per

release)

= (100) \* (50)

= 5000

To calculate the number of change requests the company is receiving, we will use this equation:

# of Change Requests = (# of ChangeItems) \* (# of Change Requests per

ChangeItem)

= (5000) \* (2)

= 10,000

To calculate the number of requesters the company is registering, we will use this equation:

# of Requesters = (# of Change Requests) / (# of Requests per Requester)

= (10,000) / (1.5)

= 6,667 approximately

Throughput:

* 100 Product Releases
* 5000 Change Items
* 10,000 Change Requests
* 6,667 Requesters
* 70 Products (Additional Assumption)

## 7.3: File Capacity

1 char = 1 byte

1 int = 4 bytes

We will be using the max number of bytes for the information that is variable based on the customer information (name, email, etc.).

For Reference:

Product Name: 1-10 chars

Release ID: 1-8 chars

Change ID: Unsigned Int (0-999999)

Email Address: 1-24 chars

Department: 1-12 chars

Description: 1-128 chars

Priority: Unsigned Int (ranging from 1 - 5)

Format of the Phone Number: US / Canada +1 ### ### ####

Dates: YYYY/MM/DD

Requester Name : 1-30 chars

**Calculations:**

Requester = Change ID + Name + Email + Phone Number + Department

= 4 + 30 + 24 + 15 + 12

= 85 bytes

Change Request = Requester ID + Change ID + Date of Request

= 4 + 4 + 10

= 18 bytes

Change Item = Change ID + Product Name + Desc + Status + Priority + Release ID

= 4 + 10 + 128 + 12 + 4 + 8

= 166 bytes

Product Release = Product Name + Release ID + Release Date

= 10 + 8 + 10

= 28 bytes

Product = Product Name

= 10 bytes

# of Requesters = 6,667 (from previous subsection calculation)

# of Change Requests = 10,000 (from previous subsection calculation)

# of Change Items = 5000 (from previous subsection calculation)

# of Product Releases = 100 (from previous subsection)

# of Products = 70 (from previous subsection)

Total Data Size = [(Requester Data Size \* # of Requesters) + (Change Request Data\*

# of Change Requests) + (Change Item Data \* # of Change Items)

+ (Product Release Data \* # of Product Releases) + (Product Data

\* # of Products] / bytes per kilobyte / kilobytes per megabyte

= [(85 \* 6,667) + (18 \* 10,000) + (166 \* 5000) + (28 \* 100) + (10 \*

70] / 1024 / 1024

= [566,695 + 180,000 + 830,000 + 2,800 + 700] / 1024 / 1024

= [1,580,195] / 1024 / 1024

= 1.5 MB approximately

With the assumption that the number of lines of code will be around 2000, it would be safe to assume that the program will use up under 200KB in disk space.

This would mean that the program would take up approximately 1.7 MB.

# Section 8: Prototype and Future Releases

## 8.1: Prototype Considerations

Minimally a first prototype of the program will need to provide the ability to create change requests and change-items. It should be able to create a report of all change-items, all ‘new’ change-items, and all change-items that are not ‘cancelled’ or ‘done’.

The prototype does not need to be able to collect requester information or create change requester objects. The prototype would benefit from implementing the process to create a new user, to emulate use cases and task flow. It should be possible to update a change-item’s status, but other update processes are not needed in the prototype. The prototype does not need to validate the integrity of any user provided data.

With this rudimentary prototype task flow and UIs can be tested to see what works. The prototype can help find a UI design that the customer approves of and finds suitable.

## 8.2: Future Release Considerations

In a future release change-items should be given some additional desirable attributes such as the severity of the bug, the predicted and real cost in time and money to implement a change, as well as the type of change. These changes help document a product’s development. They also help to inform budgets and cost estimates.

Reporting should be expanded upon. The system should have operations to list all change-items for a product, all change-items for all products, and all change-items that are not done or cancelled for all products. This will require the small overhead of leaving space for product name in a change-item’s printout, as reports may now contain change-items from more than one product. All should be easy to implement as expansions on existing report functions.

Change-items should have tags and be searchable by their tags. When a clerk receives a change-request, they must view all change-items corresponding to the product the request is for. They need to see if a new change-item needs to be created to accommodate the request, or if the request has already been made and has a corresponding change-item. For every change-request made the clerk may have to read the descriptions of hundreds of existing change-items.

As an example, if a bug is reported the clerk will need to sift through all change-items about bugs and feature requests. In this example, any change-item about a feature request is irrelevant to the change-request they are handling, their inclusion wastes time and screen space. The user should be able to filter relevant change-items. Increasing the size of a change-item so that the clerk can more quickly handle change-requests seems like a worthwhile tradeoff.

# Section 9: Acceptance Criterion

## 9.1: Hard Requirements

The program will meet the throughput and response time specifications defined under the “Performance and Capacity,” header.

The program will satisfy all use cases defined by the customer.

## 9.2: Goals

The program should deliver a user-friendly interface. The program’s UI will be prototyped and approved by the customer. Frequently used processes should be accessible quickly with minimal inputs. Menu trees should be intuitive. Organisation of printouts, and use of screen real estate should be considered.

The program hopes to streamline the customer’s development workflow and increase productivity by organising reporting bugs and feature requests to programmers and management.

## 9.3: Acceptance Test Requirements

The program will be tested on a workstation matching the specifications of the workstation described in the development environment. Testing will be performed by the development team.

All use cases described by the user will be tested. All possible state changes of the program will be tested. Functions will be tested as developed to show expected I/O behaviour and exception handling.

# Appendix

Attribute

* some piece of information; a measurement, quality, or component

Object

* Something described by its collection of attributes

Class

* grouping of an object and operations that can be performed on that object

I/O

* equivalent to: input / output

State

* Attribute used to determine how to treat an instance of an object

Char, Int

* Types of data in C, and C++, the languages used to develop the program

.

EOF Notes:

Formats of data members

Product name: 1-10 chars (alphanumeric)

release ID: 1-8 chars

product ID: 1-10 chars

email: 1-24 chars

Dept: 1-12 chars

Description: 1-128 chars

Priority: 1-5

format of the phone number: us / canadian +1 ### ### ####

Change ID : unsigned numeric 0-999999

Requester Name : 1-30 chars

Questions:

1. Who needs to be informed after a successful change has been implemented (in both scenarios - bug and new feature) ?
2. Does everyone using the product need to be informed or just the person who asked for the bug/change to be fixed
3. What would be in the description of the change
4. What is the format of the phone number

: us / canadian +1 ### ### ####

1. What if the phone number is foreign and is not in the same format
2. What if the person wants to stay anonymous
3. What to do when two issues have the same priority (bug first or issue first)
4. Should priority simply be a numeric value? 1-5
5. Can one change request be for multiple products?
6. Can one change request be for multiple change items?
7. Format of product ID, and release ID?
8. Can requester info be changed