# ECSC503.Y – Software Development Group Project - Performance Monitor - Group 2 Report

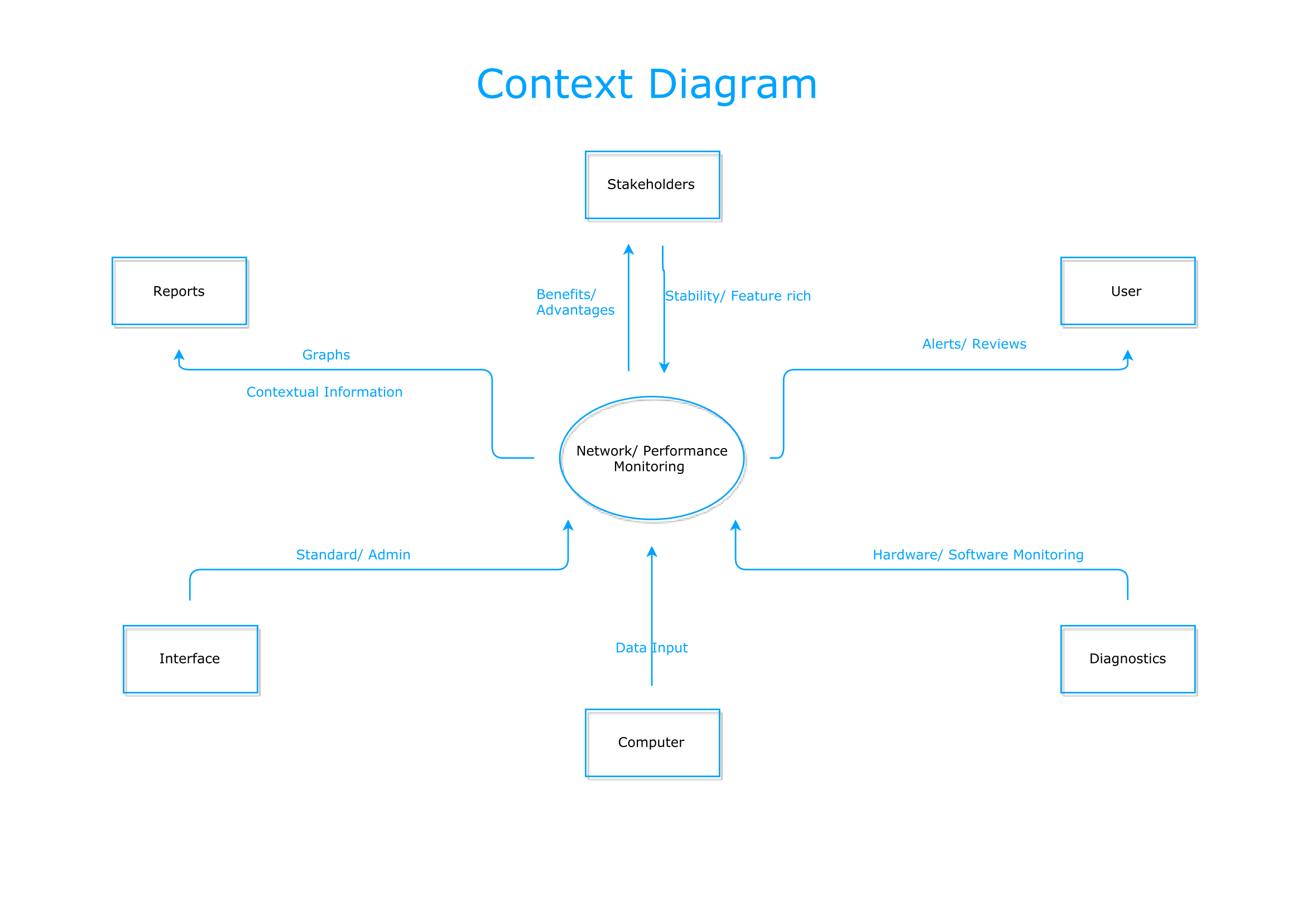
## C:\Users\Reaper\Dropbox\westminster\YearTwo\Semester1\ecsc503.y-SoftwareDevelopmentGroupProject\uow-gp-1\Logo\GroupLogo.pngContents page

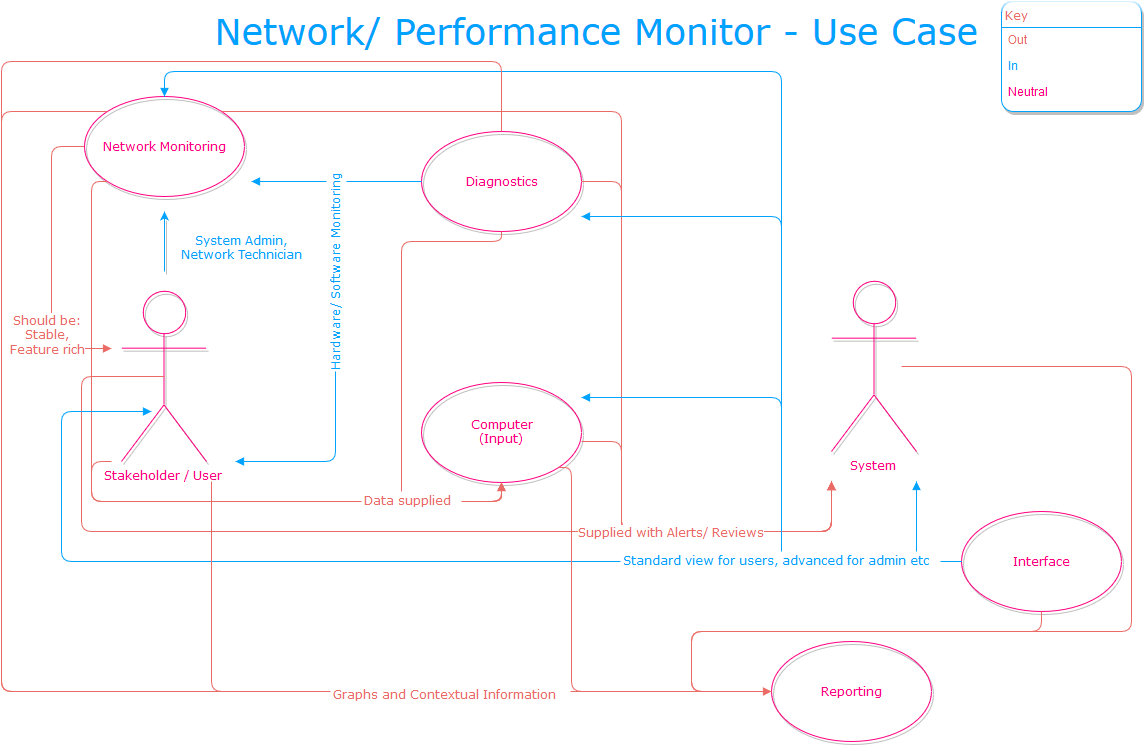
Problem scope

This report is to detail how our group came to include the aspects in our Network/ Performance Monitor.

**Aims**

The aim of our project is to provide a stable and efficient interface for standard/administrative users to monitor their system performance. Within the report, we will highlight how the system will be developed and what we have taken into consideration. The diagrams we create will specify the interactions between multiple components of the system and the users.





Use Case Scenarios

Use case scenarios are beneficial for a team to evaluate what could potentially happen in a certain environment and what may happen when a problematic variable is introduced. The idea behind scenarios is to make the end product adaptable by using forethought in order to apply a solution for a situation that may occur.

Initially we considered who may want to use the program, this could be one of the following:

* Advanced technical user – System administrator
* Technically competent user – standard user
* Enterprises

We then questioned what the users may want to do with the program, as it is a performance monitor this can entail many different things that fall under the same scope. After some reflection, the group decided that these were the most important reasons as the why the program may be used and what it may be used for.

* Network Monitoring – monitor the throughput of the network and any potential traffic problems
* System Health – general performance overview, CPU usage, System uptime etc
* Troubleshooting – check link states, possible system issues such as RAM usage being too high
* Reporting/ Reviews/ Alerts – User notifications (implemented to admin class users) utilised for troubleshooting
* Prevention of future problems – using performance monitor to precondition the system environment

Situation Flow

Making a basic situation flow, we used the scenario that would entail a normal usage day where there would be no issues on either the programs side or the usage side.

Basic Flow

**Use Case**: Performance Monitoring

**Description**: Advanced/Standard User wants to monitor the performance of the system

**Actors**: User, System

**Preconditions**: Appropriate account is used; program has access to hardware monitoring sensors

User wants to see how system is performing.

System detects user input, authenticates user.

Depending on user type, appropriate interface is loaded for User (basic/advanced).

Standard user gets basic overview of system performance, Advanced user gets reporting, reviews and separate tab for network performance.

User logs off from program.

Alternative Flow

In our alternative flow, we identified the issue of authenticating the user. If the user failed to authenticate themselves there needed to be a resolution and also a way to protect the system from misuse.

Step 2 – Failure to authenticate correctly causes the program to terminate after three bad attempts.

Exception

In an attempt for the user to regain access to the program, there were some solutions that we thought could be implemented.

User could restart the program after it terminates and use a ‘*reset password*’ button, the problem this poses is a reset is an easy system bypass. To remedy this the group proposed an idea:

If password is reset, user will enter answers to two/three security questions, and possibly a randomly generated alphanumeric string that must be entered.

Post Conditions

The program should be able to identify critical changes from user input; if the user has changed their password but it was lost on the restart of the program, it is flawed. This issue reiterates when an advanced user changes some settings but they are not saved.

On user input, password is saved and then possibly backed up?

Admin user settings are saved whenever settings are modified.

Negative Situations

Careful consideration of the use case, allows us to identify possible negative situations that could occur at certain crucial moments of the program’s development/ operation.

Exceptions

Events occurring at normal operation of the program that would interrupt progress:

Application does not start or crashes during operation

Advanced/ Standard user views are not given to appropriate user i.e. Advanced user gets basic view

Program fails to retrieve system statistics

A password that is reset has not been saved, meaning that when the user enters the new password it is incorrect

Settings that are changed have not been saved

Intentional Threats

Events that are caused by the action of hostile stakeholders or issues within the team:

Stakeholders reactively change desired features of the program

Changes are made to the program without members consulting each other

Unwanted Scenarios

Events that are undesirable:

User correctly authenticates, but the program terminates on login or crashes

When the user fails to authenticate correctly, but the program does not terminate

The team does not deliver the program on time

The team fails to meet targets and/or delivers the program with missing essential features

Sequence Diagram

Functional Requirements

Functional requirements are ‘System shall do <task> here”. As opposed to non-functional, “where system will be <behaviour>” in a certain manner.

**Standard**

Users will have an overview of system performance.

Users can report/ view own faults.

User is able to change forgotten password.

**Admin**

Administrative users will have views of individual performing components.

Administrative users are able to make/ save changes.

Administrative users can modify/ add/ remove, users respectively.

Administrative users have access to reports and alerts from system.

<https://en.wikipedia.org/wiki/Functional_requirement>

Class Diagram