Kalafong Provincial Tertiary Hospital

Gynaecological Patient Information management System:

Architectural Requirements

Team Pentec:

Ruth Ojo 12042804
Liz Joseph 10075268
Trevor Austin 11310856
Maria Qumayo 29461775
Lindelo Mapumulo 12002862

Pentec

Final Version

August 28, 2015

Contents

1 Introduction

2 Architectural requirements

2.1 Architectural Scope

2.1.1 Persistence

2.1.2 Reporting

2.1.3 Process execution

2.2 Access and integration requirements

2.2.1 Human access channels

Human Access Channels are all the various ways a user may interact with and access the PIMS.

1. Mobile Phone:
    - This mode of access will allow for better mobility and portability as the user can fill in information as they perform procedures. The user will have to use their own data to access the system.
2. Tablet:
    - This mode of access, same as the mobile phone, will also for mobility and portability.
3. Desktop Computer:
    - This will be the least common way for the user to access the PIMS, as the Hospital does not have a centralized computer.
4. Laptop:
    - This mode of access is more portable than the desktop computer.
    - The user will have to make use of a modem to access the Internet as the Hospital does not Wi-Fi access.

2.2.2 System access channels

System Access Channels are the means by which other systems will access the services offered by the Kalafong PIMS. At the moment, no system access channels are required.

2.2.3 Integration channels

The Kalafong PIMS will need to be integrated with:

- The NHLS website in order to retrieve diagnostic codes form the website
    - This functionality is to be executed at a later stage, outside the realms of the project
- The University of Pretoria website by linking the two websites

- The existing departmental Microsoft Access database
- The individual datasets that will be part of the system

2.3 Quality requirements

Critical Quality Requirements

2.3.1 Security

Description

This is the most critical requirement for the patient information management system. Patient's information should not only be confidential, but levels of user authorisation must exist. The existing information should not be modifiable by users that have no access to the information; this also applies to outside intruders.

Justification

The whole system should not be penetrable by an intruder; the general public should not have access to any of the information about the patients. The system should also make sure that each user can only access the attributes that applies to them.

Mechanism

1. Strategy:

    - Encryption: Patient data is confidential and should be encrypted. Not every patient has to be encrypted, but the data should not imply to which patient it belongs to. The patient's personal information could be encrypted instead of all the attributes.
    - Authorization: A user's access level and identity needs to be determined before they can access the system.
    - Store log information: Although this applies to audibility, it helps to know the nature of a security threat. If new threats are noted, more security features can be implemented.

2.3.2 Usability

Description

This ensures that a user will be able to use the system, with ease. The system should provide support to the user.

Justification

Patient information management system is user-oriented. How the users interact with the system is a critical, and this should be done with little to no effort. The system should appear easy to use and should not, at any point, baffle the users.

Mechanism

1. Strategy:

   - A tutorial on how the patient information management system works. A user can be initiated into the system, the first time they use it. Or they can enable the tutorial until they're familiar with the functionality.
   - Enable the user to troubleshoot their problems. Frequently asked questions or frequent problems could assist with this aspect. A user will be provided with predefined help options such that they will not need to contact the system's administrator, for assistance.
   - Provide descriptive headings that make navigation easier. Headings should not be ambiguous. A user should know what to expect when they select a certain heading.
   - Error signals should be displayed to the user, if some user-inflicted error occurs. The necessary steps to rectify this problem must be provided.
   - A user should be able to undo their action, should they be aware of their mistake.

2. Architectural Pattern(s):

   - Model-View-Controller: This separates the user interface from the rest of the system (Bass and John). A user should only interact with a simple interface that was designed for them. This is describable for patient information management system because the users don't necessary have an adept understanding of the lower levels of the system.

2.3.3 Scalability

Description

Scalability is an essential aspect of a system and is the ability of a system to be easily enlarged in order to accommodate a growing amount of work.

Justification

The PIMS should allow for hundreds of concurrent users, as such the system must be able to handle such a number without breaking down or reducing performance.

Mechanism

1. Strategy:

   - Clustering: using more resources by running many instances of the application over a cluster of servers or instances, to ensure system resources are not strained by a high workload.
   - Efficient use of storage: data storage can be efficiently used through compression of the data (reducing data size to make room for more) paging (ensuring that primary storage is used only for more crucial data) as well as de-fragmentation (organizing the data into continuous fragments and free more storage space). by ensuring that no data duplications occur, storage space can be conserved, thus the load on system resources will be reduced.
   - Efficient persistence: through indexing and query optimization, the amount of system power used to persist a database will be reduced, as data retrieval will be quicker and costly queries will be done without, thus also reducing system load. In addition, connections can be grouped and accessed via a central channel in order to aid persistent storage to the database.
   - Load Balancing: by spreading the systems load across time or across resources the load on the system can be distributed, therefore no system resource will be heavily strained. In the case that the limit for a server has been reached, a new instance or so will have to be created in order to handle the number of increasing

requests. On the other hand, if the usage of a server is way below the capacity, the number of instances will have to be reduced.

- Caching: to ensure no duplication or repeated retrieval of frequent objects or queries; a separate module can facilitate caching; thus system resources will not be used up unnecessarily.

2.3.4 Reliability and Availability

Description

The PIMS should be accessible at almost all times, in particular during the peak operational hours of the hospital. This accessibility will be limited to the hospital network only, so as to ensure no information can be changed without approval.

Justification

The reliability and availability requirements are very important seeing as the information to be kept on the system is highly valuable to the medical staff, as the need up-to-date information concerning the patients they are dealing with (lives could be at risk). With this in mind, only a downtime of less than 2 hours, at most twice a month will be allowed, so as too allow for the medical staff to maximally use the system. A high reliability rate is recommended to ensure that users do not encounter any errors and/or data corruption in their use of the system. The only leeway that will be given for errors, is to have at most one.

Mechanism

1. Strategy:

    - Clustering: using more resources by running many instances of the application over a cluster of servers or instances; therefore if any server should fail, the reliability and availability of the system will not be compromised.
    - For reading from and writing to the database, we will ensure that no parallel updates are possible through enforcing the use of a single object to stream all database transactions; thus reducing inaccuracy that would be a result of data redundancy.
    - Use of more resources: This would heavily reduce system downtime, as a temporary server can be run while the other is maintained.

2.3.5 Integrability

Description:

The Patient Information Management System should be integrated with ease to the Universities website as well as the NHSA web site.

Justification:

The Patient Information Management System is primarily designed for the kalafong medical staff and is required to be accessable on both the University's website and the NHSA website. For security perposes we limit it's intergaration to just these two sites.

Mechanism:

1. Strategy:
   The contracts based decoupling strategy is used to implement integrability and extensibility on the system by providing sections and different functionality between the back end system and the host site.

2.4 Architectural responsibilities

2.5 Architecture constraints

Important Quality Requirements

2.5.1 Maintainability

Description

The PIMS should be easily Maintainable in future. Thus needs to be flexible and extensible.

Justification

Software always needs new features or bug fixes. Maintainable software is easy to extend and fix, which encourages the software's uptake and use.

Mechanism

1. Strategy:

- Open-source resources: using open source resources to minimize update costs in the future.
- Iterative development and regular reviews: This will help us improve system quality.
- Prevention is better than cure: Here we get others(lecturers) to review your code, to make sure its clean.The cleaner our code, the cheapre it is to maintain.
- Version control:Thsi will help keep our code, tests and documentation up to date and synchronised. It will also help us keep track of progress.
- Documentation: Relevant documentation will help future developers understand the software and system as a whole.

2.5.2 Monitorability

Description

Audititabily or Monitor-ability is a software review where one or more auditors/monitors who are not members of the software development and organization team conduct "An independent examination of the software product to assess compliance with specifications, standards, contractual agreements, functional requirements and other criteria according to the development specification. Software Monitor-ability and audiability is different from testing or peer reviews because they are done by personnel external to and independent of the software development organization.

Justification

Although it is important for the PIMS to be monitorable, it is not crucial. It may be Monitored because of the following:

- Multiple and concurrent users, thus making it prone to various malfunctions.
- Form filling and Statistical quering needs to be done in real time at all times to ensure relevance of topics and subject matters.
- Sufficient feedback and updates of the site's state must be provided to the users.
- General software control and application usage..

Mechanism

1. Strategy:

   Auditability and monitor-ability will be achieved by allowing any third party software auditor or monitor support group reviewing the Buzz Space examining it specifically for the aspects of the functional requirements as given by the client. Information such as the systems state and processes in complaints with the specification given. One such auditor may be from a well-established organisation, for example Oracles PeopleSoft enterprise which is UPs current used application.

2. Pattern/Tools:

   Tools like SMaRT, a workbench for reporting the monitor-ability of Service Level Agreements for software services such as Buzz Space may be used. SMaRT aims to clearly identify the service level the service level commitments established between service requesters and providers. This monitoring infrastructure can be used with mechanical support groups in the form of a SMaRT Workbench Eclipse IDE plug-in for reporting on the monitor-ability of Service Level Agreements.

2.5.3 Testability

Description

Testability is a measure of how well system or components allow you to create test criteria and execute tests to determine if the criteria (pre and post conditions)are met. Testability allows faults in a system to be isolated in a timely and effective manner.

Justification

It is extremely important to conduct test cases for every component that will be intergrated or incorporated into PIMS. This ensures consistency in the system,and enables faults and/or loopholes to be picked up and resolved in good time.

Mechanism

1. Strategy:

- Unit testing: Unit testing and integration testing will be conducted using moch and unit.js
- White-box tests internal structures or workings of an application. This requires the explicit knowledge of the internal workings of the PIMS.

2. Pattern:

- Layering: Simplify testability since high level issues will be separated from low level issues. This level of granularity makes the system to be easily testable on every layer separately.
- Model View Controller: The PIMS model will be separate from the view and the controller, so that it is simpler to have separate test criteria testing different cases. This separation simplifies the development cycle since the model, view or the controller could be tested independently.

2.5.4 Performance

3 Architectural Patterns and Styles

3.1 Model-View-Controller (MVC)

Model-View-Controller is an architectural pattern that divides software applications into three interconnected parts.

The controller is responsible for translating requests in to responses (Nadel, 2012). The controller 'inserts' the response back into the view, which could be html or it's equivalents.

The view translates the response, from the controller, into a visual format for the client (Nadel, 2012). The view also sends requests to the controller.

The model's task is to maintain state and give methods for changing this state. Typically, this layer can be decomposed to:

- Service layer - provides high-level logic for dependent parts of an application.
- Data access layer - services objects invoke this layer, which provides provides access to the persistence layer.

- Value object layer - provides data-oriented representations of terminal nodes in the model hierarchy.

3.2 Technologies

3.3 Platform

3.3.1 Node.js

The system will be deployed in a Node.js environment. Node.js allows for queued inputs and since the system revolves around multiple inputs possibly going through at any particular time this provides a huge advantage over most other systems. Node.js also includes a wide range of modules through NPM (Node package manager). Some examples of these include Express, AngularJS, mongo, mongoose, and many more. Node.js processes programs asynchronously and thus is a no-interrupt driven language. This is as advantage because, as stated above, it allows for queued inputs. Node.js is written in JavaScript which means it is ubiquitous which is an obvious advantage.

Node.js relies heavily on callbacks to allow for synchronization which is an obvious hindrance for programmers that are not strong with recursion or callbacks. Although this con is outweighed by the advantages of the system.

3.4 Framework

3.4.1 Express

Express is a lightweight, high performance HTTP which supports URL configurable routing. This allows for a more professional look as well as improvements in maintainability and flexibility.

3.5 AngularJS

Angular is front-side development framework that applies the MVC architecture pattern. Angular speeds up client-side templating and allows for the programmers to write less code. Angular allows for dependency injection and is unit testing ready. This is a far better means than the traditional way of testing web applications by creating individual test pages that petition one component. This aids in usability, maintainability and performance.

3.6 Broadway

This framework creates a more flexible system as it allows the client to possibly implement further plug-ins to the web application.

3.7 Crypto

Crypto is an npm module that allows for multiple forms of encryption. It is lightweight and provides much needed security for the web application. It offers a way of encapsulating secure credentials over a secure HTTPS or HTTP connection.

3.8 Express-Validator

This npm package is an Express server middle-ware for the npm validator module. It is used for the purposes of form validation, particularly when making a post request to the server. This technology aids Usability, as it allows for the application to give feedback to a user concerning an input made.

3.9 Node-mailer

This npm package is for the notifications component of the system, allowing for the sending of email notifications to patients.

3.10 Database

- MongoDB - is a NoSQL document store. MongoDB is used especially for applications that need store large volumes of data, which is mandatory for PIMS.
  MongoDB ensures that scalability, a critical requirement, is enforced. This is due to the highly cache-able persistence environment. Concurrency is also handled via the locking mechanisms. Multiple read access and a single write operation is allowed.

3.11 Unit Testing

3.11.1 Mocha

- Mocha was selected as our primary unit testing framework running for Node.js and the browser, to assist in the simplification of asynchronous testing.

Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. We found that Mocha offered browser support which proves useful for our project as we are testing our application on different browsers and a javascript API for running tests which assisted us in identifying our errors more visibly and made the framework much easier, both, to understand and use. Mocha is also very popular and so there is an online community of users offering assistance and suggestions, it is also very configurable. Mocha assisted us to describe our test suites.

3.11.2 Chai

- We selected Chai as it is an assertion and expecting library for node and the browser given that we are creating a web application and using javascript as our testing framework; Chai pairs them both very well. Chai also works well with Mocha and other unit testing frameworks as we had decided to use more than one to achieve readability for the output of our unit testing program. Chai assisted us to perform all kinds of assertions against our javascript code.

3.11.3 Should

- Should is an expressive, readable, test framework agnostic, assertion library
  The main goals of the Should library is to be expressive and to be helpful. It helped keep our test code clean, and add meaning to error messages.

3.12 Object Data Model

- Mongoose - is an object modelling environment for MongoDB and Node.js. It enforces contracts and structure via validation, provides connection pooling, which improves scalability. Automatic object to document mappings is also supported.

3.13 Templating Engine

Jade: The HTML code will be generated by the template engine Jade. This allows for minimal code and speeds up the entire process of writing HTML pages. Jade improves maintainability as the views are more readable and writeable.

3.14 Application Servers

- TBA

3.15 Web front-end

The web front-end will make use of the following technologies:

- Bootstrap
- HTML
- CSS

3.16 Operating System

- Windows
- Possibly Android OS

3.17 Dependency Management

- NPM
  - This will be used to build the project and ensure that any component is not IDE-specific
  - The details pertaining to each of the packages are within a package.json file

3.18 Web Services

- SOAP-based

3.19 APIs

3.20 Other

- Heroku - A cloud based application platform.

3.21 Tactics and Strategies

3.21.1 Scalability and Performance

- Optimize repeated processes.
- Reuse resources and results.
- Reduce contention by replicating frequently used resources.
- Clustering.
- Efficient use of storage.
- Load Balancing.
- Caching.
- Use REST (Representational State Transfer) to make BuzzSpace into a scalable web service.

3.21.2 Security

- Authenticate users by requesting username and password when interaction with the system begins.
- Authorize users checking if a user has the rights to access and modify either data or services.
- Encryption to maintain confidentiality of data.
- Input Validation to detect malicious attacks.
- Auditing and logging for identifying and recovering from attacks.

3.21.3 Usability

- Usability is enhanced by giving the user feedback as to what the system is doing.
- Descriptive Error messages must be provide along with the necessary steps to address the errors.
- System must respond to actions performed by the user.
- Separate the user interface from the rest of the application using Model-View-Controller.

3.21.4 Integrability

- Use modular programming to modularize the system.
- Use REST (Representational State Transfer) to decouple BuzzSpace from other software that may need its services.

3.21.5 Maintainability

- Use modular programming to modularize the system.
- Use object-oriented programming to sub divide the sub-system features.

3.21.6 Monitor-ability

- Monitor-ability can be enhanced by using fault detection tactics:
  1. Ping/echo: One component issues a ping and expects to receive back an echo, within a predefined time, from the component under scrutiny.
  2. Heartbeat: One component emits a message periodically and another component listens for it. If the heartbeat fails, the originating component is assumed to have failed and a fault correction component is notified.
  3. Exceptions: These are raised when an anomaly in a component occurs, encounter an exception when a fault is detected.

3.21.7 Reliability

- Reliability can be enhanced by using fault recovery and preventions tactics:
  1. Active redundancy: All redundant components respond to events concurrently. All redundant components will have the same state. When a fault occurs in the responding component, the system will switch to the next redundant component, minimizing downtime.
  2. Checkpoint/rollback: the states of the components will be recorded periodically or in response to certain events. When a fault occurs, the system should be restored to the previously consistent state or checkpoint.
  3. Transactions: The system should bundle several sequential steps in such a way that the entire bundle can be undone at once.

3.21.8 Testability

- Enhanced testability by recording the information that enters the system and using it as input into the test harness, and recording the output of the system components.
- Separating the interface from the implementation to enable substitution of implementations for various testing purposes.
- Creating a specialized testing interfaces to capture variable values to a system component and also seeing the output of the component in order to detect faults.
- The components can maintain useful information regarding its execution internally and then be viewed in the testing interface.