Methodology for the Development and Deployment of a Hospital Management Information System using Free Open Source Software

M. Tariq Badsha
SEECS, NUST
tbadsha@yahoo.com

Abstract

The paper examines the methodology for developing a Hospital Management Information System (HMIS) on Free Open Source Software (FOSS) and identifies the areas of improvement in the design and methodology. In addition to the methodology the paper presents the broad contours of the software solution and the extent to which existing FOSS components have been used. The paper also provides the rationale and benefits of adopting an in-house development model using FOSS. While the target implementation is an HMIS, the general requirements are no different than a Management Information System (MIS) for a small to medium size enterprise, and thus these enterprises can migrate to FOSS based applications quite economically by adopting the model presented in this paper. With the recommendations made in this paper a more robust application can be delivered, which will also be more conducive for adoption by FOSS developer community.

1. Introduction

Free Open Source Software (FOSS) applications are affordable, reusable, and provide flexibility to organizations in customizing them to suit their needs. In 2003, while FOSS technology was not as mature as today a public sector hospital in Ireland experienced a cost saving of 13M Euros over 5 yeas by migrating to Open Source [13]. More recently a web application ”EpiVue” was developed in the state of Washington, USA by integrating open source technologies and public health data to create a web information system for public health application [20]. In the regional context Philippines has developed a generic public health software on FOSS, that can be customized and linked with various systems [19]. Unfortunately, the adoption of FOSS has been slow in Pakistan even though the Federal Government set up an Open Source Resource Center [6] in 2003, to provide training in FOSS technologies and assist organizations in migrating to FOSS, yet only 24 organizations have taken advantage of this program in migrating from proprietary systems to FOSS.

The migration of Northwest General Hospital (NWGH) from proprietary to FOSS technology offers a good case study as to the viability of migrating to FOSS. NWGH is a 200-bed private hospital, which offers outpatient as well inpatient healthcare services. The hospital is currently a post-graduate teaching facility, medical and nursing colleges are also planned to be added in the near future. The hospital had contracted the development of a Hospital Management Information System (HMIS) to a small private company, which developed and deployed the HMIS in the hospital but it was fraught with several problems and limitations. The hospital, after evaluating several options, decided to develop a new HMIS based on Open Source Technology.

The paper discusses the reasons that motivated the hospital to adopt FOSS and then briefly outlines the architecture and design of the HMIS. This is followed by a discussion on the methodology and other aspects of implementation. The paper then discusses the findings on the design, methodology, and other organizational factors. As explained in Section 8 the discussion on the migration of NWGH is equally relevant to any small or medium enterprise. The discussion on the methodology, while identifying the gaps, also highlights that FOSS solutions can be deployed at a relatively low cost. The paper serves two major purposes. One is to help organizations interested in migrating from proprietary technology to Open Source; second, to improve the process of migration, thereby improving the quality and consistency of such initiatives.

2. Problem statement and solution alternatives

The hospital had already invested a fair amount of time and money into the development of an HMIS. The operational data for nearly a year was also captured in the format specific to that system. Since the deployed solution was falling short in various areas, the hospital had to either improve this solution to an acceptable level or deploy a new one. The next section discusses the various options available to the hospital.

2.1. Improve the system in use

The hospital carried out a third party review of the existing solution to determine if it could be salvaged. The review included an examination of the system interfaces and functionality as well as interviews with the users. The process revealed that the software design was very weak and parameters which should have been configurable, were coded in the software. The software also lacked several features required by the end users. Hence, improving the existing solution would have required a redesign of the software. Since the source code was not provided to the hospital, the redesign would have to be done by the same vendor who had developed it in the first place. Subsequently, the hospital would be captive to the vendor for support and maintenance. The vendor, in turn, had to rely on a single developer who had worked on the software. This was an extremely risky proposition and was rejected.

2.2. Customize other existing applications

The development team searched for FOSS packages that could be modified to meet the requirements. Two packages, Java Hospital Information System (JHIS) [3] and arpitsoft [1], were studied in detail but it was found that the business processes of the hospital did not mesh with these packages. For instance, in JHIS there were significant differences in the inventory management, accounting, patient registration and diagnostics. JHIS also lacked functionality in the area of patient care, Operation Theatre scheduling, and other in-patient handling functions. The HMIS by arpitsoft was geared more towards family medicine and primary healthcare in the context of the socio-economic environment of a developed country. For instance, arpitsoft includes modules for housing conditions, substance abuse, and education background. Other features which were not of interest to NWGH included focus on genetics, hereditary risk factor, specific standards of diseases and medical records. It would have taken more effort to retrofit these packages than developing the business logic from scratch. As pointed out in [12], there are several differences in the financial, technological and human resources between the South and North, which make it impractical to directly adopt IT solutions that were designed and tested in developed countries. The social welfare system, health insurance, patient privacy laws, malpractice laws are just a few such differences that make the business process in Pakistan different. Commercial solutions that are highly configurable are not only extremely expensive but also very complex for adoption by a small or midsize business.

2.3. Outsource development

The other option was to outsource the development of a new system. There are only 8 IT companies in Peshawar and all of them are small companies with very limited organizational depth [7]. Such companies are very sensitive to personnel changes, and in most cases even the resignation of a single employee can seriously impact the quality of service. Thus it was too risky to outsource the work to a local company. On the other hand, the financial impact of outsourcing the work to a major IT company, not present locally, was prohibitive.

2.4. In-house development

Hence the only viable option was to develop the solution in-house. The issue of organizational depth as mentioned in the context of small IT companies was also applicable to this approach because the in-house team of developers was small and any changes in personnel could severely impact the support of the system. Hence, it was decided to develop the solution in open source and build a FOSS development community around it. While Hill has presented a case that Open Source development should not be funded [15], there are strong counter-arguments that in order to jump start the development process a core development team of paid developers is required [11]. Once the system is developed then an opportunity to work on a real life application would give impetus to senior students to get involved in Open Source development and at the same time provide hedging to the hospital against changes in the core development team. It is envisioned that the requirements of the hospital would continue to evolve over the next few years and thus it would require several releases before a finally accepted solution emerges. An in-house development team along with a community of FOSS developers would be required to support this maturity phase.

3. Proposed Solution

The scope of the solution and the constraints have a direct bearing on the methodology. It is, therefore, pertinent to discuss the features of the proposed solution and the constraints of time, budget and human resource under which the solution had to be delivered.

3.1. Schedule

Initial consultations on the replacement or revamping of the MIS began in the fourth quarter of 2009. The hospital wanted to have a replacement within 6 months, as they were paying maintenance on a product that was likely to be discontinued. The hospital management had to be convinced that 6 months is not a feasible target and a more realistic target would be 9 -12 months from the start of activities. In the first quarter of 2010 the first member of the development team was hired and the development process formally kicked off.

3.2. Budget and human resource

The reference for the development cost was the maintenance cost of the existing proprietary system. As the hospital was already incurring the cost for the IT team, comprising of a senior developer and 4 network support personnel, the incremental cost of the development was the hiring of additional developers. This cost was to be kept within the range of the maintenance cost. Hence, it was envisioned that 2-3 developers would be hired for the project. The existing IT Manager was to lead the team. In addition, the 4 network/system support personnel at NWGH were to assist the team. The Open Source Resource Center [6] and an external consultant were to provide advice and general guidance.

3.3. Scope

The MIS is the complete software solution for all the administrative and patient care functions of the hospital. The application consists of 9 major modules:

1. Front-desk, which is the module for registering a patient and scheduling appointments. This module creates the patient record, which is the main pivot around which the work flow links other records.

2. Pathology, which deals with the scheduling of lab tests and links the results of the lab tests to the patient record.

3. Radiology, has pretty much the same functions as the Pathology module, except that it handles CT-scan, MRI, X-Ray, and Ultrasound. Currently the diagnostic machines are not connected directly to the system and data has to be manually entered from the print-outs. This limitation is due to budgetary constraints which prohibited the purchase of the optional software available from the equipment manufacturers.

4. Admissions, for administering the admission and discharge of in-patients.

5. In-Patients, maintains the record of the patient care throughout the treatment, including physician reports, daily charts, diagnostics, blood requests, Pre-OP and Post-OP procedures, medication and follow-up instructions.

6. Blood Bank, maintains a database of the donors, screening history, blood stock inventory, and all the functions for the management of the Blood Bank.

7. HR, which handles the personnel management of the employees of NWGH. It also has the capability to manage recruitment and keep a record of the hiring processes.

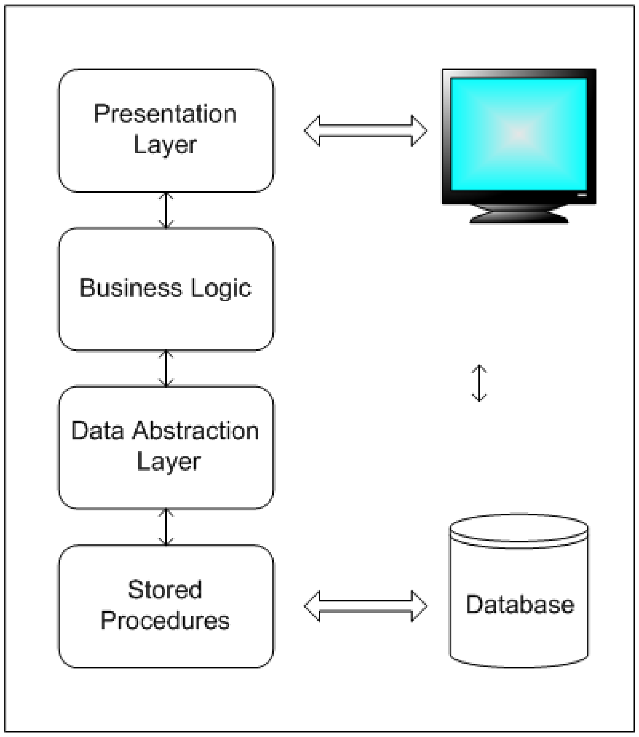

Figure 1. Layered Architecture

8. Finance Management System, takes care of all the accounting functions. Other modules, like Pharmacy, that affect the Assets or Revenue stream also update the financial information.

9. Pharmacy, is essentially an inventory and stores module for the medicines and supplies. The hospital has a Main store, which receives bulk medicine from the suppliers and Satellite stores that dispense the medicine and supplies. The module manages both these stores as well as the interaction between the two. Appropriate updates are made to the patient record as medicine and supplies are dispensed to the patients.

3.4. Design

As shown in Figure 1 the software application is based on a 4-tier architecture, with MySQL database version 4.5 at the lowest level. The front end has been developed in PHP version 5.0. A Data Abstraction Layer (DAL) is used between the Business logic and the Database. The Presentation and Business logic form the other two layers but the segregation between the two layers is not concrete as, in some situations, the Business Logic Layer has been coded within the same pages as the Presentation Layer. The Data Abstraction Layer has been included through include tags and has not been compiled as a separate DLL. The application has been developed in the Netbeans Integrated Development Environment [5] with the use of libraries and code segments from phpbuilder [8], phpclasses [9], jquery [4], and ajax [2].

The solution could very well be implemented in one database but due to the fact that the hospital has small servers, it was decided to have five separate databases so that if the database size grows then it can be distributed over different machines. The business logic is supported by the following four databases, while a separate database is maintained for logging:

1. Database of Patient Records
2. Accounting Data
3. Human Resource Data
4. Pharmacy

There are a total number of 180 tables spread over the five databases. Each data table is supported with five stored procedures to perform CRUD (Create, Read, Update and Delete) operations. Each table is further supported with the Primary key generation table.

For all errors and exceptions the same page format is displayed with the error specific information, providing the users a consistent view. Moreover a Java script dialogue box is also generated to prompt the user for additional information or comments.

3.5. Security

The connection to the database is established via DAL procedures through a connection string. Since it is a single deployment environment, the connection string is currently hard coded with the host name, user name and password.

The authentication mechanism for the application is done at two levels. First, the user gets authenticated as a valid Database User. The record for the database users is kept separately in an encrypted XML file that is maintained through a separate stand-alone utility developed in-house. A second layer of security is implemented through database tables, where roles and group membership of each user are defined. Each front end form is defined as an Activity with access rights like Add, Delete, Update and Print defined for each user. The access rights of tables are maintained through an administrative interface of the application.

4. Methodology adopted

There is a fair amount of literature on the recommended methodologies for Open Source Projects but as pointed out in [17], these methodologies are applicable to projects where a diverse community of developers, typically dispersed geographically, participate in the process. As teams become smaller and co-located, then the standard development methodologies are applicable. In case of the development at NWGH even the standard methodology had to be adjusted due to the small size of the team and the constraints of time. In Section 8 we will discuss what risks are introduced by deviating from the methodology. As a reference methodology SCRUM [10] was used. The project started off with one Project Manager and one developer only. The Project Manager also worked as the lead developer. Another developer was hired but quit shortly thereafter and it was only towards the tail end of the project that another developer was taken on board. Thus, for most part of the project there were two resources for code design and development. We will examine the methodology and the role of the other actors that supported the team.

In SCRUM methodology the role of Product Owner is to decide which features are to be incorporated in the release. This role is typically weak in most IT projects in Pakistan. Part of the problem is in the hierarchical thinking of organizations. The role of the Product Owner is not explicitly defined and even when a focal person is defined, he/she is assumed to be more of a conduit for user requirements rather than someone who can make a qualitative judgment. The executive authority makes the decisions but does not have enough background in IT to evaluate the cost/benefit/risk of including certain features. The CEO of the hospital took personal ownership of the project but, being a medical doctor, lacked the technical IT skills. The problem was solved by forming a virtual Product Owner with the CEO and the external consultant working as a team. For this model to work the executive authority must have confidence and trust in the person providing technical guidance. The consultant held a joint meeting, once a month, with the user representatives, executives, and the IT team. In addition to that, the CEO and the IT team discussed issues with the consultant by phone. This level of interaction was generally adequate but the absence of a full time Product Owner was felt towards the tail end of the project when time was getting tight and departments were forwarding requests for changes in the software. Those users who had very little exposure to IT systems would remain focused on the functionality they required but those users who had some experience with IT went beyond the domain of functionality and proposed changes to design, thereby, causing reworks and delays. In the presence of a full time Product Owner these problems would be resolved as and when they arose.

The system requirements were not documented in a single requirements document but flow charts of the major processes were kept in flat files. The database tables were also documented in spreadsheets in flat files. These files served the purpose that the developer knew what to develop and the tester could use it as a reference, however, the level of detail was not enough for an independent tester to do testing. As the team was very small, the developer and tester used their general understanding of the user requirements to augment the lack of detail.

The existing system was very useful in generating system requirements. Whilst it lacked several features and was not scalable, it provided the baseline requirements. It served the purpose of a prototype and allowed users to provide feedback on the look and feel as well as the features. The shortcomings of the existing system helped directly in formulating the requirements, as the support team was instructed to convey all the trouble tickets to the development team. The development team examined the trouble tickets to separate those that were bugs in the system as opposed to those that were either missing features or configuration limitations. The data from the trouble reports along with the users' wish lists were added to the requirements of the new system. Similarly, in case of reports, the existing reports served as the baseline and it was easier to get user input on the desired reports in the context of the existing reports.

The concept of Change Control Committee was introduced to NWGH management, and the danger of unabated changes to the system was realized by the hospital management. A Change Control Committee was set up but was not operational in the development phase of the project.

5. Testing and Evaluation

The application was developed in modules as listed in Section 3.3. Each module was put through three levels of testing. The Unit and Functional Testing was done by the developer, while the Integration testing was done by the developers as well as the users of the system. As each module passed the functional testing, it was integrated with the system on a test server. The developers would test it for stability and functionality, and fixes were made as required. Following that, the integrated release up to that point, with a test database, would be exposed to the relevant users of the hospital. The users were provided with mobile telephone numbers of the developers so that any anomalies could be reported and observed immediately.

6. User Training

We have already seen that the support team helped in identifying the requirements. The support team also helped in user training. The development team prepared training materials and trained the support engineers in the use of the new system. As the user training was basically the use of a few forms pertinent to the particular function, it did not pose much problem to the support personnel in gaining proficiency in the system and later demonstrating the same to the end users. The training was given in groups of users dealing with the particular functional module. Other than the front-desk users, who are responsible for the initial registration of patients, the other groups were comfortable in using the system after a total of 6 hours training. The front-desk group was given 10 hours of training. This could be attributed to the fact that users already had exposure to the proprietary system for nearly a year.

7. Data migration and cutover

The data tables and the data entity relationships of the proprietary database were not available to the hospital. Hence, the previous vendor was asked to provide the data in a Comma Separated Values (CSV) file or some other standard format. A small utility was written to pick the data from the CSV file and import it into the new system. The mechanism for data migration was tested and ready to be implemented but postponed till right before the cutover, at the end of the parallel run.

The data was classified in two sets. One, the data that was not expected to change on a day-to-day basis, and the other, that data which was very dynamic. For instance, the personnel data in the HR database was not expected to change in the short window, during the parallel run. Similarly, the Pharmacy database, which contained the list of drugs and supplies along with the suppliers, was also relatively static. If any changes, like the termination or joining of an employee, affecting the static data happened during the parallel run, it was to be postponed until the completion of the cutover. However, patient records and financial data were very dynamic.

The cutover from the proprietary system to the new system was to be done after a successful 2-week parallel run. The cutover plan is summarized in Figure 2, which shows three databases going through different states during the process of the cutover. The state of the database is shown in a rectangle and the processing in ovals. We start off with three databases.

1. The database of the existing system, which is in production prior to cutover and hence both the static and dynamic data is current.

2. The database of the new system, which was to go into production after the cutover. The static part of the database is current and not to be changed until the cutover is complete.

3. The third database has the same structure as the new system but contains only sample dynamic data for testing, while the static data is kept current to facilitate testing during the parallel run.

The reason that the parallel run was done with a test database of the new system was to minimize disruption to the operations. If the production database of the new system were to be used during the parallel run then it would

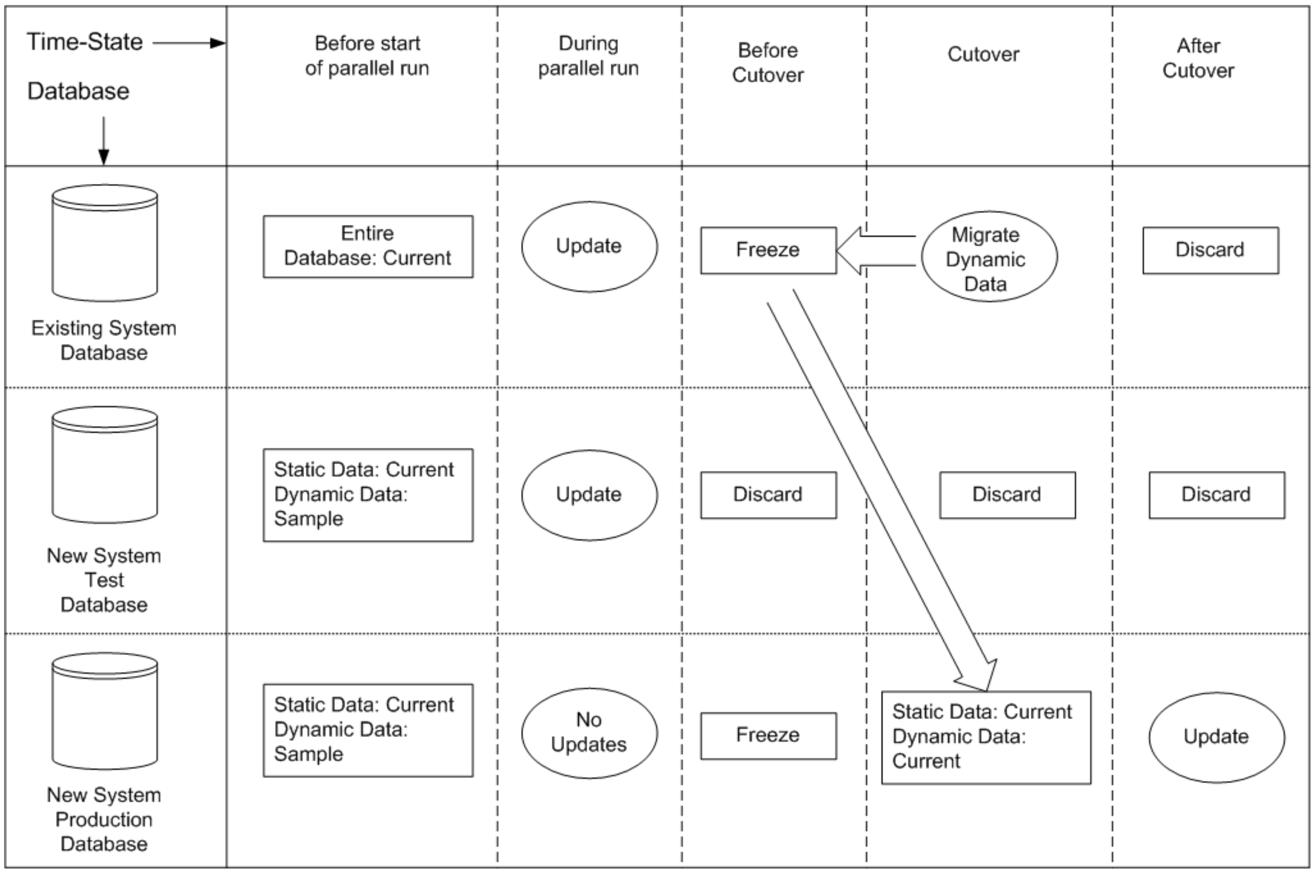

Figure 2. Data Migration for Cutover

have been mandatory that each and every update must be made to both databases. In the current scenario an unforeseen problem with the new system did not stall the operations while the problem was being fixed. After the completion of the parallel run, and just before the cutover, the production database was frozen and the dynamic part of the data was migrated to the new production database, while the test database was discarded. During the data migration no database updates were made to any of the databases. Once the cutover was complete the old production database was discarded and all further updates were made to the new system database only.

8. Findings

The challenges faced in deploying FOSS HMIS in Western Cape, South Africa [18], Mali [12], Tanzania and Zanzibar [16] are quite different and much more than those faced by NWGH. The fundamental difference is that these projects were in government hospitals which were part of a bigger healthcare system. For instance, in Zanzibar the scope included one referral hospital, three district hospitals and four cottage hospitals. Thus three tiers were involved with varying degrees of IT readiness and divergence of standards. Hence, issues like fragmentation of data, standardization of data, consensus building, and integration of heterogeneous systems had to be dealt with. In Mali, resistance to change and the language barrier for a French speaking population set additional barriers. Dealing with diversified needs of different user groups was one of the main challenges brought out in [18]. Therefore, a traditional participatory approach, as supported by [18] and proposed by [14] worked very well for NWGH. One of the barriers identified by [12], namely, the absence of a single contact for free tools, was applicable to some degree. Even though OSRC provides central services but, being a very small organization, cannot compete with the sales support provided by commercial companies.

If we compare the experience of deploying HMIS at NWGH with the challenges faced by the above referenced cases, it becomes clear that the HMIS deployment at NWGH is closer to deploying an MIS at any small to medium size enterprise, including private hospitals and clinics.

It is commendable that a very small development team delivered a workable solution in nine months. While it is understandable that under the given constraints an elaborate development methodology could not have been applied, however, certain improvements in the methodology and design are suggested.

8.1. Methodology gaps

As discussed in section 4 the development team did not rigorously follow a formal methodology. While the team has been successful in delivering a solution within the target time frame but certain gaps in the methodology created vulnerabilities in the process.

The first observation is that weakness in documentation of user requirements caused some reworks and delays - particularly in the final phases of the project. If the user requirements were documented and signed off by the respected user groups, it would have introduced some additional overhead but resulted in fewer reworks and a more robust process. Documenting and numbering the system requirements would have also improved the testing process. In the absence of a numbered list of user requirements the tester relied on frequent user feedback. This method of testing compromises the completeness of testing, in that the user is typically interested in the most frequently used features, while some unusual requirements and boundary conditions can be left out. Furthermore, as the quality of feedback varies for user to user, reliance on user feedback adds additional variance to the process.

In case of NWGH, testing was carried out by the same person who developed the module. For better quality assurance (QA) the QA function should be carried out by an independent tester, for which it is necessary to have complete documentation of the requirements. Formal QA and Test Plans were also not developed and quality metrics were not defined, which potentially compromised the quality of the software. In the absence of a formal QA function, the judgment of the developer and the user feedback determined whether the module met the required quality level. On the other hand, if specific quality metrics were defined and met then future bugs and reworks would have reduced.

The database design and data entity relationships were documented in flat files. As the team was extremely small, there was very little demand on intra-team communication and for this reason this method of documentation sufficed. However, it is recommended that system engineering tools should have been used. In the given constraints the development team neither had the time to evaluate FOSS tools nor did they have the budget to purchase the required tools. Any other small or medium enterprise that is migrating to Open Source would most likely have the same constraints. Hence, Open Source Resource Center [6] should carry out a study on the FOSS tools that are available for software engineering and maintain this information on their web site. This would encourage future migrations to be done in a more structured manner. Standard design documentation would also facilitate in creating a FOSS development community around that technology. In case of NWGH they would have to produce these documents retroactively.

The trouble ticketing system can provide usefule data on the quality the software. It is recommended that NWGH use this system to log all software problems in the production environment.

8.2. Design improvements

In terms of design, the segregation between the layers could have been more concrete. Though a DAL has been used to separate the Business logic from the Database, the implementation was done through include tags. The same could have been implemented through the use of DLLs which, being compiled code in a separate file would have provided a concrete division between the two layers and resulted in a more maintainable and optimized solution.

In the current implementation, separate stored procedures are developed for each table in the database. This has the advantage that the stored procedure is very simple and could be replicated with a simple copy and paste of the same code with a few substitutions. However, the number of stored procedures could have been reduced significantly if generic procedures were made, that would accept the table name as a parameter. The trade-off in terms of complexity would have been offset by the elegance of design and the long term maintainability of the application.

The development team attempted to reuse existing code but it is not clear if the search was exhaustive. As mentioned in section 3.4 existing code was used for database access, graphics and forms but given the large number of Open Source applications in PHP, more of the business logic could have been implemented in existing code rather than re-writing it from scratch. From a practical consideration it cannot be expected that developers operating under similar conditions could afford any more time to research the availability of existing code for reuse. It is, therefore, proposed that if code reuse is to be increased in future migrations then a central resource, like OSRC, needs to facilitate this process by keeping an index to FOSS applications that are more suitable to the local environment.

User Requirements, Project Management, and Technical Design are distinct roles. It is difficult to give due focus to each when these roles are performed by the same individual. For instance, certain design limitations may exist in the system but those would not become apparent till the load on the system is increased, the data sets increase beyond a certain threshold, or some other unusual scenario is encountered. For example, in a multi user environment with a large dataset, issues of database locking, conflicts and optimization could surface as the load and the number of users increases. With a team of two, this separation was not possible but for future Open Source implementations it is recommended that slightly larger teams should be planned in order to avoid this limitation.

8.3. Organizational factors

Organizations that have well established IT departments and have purchased software services from multi-national corporations or large local companies realize that a professional IT environment requires significant capital outlay and a proportionate annual maintenance budget. Small and Medium size businesses in Pakistan though relatively comfortable in funding hardware, particularly desktop PCs and laptops, seem to grossly underestimate the allocations for software. In part, this may be due to lack of awareness but the availability of pirated software may also be influencing this thinking. The time required for developing a bespoke solution is also underestimated because senior management of small and medium businesses are not familiar with the methodology of a full software development life cycle (SDLC). Businesses targeted by large IT companies get educated by the marketing and sales teams of the IT companies. However, smaller businesses do not have the financial promise to attract professional IT companies. Hence, such companies are exposed to individual developers or very small companies who are keen to secure the business, even if it is at a cost of cutting out major components of the SDLC. Invariably, they face problems in the long run and sometimes repeat the mistake. Small and medium business must realize that allocating realistic time and resources for software development would result in savings over the long run. While NWGH did not place a hard and fast limitation on the budget but the general understanding was to keep the development team limited to 2-3 developers.

Another major issue of small and medium enterprises is the absence of an experienced IT person in the organization as head of the IT operations and development. In small businesses the IT function could be assigned to a variety of positions, ranging from Head of Finance and Administration to the senior most network/system administrator. This arrangement works as long as the functions are limited to day-to-day operations of an established IT environment. The difficulty arises when the organization tries to move on to a new technology or to upgrade the application. Large organizations can afford to engage a consultant to plan and oversee the expansion/upgrade, smaller organizations have no option but to entrust the task to the person heading the operations, who has limited experience in SDLC and managing user expectations. Though the team lead at NWGH had exposure to SDLC but as mentioned in Section 4 the role of the Product Owner was met by a combination of the CEO and the consultant. Small enterprises should consider a similar model by having a virtual Product Owner, which would be a combination of a senior executive with authority and a seasoned technical resource person. As quick and accurate decisions are paramount for agile methodologies, it is important that the executive must trust the judgment of the technical person.

As mentioned above, in the context of documentation, several follow-up activities must be carried out before a FOSS developer community can take root around this application. The institutions offering IT education in and around Peshawar must be engaged and made aware of the benefits of getting involved in FOSS projects. Seminars and workshops must be held to increase awareness and impart training.

Notwithstanding the above mentioned improvements in the process, NWGH has been successful in migrating to open source technology with the help of an in-house team. The user requirements will continue to evolve as they gain exposure to the system and by having an in-house team the hospital can bring out periodic releases of the software. NWGH also offers post-graduate teaching and research. Hence, various reports of clinical data are required. With an in-house team they can generate a variety of customized reports from the data repository. This would not have been possible with the previous software where the hospital neither had access to the database nor the organization of data.

9. Conclusion

The implementation of a FOSS Hospital Management Information System at NWGH has demonstrated that an affordable software solution can be developed in-house for small and medium size businesses that do not have the budget to pay for expensive solutions from major companies. The same in-house development team can continue to refine the solution over successive releases in order to cater for the growing needs of the organization. Small and medium enterprises need to be sensitized to the resources required for developing and maintaining a professional application so that the development team is given adequate time and resources.

It has been observed that most small and medium size businesses in Pakistan are not in a position to articulate their IT needs without gaining exposure to an IT system. For this reason waterfall methodology does not seem to work in such situations and agile software development methodologies, where user requirements can be incorporated throughout the life cycle, are more suitable. Unlike waterfall methodology, agile software methodologies do not have a crisp boundary between requirements and downstream development, which poses an inherent risk of unending user requirements; thereby jeopardizing the completion of the project. The implementation at NWGH confirmed this assertion and also highlighted the importance of having a process that puts a cap on the user requirements in the first release.

Organization considering migration from proprietary to Open Source can benefit from the existing system, no matter how inadequate, by using it as a vehicle for identifying gaps and generating requirements for the new system. Reuse of existing FOSS applications can be increased if a central resource provides support in identifying FOSS applications that are relevant in the local context.

References

[1] Advance hospital management. [Online]. Available: http://sourceforge.net/projects/arpitsoft/, last accessed October 28, 2010.

[2] Ajax, the collaborative applcation platform. [Online]. Available: www.ajax.org, last accessed October 24, 2010.

[3] Java hospital information system jhis. [Online]. Available: http://sourceforge.net/projects/jhis/, last accessed October 28, 2010.

[4] Jquery write less do more javascript library. [Online]. Available: www.jquery.com, last accessed October 25, 2010.

[5] Netbeans ide. [Online]. Available: http://netbeans.org/, last accessed October 25, 2010.

[6] Open source resouce center, a project of pakistan software export board. [Online]. Available: www.osrc.org.pk, last accessed October 25, 2010.

[7] Pakistan software export board. [Online]. www.pseb.org.pk, last accessed October 25, 2010.

[8] Php builder. [Online]. Available: www.phpbuilder.com, last accessed October 25, 2010.

[9] Php classes. [Online]. Available: www.phpclasses.org, last accessed October 25, 2010.

[10] Scrum.org improving the profession of software development. [Online]. Available: www.scrum.org, last accessed October 25, 2010.

[11] T. Badsha. Funding foss projects to increase participation. [Online], September 2008. Available: http://wiki.sugarlabs.org/go/Sugar_Labs/Funding, last accessed October 20, 2010.

[12] Cheick-Oumar Bagayoko, Jean-Charles Dufour, Saad Chaacho, Omar Bouhaddou, and Marius Fieschi. Open source challenges for hospital information system (his) in developing countries: a pilot project in mali. *BMC Medical Informatics and Decision Making*, 10(1):22, 2010.

[13] B. Fitzgerald and T. Kenny. Open source software in the trenches: Lessons from a large-scale implementation. In *Proceedings of the Twenty-Fourth International Conference on Information Systems, Seattle*, December 2003.

[14] Joan Greenbaum and Kim Halskov Madson. Small changes: Starting a participatory design process by giving participants a voice. In Douglas Schuler and Aki Namioka, editors, *Participatory Design Principles and Practices*, pages 289–298. Lawrence Erlbaum Associates, Inc., 1993.

[15] B. M. Hill. Problems and strategies in financing voluntary free software projects. [Online], June 2005. Available: http://mako.cc/writing/funding_volunteers, last accessed October 24, 2010.

[16] F. T. Igira and O. H. Titlesta et al. Designing and implementing hospital management information systems indeveloping countries: Case studies from tanzania zanzibar. In *Helina 2007/12th JFIM*, 2007.

[17] Jacques Lonchamp. Open source software development process modeling. In Victor R. Basili, Silvia T. Acua, and Natalia Juristo, editors, *Software Process Modeling*, volume 10 of *International Series in Software Engineering*, pages 29–64. Springer US, 2005. 10.1007/0-387-24262-7_2.

[18] P. V. Shaw. Considering the options for hospital management information systems. In *Helena 12th E-Health Conference*, 2007. Bamako, Mali.

[19] H. Tolentino, A. Marcelo, P. Marcelo, and I. Maramba. Linking primary care systems and public health vertical programmes in the philippines: An open-source experience. In *AMIA Annu Symp Proc.*, pages 311–315, 2005.

[20] Qian Yi, Richard E Hoskins, Elizabeth A Hillringhouse, Svend S Sorensen, Mark W Oberle, Sherrilynne S Fuller, and James C Wallace. Integrating open-source technologies to build low-cost information systems for improved access to public health data. *Int J Health Geogr*, 7:29, 2008.