

COS301 Mini Project Testing

Notification

Johan Marx 12105202
Tim Kirker 11152402
Edwin Fullard 12048675
Matthew Russell 12047822
Jessica Lessev 13049136
Hanrich Potgieter 12287343
Roger Tavares 10167324
Thinus Naude 13019602
Paul Engelke 13093500
Trevor Austin 11310856

Git repository link:
[https://github.com/DianMarx/
notificationTesting](https://github.com/DianMarx/notificationTesting)

Version April 24, 2015

Contents

1	Introduction	2
2	What we tested	2
3	Functional Testing	3
3.1	Register for Notification Use Case	3
3.2	Deregister for Notification Use Case	4
3.3	Send message	5
4	Non-functional testing/assessment	7
4.1	Performance	7
4.2	Scalability	8
4.3	Plug-ability (Maintainability)	8
4.4	Reliability	10
4.5	Integrability	12
4.6	Usability	15
4.6.1	Notification A Usability Testing	15
4.6.2	Notification B Usability Testing	17
4.6.3	Remarks	18

1 Introduction

This document was compiled by our group and was produced as a whole by the team.

We were tasked with testing the two Buzz Notification modules. The functionality provided by the buzzNotification module should be focused on a user registering to receive a notification messages.

2 What we tested

Our testing involved looking at the two notification modules and comparing it to the specification. The use cases we will be evaluated can be seen in figure 3

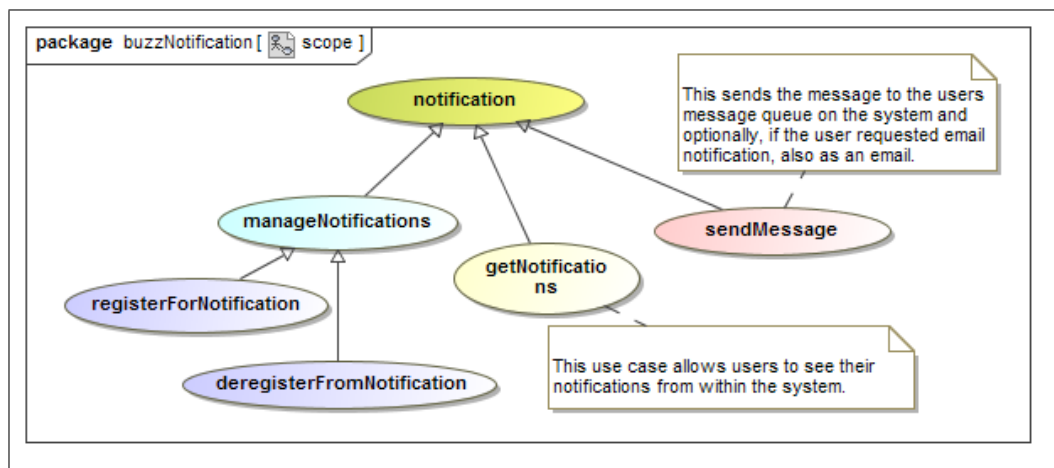


Figure 1: The scope of the buzzNotification module.

3 Functional Testing

We tested each use case and made conclusion based on what we found. For all use cases of Buzz Notification A and B we will either that the use case a success or we will provide a list of violations of the contract requirements (e.g. pre- and post-condition violations or data structure requirement violations)

3.1 Register for Notification Use Case

We found the following problems with regards to the "Register for Notification" use case of the module.

Notification A

The notification module for team A had the following violations

1. Pre condition violations:
 - Team A provided no checks for pre-conditions.
2. Data structure requirement violations:
 - None found.

Notification B

The notification module for team B had the following violations

1. Pre condition violations:
 - None found.
2. Data structure requirement violations:
 - None found.

Remarks

Team A provided no checks for pre-conditions before writing data to the database, in every case the function returns with a value of true, making it appear to succeed even when there was no data written to the database at all, and thus no registration actually happened.

Team B had no pre-condition violations. Any data that violated pre-conditions was correctly checked for and handled, registration of the user is only made if all the pre-conditions are successfully met.

3.2 Deregister for Notification Use Case

We found the following problems with regards to the deregisterFromNotification use case of the module.

Notification A

The notification module for team A had the following violations

1. Pre conditions:

- Check that the user is registered.

The function attempts to remove a record from the database and an error is thrown from the database. If error is null then it shows that such a record exists thus the user was registered. If the error is not null then the record did not exist and thus the user is not registered to notification. Thus the pre-condition is checked for but not in the best way possible.

Thus no pre-condition violation.

2. Post conditions:

- The function returns based on success or failure.

This post-condition is violated as the function returns true as long as the function terminates. Thus the function return is not an indication as to whether the execution was a success or failure.

- The user is de-registered from notification. By making changes to database.

The function removes the record corresponding to the input parameters. Thus the post-condition is not violated.

3. Data structure requirement violations:

- The data structure is violated in the sense that there is no return object. Only a set boolean value is returned.
- The function itself is well structured and easy to follow.

Notification B

The notification module for team B had the following violations

1. Pre condition:

- Check that the user is registered.

The function takes the object it receive as a parameter and checks that the details and the user exists in the database and thus is registered for notification. Thus the pre-condition is met.

2. Post conditions:

- The function returns based on success or failure.

This post-condition is violated as the function does not return any value. This means that the user does not know whether the function was a failure or success.

- The user is de-registered from notification. By making changes to database.

The function removes the record corresponding to the input parameters. Thus the post-condition is not violated.

3. Data structure requirement violations:

- The data structure is violated in the sense that there is no return object.
- The function is very long with a lot of nested if-statements thus poor structure.

Remarks

Both teams' functions achieved the final goal which is to deregister a user from notification. Both teams violated the post-conditions by not returning an object indicating whether the execution was a success or not.

3.3 Send message

We found the following problems with regards to the Send message Use Case of the module.

Notification A

The notification module for team A had the following violations

1. Preconditions:

- Check that the user is registered

A function is used to determine whether or not the user has been registered to receive notifications.

- sends user email
2. Postconditions:
 - Function returns a message based on success or failure
A message is created
 - An email is sent to the user
The message is sent in an email to the registered user
 3. Data structure requirement violations:
 - It did not take into account callback functions thus the data structure requirements are violated.

Notification B

The notification module for team B had the following violations

1. Preconditions:
 - Checks that the user is registered
A function is used to determine whether or not the user has been registered to receive notifications.
 - Takes in a JSON string
It takes a string in the format FROM, TO, SUBJECT, PLAIN, HTML
2. Postconditions:
 - A user is sent an email
A user is sent an email using a JSON string, the email contains the recipients and the message.
3. Data structure requirement violations:
 - Makes use of callback functions thus nothing is being violated.
 - The email function itself is well structured and easy to follow.

Remarks

Notification A did not fully do what was required in the functional requirements. However Notifications B seemed to have managed to cover all aspects of the functional requirements

4 Non-functional testing/assessment

Problems regarding performance, scalability, maintainability, reliability, usability as well as evidence/proof of said problems.

4.1 Performance

The notification module had the following problems regarding performance of the notification system both A and B.

Notification A

- No test file. To start off with, Notification A had no test file to actually test the performance with.
- Hardcoded Notification A has 306 lines of coding more than Notification B. It can also be noted that they had a lot of commented out work in their file, but it can be seen that it actually belongs there(so we are assuming they hid it for demo purposes). They did a lot of hardcoding, where as Notification B found a way of streamlining the process.

Notification B

- Buggy Test file Even though they had a test file for the coding, which was configured with nodeUnit to do unit tests etc., The file did not finish the test. It started testing, had no assertions, connected to the server, but then just stopped. This is mainly because of the .listened function which never ends automatically.
- Large file Compared to Notification A, Notification B has a 6.67x larger package file(20kb against 3kb). Although this is still quite small overall, it just shows that it could still have been programmed more compactly.(It has to be noted though that they do have unit tests, unlike Notification B).

Remarks

The overall impression of both teams is that they did not meet the requirements that were needed. Although the performance was not the worst issue they had, there is still a lot of room for improvement on both teams' work before a fully deployable product can come out of it.

4.2 Scalability

In regards to scalability of the notification module for both A and B the system should have had the ability to work for all Computer Science modules at a University of the size of the University of Pretoria. Each teams module is discussed below

Notification A

The notification A module will be scalable for any size as it is a proper NodeJS module, meaning one could launch multiple instances over a cluster of servers without any conflict or problem.

A problem could however arise i.t.o. scalability as the module does not cater for rapid switching of the database it uses. This could lead to problems down the line where you would need to manually intervene if you would want to support more then once database connection/location.

Notification B

The notification B module will not be scalable as it is an express application and as such wont be able to be included into another NodeJS application. This is because the functionality of the notification system is intertwined with a Express web server code. In their main file at the top they have the following code:

```
var express = require('express'),
app = express();
```

This will result in each inclusion of the notification module having a full blown express application. It would be a extreme waste of system resources to scale this module.

Remarks

Neither of the teams did what was required fully by the architectural requirements but Notification A does allow for scalability. There were no major problems or concerns in regard to Notification A unlike notifications B which would not be scalable.

4.3 Plug-ability (Maintainability)

As defined in the Architectural Requirements Document all modules should be designed and developed in such a way that they are modular allowing for

sub modules to be added or even removed during development.

This allows for a more maintainable system as Diagnostics of potential flaws and function clashes can now be eliminated as the system can be tested as individual parts or even as a whole allowing for a more stable module.

Team A and B's modules will each be discussed below with regards to this non functional requirement:

Notification A

- The initial problem with Notification A is that all the code is part of the same Java Script file suggesting that the whole module is dependent on itself and if one function is providing the wrong functionality the whole module will fail or produce incorrect output.
- Opening the Java Script file one can immediately see that this module is not made of sub modules and testing is done by commenting out of code. This can produce many errors and is hard to test since you don't know which line of code is producing errors or is providing incorrect results.

Notification B

- Notification B has sub-divided the whole notification module into smaller more manageable sub modules allowing for individual testing.
- Each sub module can then be altered and tested before integrating into the final module allowing the system to be maintained at a constant stable version. New functionality can now be added via new submodules allowing for plug ability.

Remarks

Notification A did not meet the non functional requirement maintainability as their code is not modular while on the other hand Notification B made an excellent job to ensure that their code remains as modular as possible.

4.4 Reliability

Notification A

Below are some of the issues that could be identified in the module provided by Notifications A, in terms of reliability.

- **Function Return Values:**

All functions available in the module's API should return some indication of success. However, in the case of this module, all the functions that return a value, only return true (proof of which can be found on lines 137, 169, 215, 260, 305). This means that regardless of what the outcome of the function's execution is, the caller will only perceive it as having succeeded — which is an issue of reliability: if an error occurs and needs to be handled by the system, nothing can be done because the function does not notify the system of erroneous execution.

- **Callback Usage:**

A follow-on of the previous point is the usage of callbacks. No provision has been made for the use of callback functions and therefore any information provided by asynchronous execution cannot be conveyed to the caller. So any return value, regardless of whether it may be true or false, cannot be trusted as the asynchronous part of the function will most likely not have been completed before it returns to the caller, i.e. the return value is unreliable.

Below is a listing of the function signatures that require callbacks, but do not make provision for them:

```
line 94  : notification.notifyRegistration(jsonObject)
line 140 : notification.notifyDeregistration(jsonObject)
line 172 : notification.notifyNewPost(jsonObject)
line 218 : notification.notifyDeletedThread(jsonObject)
line 263 : notification.notifyMovedThread(jsonObject)
line 308 : notification.appraisalRegister(jsonObject)
line 321 : notification.appraisalDeregister(jsonObject)
line 336 : notification.appraisalNotify(jsonObject)
line 390 : notification.sendNotification(jsonObject)
```

- **Logging to Console:**

In the cases where errors occur and in some cases where successful execution occurs, information about the success or failure is lost to the system as it is only logged to the console(not reliable as a persistence tool) instead of being returned in some form that the caller can interpret. This is another issue of reliability as the system cannot determine

whether or not a user will now receive notifications or if a user has been deregistered from receiving notifications, etc. An example of the above can be found starting on line 321:

```
notification.appraisalDeregister = function (jsonObj) {
  appraisals.remove({
    notification_StudentID: jsonObj.studentID,
    notification_AppraisalType: jsonObj.appraisalType
  }, function (err) {
    if (err != null)
      console.log(err);
    else
      console.log("user has been removed from table");
  });
};
```

Notification B

Below are discussed some of the issues identified, in terms of reliability, for the Notifications B module.

- **Function Return Values:**

In many cases, no values are returned at all, which forces the system to assume that the call executed correctly. This causes the system to be unreliable as there is no guarantee that the function call succeeded or not. An example of this is shown below and is taken from Appraisal-NotifyMe.js (lines 37 - 58):

```
function addAppraisalToDB(details)
{
  Subscription.findOne(
  {
    'user_id':details.post_user_id

  },function(err,docs){...});
}
```

Some code has been removed from the above, as not to clutter the document with redundant code, and has been replaced with '...'.

- **Callback Usage**

In most cases, like Notifications A, no provision has been made for callback functions to asynchronous function calls. This results in no means of obtaining reliable result information from the functions. Below is a list of the files containing the functions that fail to make provision for callback functions where they are needed:

```
AppraisalNotifyMe.js
DailyNotif.JS
DeleteNotif.js
StandardNotification.js
```

- **Logging to Console:**

As with Notifications A, information that should be returned to the caller is logged to the console where it will be lost and not serve any useful purpose. An example of this is as follows, taken from DeleteNotif.js (lines 159 - 170):

```
newNotif.save(function(err,newNotif)
{
    if (err)
    {
        success = false;
        console.log("Error Adding Notification ");
    }
    else
    {
        success = true;
    }
});
```

Reliability might be improved by incorporating logged information in a result object (as specified in the specification) so that the system can be informed if any issues arise and thus provide a reliable service.

Remarks

A number of problems pertaining to reliability have been identified across both modules. And in comparison, these issues are the same in both cases. It would be highly recommended that these issues be addressed accordingly as to improve the reliability of the module and thus the system as a whole.

4.5 Integrability

The system should be able to easily address future integration requirements by providing access to its services using widely adopted public standards. We found the following problems with regards to the "Integratability" use case of the module.

Integratability A

Notification A is not integratable because of the following reasons.

- **Installing required Packages.** There is no way of installing the required packages. This degrades the quality of the integratability of the system as one has to manually figure out which packages to include in the system.
- **Dependancy Injection** There is no dependancy injection.
- **Unit tests** There was no supplied unit test. Unit test are critical to determine whether or not we can integrate it into a larger system.

Integratability B

Notification B is not integratable because of the following reasons.

- **File Structure** Each function is placed in a separate file. There is no common module to integrate that will allow access to all the capabilities of notifications.
- **Installing required packages** One cannot easily install the dependencies that is required by Notifications.
- **Dependancy Injection.** There is no dependancy injection.
- **Unit tests** There is a file called test.js that is an attempt at unit testing but no proper unit testing was applied. They should have used something similar to Unit.js. ??

```

1 var send = require('./Email.js');
2 var DeleteNotification = require('./DeleteNotif.js');
3 var DailyNotification = require('./DailyNotif.js');
4 var AddAppraisalToDB = require('./AppraisalNotifyMe.js');
5 var StandardNotification = require('./StandardNotification.js');
6
7 var express = require('express'),
8     app = express();
9 var bodyParser = require('body-parser');
10 app.use(bodyParser.urlencoded({extended: true}));
11 var http = require('http'), fs = require('fs');
12
13 // as only one page can use res.sendFile to render the page which will contain the drop downs
14 app.post('/', function (req, res) {
15     res.sendFile('test.html');
16 });
17
18 //Builds the content used to send the email using the appraisal type
19 app.post('/appraisal', function (req, res) {
20
21     var appraisal = req.body.appraisal;
22     var user = req.body.User_id;
23     var post = req.body.Post_User_id;
24     var thread = req.body.Thread_id;
25
26     console.log(appraisal+user+thread+post);
27     var options = {
28         current_user_id: user,
29         post_user_id: post,
30         appraisedThread_id: thread,
31         appraisalType: appraisal
32     }
33
34     res.sendFile('test.html');
35     AddAppraisalToDB.addAppraisalToDB(options);
36
37 });
38

```

Figure 2: Unit Tests

- database issues There is no way to access the specified database and this also contributes to the integratability. They should supply a way to specify the database to be used. ??

```
12 var notification = require('./Notification\B/test.js');
13
14
15
Notification Packages — node — 119x24
le-stream,
npm WARN invalid dependency which is version 1.0.33
body-parser@1.12.3 node_modules/body-parser
├── content-type@1.0.1
├── raw-body@1.3.4
├── bytes@1.0.0
├── depd@1.0.1
├── iconv-lite@0.4.8
├── on-finished@2.2.1 (ee-first@1.1.0)
└── type-is@1.6.1 (media-typer@0.3.0, mime-types@2.0.10)
Hanrichs-MacBook-Pro:Notification Packages hanrich$ node test.js
Testing integrability
connection error: { [Error: Trying to open unclosed connection.] state: 2 }
connection error: { [Error: Trying to open unclosed connection.] state: 2 }
connection error: { [Error: Trying to open unclosed connection.] state: 2 }
connection error: { [Error: Trying to open unclosed connection.] state: 2 }
connection error: { [Error: Trying to open unclosed connection.] state: 2 }
connection error: { [Error: Trying to open unclosed connection.] state: 2 }
Server is running.
connection error: { [MongoError: connect ECONNREFUSED] name: 'MongoError', message: 'connect ECONNREFUSED' }
connection error: { [MongoError: connect ECONNREFUSED] name: 'MongoError', message: 'connect ECONNREFUSED' }
connection error: { [MongoError: connect ECONNREFUSED] name: 'MongoError', message: 'connect ECONNREFUSED' }
connection error: { [MongoError: connect ECONNREFUSED] name: 'MongoError', message: 'connect ECONNREFUSED' }
```

Figure 3: Database connection issues

Both modules suffer from common symptoms when it comes to Integrability. I therefore conclude that neither system is integratable and needs to look at dependancy injection. Each package should at least include a npm readme file to specify the packages are required.

Remarks

Notification B is not Integratable at all. There is no provision for dependancy injection.

4.6 Usability

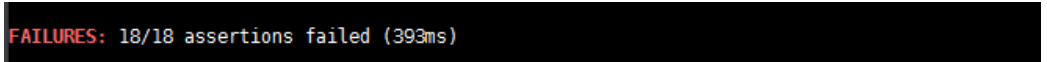
The issues related to usability and the testing of usability are included below:

4.6.1 Notification A Usability Testing

- The first and foremost problem is there is no way to test if the notification section is usable as no testing infrastructure for usability is provided. The team would need to implement just a basic interface to allow users to actually use the functions they have implemented. Thus, without this in place the notification module has no real usability. In order to provide a full usability testing section, the authors have

assumed that there is a functioning interface for some of the points below.

- There was no testing interface that functioned correctly. This prevented further usability tests as the tester could not see if the functions were procedurally correct. All 18 of the 18 tests failed when attempting to use nodeunit(the testing infrastructure that was provided). This can be seen in the figure 4. This shows further that the module, even with a functioning interface, would not be able to have any usability.



```
FAILURES: 18/18 assertions failed (393ms)
```

Figure 4: Usability test failure

- The call-back nature of many of the functions, as mentioned in previous sections, is lacking. Many of the functions that are called return Boolean values but they do not use call-back functions in order to return these functions. This in combination with the asynchronous nature of JavaScript, or node.js, causes the values to always return true values. With relation to usability, the user would always assume that when they register for a notification, unregister, or even opt in for e-mail notifications. This proves to be false as the functions do not return properly which can cause frustration for the user as the program does alter values as expected. The usability for the function calls within the module is null.
- Similar to the above point, none of the functions throw errors. This prevents notification to user of problems within the application. The user would become frustrated as they would not be able to understand why the program is not functioning correctly. The user would not be able to provide specific debugging information as feedback to the developers.
- NPM install of the module does not function at all(See Figure 5 below) . So with regards to usability for developers the module has failed.
- The programs code is poorly commented but at least maintains some concept of simplicity.

```

C:\> .\WebstormProjects\test-npm install https://github.com/RendaniDau/COS301PhaseNotificationA/tarball/master
npm ERR! Windows_NT 6.1.7601
npm ERR! argv "C:\\Program Files\\nodejs\\node.exe" "C:\\Program Files\\nodejs\\node_modules\\npm\\bin\\npm-cli.js" "install" "https://github.com/RendaniDau/COS301PhaseNotificationA/tarball/master"
npm ERR! node v0.12.1
npm ERR! npm v2.5.1
npm ERR! file C:\> .\WebstormProjects\test-npm-11900-289f4807\unpack-51ee5471d631\package.json
npm ERR! code EJSONPARSE

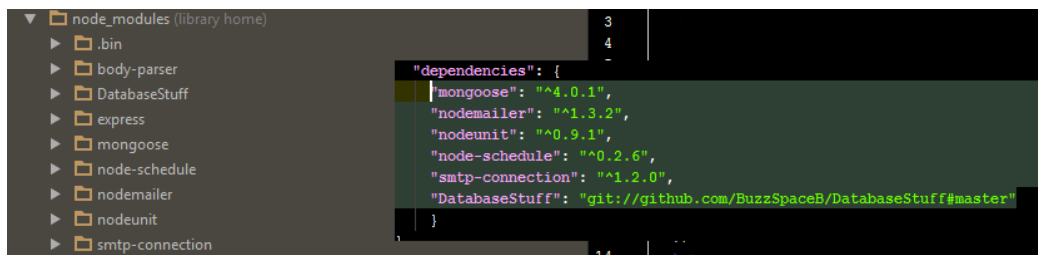
npm ERR! Failed to parse json
npm ERR! Unexpected string
npm ERR! File: C:\> .\WebstormProjects\test-npm-11900-289f4807\unpack-51ee5471d631\package.json
npm ERR! Failed to parse package.json data.
npm ERR! package.json must be actual JSON, not just JavaScript.
npm ERR!
npm ERR! This is not a bug in npm.
npm ERR! Tell the package author to fix their package.json file. JSON.parse
npm ERR! Please include the following file with any support request:
npm ERR! C:\> .\WebstormProjects\test-npm-debug.log

```

Figure 5: NPM Failure

4.6.2 Notification B Usability Testing

- The usability for developers is lacking. The npm install of the module does not give all the required files for testing to work. The tester had to manually install some of the modules as they required to see if the testing program runs. Usability on a developer level is substandard.



```

{
  "dependencies": {
    "mongoose": "^4.0.1",
    "nodemailer": "^1.3.2",
    "nodeunit": "^0.9.1",
    "node-schedule": "^0.2.6",
    "smtp-connection": "^1.2.0",
    "DatabaseStuff": "git://github.com/BuzzSpaceB/DatabaseStuff#master"
  }
}

```

Figure 6: Dependency Failure

- Notification B did provide a testing file which seems to emulate the idea of sending e-mail notifications. The problem with this system is that many of the functions fail to execute. This can cause frustration for the users as they would not receive expected notifications from services. Thus the required usability is not met.
- In aspects of programmer usability, the code is poorly commented in some sections and near perfect in others. The code is poorly structured and creates an overall impression of disorganisation
- Similar to above with regards to callbacks and thrown errors. The difference being that some errors are thrown.

← → ↻ 127.0.0.1:5000

Apps For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks ne](#)

Testing Email Functionality

Plain Notification

Appraisal Notification

Select the type of appraisal:

Delete Notification

Select the thread to be deleted:

Would you like to send a notification about the delete ☒ Yes ☐ No

Enter the reason for deletion:

Figure 7: Emulated Interface

4.6.3 Remarks

The overall impression of both notification sections is that they did not meet the requirements that were needed. The usability was substandard and a lot of improvement is needed for a fully deployable product.