

# Architecture Requirements

## COS 301 ASSIGNMENT GROUP 2 B

Duran Cole (13329414)  
Johannes Coetzee (10693077)  
Estian Rosslee (12223426)  
Edwin Fullard (12048675)  
Herman Keuris (13037618)  
Martha Mohlala (10353403)  
Motsoape Mphahlele (12211070)  
Xoliswa Ntshingila (13410378)

10 March 2015

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Architecture Requirements</b>	<b>2</b>
2.1	Architectural Scope . . . . .	2
2.2	Quality Requirements . . . . .	2
2.2.1	Critical: . . . . .	2
2.2.2	Important: . . . . .	3
2.2.3	Nice to have: . . . . .	5
2.3	Architectural Constraints . . . . .	5
<b>3</b>	<b>Architectural patterns or styles</b>	<b>6</b>
3.1	Layered Architectural Style . . . . .	6
<b>4</b>	<b>Architectural tactics or strategies</b>	<b>7</b>
4.1	Fault Detection . . . . .	7
4.1.1	Ping / Echo . . . . .	7
4.1.2	System Monitor . . . . .	8
4.1.3	Exception Detection . . . . .	8
4.2	Fault Recovery . . . . .	9
4.2.1	Preparation and Repair . . . . .	9
4.2.2	Reintroduction . . . . .	10
4.3	Fault Prevention . . . . .	11
4.3.1	Removal from Service . . . . .	11
4.3.2	Transactions . . . . .	11
4.3.3	Process Monitor . . . . .	11
<b>5</b>	<b>Use of reference architectures and frameworks</b>	<b>11</b>
<b>6</b>	<b>Access and integration channels</b>	<b>12</b>
6.0.4	Human access channels . . . . .	12
6.0.5	System access channels . . . . .	12
6.1	Integration Channel Used . . . . .	12
6.1.1	REST - Representational State Transfer . . . . .	12
6.2	Protocols . . . . .	13
6.2.1	HTTP - Hypertext Transfer Protocol . . . . .	13
6.2.2	IP - Internet Protocol . . . . .	13
6.2.3	SMTP - Simple Mail Transfer Protocol . . . . .	13

6.2.4	TSL - Transport Layer Security . . . . .	13
<b>7</b>	<b>Technologies</b>	<b>13</b>
7.1	Programming technologies . . . . .	13
7.1.1	Java EE . . . . .	13
7.1.2	JSF - Java Server Faces . . . . .	14
7.1.3	JPA - Java Persistence API . . . . .	14
7.1.4	JPQL - Java Persistence Query Language . . . . .	14
7.1.5	Maven . . . . .	14
7.1.6	AJAX - Asynchronous JavaScript and XML . . . . .	14
7.2	Web technologies . . . . .	15
7.2.1	HTML - Hypertext Mark-up Language/XHTML - eX- tensible Hypertext Mark-up Language . . . . .	15
7.2.2	CSS - Cascading Style Sheet . . . . .	15

# 1 Introduction

This section deals with the software architecture requirements of the Buzz system being designed. It handles all aspects of the system's design which form part of its non-functional requirements, in particular the requirements of the software architecture on which the application's functional aspects are developed. This includes:

- The architectural scope.
- Quality requirements.
- The integration and access channel requirements.
- The architectural constraints.
- Architectural patterns and styles used.
- The architectural tactics and strategies used.
- The use of reference architectures and frameworks.
- Access and integration channels.
- Technologies used.

## **2 Architecture Requirements**

### **2.1 Architectural Scope**

The following responsibilities need to be addressed by the software architecture:

1. To be able to host and provide an execution environment for the services/business logic of the Buzz system
2. To provide an infrastructure for a web access channel
3. To provide an infrastructure that provides a mobile access channel
4. To provide an infrastructure that handles persisting and provides access to domain objects
5. To integrate with a LDAP repository.
6. To provide an infrastructure that allows module plugability in such a way that core modules are independent from add on modules and vice-versa
7. To provide an infrastructure to integrate the system with other systems such as the CS department's website and the Hamster marking system

### **2.2 Quality Requirements**

The following quality requirements are in order of the most important to least important.

#### **2.2.1 Critical:**

##### **Usability**

The average student should be able to use the Buzz system without any prior training and not be discouraged to use the system again.

- The Buzz system interface must be efficient and easy to use and navigate.

- Initially only English needs to be supported, but the system must allow for translations to the other official languages of the University of Pretoria to be added at a later stage.

### **Scalability**

It is important that a large number of users can be accommodated by the Buzz system.

- Manage resource demand to ensure that every user of the Buzz system will be presented with a satisfactory experience.
- Scale out resources to ensure that resource demand be met.
- The system must be able to operate effectively under the load of all registered students within the department of Computer Science and guest users.

### **2.2.2 Important:**

#### **Availability**

It is important that the Buzz system is always accessible by the user.

- Adequately test each module of the Buzz system to minimize downtime to ensure that the system can function even if a module is down.
- The Buzz system is accessible by multiple platforms and browsers.

#### **Integrability**

The Buzz system must be able to communicate with the existing CS modules without changing the Buzz system.

- Must be able to integrate with existing systems and systems which may want to be added.
- Must be able to integrate with the existing CS website.

## **Maintainability**

The Buzz system must be maintained to keep the system in as safe state.

- Rollback to a previous state in which the Buzz system was safe. Lets us recover from an error or a system fault.
- Corrective maintenance is to correct discovered problems in the Buzz system after the system has been implemented.
- Perfective maintenance is to improve performance or maintainability after the Buzz system has been implemented.

## **Testability**

The modules of the Buzz system must be tested thoroughly before they are deployed to the final system.

- The Buzz system must use test driven development.
- Each service provided by the system must be testable through a unit test that tests:
  - That the service is provided if all pre-conditions are met, and
  - That all post-conditions hold true once the service has been provided.

## **Monitorability and Auditability**

The Buzz system must keep track of system and user activities that can be audited at a later stage.

- Each action on the system must be recorded in an audit log that can later be viewed and queried.
- Information to be recorded must include:
  - The identity of the individual carrying out the action.
  - A description of the action.
  - When the action was carried out.

### **2.2.3 Nice to have:**

#### **Security**

Sensitive information of the users must be kept secure.

- Minimize access and permissions given to users who do not have the required privileges.
- All communication of sensitive data must be done securely through encryption and secure channels such that unwanted users are unable to compromise the Buzz system.
- All system functionality is only accessible to users who can be successfully authenticated through the LDAP system used by the department of Computer Science.

#### **Performance requirements**

Minimize the response time of requests from the Buzz system to ensure that the Buzz system would work efficiently.

- The Buzz system must have a large throughput number such that the rate at which incoming requests are completed is high.

## **2.3 Architectural Constraints**

The following architecture constraints have been imposed:

1. The system must be developed using the following technologies
  - Development must take place on a distribution of the Linux operating system
  - The system must be developed using the Java-Enterprise Edition (Java-EE) development platform and the Java Server Faces (JSF) architectural framework
  - Persistence to a relational database must be done by making use of the Java Persistence API (JPA) and the Java Persistence query language (JPQL)



- The unit tests should be developed using using an appropriate unit test framework such as JUnit (a unit testing framework for the Java programming language).
2. The system must ultimately be deployed onto a GlassFish application server running on the cs.up.ac.za Apache web server.
  3. The system must be decoupled from the choice of database. The system will use a MySQL database.
  4. The system must expose all system functionality as restful web services and hence may not have any application functionality within the presentation layer.
  5. Web services must be published as either SOAP-based or Restful web services.

## 3 Architectural patterns or styles

### 3.1 Layered Architectural Style

- The Buzz system can at its simplest be implemented across the following three layers:
  1. Client access layer
    - Provides the front-end/interface through which the client interacts with the system
    - Thin client in the form of a web interface which will simply handle input from the user and display appropriate output
  2. Business logic layer
    - Provides the back-end services of the system
    - Manages authentication via communication with CS LDAP
    - Manages access to the persistence database
    - The aim of having the back-end managing the data is to achieve higher security.
  3. Infrastructure layer

- Includes the framework upon which the system is built and provides integration with other systems
- The main benefits of layered architectural style:
  - it abstracts the view of the system as whole while providing enough detail to understand the roles and responsibilities of individual layers and the relationship between them.
  - separation of concerns across different layers improves cohesion and reduces complexity
  - improved maintainability through the ability to have separate layers developed independently by different teams
  - testability is improved by separation of concerns leading to smaller independent modules which can be tested

## 4 Architectural tactics or strategies

### 4.1 Fault Detection

#### 4.1.1 Ping / Echo

- Ping / Echo refers to an asynchronous request/response message pair exchanged between nodes, used to determine reachability and the round-trip delay through the associated network path. We can use it to ensure that the students are connected and up to date with the latest version of a Buzzspace before they post something that might already have been posted.
- The quality requirements addressed:
  - Security, as it can confirm on security policies between ports and that no faults have occurred.
  - Availability, allows us to note whether server is available or not.
  - Reliability, detected faults can then be acted upon.
  - Monitorability and Audibility, To log faults detected.
  - Testability, allows errors to be found and dealt with.

#### 4.1.2 System Monitor

- Watchdog is a hardware-based counter-timer that is periodically reset by software. Upon expiration it indicates to the system monitor of a fault occurrence in the process.
- Heartbeat is a periodic message exchange between the system monitor and process. It indicates to system monitor when a fault is incurred in the process.
- Both of these can be used to prevent zombie threads (ones that expired or crashed on the client side) to clog up the server by taking up valuable resources.
- The quality requirements addressed:
  - Availability, allows us to note whether server is available or not and view current system processes.
  - Monitorability and Audibility, To monitor system and log activities and changes.

#### 4.1.3 Exception Detection

- System Exceptions are raised by the system when it detects a fault, such as divide by zero, bus and address faults and illegal program instructions.
- Parameter Fence incorporates an A priority data pattern (such as 0xdeadbeef) placed immediately after any variable-length parameters of an object. It allows for runtime detection of overwriting the memory allocated for the object's variable-length parameters.
- Parameter Typing employs a base class that defines functions that add, find, and iterate over Type-Length-Value (TLV) formatted message parameters and uses strong typing to build and parse messages.
- All exceptions needs to be handled in order to prevent any critical errors or failures which can lead to a system compromise.
- The quality requirements addressed:

- Security, Allows us to detect possible security errors and threats such as: buffer overflows, possible attacks, etc.
- Maintainability, Allows us to find and fix exceptions which could further lead to system errors and problems.
- Monitorability and Audibility, To monitor system and log all exceptions thrown by either system or user activities.
- Testability, allows exceptions to be found and dealt with.

## 4.2 Fault Recovery

### 4.2.1 Preparation and Repair

- Voting
  - Triple Modular Redundancy is three identical processing units, each receiving identical inputs, whose output is forwarded to voting logic. It then detects any inconsistency among the three output states, which is treated as a system fault.
- Active Redundancy
  - Configuration wherein all of the nodes (active or redundant spare) in a protection group receive and process identical inputs in parallel. The redundant spare possesses an identical state to the active processor, so recovery and repair can occur in milliseconds.
- Passive Redundancy
  - Configuration wherein only the active members of the protection group process input traffic, with the redundant spare(s) receiving periodic state updates. This achieves a balance between the more highly available but more complex Active Redundancy tactic and the less available but significantly less complex Spare tactic.
- Spare
  - Configuration wherein the redundant spares of a protection group remain out of service until a fail-over occurs. Then it initiates the power-on-reset procedure on the redundant spare prior to its being placed in service.

- The quality requirements addressed:
  - Monitorability and Auditability, faults are detected and logged and allows us to backtrack to an earlier state or prevent faults and recover.
  - Availability, it allows the most current operational state to be available.
  - Maintainability, allows us to backtrack to a previous working state.

#### 4.2.2 Reintroduction

- Shadow
  - Operates a previously failed or in-service upgraded component in a shadow mode for a predefined duration of time and when a problem is incurred it reverts the component back to an active role.
- State Resynchronization
  - When it's implemented as a refinement to the Active Redundancy tactic, it occurs organically as active and standby components that each receive and process identical inputs in parallel.
  - When it's implemented as a refinement to the Passive Redundancy tactic, it's based solely on periodic state information transmitted from the active component(s) to the standby component and involves the Rollback tactic. It then allows the system's control element to dynamically recover its control plane state from its network peers and periodically compares the state of the active and standby components to ensure synchronization.
- Rollback
  - Rollback is a checkpoint based tactic that allows the system state to be reverted to the most recent consistent set of checkpoints.
- The quality requirements addressed:

- Testability, allows the system to remain intact while new components are tested.
- Maintainability, allows us to backtrack to a state where components were working correctly.

## **4.3 Fault Prevention**

### **4.3.1 Removal from Service**

- Places a system component in an out-of-service state that allows for mitigating potential system failures before their accumulation affects the service.
- The quality requirements addressed:
  -

### **4.3.2 Transactions**

- Ensures a consistent and durable system state. The Atomic Commit Protocol can be used and it is most commonly implemented by the two-phase commit.
- The quality requirements addressed:
  -

### **4.3.3 Process Monitor**

- Monitors the state of health of a system process and ensures that the system is operating within its nominal operating parameters.
- The quality requirements addressed:
  -

## **5 Use of reference architectures and frameworks**

## 6 Access and integration channels

### 6.0.4 Human access channels

- Must run on all major web browsers (Mozilla Firefox, Microsoft Internet Explorer, Opera, Google Chrome, Safari)
- Must be able to access from any computer regardless which operating system it uses ( e.g. Windows, Linux, etc.), tablets and phones (i.e. mobile/Android devices)
- The user can easily use the Buzz system to communicate with both the CS LDAP server (when registering and logging on and off of their profiles) and the CS MySQL database to access course and module details.

### 6.0.5 System access channels

- The Buzz system can be accessed through a direct web page (i.e. using http) or through a link from the CS web page.
- The Buzz system must easily communicate with both the CS LDAP server (to retrieve class lists and student information) and the CS MySQL database to access course and module details.

## 6.1 Integration Channel Used

### 6.1.1 REST - Representational State Transfer

- Uses standard HTTP and thus simpler to use.
- Allows different data formats where as SOAP only allows XML.
- Has JSON support
  - faster parsing.
- Better performance and scalability with the ability to cache reads.
- Protocol Independent, can use any protocol which has a standardised URI scheme.

## **6.2 Protocols**

### **6.2.1 HTTP - Hypertext Transfer Protocol**

- Used to send web pages from server to workstations.

### **6.2.2 IP - Internet Protocol**

- Allows Communications between users.
- In charge of sending, receiving and addressing data packets.

### **6.2.3 SMTP - Simple Mail Transfer Protocol**

- Sends emails.
- MIME (Multi-purpose Internet Mail Extensions) which allows SMTP to send multimedia files.

### **6.2.4 TLS - Transport Layer Security**

- Alternative to SSL
- Newer and more secure version of SSL.

## **7 Technologies**

### **7.1 Programming technologies**

#### **7.1.1 Java EE**

- Java is a platform independent language which can be run on any operating system thus allowing us to run website of a Linux machine.
- Contains multiple libraries that can be used that can provide functionality to the system.



### **7.1.2 JSF - Java Server Faces**

- Allows Component based user Interfaces.
- Gives the ability to create stateless views, page flows and portable resource contracts

### **7.1.3 JPA - Java Persistence API**

- Describes how relational data is managed.
- Supports embedded objects.
- Allows shared objects to be cached.

### **7.1.4 JPQL - Java Persistence Query Language**

- Structured query language is used to query the database.
- Allows the ability to get very specific data and perform operations on returned data by using query language.
- Provides the functionality of query caching.

### **7.1.5 Maven**

- Uses Project Object Model.
- Allows project documentation and reporting to be managed by central piece of data .
- Provides build environment.

### **7.1.6 AJAX - Asynchronous JavaScript and XML**

- JavaScript
  - Provides the functionality to edit the Document Object Model(DOM).
  - JQuery will be included as a JavaScript library, allowing the use of extended functions and methods.
- XML - eXtensible Mark-up Language

- Platform independent.
- was created to store and transfer stored data.
- Easily parsing.

## **7.2 Web technologies**

### **7.2.1 HTML - Hypertext Mark-up Language/XHTML - eXtensible Hypertext Mark-up Language**

- Standard web language.
- Easy to write pages.

### **7.2.2 CSS - Cascading Style Sheet**

- Used to provide styling to HTML/XHTML web pages.