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)

February 2015

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

2.2.1 Scalability

- Manage resource demand.
- Scale out resources.
- 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 Performance requirements

- Throughput: The rate at which incoming requests are completed.
- Manage resource demand.
- Minimize response time.

2.2.3 Maintainability

- Regression: Ability to backtrack to a state in which system was safe.
- Corrective maintenance: Reactive modification of the system performed after delivery to correct discovered problems.
- Adaptive maintenance: Modification of the Buzz system performed after delivery to keep the system usable in a changed or changing environment.
- Perfective maintenance: Modification of the Buzz system after delivery to improve performance or maintainability.
- Preventive maintenance: Modification of the Buzz system after delivery to detect and correct latent faults in the system before they become effective faults.

2.2.4 Availability

- Prevent faults.
- Detect faults.
- Recover from faults.
- Cross platform functionality.
- Cross browser functionality.

2.2.5 Reliability

- Prevent faults.
- Detect faults.
- Recover from faults.

2.2.6 Security

- Detect attacks from unwanted and unauthorised users.
- Resist attacks from unwanted and unauthorised users.
- Recover from attack from unwanted and unauthorised users.
- 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.
- All system functionality is only accessible to users who can be successfully authenticated through the LDAP system used by the department of Computer Science.

2.2.7 Monitorability and Auditability

- Logs system activities such as the time a user logged into/out of the system
- 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.

2.2.8 Testability

- Controllability: The degree to which it is possible to control the state of the component under test as required for testing.
- Understandability: The degree to which the component under test is documented or self-explaining.
- 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.

2.2.9 Usability

- Efficiency
- Ease of use
- Easy to navigate between the system's various web pages.
- Satisfaction (How pleasant is it to use the system?)
- The average student should be able to use the system without any prior training.
- 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.

2.2.10 Integrability

- 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.

2.3 Integration and access channel requirements

2.3.1 Human access channels

- Must run on all major web browsers (Firefox, Internet Explorer, Opera, Chrome, Safari)
- Must be able to access from computer (desktop & laptop), tablets and phones (i.e. mobile/Android devices)
- Must be able to able to handle thick clients (like a student's home PC) and thin clients (like the PC's in the Informatorium)
- Must support all major web standards such as the standards published by the International Organization for Standardization (ISO), Request for Comments (RFC), Document Object Models (DOM), proper use of HTTP, stylesheets (especially Cascading Style Sheets (CSS)) and markup languages, such as Hypertext Markup Language (HTML) and Extensible Hypertext Markup Language (XHTML).

2.3.2 System access and integration 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 be integrated seamlessly with both the CS LDAP server (to retrieve class lists and student information) and the CS MySQL database to access course and module details.

2.4 Architectural Constraints

The following architecture constraints have been introduced largely for maintainability reasons:

1. The system must be developed using the following technologies
    - The system must be developed using the Django web framework.
    - Persistence to a relational database must be done using the Object-Relational Mapper bundled with Django.
    - The unit tests should be developed using the Django unittest module.

2. The system must ultimately be deployed onto a Django application server running within the cs.up.ac.za Apache web server.

3. The system must be decoupled from the choice of database. The system will use the 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.

Alternative Architectural Constraints:

1. The system must be developed using the following technologies

    - 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 will use ISO OSI layers and its communication protocols. For example,when accessing the Buzz space the user opens the web browser and use HTTP to directly request the Buzz space web page. This happens on the application layer.
- Layered architectural style supports N-tier architectural style which can improve scalability.
- 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 which improves maintainability of the system.
  - it improves performance and fault tolerance.
  - it increases testability.
- 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
  2. Business logic layer - Provides the back-end services
  3. Infrastructure layer - Includes the framework upon which the system is built and provides integration with other systems

3.2 Virtual Machine Pattern *

- The Buzz system must be portable to run on any computing platform.That is, the system must be able to run on home desktop, PCs on campus, and different mobile devices. It must also run on different web browsers on different operating systems.
- Virtual machine pattern provides:

  - portability
  - pluggability

3.3 Client/Server Architectural Style

- Client/Server allows multiple clients to access the system using N-tier architectural style. The benefit of using N-tier architectural style is that it improves the scalability of the system.
- The server side is the back-end of the system which manages authentication via communication with CS LDAP, access to the persistence database, and the other background services. The aim of having the back-end managing the data is to achieve higher security.
- The client side for the Buzz system will be a thin client in the form of a web interface and will simply handle input from the user and display appropriate output.

3.4 N-tier/3-tier Architectural Style *

- The N-tier/3-tier architectural style provide improved:
  - scalability
  - availability
  - maintainability
- And resource utilization

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 confirms 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 where working correctly.

4.3 Fault Prevention

- 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.

- 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.

- Process Monitor
  Monitors the state of health of a system process and ensures that the system is operating within its nominal operating parameters.

5 Use of reference architectures and frameworks

6 Access and integration channels

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

- Standard web language.
- Easy to write pages.

6.2.2 PHP

- Allows dynamic pages to be built.
- Easy integration of JavaScript and HTML with PHP functions.

6.2.3 IP - Internet Protocol

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

6.2.4 SMTP - Simple Mail Transfer Protocol

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

6.2.5 TSL - Transport Layer Security

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

7 Technologies

7.1 Programming technologies

- AJAX (Asynchronous JavaScript and XML).
- The JQuery library which will be used with our JavaScript.
- Python (used in the Django Framework).
- Java programming language

7.2 Web technologies

- HTML

7.3 Other technologies

-