**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**1. Introduction**

**1.1. Purpose of Document**

The purpose of this document is to provide a specification of the proposed design for a Bug Tracking System. It will document the basic structure, requirements (both functional and non-functional), design constraints and other factors needed to provide a complete description of the proposed system design. It is also intended to detail the motivation and reasoning for developing such a system – the business case.

The intended audience of this document includes the software development team, client(s) for whom the system is being developed and potential users of the completed system.

**1.2 Scope**

The software system to be implemented is a general-purpose Bug Tracking System (referred to as a “BTS” for the remainder of this document) for software systems. By “general-purpose” it is meant that the BTS will not be designed to cater for application to any specific software system, rather it will be designed specifically to be applicable across a wide-range of contexts where software bugs are encountered.

The aim of the BTS is to provide the organisation using it with a uniform process (and associated interface) for the reporting of bugs within some software system that organisation is responsible for. The BTS will also facilitate the management of the bug throughout its lifecycle, including the patching of the bug and verification of any proposed solutions. The intent is to streamline software bug handling to maximise efficiency and reduce costs.

The BTS is to serve four distinct classes of user, each class representing their role within the system:

* **Reporters:** Users whose purpose within the system is submitting reports describing any bugs discovered. This group will consist primarily of testers and end-users.
* **Developers:** These users are assigned bugs and are responsible for fixing them.
* **Triagers:** Triagers manage administrative actions related to each bug report, including checking for duplicate reports, invalid bugs and assigning bugs to a suitable Developer.
* **Reviewers:** Once a Developer has patched a bug**,** Reviewer users check if the patch adequately addresses the issues described within the initial report – i.e. to confirm whether a bug has been fixed.

Each of these user classes (to be referred to collectively as “user types” for the remainder of this document) will have their own set of access permissions to certain resources and subsystems within the BTS that correspond to their role. In addition to this a system administrator will have access/control of the entire system, with their purpose being to maintain correct operation of the BTS and its subsystems.

The BTS will be developed in a modular fashion, split into four independent subsystems:

* **Profile Subsystem:** Manages user profiles and associated information, including personal details and user types.
* **Bug Subsystem:** Manages bugs, tracking their status through their lifecycle and facilitating necessary communication between users along the process.
* **Search Subsystem:** Allows users to find bugs within the system using a keyword search.
* **Reporting Subsystem:** Manages the generation of statistical reports detailing activity within the BTS, as well as the tracking and collation of the statistics required. These reports are not to be confused for the bug reports managed within the Bug Subsystem.

This document aims to address requirements for all user types as they interact with the BTS and its subsystems.

**1.3 Definitions, Acronyms and Abbreviations**

* GUI (Graphical User Interface): the method of facilitating interaction between user and system through the visual representation of concepts and objects on the computer’s screen, rather than simply through text or otherwise.
* BTS (Bug Tracking System): the system to be developed, as described throughout this document.
* Java is a popular programming language and development platform, used widely for its designed ease in deploying Java applications across many platforms and environments.
* PC (Personal Computer) is a catchall term for computer devices intended for individual use, rather than larger-scale commercial or industrial applications.
* “Bug“ is computer jargon for any defect in design or implementation of software that results in unintended and/or undesirable results.

**1.4 Overview**

The remainder of this document is split into two sections as follows:

* The **Overall Description** will contain a general-level description of the system factors that affect requirements and the product design. This information should serve as background understanding for the specific requirements.
* The **Specific Requirements** section details all requirements necessary for the development team to satisfy for the final deliverable to meet client needs.

**2. Overall Description**

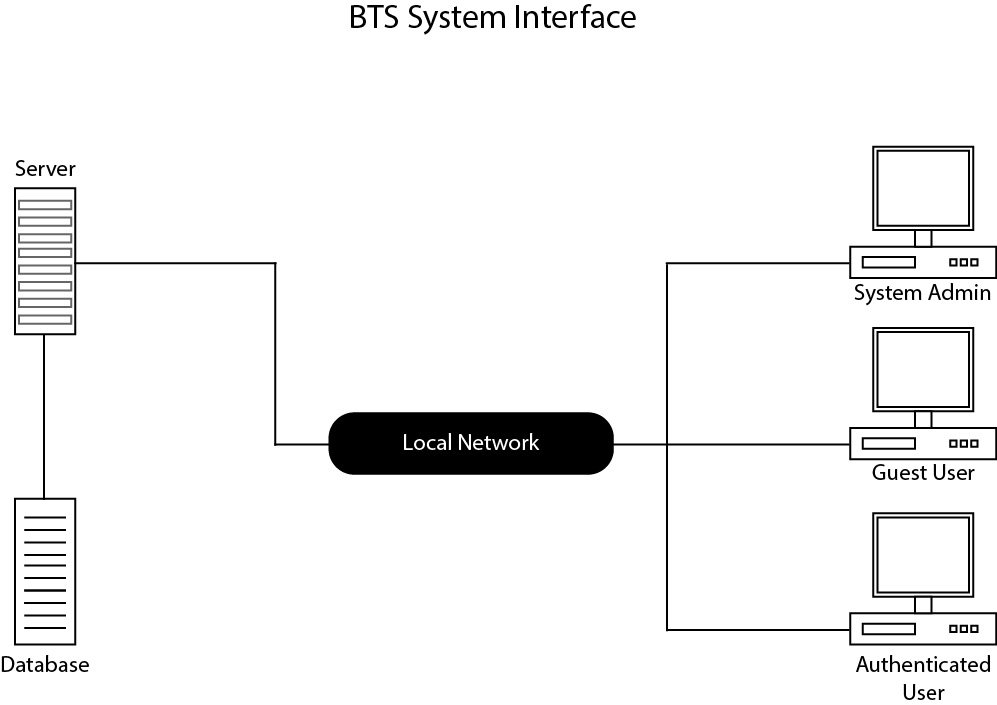
**2.1 Product Perspective**

In the absence of a dedicated system for bug reporting and management, software bugs are often addressed in an informal fashion with no sense of structure or process. For example, an organisation may rely on emails from users to be notified of bugs, with no way to assess the quality of their claims. Furthermore with no structured role system in place or unified interface for people to use, it can be very difficult to prioritise, track and/or finalise all efforts required to address a particular bug. Some organisations may even spend resources on organising a team to handle bugs that needs to coordinate in person due to the absence of a computer-based solution. More often than not this results in wasted time and finances for the organisation, as well as poorer outcomes in regards to the bugs themselves.

As such there is a need for a tool that streamlines the process of finding and addressing bugs. By providing a structured system with a unified interface to maximise efficiency and effectiveness, the BTS aims to help avoid these potential pitfalls simply through its own design.

2.1.1 *System Interfaces*

The BTS will be implemented through a Java application that can be used by an organisation internally through their own local network.



Anyone using a PC with local network access and Java compatibility an access the system – reports can be viewed (upon request) and searched by non-authenticated users, denoted as *Guest User* above*.* A *Guest User* with an account in the *Profile Subsystem* can log in to access their user type specific system features, considered here as an *Authenticated User.* It is planned to allow access permissions to be configurable by a *System Admin* if this behaviour is not desired at a point in the future.

The *Server* serves as a connection between users and the *Database*, which contains all report, profile and bug data. The *Server* handles user requests, retrieves necessary data from the *Database* and returns results that the application can interpret and display on screen.

2.1.2 *User Interfaces*

All user interfaces will be implemented in the form of a GUI accessed through an executable Java file (stored locally). Different user types will have unique interface features corresponding to their access permissions, but each of these will be implemented as a variation of the one core interface design.   
  
All user access to the *Database* is indirect, performed only by sending requests to the *Server* through the GUI.

2.1.3 *Hardware Interfaces*

All components are to be executed on a Java compatible PC.

2.1.4 *Operations*

It is intended that the system be operable without requiring any specific technical knowledge beyond basic computer operation.

Server and database installation/maintenance will not require any skills that a network administrator would not already be expected to understand how to do.

**2.2 Product Functions**

The primary function of the BTS is to facilitate the organised reporting, management and eventual patching of bugs found within a particular software system. These functions are encapsulated in the user types built into the system, and as such we have performed a use case analysis and breakdown to describe these functions in detail. This information can be found in section **WHERE EVER WE PUT NATHAN’S USE CASE STUFF**

**2.3 User Characteristics**

The users of the BTS are system admins, reviewers, developers, triagers and reviewers. For clarity the software system clients will be applying the BTS to will be referred to as the “target system” in this section.

* System admins have detailed knowledge of the BTS and its subsystems at a technical level, as they are responsible for system configuration and management. They do not necessarily have detailed knowledge of the target system.
* Reviewers do not require any particular level of knowledge or expertise other than that needed for the basic operation of the application. They are either users of the target system who wish to contribute information regarding bugs they have identified, or testers employed to find bugs in the target system. It is assumed they have familiarity with the target system, but the reputation system will allow the quality of a user’s information to be assessed easily.
* Developers are skilled programmers - knowledgeable of the target system and the underlying technologies it is built upon, such that they are prepared to conceptualise and implement patches to fix any bugs identified.
* Triagers have strong general-level knowledge of the target system, such that they can determine which reports are valid and identify which Developer(s) are best suited to be assigned a particular bug. They do not necessarily need to have any deep technical understanding of the target system however.
* Reviewers need to have strong understanding of the target system’s functionality so they can reliably identify the quality of a patch and whether the problem has been addressed.

**2.4 Constraints**

The BTS will operate strictly under the following constraints:

* Access control. The system will allow each user access to only the parts of the BTS that relate to their role within the system.
* Authentication security. The system will enforce this access control through strict authentication security – users must sign in to access their role-specific features.
* The system will be developed to run with an up-to-date Java installation at the time of release.

**2.5 Assumptions and Dependencies**

* All potential users of the BTS will maintain an up-to-date Java installation.
* All users will have basic computer operation skills, including mouse and keyboard operation.
* All users of the BTS will have reading comprehension of the English language and familiarity with domain-specific terminology.
* The client will have enough database storage to support the scale of implementation they wish to achieve.

**3. Specific Requirements**

Each requirement of the BTS has been ranked in terms of its importance to the functionality of the system, from most to least important according to the following naming convention:

* **Critical:** Critical requirements are aspects fundamental to the functionality of the BTS, necessary elements that must be implemented first.
* **Essential:** Essential requirements address important functionalities of the BTS that must be implemented once all Critical requirements have been satisfied.
* **Desirable:** Desirable requirements relate to functionalities that are necessary parts of the desired system design that are not critical to the core functionality of the BTS. These requirements should be addressed once all Critical and Essential requirements are successfully implemented.
* **Optional:** Optional requirements are not necessary parts of the system functionality, but rather represent possible improvements that enhance the functionality of the BTS. These requirements are to be considered only once all other Critical, Essential and Desirable requirements have been fulfilled.

**3.1 Functional Requirements**

Functional requirements will be divided into sections corresponding to each of the subsystems as described in section 1.2, as well as *General System Requirements* for any requirements that apply across the entire system.

3.1.1 *General System Requirements*

|  |  |
| --- | --- |
| **Requirement #: 3.1.1.1** | **Importance: Critical** |
| **Description:** Thesystem shall provide *Unauthenticated Users* with a GUIallowing submission of authentication details, consisting of text input fields and a submission button. | |
| **Rationale:** *Unauthenticated Users*want to log in to their profile to access specific features restricted to their user type. | |
| **Source:** *Unauthenticated Users.* | |
| **Fit Criterion:** The input fields are visible, and input data is sent to the Profile Subsystem successfully upon clicking the submission button. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.1.2** | **Importance: Critical** |
| **Description:** The system shall provide all users with a GUI allowing the use of the Search Subsystem to find bug reports, consisting of a text input field in which keywords can be entered, a menu allowing selection of the attribute the search is to be based on and a submission button. | |
| **Rationale:** All users want to be able to search through bug reports in the system so they can view them. | |
| **Source:** All users. | |
| **Fit Criterion:** The input fields are visible, and input data is sent to the Search Subsystem successfully upon clicking the submission button. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.1.3** | **Importance: Desirable** |
| **Description:** Upon logging in, a list of ‘of interest’ items is generated and displayed on screen, consisting of new behaviour that the user needs to be aware of. This includes any new bug report assignments (see 3.1.3.5), activity on reports the user has marked their interest in (see 3.1.3.8) and activity on any reports that user may have submitted. This screen can be accessed again at any later point through a GUI button. | |
| **Rationale:** Users want to be able to access the information most relevant to them as easily as possible, and want to be notified when activity occurs that relates to their role within the system. | |
| **Source:** All users. | |
| **Fit Criterion:** The list is displayed correctly upon logging in, and can be viewed again through the GUI successfully. | |
| **Dependencies:** Users must have some valid ‘of interest’ items associated with their account for the list to appear. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.1.4** | **Importance: Optional** |
| **Description:** The system shall provide all users with a list of the five most recently added bug reports for convenient access, displayed in the initial screen shown after logging in. | |
| **Rationale:** Users want to be aware of newly added bug reports. | |
| **Source:** All users. | |
| **Fit Criterion:** The five most recently added bug reports can be viewed on screen. | |
| **Dependencies:** There must be bug reports in the system to list. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.1.5** | **Importance: Optional** |
| **Description:** The system shall display a *Reporter* user’s Reputation score next to the user’s name for all instances where the user’s name is displayed on screen throughout the system. | |
| **Rationale:** Users want to be able to efficiently find out other users’ Reputation score to assist in judging the value of a contribution. | |
| **Source:** All users. | |
| **Fit Criterion:** The reputation score displays correctly in all instances. | |
| **Dependencies:** None. | |

3.1.2 *Profile Subsystem Requirements*

|  |  |
| --- | --- |
| **Requirement #: 3.1.2.1** | **Importance: Critical** |
| **Description:** The system will store a set of attributes describing each user in the system. These apply for all user types and include username, password, user type, post counts, email, bugs they are working on, etc. | |
| **Rationale:** Users want to have consistent information in the system describing them and tracking their behaviour. | |
| **Source:** All users. | |
| **Fit Criterion:** Information is stored in a persistent fashion and is associated with the correct account. | |
| **Dependencies:** The user must exist in the system before any attributes can be set. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.2.2** | **Importance: Critical** |
| **Description:** Unregistered users can complete a form to apply for a user account. This form will be accessed through a GUI button that will direct the user to the form (consisting of many text input fields corresponding to attributes as described in 3.1.2.1), which they can then fill in and submit. This information is forwarded to a *System Admin* who will decide if the application is legitimate and complete registration using the process provided through 3.1.2.4 if so. | |
| **Rationale:** People want to be able to be granted permission to required system features that can only be accessed while signed in. | |
| **Source:** Potential new users external to the system. | |
| **Fit Criterion:** The information as submitted is successfully forwarded to the *System Admin.* | |
| **Dependencies:** A *System Admin* user must already exist in the system. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.2.3** | **Importance: Essential** |
| **Description:** The system shall provide *System Admin* users with a GUI to allow the configuration of access permissions for each user type. This will consist of several fields indicating permissions for each user type, and a save button to commit any changes made. | |
| **Rationale:** *System Admin* wants to alter access control behaviour for user type(s) in the system. | |
| **Source:** All users. | |
| **Fit Criterion:** Parameters as they are represented on-screen initially are indicative of current system parameters, and pressing the save button successfully commits any changes to the system. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.2.4** | **Importance: Critical** |
| **Description:** The system shall provide *System Admin* users with a GUI to allow the creation of a new user profile. This will consist of input fields for user attributes (refer to 3.1.2.1), and a submission button to commit the data to a new account in the Profile Subsystem. | |
| **Rationale:** *System Admin* needs to be able to create new user accounts. | |
| **Source:** All users. | |
| **Fit Criterion:** The account is successfully stored in the system with the correct details as they were entered. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.2.5** | **Importance: Essential** |
| **Description:** The system shall provide *System Admin* users with a GUI to allow the editing of any existing user profile within the Profile Subsystem, including the deletion of the profile. | |
| **Rationale:** System Admin needs to change attributes of a particular user profile. | |
| **Source:** All users. | |
| **Fit Criterion:** Any changes made are successfully saved within the system. | |
| **Dependencies:** The to-be-edited profile must exist in the Profile Subsystem. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.2.6** | **Importance: Desirable** |
| **Description:** The system will track a Reputation attribute for *Reporter* users. Other (non-*Guest*) users can alter this attribute by interacting with a “+1 Reputation” button found in each *Bug Report*. This button can only be used once per user, per bug report. | |
| **Rationale:** Users want to be able to reward quality contributions and see at a glance the general quality of a particular user’s contributions. | |
| **Source:** All users. | |
| **Fit Criterion:** The Reputation score must be persistent across the system and correctly incremented by the “+1 Reputation” button. | |
| **Dependencies:** The user profile must still exist for the “+1 Reputation” button to operate. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.2.7** | **Importance: Essential** |
| **Description:** The system will provide a GUI allowing users to alter their profile’s attributes. | |
| **Rationale:** Users want to have up-to-date information associated with their account. | |
| **Source:** All users. | |
| **Fit Criterion:** Upon submitting changes the associated attributes are changed correctly within the database. | |
| **Dependencies:** None. | |

3.1.3 *Bug Subsystem Requirements*

|  |  |
| --- | --- |
| **Requirement #: 3.1.3.1** | **Importance: Essential** |
| **Description:** The system will track the status of bug report through the following set of attributes of binary nature, each with their own permissions for edit access (listed in parentheses):  Open / Closed (*Triager* only)  Unassigned / Assigned (*Triager* only)  Fixed / Not Fixed (*Reviewer* only) | |
| **Rationale:** Users need to be able to see the state of a bug in order to decide what action needs to be taken. | |
| **Source:** All users. | |
| **Fit Criterion:** The attributes are tracked correctly and can be changed only by those with appropriate permission. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.3.2** | **Importance: Critical** |
| **Description:** The system shall allow users to post text comments on a bug report, which are to be displayed below the bug report itself. The user name of the account posting the comment and the date/time the comment was posted will be displayed along with the submitted text. The system will facilitate the submission of comments through GUI features, consisting of a text input box and a submission button. | |
| **Rationale:** Users want to be able to discuss bug reports and communicate with other interested users. | |
| **Source:** All users. | |
| **Fit Criterion:** Submitted comments are saved and display correctly. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.3.3** | **Importance: Essential** |
| **Description:** The system will allow users to edit or delete any comments they have already made. Buttons will be visible by each comment that user has made, that will either open the comment GUI as described in 3.1.3.2 with the original comment content in the text input box, or a confirmation check if the delete button is clicked. | |
| **Rationale:** Users want to be able to correct errors, refine and/or remove comments after they have been posted. | |
| **Source:** All users. | |
| **Fit Criterion:** The edited comment appears in the place of the original comment correctly – or is removed from the comment list if deleted. | |
| **Dependencies:** Users must be logged into the account that posted the comment to access these features. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.3.4** | **Importance: Critical** |
| **Description:** The system shall allow *System Admin, Reviewer* and *Triager* users to delete bug reports posted by other users that are invalid, corrupted or otherwise not appropriate for storage on the system. This will be allowed through GUI features available only to these users. | |
| **Rationale:** Users want to avoid unnecessary reports that are of no use. | |
| **Source:** All users. | |
| **Fit Criterion:** Posts are successfully removed. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.3.5** | **Importance: Critical** |
| **Description:** The system shall allow *Triager* users to assign a bug to a particular developer. The bug report will then be added to the *Developer* user’s profile, and that user will be shown a notification upon signing in to notify them (refer to 3.1.3.3). | |
| **Rationale:** *Triager* users want to find appropriate *Developers* to who are best qualified to address a particular bug and assign them the task of working on a patch. | |
| **Source:** Triager | |
| **Fit Criterion:** Attributes are updated accordingly and notifications are displayed. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.3.6** | **Importance: Optional** |
| **Description:** The system will generate and send a notification email to *Developer* users when they have been assigned a bug to work on by a *Triager.* | |
| **Rationale:** *Developer* users want to be as aware of new jobs as soon as they come in. | |
| **Source:** Developer | |
| **Fit Criterion:** An email with the correct details is sent to the address recorded in the *Developer* user’s profile information. | |
| **Dependencies:** The *Developer* user needs to have a valid email address recorded. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.3.7** | **Importance: Critical** |
| **Description:** When requested, the system can generate a visual representation of all bug report information on screen in an organised, consistent format across all reports. This includes all submitted report data, username of submitter, time of submission, report status, GUI interaction elements and comments. | |
| **Rationale:** Users want to see reports and perform relevant actions. | |
| **Source:** All users. | |
| **Fit Criterion:** Reports are displayed correctly on screen. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.3.8** | **Importance: Desirable** |
| **Description:** Users can register their interest in a particular bug report. This is achieved through clicking on an “interested” button displayed with each bug report. Doing so will cause the user to be notified when new comments and patches are posted, as well as any updates to the report’s status. Refer to 3.1.3.3 for notification method details. | |
| **Rationale:** Users want to be able to easily track bugs they are interested in throughout their lifespan. | |
| **Source:** All users. | |
| **Fit Criterion:** Clicking the button will result in notifications displaying correctly. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.3.9** | **Importance: Critical** |
| **Description:** *Developer* users can upload patch files containing implementation of potential fixes to a bug, which can be downloaded by *Reviewer* users for testing and verification purposes. | |
| **Rationale:** *Reviewer* users want to be able to confirm whether a patch is satisfactory, and *Developers* need to know if their proposed solution is working. | |
| **Source:** All users. | |
| **Fit Criterion:** Uploaded files are correctly stored in the database and can be accessed successfully by those with the correct permissions. | |
| **Dependencies:** The server and database must not experience any failures during the upload/download process. | |

3.1.4 *Search Subsystem Requirements*

|  |  |
| --- | --- |
| **Requirement #: 3.1.4.1** | **Importance: Critical** |
| **Description:** The system shall provide a keyword matching search function, which will provide a list of all bug reports in the system that contain the input keyword(s) in the report’s metadata. The data attribute to be searched through can be chosen by the user (for example title or username). This is implemented in GUI as described in 3.1.1.2. | |
| **Rationale:** Users want to be able to find particular bug reports. | |
| **Source:** All users. | |
| **Fit Criterion:** The keyword search finds all matching results without any incorrect matches and returns the list of reports. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.4.2** | **Importance: Desirable** |
| **Description:** The system will provide an additional search field that will allow more refined search results by setting a condition for a particular attribute that results must match. For example one may only want to see reports that are set as ‘Not Fixed’ (refer to 3.1.3.1). This will work in addition to the keyword search described in 3.1.4.1 | |
| **Rationale:** Users want to find specific subsets of the report database. | |
| **Source:** All users. | |
| **Fit Criterion:** The added condition will correctly augment search results without false positives or missed correct results. | |
| **Dependencies:** None. | |

3.1.5 *Reporting Subsystem Requirements*

|  |  |
| --- | --- |
| **Requirement #: 3.1.5.1** | **Importance: Critical** |
| **Description:** The Reporting Subsystem must have access to all data attributes within the system that are required for generating reports and graphs. | |
| **Rationale:** The reports and graphs need correct data to have any meaning. | |
| **Source:** System | |
| **Fit Criterion:** All report generating processes must run without error, to produce accurate reports and graphs. | |
| **Dependencies:** The required data must exist in the system if the processes are to run correctly. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.5.2** | **Importance: Critical** |
| **Description:** The system must provide a GUI to select what report/graph is to be generated, along with any desired parameters/conditions the generating method may need in order to run correctly. These methods shall not be permitted to run unless the user has set all information required. | |
| **Rationale:** Users want to be able to choose what outputs they want from the subsystem. | |
| **Source:** All users. | |
| **Fit Criterion:** Generated output is consistent with what is shown onscreen in the query form before submission. | |
| **Dependencies:** None. | |

|  |  |
| --- | --- |
| **Requirement #: 3.1.5.3** | **Importance: Optional** |
| **Description:** The system can export graph datasets in .csv form if applicable. | |
| **Rationale:** Users may want to use data in external applications rather than just viewing information. | |
| **Source:** All users. | |
| **Fit Criterion:** Correct data is exported in correct format and can be saved locally by the user. | |
| **Dependencies:** None. | |

**3.2 Non-Functional Requirements**

|  |  |  |
| --- | --- | --- |
| **Requirement #: 3.2.1** | **Req. Type: Usability** | **Importance: Essential** |
| **Description:** The system shall be operable by all non-admin user types after a total of 1 hour of training. | | |
| **Rationale:** The system is not particularly complex, and to dedicate further resources to training is wasteful. | | |
| **Source:** All non-admin users. | | |
| **Fit Criterion:** Once training is complete, on average users should not report more than two events where there were uncertainties in how to perform an action within the system. | | |
| **Dependencies:** None. | | |

|  |  |  |
| --- | --- | --- |
| **Requirement #: 3.2.2** | **Req. Type: Performance** | **Importance: Essential** |
| **Description:** The system shall respond to any user interaction within 10 seconds. | | |
| **Rationale:** Users want systems to respond quickly so they can work efficiently. | | |
| **Source:** All users. | | |
| **Fit Criterion:** No interaction should have a response time greater than 20 seconds. | | |
| **Dependencies:** None. | | |

|  |  |  |
| --- | --- | --- |
| **Requirement #: 3.2.3** | **Req. Type: Usability** | **Importance: Essential** |
| **Description:** The system shall be packaged with supporting documentation to assist users with any issues they may have operating the system. | | |
| **Rationale:** Some users may need guidance to learn the system, or will forget important information over time. | | |
| **Source:** All users. | | |
| **Fit Criterion:** Help files are available to all users with access to the system. | | |
| **Dependencies:** None. | | |

|  |  |  |
| --- | --- | --- |
| **Requirement #: 3.2.4** | **Req. Type: Performance** | **Importance: Critical** |
| **Description:** Outside of planned maintenance or unexpected downtime, the system should be available 24 hours a day, 7 days a week. | | |
| **Rationale:** Users want to be able to access the system whenever is most convenient to them, and to be able to access the system for as long as they require. | | |
| **Source:** All users. | | |
| **Fit Criterion:** Any user with access to the application will be able to access the full expected functionality at any time. | | |
| **Dependencies:** None. | | |

|  |  |  |
| --- | --- | --- |
| **Requirement #: 3.2.5** | **Req. Type: Reliability** | **Importance: Essential** |
| **Description:** The system should run reliably within an acceptable failure rate. | | |
| **Rationale:** Users expect the system to operate correctly at all times, within realistic bounds. | | |
| **Source:** All users. | | |
| **Fit Criterion:** The mean time between failures should on average never fall below two weeks, and take no longer than 24 hours to recover from failure. | | |
| **Dependencies:** None. | | |

|  |  |  |
| --- | --- | --- |
| **Requirement #: 3.2.6** | **Req. Type: Performance** | **Importance: Desirable** |
| **Description:** Search queries should take no longer than 15 seconds to complete. | | |
| **Rationale:** Users want results as quickly as possible. | | |
| **Source:** All users. | | |
| **Fit Criterion:** Time elapsed between a search query being sent and results displaying on screen should not exceed 25 seconds, and on average should not exceed 15 seconds. | | |
| **Dependencies:** None. | | |