
UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

# COS301 Mini Project Testing Infrastructure

Matthew Gouws *u11008602*
Andrew Parkes *u12189139*
Axel Ind *u12063178*
Patience Mtsweni *u11116774*
Khathutshelo Shaun Matidza *u11072157*
Matthew Nel *u10126229*
Ephiphania Munava *u10624610*

Here's a link to Github.
`https://github.com/MatthewGouws/COS301_Testing_infrastructure`

Version 0.1-alpha
April 24, 2015

# 1 History

| Date | Version | Description |
|---|---|---|
| 21-04-2015 | Version 0.1 | Document Template Created |
| 22-04-2015 | Version 0.1.1 | Added Authorization for B |
| 22-04-2015 | Version 0.1.2 | Added Authorization for A |
| 22-04-2015 | Version 0.1.3 | Added Notification Table |
| 22-04-2015 | Version 0.1.4 | Added introduction |
| 22-04-2015 | Version 0.1.5 | Added uses cases for Buzz B |
| 22-04-2015 | Version 0.1.6 | Added uses cases for Buzz 1 |
| 22-04-2015 | Version 0.1.7 | Fixed Authorization formatting |
| 23-04-2015 | Version 0.1.8 | Added Space use cases |
| 23-04-2015 | Version 0.1.9 | Added CSDS use cases |
| 24-04-2015 | Version 0.2.0 | Added availability and security non-functional requirements |
| 24-04-2015 | Version 0.2.1 | Added Manageability and Reliability |
| 24-04-2015 | Version 0.2.2 | Added Monitor ability and Auditability and integrability |
| 24-04-2015 | Version 0.3 alpha | Proposed Alpha testing documentation |
| 23-04-2015 | Version 0.3.1 | Added CSDS Scalability, Testability and Maintainability |

# Contents

**1 History** **1**

**2 Introduction** **3**

**3 Purpose** **3**

**4 Project Scope** **3**

**5 Functional** **4**
5.1 use cases and results . . . . . 4
5.1.1 Authorization . . . . . 4
5.1.2 Notification . . . . . 5
5.1.3 Spaces . . . . . 6
5.1.4 The Buzz-Data-Sources Module . . . . . 7

**6 Non-functional** **7**
6.1 Performance . . . . . 7
6.2 Scalability . . . . . 8
6.3 Maintainability . . . . . 8
6.4 Reliability . . . . . 9
6.5 Usability . . . . . 9
6.6 Availability . . . . . 10
6.7 Manageability . . . . . 10
6.8 Security . . . . . 11
6.9 Monitor ability and Auditability . . . . . 11
6.10 Integrability . . . . . 11
6.11 Testability . . . . . 11

**7 References** **11**

# 2 Introduction

This document contains: Part 1 the functional testing phase for each mid-level parts Buzz A and Buzz B. Each section will show the success or the failure of each part. This contains all violations of the contract requirements. Pre- and post- conditions should be tested for all the violations and the data structure requirements. For all the testing, an analysis report of the percentage cases will be given that will depict the amount of work done and the successfulness of the sections in the implementation.

Part 2 the non-functional testing phase. This part contains the performance, scalability, maintainability, reliability, usability of the application and problems associated with the system.

# 3 Purpose

The purpose of this task was to test functionality provided by mid-level integration for infrastructure, which consisted of Notification, Authorization, Spaces and CSDS.

# 4 Project Scope

The scope of the integration for infrastructure was to combine all functional teams' code in a manner which could be used by top level integration. From what has been discovered and explained further in this document it shows that both teams A and B have failed to do so. Team A was very difficult to try and decipher. With missing dependencies, while Team B only had mock functionality.

# 5 Functional

## 5.1 use cases and results

### 5.1.1 Authorization

| Use Case(s) | Buzz A | Buzz B |
|---|---|---|
| addAuthorizationRest - Adds an authorization restriction for a user role in a particular buzz space. | Could not NPM start as stated in README due to missing dependencies, as a result could not run | Only Mock functionality, does not run |
| updateAuthorizationRest - Facilitates editing of authorization restrictions. | Could not NPM start as stated in README due to missing dependencies, as a result could not run | Only Mock functionality, does not run |
| removeAuthorizationRest - Removes an authorization restriction for a user role from a buzz space. | Could not NPM start as stated in README due to missing dependencies, as a result could not run | Only Mock functionality, does not run |
| getAuthorizationRest - Retrieves the authorization restriction to enable users to select a restriction to update. | Could not NPM start as stated in README due to missing dependencies, as a result could not run | Only Mock functionality, does not run |
| isAuthorized - Queries the services a user may access in order to customize the UI for the user. | Could not NPM start as stated in README due to missing dependencies, as a result could not run | Only Mock functionality, does not run |

### 5.1.2 Notification

| Use Case(s) | Buzz A | Buzz B |
|---|---|---|
| Daily Email - Sends Daily Email. | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Delete Notification - Checks if the user should receive a notification | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Edit Notification Settings - Edits the notifications | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Web Notification - returns a list of notifications for the specified user | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Register For Notification - Allows a user to register for notifications on a thread, to specified users | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Standard Notification - When a user adds a new thread it sends notifications to a list of registered users | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |

### 5.1.3 Spaces

| Use Case(s) | Buzz A | Buzz B |
|---|---|---|
| Create Buzz Space - Creates and adds the buzz space to the activated list of buzz spaces. | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Close Buzz Space - Receives buzz to close and then removes the buzz space from the list of activated buzz spaces. | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Assign Administrator - Gets the user to be assigned to be administrator then checks if it is administrator and adds the user to the list of administrators. | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Remove Administrator - Receives the user to be removed then removes the user from the list of admin. | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Is Administrator - Receives the user to be checked then searches the admin list for the user. | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Get User Profile - Searches for the user that is queried and returns the user searched for. | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |
| Register On Buzz Space - Registers the user on buzz spaces and stores the user in the database. | Could not npm install, Missing dependencies, thus would not run | Only Mock functionality, does not run |

### 5.1.4 The Buzz-Data-Sources Module

| Use Case(s) | Buzz A | Buzz B |
|---|---|---|
| Login and administrative user - The system should authenticate against the CS DataSources within which authentication credentials are currently stored. | Gets connection to DataSource, Authenticate user against the DataSource and userID is returned. | Gets connection to DataSource, Authenticate user against the DataSource and userID is returned. |
| getUsersRolesForModule - This function is to query the user roles for a particular user. | The function successfully meets the requirements it has local copy to the userID, ModuleID and roles array. It returns user roles for module request. | The function fails to meets the requirements it has local copy to the userID, ModuleID and roles array but it *does not* return user roles for module request. |
| getUsersWithRole - It retrieves all the users that have a particular role like a teachingAssistant role for a particular module. | The function fails to meets the requirements has local copy to the userID, ModuleID and roles array but it *does not* return user roles for module request. | The function fails to meets the requirements has local copy to the userID, ModuleID and roles array. It *does not* returns user roles for module request |

# 6 Non-functional

## 6.1 Performance

The use of Nodejs allows the B Infrastructure system to run in a multi-threaded manner with very high performance. Furthermore, if the final system were to make use of MongoDB (as specified), an appropriate performance increase would be achieved, indicating minimal-bottlenecking in the system overall.

The Integration Team also provided code which was highly streamlined, with no recursive methods or unnecessary process iterations. Suitable response times can be expected from the system for any reasonable number of concurrent users.

As stated for the B system, A would run similarly in performance due to the use of NodeJS and MongoDB, With the code very difficult to track down and run, the true performance could not be tested for the system, but given a decent entry level server would be expected to handle multiple client connections at once.

## 6.2 Scalability

By its nature, Nodejs and its subcomponents provides a high level of intrinsic scalability, due in no small part, to the streamlined use of client-server architecture.

The work of the mid-level integration teams for both A and B included the appropriate use of core Nodejs functionality to create Service Objects, facilitating easy implementation of a single or several thousand Spaces. Multithreading is implemented and no limitting bounds are set on the potential permissible Space IDs.

At present every function at the midlevel operates with O(logn) complexity (as the number of tasks increase, the odds of a given task already having been performed and, thus needed no additional complex work, increases significantly)

The system implemented by CSDS buzz A supports the ability to host buzz spaces for all Computer Science Modules at a University of the size of the University of Pretoria furthermore it is possible to, in the future, scale the architecture. Whereas CSDS B fails to provide scalability.

## 6.3 Maintainability

Although the code appears simplistic and highly maintainable, minimal commenting was provided by the Integration Team. The current code is human-readable, but in the pursuit of long-term maintainability by multiple parties, the lack of authors notes could be seen as a potential stumbling block to future programmers.

That notwithstanding, the code provided by the Integration Team clearly makes strong distinctions between individual components and would, for example, allow modifications to the registerOnBuzzSpace and closeBuzzSpace processes independently and with suitable modularisation for easy further modification by a third party.

Due to the fact that the code is very difficult to access in the repository for A it is highly unlikely it can be called maintainable. However unit tests seem to have been completed and thus would be easy to add specific extra cases to the code while ensuring it does not break the known working code.

When looking at the CSDS implemented by both groups it is clear to see that it is easy to maintain the system in the future, besides the fact that there is insufficient comments, future developers will be able to easily understand the system because it is implemented simply, the technologies chosen for the system are reasonably expected to be available for a long time and developers will be able to easily and relatively quickly change aspect of the systems functionality as well as add new functionality.

## 6.4 Reliability

Due to the lack in all aspects of Buzz B the reliability is non-existent due to missing code, mock functions and not being able to run the packages.

The Integration Team of Buzz B have implemented a highly reliable way to integrate Spaces. An appropriate message is logged when an access attempt is made and several comprehensive exceptions are thrown by the system when registerOnBuzzSpace, closeBuzzSpace and other processes fail.

It may be advisable, however, to increase the number of exceptions that may be thrown to better inform the system as to the problem that has occurred. For example: rather than stating: "BuzzSpace 7A is closed or does not exist." It may be prudent to have 2 errors: BuzzSpace 7A is closed and BuzzSpace 7A does not exist.

## 6.5 Usability

Given, the admittedly sparse, template provided by the Integration B Team, it is clear that Buzz Bs modularised functionality would be highly useable at mid-level. All modules are given descriptive names, all variable names are human-readable and meaningful, and programming logic is clear throughout.

The only remark on improvement has already been mentioned in Maintainability and would involve the addition of more comments to better guide a first-time user.

Due to the fact that the code is barely traceable to where it is located, it is very difficult to use this package. Insight into the development would be

required to allow a user to be able to use the code correctly and how to use the code before production could begin.

## 6.6 Availability

One would expect the availability of the Buzz B System to be almost 100 percent. The code itself is minimal and, given appropriate decoupling from the implementation of Modules at a lower level, one could expect and changes that need to be made at the mid-level to be performed very quickly, while low-level changes to implementation could be accounted for suitably quickly.

Because Services are implemented as discrete objects, it would also significantly reduce downtime that may be caused by the failure of a single space, if that space could be recreated or reconnected to in a suitably dynamic and time-sensitive way.

Considering the fact that the system was going to be hosted online on successful completion, its availability would've been archived since it would be in an operable and committable state at any random time. However this is not the case with the current state of the system.

## 6.7 Manageability

Given the fairly abstract and well-decoupled nature of Buzz B Infrastructure, the Manageability of the component is fairly high and would facilitate easy management of (unlikely) logical errors in the Buzz B system.

Again, the one negative aspect of the emergent manageability of the system is the lack of commenting which makes the code unnecessarily hard to read and would limit the ability of the a programmer, either inspecting code manually or through a comment documentation system (e.g. Doxygen), to quickly grasp nuances in logic that may be affected by minor changes.

The manageability of part A is by no means possible. Without being able to get Buzz to run it is impossible to have any manageability in the system. For Buzz B it is highly manageable in its current state due to all the mock functions but since the code does not produce any significant functionality it is thus overlooked.

### 6.8 Security

Authorization is the module that was supposed to provide the degree of security in this system. Both authorization A and B failed to run due to missing dependencies and only mock functionality provided for the other. Considering this, resistance to, or protection from, harm failed.

### 6.9 Monitor ability and Auditability

In both systems Buzz A and B there does not seem to be any logs or error messages to help with monitoring of the system. Since there is not expected outcome for each we are unable to test to see how auditable the systems are and thus both systems are not monitor able nor are they auditable.

### 6.10 Integrability

Since there is none of any of the above parts the integrability of the system is not possible. With unreliable systems that are not manageable, without it being usable the system in both Buzz spaces A and B they are not integrable

### 6.11 Testability

Most of the services offered by CSDS A are testable through unit tests testing components in isolation using mock objects as well as integration tests where components are integrated within the actual environment which tests them. CSDS B implements a few of these but fails to for most of the requirements.

## 7 References


- http://searchdatacenter.techtarget.com/definition/scalability
- http://stackoverflow.com/questions/9420014/what-does-it-mean-scalability
- http://en.wikipedia.org/wiki/Reliability,_availability_and_serviceability_%28computing%29
- http://whatis.techtarget.com/definition/Reliability-Availability-and-Servicea