# Testing - Alpha Integration

## Members


14395283 Alberts JD (Josef)
14011396 Grobler A (Arno)
14035538 Heins D (Dillon)
12031748 Khumalo S (Sandile)
12017800 Maree AA (Armand)
14103207 Mogase L (Lethabo)
14222583 Molefe KP (Keletso)
13278012 Muranga KJ (Kudzai)
13040686 Nyuswa ML (Maluleki)
12059138 Saaiman C (Christiaan)
13027205 Smallwood DW (Duncan)



University of Pretoria
COS 301
20 April 2016

# Contents


**1 Introduction** **1**
1.1 Testing . . . 1

**2 Functional Testing** **2**
2.1 Reporting . . . 2
2.1.1 Implementation . . . 2
2.1.2 Short-comings . . . 2
2.1.3 Missing . . . 2
2.2 Notifications . . . 2
2.2.1 Implementation . . . 2
2.2.2 Short-comings . . . 2
2.2.3 Missing . . . 2
2.3 People . . . 2
2.3.1 Implementation . . . 2
2.3.2 Short-comings . . . 3
2.3.3 Missing . . . 3
2.4 Publications . . . 3
2.4.1 Implementation . . . 3
2.4.2 Short-comings . . . 3
2.4.3 Missing . . . 3
2.5 Interface . . . 3
2.5.1 Implementation . . . 3
2.5.2 Short-comings . . . 3
2.5.3 Missing . . . 3
2.6 Integration . . . 4
2.6.1 Implementation . . . 4
2.6.1.1 Use Case Prioritisation: . . . 4
2.6.1.2 Service Contracts: . . . 4
2.6.1.3 Required Functionality: . . . 4
2.6.1.4 Process Specifications: . . . 4
2.6.1.5 Domain Models: . . . 4
2.6.2 Short-comings . . . 4
2.6.3 Missing . . . 4

**3 Architectural Compliance** **5**
3.1 Reporting . . . 5
3.1.1 Software Architecture Adhered . . . 5
3.1.1.1 Flexibility: . . . 5
3.1.1.2 Scalability: . . . 5
3.1.1.3 Testability: . . . 5
3.1.1.4 Deployability: . . . 5
3.1.2 Software Architecture Partially Adhered . . . 5
3.1.2.1 Maintainability: . . . 5
3.1.2.2 Performance requirements: . . . 5
3.1.2.3 Integrability: . . . 5
3.1.3 Software Architecture Not Adhered . . . 5
3.1.3.1 Reliability: . . . 5
3.1.3.2 Security: . . . 5
3.1.3.3 Auditability: . . . 5
3.1.3.4 Usability: . . . 5

3.2 Notifications . . . 6
3.2.1 Software Architecture Adhered . . . 6
3.2.2 Software Architecture Partially Adhered . . . 6
3.2.3 Software Architecture Not Adhered . . . 6
3.3 People . . . 6
3.3.1 Software Architecture Adhered . . . 6
3.3.2 Software Architecture Partially Adhered . . . 6
3.3.3 Software Architecture Not Adhered . . . 6
3.4 Publications . . . 6
3.4.1 Quality Requirements . . . 6
3.4.1.1 Flexibility . . . 6
3.4.1.2 Maintainability . . . 6
3.4.1.3 Scalability . . . 6
3.4.1.4 Performance Requirements . . . 6
3.4.1.5 Auditability . . . 6
3.4.1.6 Reliability . . . 6
3.4.1.7 Security . . . 6
3.4.1.8 Testability . . . 6
3.4.2 Software Architecture Adhered . . . 7
3.4.3 Software Architecture Partially Adhered . . . 7
3.4.4 Software Architecture Not Adhered . . . 7
3.5 Interface . . . 7
3.5.1 Software Architecture Adhered . . . 7
3.5.2 Software Architecture Not Adhered . . . 7
3.6 Integration . . . 7
3.6.1 Software Architecture Adhered . . . 7
3.6.1.1 Modularity . . . 7
3.6.2 Software Architecture Partially Adhered . . . 7
3.6.2.1 Maintainability . . . 7
3.6.2.2 Testability . . . 7
3.6.3 Software Architecture Not Adhered . . . 7
3.6.3.1 Reliability . . . 7
3.6.3.2 Security . . . 7
3.6.3.3 Auditability . . . 7
3.6.3.4 Deployment . . . 7
3.6.3.5 Architectural Patterns . . . 7

# 1 Introduction

This project is a Research Support System. Its main purpose is to keep track of many aspects that are regularly needed by researchers and an easy way for research heads or HOD of the respective facility to be able to see progress that has been made by the researchers in their team.

The scope for this project is only the Computer Science department and its respective lecturers and leaders. General uses of this system should include keeping track of:

- Research Papers
- Reports, of which has different types
- Research Groups
- Running Costs
- Historical publications
- List of Authors
- List of Users
- Units

## 1.1 Testing

Software testing plays an important part of the software development pipeline and is necessary for creating maintainable and scalable programmes. It has many different objectives and goals but ultimately it comes down to:

- Finding defects in the program that was developed and possibly finding solutions to those problems
- To ensure the level of quality has been maintained throughout the program
- Find potential security flaws and vulnerabilities
- Ensuring usability
- Ensuring the program has lived up to the system requirements set out by the clients.

Testing ensures that the project, which should be in its final stages of development, is ready to be rolled out to the clients and that it meets the requests made by the clients. Thus certain fields of the program could be addressed. These fields are:

- Flexibility
- Maintainability
- Scalability
- Performance requirements
- Reliability
- Security
- Auditability
- Testability
- Usability
- Integrability
- Deployability

# 2 Functional Testing

## 2.1 Reporting

### 2.1.1 Implementation

The integration system gains access to the reporting module via an interface called IReporting. This provides the system with two functions, each to get a different category of report (Progress or Accreditation Unit). The team has implemented all the functions necessary to integrate the Reporting module and all functions run as expected. Comprehensive unit tests have been written by the Alpha team and all tests pass - this includes tests which execute a request for a report without trouble, as well as requests which expect exceptions. The Alpha Team has implemented custom Exceptions for invalid Report parameters among other things, and these Exceptions are caught and handled accordingly. The integration of the reporting module includes a REST layer which consumes a JSON Object (request) and generates a Response instance to return. They utilized the Spring framework for dependency injection and stateless entity beans are used for the different entities.

### 2.1.2 Short-comings

There does not appear to be any shortcomings in the integration of the reporting system.

### 2.1.3 Missing

There are no functions missing.

## 2.2 Notifications

### 2.2.1 Implementation

Unit tests were performed for all the functionality within the notification module.Dependency injection was used through the spring framework and everything worked.J unit was used accordingly and all the test cases that were written passed.

### 2.2.2 Short-comings

There is only one short-coming, the notification is sent out correctly, but the scheduling does not work accordingly.

### 2.2.3 Missing

There is no missing functionality, just that the above mentioned is not working correctly.

## 2.3 People

### 2.3.1 Implementation

The people module is accessed via an interface specified as IPeople. This is then implemented in the defaultImpl package where all methods return null, enabling the system to compile and pass all build tests. The functionality of the People module has been extensively mocked and tested within the testing framework. Expected values were checked to make sure the correct objects were being handled and returned. They were sure to include all the exceptions that needed be thrown if an error occurred and showed comprehensive use of the assert library functions. Alpha Integration had an extensive exception library, making sure to have an exception class for each and every possible exception that may occur. The seamless integration of this module is apparent that the layout was well thought out. A request object is sent in, encapsulating various necessary information to perform a function. A response object is returned to notify the system of the status of the function's execution.

### 2.3.2 Short-comings

Short-comings aren't present in the testing and integration of the People module. Everything was covered to the edge and beyond.

### 2.3.3 Missing

There are no functions missing.

## 2.4 Publications

### 2.4.1 Implementation

The integration functionality to get a publication works, it takes in a json object request and returns a response with is a json object.if the publication does not exist the is a catch an exception or if the request in not valid it will catch an exception. for that they use a rest layer. This is the only functionality implemented by the integration team for publication.

### 2.4.2 Short-comings

There were no short comings in the get publication function

### 2.4.3 Missing

There is no other functionality to modify a publication such as to change publication state ,add publication type, modify publication type and also to getPublicationForPerson,getPublicationForGroup and calculate Accreditation points for both group and person.

## 2.5 Interface

### 2.5.1 Implementation

The integration functionality, in terms of integrating the rest of the back-end with the interface or front-end for both the web interface and android, was to have communication between the front and back-ends through AJAX calls to get or post new information to be displayed or stored. The web interface used ember and mirage server to handle the calls and create a mock database. Requests are made through user interaction which will send either GET requests if the user wants to retrieve information to display your use and will use POST requests if the user wants to update the database. other requests such as PUT, PATCH and DELETE which is needed to be compliant with the RESTful API.

### 2.5.2 Short-comings

Since the integration team did not implement any integration of front and back-end, there was no code to test.

### 2.5.3 Missing

There is no functionality put in place for the integration of web or android code with any of the other back-end modules and so the section as a whole is missing.

## 2.6 Integration

### 2.6.1 Implementation

The system as a whole has been implemented well in terms of functional requirements. Service contracts have been mapped onto interfaces for each of the individual modules. The scope of functionality for each module has been adhered to and provisions have been made to ensure the system caters for all of these required functionalities. Domain models have been followed correctly and care has been taken in order to implement custom exception, request and response objects.

**2.6.1.1 Use Case Prioritisation:** Critical, important and nice-to-have use cases have been mapped onto interfaces for the most part. The most critical use cases have been catered for.

**2.6.1.2 Service Contracts:** As mentioned before the service contracts have all been mapped correctly onto interfaces. All service contracts and modules have been accounted for. Pre-conditions and post-conditions have been adhered to and are tested through the use of mock objects and test cases for each individual module. The request and result data structures have been created and implemented correctly.

**2.6.1.3 Required Functionality:** Provisions have been made to ensure that the services offered by each module are accounted for. The integration code has tests to insure the dependency which is injected implements as well as adheres to these required functionalities correctly.

**2.6.1.4 Process Specifications:** The integration code has been implemented to obey process specifications. All services start with a request. There are pre-conditions for each service which lead to different exceptions which could be thrown or which lead to the service being terminated. If all pre-conditions have been met the service returns the correct values. Tests have been implemented to ensure this flow of execution is carried out correctly.

**2.6.1.5 Domain Models:** As mentioned before the domain models have been adhered to which in turn has lead to the successful implementation of the service contracts, required functionality and process specifications in terms of data structures implemented.

### 2.6.2 Short-comings

There are very few short-comings with regards to the integration as a whole in terms of functional requirements.

### 2.6.3 Missing

In terms of functional requirements for integration there is not much missing, if anything at all.

# 3 Architectural Compliance

## 3.1 Reporting

### 3.1.1 Software Architecture Adhered

**3.1.1.1 Flexibility:** Due to the independence between the integration module and the reporting module, this part of the system is quite flexible. One could swap out the reporting module for a different system without too much trouble, if requests are still sent in an appropriate format (JSON). This is enhanced with the dependency injection used.

**3.1.1.2 Scalability:** The reporting module has been built to the spec of what was required. Scalability was not of importance and therefore no provision has been made for any.

**3.1.1.3 Testability:** This system is testable with built-in unit tests - as requested in the master spec. There are mock objects provided for unit testing and integration testing.

### 3.1.2 Software Architecture Partially Adhered

**3.1.2.1 Maintainability:** The system is poorly documented, which makes maintenance more difficult for a third-party developer. On the other hand, the system uses technologies we believe will be in regular use for the foreseeable future. Furthermore, despite the lack of documentation, the system does have a rather logical structure which should assist a new developer in understanding the system. Maintainability is, however, slightly hampered by the main Reporting interface, since it requires a unique function for each report category and adding a category may be difficult.

**3.1.2.2 Performance requirements:** The system has been built in a manner which, if it were to be able run, it would be able to meet the performance requirements. However seeing as it does not, performance requirements have not been met.

**3.1.2.3 Integrability:** The data does have integrability to some degree but as there is no security it is easy for that to change.

### 3.1.3 Software Architecture Not Adhered

**3.1.3.1 Reliability:** The reporting module's reliability could not be tested as it is not executable.

**3.1.3.2 Security:** There is no security or authorisation in place.

**3.1.3.3 Auditability:** There is no logging functionality evident within the reporting integration itself. Exceptions are thrown, but are either caught and ignored, or caught in the unit tests. Successes are not logged, and it does not seem any other activity is either. Calls to services are, however logged, higher up in the system but these are not specific to the reporting.

**3.1.3.4 Usability:** The system's usability is non-existent as it does not run.

**3.1.3.5 Deployability:** The system is unable to be deployed in an easy manner as it does not run.

## 3.2 Notifications

### 3.2.1 Software Architecture Adhered

### 3.2.2 Software Architecture Partially Adhered

### 3.2.3 Software Architecture Not Adhered

## 3.3 People

### 3.3.1 Software Architecture Adhered

### 3.3.2 Software Architecture Partially Adhered

### 3.3.3 Software Architecture Not Adhered

## 3.4 Publications

### 3.4.1 Quality Requirements

**3.4.1.1 Flexibility** The integration code complies to this requirement because it has the ability to deploy different versions of the system with as little down-time as possible. This is partly due to the decoupled nature of the integration and publication sections. The code also makes use of dependency injection for contract based software development.

**3.4.1.2 Maintainability** The code has very little to no documentation, which makes it very difficult for developers that did not on the system initially to be able to understand the code.

**3.4.1.3 Scalability** The system is able to store a large amount of the department's publication meta data on the database without any difficulty.

**3.4.1.4 Performance Requirements** All the the services of Publication operate within the required time of 0.2 seconds.

**3.4.1.5 Auditability** The system does not log any of the requests, responses and exceptions of the publication services.

**3.4.1.6 Reliability** The service requests are fully implemented and can be fully executed for each service contract for the publication section. It also supports commit and rollback functionality for the database.

**3.4.1.7 Security** The integration code does not check the authorisation of the user for the services that only should be used by specific users. For example, the addPublicationType service should only allow administrators to use it. This is not implemented.

**3.4.1.8 Testability** Junit is used for out-of-container testing to test the publication section. Embedded container unit testing is not used, however, this is listed as a nice-to-have.

### 3.4.2 Software Architecture Adhered

### 3.4.3 Software Architecture Partially Adhered

### 3.4.4 Software Architecture Not Adhered

## 3.5 Interface

### 3.5.1 Software Architecture Adhered

Since there is no interface in place there is no architecture that was adhered to.

### 3.5.2 Software Architecture Not Adhered

- The web interface is supposed to use the Ember.js engine, but there are no traces of any Ember.js code, a server that would house this component or some interface where connections can be established to.
- There is also no trace of a connection point for an Android interface to make requests to.

The integration system does not accommodate the interfaces on any level, thus no software architecture requirements can be tested on this section

## 3.6 Integration

### 3.6.1 Software Architecture Adhered

**3.6.1.1 Modularity** The system has a very large amount of modularity. In some sections it becomes unnecessarily modular. This will help with future scaling and adding new components to the system.

### 3.6.2 Software Architecture Partially Adhered

**3.6.2.1 Maintainability** The system is fairly hard to understand and future developers are going to struggle in understanding the code since there is very little to no documentation and comments.

**3.6.2.2 Testability** Although some of the components of the system has the ability to be tested, not all components have this functionality. One can thus not be sure whether the entire system is reliable.

### 3.6.3 Software Architecture Not Adhered

**3.6.3.1 Reliability** This system is not reliable at all since the code does not even compile. There is also no trace of error checking.

**3.6.3.2 Security** There is supposed to be functionality that would only allow certain individuals to do certain actions but there is no functionality that would require an individual to sign in. This would obviously allow unauthorized individuals to request sensitive information without hassle.

**3.6.3.3 Auditability** The system has no functionality to log requests and responses made by the system. Thus no one can be held accountable for certain actions that is taken.

**3.6.3.4 Deployment** The system does not deploy at all (the code does not compile).

**3.6.3.5 Architectural Patterns** REST was not adhered to properly, no services were accessible from a web front-end or a mobile-device client.