CO600 Group Project

KTWLMS: A Kidney Transplant Waiting List Management System

Tristan Cuthbert
Evdokia Kolia
Lewis Webb
Abstract

This paper discusses the technical aspects and overall lifecycle of the KTWLMS final year Computer Science group project that was aimed at developing a Kidney Transplant Waiting List Management System for the Kent and Canterbury Hospital Renal Unit. The current system (based on a number of simple spreadsheets) is inefficient and disliked by the current users so it was proposed by the development team on the project, in collaboration with the end user, to build a database driven system that was front ended by a web style interface. The paper begins by introducing the current system utilised by the hospital and takes this on further by discussing the development of a proposed design for the new system, as well as the technical reasoning behind this design. Finally, the implementation of the system is discussed as well as the strategy for testing the finished article and a summary of the projects outcomes.

1. Introduction

At the time of writing there were currently 5737 people in this country waiting for a kidney transplant [1]. In terms of the number of patients who are waiting for one, a kidney transplant is the most sought after transplant in the country. In fact, such is the need for kidney transplants that the Department of Health estimates that treating people for kidney failure in 2004-2005 cost the NHS in England more than £600 million – around 1% of its total budget. There are currently 18,000 people on dialysis and this number is rising at a rate of 5% annually costing an additional £24 million. On average, an adult will have to wait 729 days on the list before they are able to receive a transplant (139 days for children) – the longest wait of any available transplant.

The national allocation scheme uses a computerised protocol to allocate a particular kidney to an individual patient primarily based on blood group, degree of tissue matching and time spent on the waiting list. Local schemes in operation across the country vary slightly from area to area but in general tend to use very similar protocols to allocate the kidney to a local recipient.

The Kent and Canterbury Hospital is a local unit for kidney transplant. In the district in a year you will typically find approximately 300 possible transplants patients and 50 who will actually make it onto the list for a transplant. Details of the patients involved in this process are currently kept by the hospital in a simple computer file that is updated as necessary. Most of these updates will be as a result of patients changing their status throughout the process. For example – a patient may be suspended temporarily from the list of potential transplant patients due to medical reasons or if they are away on holiday etc. They may then become ‘Active’ again on the list when they return from holiday or when their medical condition improves. These changes are noted locally but also need to be communicated on a nationwide level to UK Transplant who maintains the primary list of prospective kidney transplant patients and ultimately decides who will get a transplant. Recording of these updates is crucial if an efficient system is to be run with matching datasets being kept nationally and locally.

The major objective of the Kent Kidney Patient Association is to represent people requiring renal replacement therapy who are patients of the Kent & Canterbury Hospital. Moreover, it provides further support to help face up to kidney failure and help families to cope with the stress of long-term dialysis treatment.

2. Background

The current system utilised by the Kent and Canterbury Hospital to monitor the
local list of patients awaiting a kidney transplant is driven by several Microsoft Excel spreadsheets. At the moment one of these spreadsheets (See corpus section 8.14) contains the main list of patients kept by the hospital. A patient will be assigned a specific modality, which is simply the type of treatment they are currently receiving. They will also be assigned a status, which shows which stage of the transplant process they are currently at. This could be 'Active', meaning the patient is awaiting a transplant or 'Suspended', which means the patient is temporary off the list (perhaps for medical reasons or they may be on holiday), but may become active again. There are also a few other options for status which can be found by examining the spreadsheet referred to above. The spreadsheet in section 8.15 shows patients who have been removed from the list for whatever reason and the spreadsheet in section 8.16 a list of data that is sent to the hospital by UKT regarding the medical details of specific patients.

As should be readily evident from these spreadsheets, this system is inherently inefficient for a number of reasons. Firstly there are no restrictions placed on the data that is stored within the system (i.e. there are no access rights given to different members of the hospital team). This means that potentially anyone can access the data and make amendments/additions to it. The system is also not held on a PC but on a laptop which has the drawback that it can only be accessed from the one machine that it is stored on. It has also been the case in the past that the laptop with the system on it has been mislaid (potentially stolen!).

There are no validations placed on the data either, meaning that the user can potentially enter invalid data that will make no sense when they come back to view it at a later date. There is also no way of telling what changes were made to the data in the system and when these changes were made (i.e. the system has no memory of events that have occurred). So it would be quite possible for a change to be made to a patient record and 3 months down the line when that record is next viewed, it could be discovered that the record was changed with no explanation of when or why it was changed. Therefore the system lacks any comprehensive auditing facility. This leads on to one of the biggest problems with the current system, which is one of data validity.

The main list of transplant patients in the UK is held at Guys Hospital in London. If a change is made to data on the local list, it needs to be communicated with Guys so they can update their main list (and vice versa). Often this communication is lost and the two lists kept by the hospitals end up saying different things, which is obviously not very useful in terms of keeping reliable and accurate patient records.

The spreadsheets also provide no means of searching the data for specific records (based on key fields) or for management to obtain reports on the data in order to discover useful information such as the length of time a patient has been on the list or what the patient was doing between certain dates.

The commonality these problems with the current system share is that they can all potentially be solved and themselves act as requirements for a new system. Therefore, the groups brief was to design and develop a transplant waiting list system that would take into account the flaws of the old system and try to turn them around into new functionality that could form the basis of a new system.

3. Aims

The main aim of this project was to develop a fully working system that could be used by the hospital if they wished, to simplify the management of kidney patient data.

If the KTWLMS was going to be a success then it needed to replicate (and add to) the existing functionality of the hospitals current crude spreadsheet system. This would mean satisfying the everyday needs of the hospital in the following ways:

- Allowing a nurse to see what a patient is doing (in terms of the transplant list on the current day).
- Allowing a system user to view patients with certain dialysis status (modalities), including allowing nurses to use reporting features to access this information.
- Allow a user to add a patient record.
- Allow the user to search for patients with specific blood groups.
- Allow the user to search for patients who have currently been ‘Unassigned’ a status.

Basic requirements of such as system that could help facilitate these needs would be:

- Data entry and modification with relevant validation
- Data access rights (i.e. log-on/log-off functionality
As well as these basic requirements the system would require a number of more complex components including:

- Automated Email functionality for confirming with different parties, changes made to the data.
- Detailed reporting of data, specified by the hospital.
- Audit tracking of changes made to the data over time.
- Logging of who made changes to the data and to what records they made the changes to.

Undertaking this project also gave the development team made up of ourselves, the chance to expand our programming and development horizons beyond the core Java programming that is taught at the university. This would include an expanded knowledge of web-based software implementations and an increased awareness of the planning and procedural elements that make up project management. It would also give us experience of learning how to search the relevant literature and web-sites in order to obtain help for our problems.

4. Project Planning

The group decided on a software process model to follow from the outset of the project. This was to be an Iterative model, whereby the implementation of the project would be split into a number of iterations, each with an evaluative process at the end of the iteration.

The group had decided on developing two iterations of the software – whereby the first iteration would contain the basic functionality of the system and the second the more complex functionality. The reasoning behind this was that if halfway through the second iteration it was decided that the software would not get finished, the group could deliver the first iterations attempt to the customer as a finished article (albeit with none of the more complex functionality). This would at least give the customer ‘an’ end product. The merits of this methodology were obvious and can be read in more detail in the Project Planning 2.1.5 document; however it was decided a number of months into the project that an iterative approach to development was not going to be a realistic one. With constraints being put on the group’s time, it was clear that developing iterations of the software that would undergo individual evaluations was not the best use of the group’s resources. This was also coupled with the fact that all members of the group had resorted to following a methodology they were already familiar with from previous software projects. This methodology was the ‘Waterfall’ model, and as can be seen from the Project Planning document, the group updated the process model in accordance with this decision.

The waterfall model allowed the group to work on the project in a sequential manner (in stages), making sure that errors early on in the project lifecycle were eradicated so as not to cause problems at a later stage (one of the major benefits of the waterfall model).

The group wanted to follow standards of project planning as closely as possible, without doing it purely for extra academic merit. Doing this meant laying down procedures that would be followed throughout the project lifecycle (such as the taking of minutes at meetings, storage of group emails, and the adherence to of standards of quality such as document standards) in order to ‘standardise’ the groups work.

![Figure 1 – The Waterfall Process Model](image)

An estimation of the project schedule was also completed at the beginning of the project, and updated throughout resulting in the final ‘actual’ timescale that was followed (see Project Schedule (Est) 2.2.3 and (Act) 2.2.4), showing how for a large proportion of the time the group managed to follow the schedule it imposed on itself. (See appendix C -Gantt chart for a simplified example of this project schedule).

Some basic assignment of roles was also carried out at the beginning of the project (see Configuration Management 2.1.3 document for definition of this) giving each member of the group an area in which they would be ‘expert’ on the project. The group found this beneficial as it allowed each member of the group to be accountable for
some part of the group’s efforts and ensured that all areas of the project received due attention. This was however never carried to extremes and all members of the group were afforded the opportunity to extend their responsibilities and perform different tasks.

The group also decided it prudent to set milestones for the project. These milestones would allow the group to effectively ‘sign-off’ on different stages of development and represent important aspects of the project that had been completed. Milestones also act as reviews for the project and help to document what has been achieved up to a certain date. A detailed list of milestones for the project can be found at the end of the Project Planning 2.1.5 document.

5. Technical

5.1 Approaches considered

A number of approaches to development were considered from the outset. The customer at the hospital had always had it in mind that the new system should not be a stand-alone application in the same way as the current system of spreadsheets. The desired approach was to produce a web-based system that could be run on the hospital intranet. This would of course have the advantage that it could be run from any networked PC on the hospital intranet. Standalone environments that were considered in the event were limited at Microsoft Access and Visual Basic. These of course don’t offer the flexibility and access that web-based applications afford. A web-based approach gave the group a wide range of software development environments from which to choose. PHP and Java were considered as well as a number of databases including PostgreSQL and Oracle (see Software Feasibility 1.2.3 document for more detailed discussion).

The group (in collaboration with the customer) decided on using Visual Basic.NET within Active Server Pages (ASP.NET) connecting to a Microsoft SQL Server 2005 database. This afforded the group the chance to learn new things from development with tools that we had not used together before. However, some members of the group had had experience using Visual Basic before so there would a degree of expertise working with this language.

5.2 Experimentation Phase

During the experimentation phase, the group experimented with different set-ups for producing the final software article. This allowed the group to spot any potential problems with certain set-ups before implementation activities began.

Primarily this involved trying out different combinations of database and programming language that would be used to implement the final software solution. Experiments at this point involved taking different databases and attempting to connect to them from web-pages. Databases involved in this process were Microsoft Access, MySQL, and Microsoft SQL Server.

Each experimentation was met with a degree of success (success in that successful connections were made to the databases) however it became clear of the difficulties that would be faced with certain options. MySQL for instance required a relatively large amount of code, tweaking of the GAC (Global Assembly Cache) and making references to certain .dll files in order to get a connection established. Microsoft Access and SQL Server on the other hand proved very easy to connect to with simple connection strings providing the connections. It was because of this ease of use that SQL Server was chosen as the database of choice for the system (Access being a good alternative but lacking some of the integration qualities with the IDE – see section 5.4 Visual Web Developer). Assistance on the Internet was also vast and more understandable for SQL Server over the other databases we encountered.

5.3 IIS and ASP.NET

One of the restrictions placed on the project by the hospital was that the new system had to be capable of running on Internet Information Services (IIS). IIS is a powerful web-server that provides a highly reliable, manageable, and scalable web application infrastructure for all versions of Windows Server 2003 [7]. As the worlds second most used web-server, it is perhaps the most widely used web-server for corporate websites. IIS also now provides support for Active Server Pages (ASP.NET).

ASP.NET is a technology for building powerful, dynamic web applications [6]. Its easy programming model makes building web-applications easy with an HTML like style of declarative programming that lets you build web pages with less code compared to traditional ASP or technologies such as PHP or
JSP (Java Server Pages). There is also no need to write common code for managing users as this is included as an application service. Access to web-applications is managed through the use of ASP.NET membership services. ASP.NET tries to encourage the programmer to develop applications using an event driven GUI paradigm as opposed to in the conventional web scripting fashion. This in itself makes it suitable for the project seeing as the group was unfamiliar with this more conventional approach to web-development at the start of the project.

5.4 Visual Web Developer

Microsoft Visual Web Developer 2005 Express (to give it its full name) is a lightweight easy-to-use development tool focused exclusively on the building of web-based software projects [4]. Creating web-applications from scratch is relatively easy with a useful drag-and-drop interface. It also contains a very powerful code editor that makes writing code fast, with support for a number of languages including Visual Basic, C# and J#. It is also very easy to create data driven web applications using the built in data controls and integrated access to Microsoft SQL Server 2005 Express, making the process of setting up a database that integrated with a web page very easy to do.

From a development point of view it was critical we use a tool such as Visual Web Developer. As was mentioned previously, ASP.NET requires IIS to run, and the machines on which we were developing the software run on Microsoft Windows XP Home Edition operating systems. The drawback of this is that IIS can only be run on machines running Microsoft Windows XP Professional Edition or Window Server, and therefore we would not be able to test and run any of our development work without setting up an account with an ASP.NET web hosting company. Visual Web Developer gets around this problem because it includes its own development server that can be used by developers to test their products without the need for IIS to be installed (which is a huge benefit as it removed the need to install an expensive array of software in order to run our software). It was also reassuring to know that if our software ran on this development server then it would most likely run on the target hardware at the hospital (which does run IIS).

5.5 System Architecture

When deployed on the target hardware the system architecture should mimic the diagram in Figure 3 below. This is an example of a three-tier client-server architecture [2]. The three tiers consist of a tier that provides data management services, another tier (the web server) which provides application services, and the final tier which represents a client or user machine.

The developed system should reside on the hospitals web server (which will be running IIS) which will provide account provision for all the computers attached to the hospital network. These computers (or clients) will connect to the system on the server via HTTP interaction. When data needs to be retrieved within the system, SQL queries are then made to the SQL database server which stores the patient database. The data is then returned to the clients via their connection with the main web server.

During development, the account service provision of the web server and the client connections melds into one somewhat. When the system is run, Visual Web Developer will start its own development server from which it will appear a client is connected (i.e. the development machine). SQL queries can then be made to the SQL database server.
database server in the way discussed previously in order to retrieve and update data.

Figure 3 – System architecture diagram [1]

5.6 Database design

The initial database design for the system was to contain two tables – one to store the list of active patients and another to list patients who had been deleted from the list (presumably because they had died, were unavailable for transplant or had become suspended from the list). This decision was taken at first because the group thought that for performance reasons it was inefficient querying a table that might have become cluttered with those who have died or become otherwise inactive, so it would be best to remove them to another table, as is currently the case with the spreadsheet used by the hospital (where there is a separate deleted patients spreadsheet).

However after careful analysis of this approach (and discussion with the customer) we decided it to be inefficient. Most of the queries we would be making to the database would end up UNIONing both tables anyway (i.e. a search may need to return inactive (but not necessarily deceased) patients, and so most reports will need to analyse data from both tables (even including deceased patients)). There would also be extra overhead involved in transferring patients between tables and arguably extra schema complexity. The actual process of running reports may also become more complicated, seeing as any query which refers to patients who may have died (but have been alive/active in the scope of the report) will have to refer to two tables. The number of patients entered into the system over its lifetime will also only likely be in the hundreds and not the thousands or millions so there is little need to worry about deleting patients from the system.

Our solution was to design a database with three tables in it that stored the data in a more relational way (See Database Schema Documentation 4.1.7 document and appendix A and D of this document). These tables were ‘Patient’, ‘Modality’ and ‘TransplantStatus’. The Patient table was designed to store the main personal details of a patient. The TransplantStatus and Modality tables were designed to store details of a patient’s status and modality (respectively) over time. These two tables would be linked to the Patient table by the primary key of the Patient table (NHS Number) with NHS Number appearing as a non-unique foreign key in these tables. This would allow for the tracking of a patients status and modality history over time (something the customer wanted in the original specification) seeing as the status and modality tables can contain many references to the same patient.

This new schema allowed for arguably less schema complexity and simplified reporting (seeing as the dates of modality and status changes are stored – allowing the user to search for patients who had a particular status or modality between set ranges of dates). As was mentioned in the previous paragraph, this design also allowed for built in ‘audit logging’ functionality for modality and transplant status changes, with the user effectively being able to create a report that shows the patients movements since they entered the system (something the current system lacks as it simply overwrites the previous status/modality of the patient). Also see ‘Normalization’ 3.2.4 document for details on the database normalization process.

5.7 General Design and Implementation

Much of the groups design work was focused around UML [11] [12]. This allowed the group to create detailed diagrams that could be used to explain the processes of the system and also make the implementation stage easier, (seeing as there would be diagrams and documentation replicating how the flow of data and the systems processes should behave). (See appendix E for an example of the type of diagram used during this stage).

The implementation of the system was based upon a set of requirements that were produced, through meetings and consultation with the customer. What follows is a summary of how these requirements were taken and used to make design and implementation decisions to ultimately create the system in question.
5.71 System Summary

When the application loads, the user is presented with an authentication page that requires them to log-in with a username and password (See appendix B – Diagram 1 and default 1.8 ASP file). This was created using the built in support provided by ASP.NET for Membership and Role Manager Services, which made it easy for the group to tailor it to our own needs. In addition we made use of the forgotten password, change password and create a new user classes (See ChangePassword 1.4, ForgottenPassword 1.11 and CreateUser 1.7 ASP files). Every page then has script written to check if the user is logged in, if they are not then they are redirected to the log-in page otherwise the users roles are checked against the roles needed to gain access to the page. If they do not have sufficient role privileges then they are redirected to the appropriate pages.

If a user successfully logs-in then they are presented with the main menu (See appendix B – Diagram 2 and menu 1.4 ASP file). This has hyperlinks to Patient File, Audit Trail and Reporting along with Create a new user if they have sufficient privileges. Audit Trail enables the user to see the entire history of a patient’s status and modality. It uses previous entries in the modality and TransplantStatus tables so that the user can see previous status/modality’s and start and finish dates. However, if a user needs to find out who made these changes and when they made the change, they can refer to the text log-file. Reporting functionality is covered in section 5.8.

Patient file is another menu page that has hyperlinks but this time it is for various patient operations (See PatientFile 1.15 ASP file). First is View All Patients, which shows all the patients in the database with their current status and current modality (See viewall 1.24 ASP file). It does this by making sure that the status_id or modality_id is equal to the current modality/status field in the patient table. This ensures that just the current status and modality is displayed and not any previous records. There is the ability to search for a patient (See appendix B – Diagram 3 and SearchPatient 1.18 ASP file). The customer requested that they should be able to search for a patient by NHS Number, EKHT Number or by a Surname. This search should only bring up the current status and current modality in the same way as view all patients.

In the future if we have time before we hand the system over to the hospital, we hope to implement a smart search page. This would mean that users would not have to type the full surname in and could just type the first four characters in for example. This was made clear in the user acceptance testing that it is needed as users struggled to remember long surnames with difficult spellings.

The next page in the system is where the user can add a new patient to the database (See appendix B – Diagram 4 and AddAPatient 1.1 ASP file). This page inserts the new patient’s personal details into the patient table along with the current_modality and current_status id’s. To gain these ids when the page loads we have an sqldatasource selecting the max ids in both tables and then adding one to get the id of the next record that will be inserted. We then have to insert the modality and status records into their respective tables with the id that has been calculated. We encountered some problems on this page that were due to using the form view object to insert the data.

It is a very neat tool for just inserting or updating records however when you need to customise the process it then become difficult. You cannot reference the text boxes from outside this form view and therefore when trying to write code to log to a text file and email UKT changes that the user has entered, you cannot retrieve these values from the text boxes. To get around this problem we had to insert the records and then get the user to click a second button which refreshes the page, selects the record that has just been inserted and then saves this to a log file and emails it to UKT. This is not a very neat implementation as it requires the user to click on two buttons however it was the only way we could find to implement it. However from the acceptance testing we were informed that we do not need to send this information to UKT and so it will be removed. Please see Maintenance 6.1.1 document for more details.

There are then three pages to change the records for each of the three tables. These are ChangePatient, ChangeModality and ChangeStatus (See appendix B – Diagram 5 and ChangePatient 1.5, ChangeModality 1.3 and ChangeStatus 1.6 ASP files). Firstly ChangePatient is used to update the personal details of the patient. This simply searches for a patient by NHS Number and places it in a form view with its default view set to “Edit” ready for modifying (See appendix B – Diagram 7). The group then implemented ChangeStatus and ChangeModality in the same way.

It was not until we went to create the Audit Trail page that we realised that we had implemented these pages incorrectly (See...
AuditTrail 1.2 ASP file). What we hadn’t considered when implementing these pages was that when changing a status we need to insert a new record into the database and not just modify the present record. This meant a total redesign of these pages and it took a great length of time implementing them so that they update the current_status and current_modality fields in the patient table.

5.8 Reporting

5.8.1 Crystal Reports

Crystal Reports is a powerful reporting tool that satisfies the reporting needs of this project [9]. It comes with a wide variety of tools and a comprehensive sample database to help you master the program and use it efficiently. It also comes with Online Help that gives a description of each command, dialog box, formula function and operator included within the program [10]. It also includes sample formulas, group selection formulas and many other topics related to reporting.

When working with Crystal, you will probably use Design Tab more than any other part of the program. The Design Tab is the place you do most of the initial work when creating a report. It designates and labels the various sections of the report. You can place objects in these sections where you want them to appear, specify sorting, grouping and totalling needs. The Design Tab provides a very efficient environment for designing a report because you work in the tab with data representations, not with data itself. Thus, you can add and delete fields and other objects, move them around, set up complex formulas, without tying up the computer or network resources needed to gather the data.

An example of what Crystal can do is shown here.

\[
\begin{align*}
\text{cdate(TransplantStatus.Status_Start)} & \geq \\
\text{cdate(?startDay)} & \text{ and} \\
\text{cdate(TransplantStatus.Status_End)} & \leq \\
\text{cdate(?endDay)} & \text{ and} \\
\text{TransplantStatus.Status} & = \text{?statusSelected}
\end{align*}
\]

If the formula above were entered into Crystal, it would yield a report that displayed all patients with Active status between a specific range of dates. For more details on actual reporting requirements see section 5.8.2 – Reporting Requirements.

5.8.2 Reporting Requirements

Reporting is one of the most critical processes of the system, since the hospital will use them in order to make important decisions regarding transplants and to view specific data in a user friendly manner. The hospital wants to be able to view and compare patients’ data. Taking all the user’s needs into account, we tried to design the reports in such a way that they will help the user to make their decisions.

First of all, we divided the reports into two categories and created pages from which the user could select the report they wanted to view, and then a page which would display the report. (See ReportMain 1.17 and DisplayReport 1.9 ASP files):

1. Static Reports:

The static reports display patients’ data sorted be different field such as their Surname, UKHT Number, Blood Group, Status, and Modality etc. No further processing is applied on the patients’ data. Moreover, there are reports that display only a category of patients, for example ‘display only the patients with blood group A’ or ‘display only the patients with unassigned status’. There are a great variety of Static Reports available and this is particular useful for the user, because he is able to view information about all the patients as a whole or about a specific category of patients. (See appendix B – Diagrams 8 and 9 and SearchReport 1.19 ASP file).

2. Compound Reports:

The compound reports take between one and three parameters as an input such as dates, patient’s NHS number or patient’s status or modality, pass these parameters into a formula and display only the patients that satisfies this formula (See appendix B – Diagram 6 and SearchReportDates 1.20, SearchReportDatesStatMod 1.21, SearchReportNHS 1.22 ASP files).

Secondly, almost all the reports display the data in groups and automatically calculate the group total and the grant total. At anytime the user is able to know significant information, i.e. how many patients are active or suspended.

Finally, some reports count the duration of each status and modality automatically. This is extremely useful for the user. It is vital for the user to know the duration of each status for a particular patient,
in order to get a decision for them, i.e. if they need to change their status because they has been under a certain status for too long. Furthermore, some reports count the age of each patient and this is very important, since children have high priority in the patients list for getting a kidney transplant.

5.9 System Component Integration

The group decided that the completed system should be made up of 3 main components. In order to create a working system, it was necessary that integration of these components took place. Each member of the group was made responsible for one of these components, ensuring a fair and efficient spread of the workload.

5.9.1 Component 1

Component 1 was the responsibility of Lewis Webb. It involved creating the main functionality of the system, which consisted of data entry/manipulation, Email functionality, audit tracking, logging of changes made to the patient data and who the changes were made by and also the user log-in system. It also involved the creation of the SQL Server database that was to be used by the system for data storage.

5.9.2 Component 2

Component 2 was the responsibility of Evdokia Kolia. It involved the design and creation of the report templates that the system would use to display data (within Crystal Reports) and also the creation of an online help system accessible from the main menu of the system (within Shalom help maker).

5.9.3 Component 3

Component 3 was the responsibility of Tristan Cuthbert. It involved the creation of the reporting section of the system. This required taking the reports that were created in Component 2 and integrating them to display data within Component 1. This required creating extra code that would load a report from within the main system (based on which report the user selected), and take in any extra parameters the user provided (such as a date range) so these could be fed into the report to produce the desired output. This extra code required making references to external Crystal Reports libraries that allow for integration with ASP.NET. The reporting section would be accessible from the main menu of the system.

5.9.4 Integration Process

The integration process was attempted on the development machine used to create Component 1. Component 1 contains most of the main functionality of the system so it made sense to integrate smaller Components into the larger one. This development machine also happened to be a laptop, so integrating the system on here made even more sense from the point of view that the system could be easily developed in different geographical locations and could be easily shown to third parties (i.e. the end user, other group members). It also ran IIS, which was advantageous as it meant we could test the system on a similar configuration to what the hospital would be using.

It was necessary at first to temporarily install Crystal Reports on the integration system. This was necessary in order to set-up the ODBC connection that would allow the reports to access data stored in the system database. Once this was set up, the reports could then be put onto the system.

We tried running the reports with no success at first. This was a problem for quite sometime, but was tracked down in the end to differing report sources referenced on the integration system and the system used to develop Component 3. This was solved by making all report sources refer to locations on the hard disk of the integration system. With this change made we were able to successfully complete the integration of the system.

Initial testing of the reports showed up some unexpected problems. In the main case, this manifested itself in the way that the reports were incorrectly displaying some fields and not displaying other fields at all. This problem was also tracked down, and was due to the fact that Crystal Reports doesn’t cope well with some data types that SQL Server uses. This meant that some changes were necessary to the data types used by some fields in the database. This also allowed for the creation of new reports that were previously unavailable due to the aforementioned data types.

Integration also allowed us to iron out some problems regarding OBBC connectivity between the reports and the database (realising that we had to make sure the name of the ODBC connection the reports were created with was the same as the name of the ODBC that was created on the integration system).
With this process completed, we were able to successfully integrate all 3 components to produce a fully working system.

5.10 Testing

The detailed testing strategy for this system can be found in section 4 of Detailed Design and Implementation 4.1.1 document. The results of implementing this strategy can be seen in the Testing Results 5.2.1 document and the Proof of test Results 5.2.3.

This strategy was composed of 4 different types of testing. These were:

- Unit testing
- Integration testing
- Systems testing
- Acceptance testing

To implement the first 3 strategies, we devised 2 sets of tests, one set that would test the core functionality of the system and a second set that would test the reporting functionality. In order to do this it was necessary to come up with a set of test cases that would test/cover all aspects of the software (See Testing Results 5.2.1 document for this set of cases). Each test case would need to pass in order that the test could be deemed successful and to show that that part of the system functionality was working. If the test failed then it was necessary to fix the problem and attempt a re-test which would also need to pass in order to deem the test a success.

5.10.1 Core Functionality

The testing of the core functionality of the system was designed to satisfy the majority of the requirements set out in the testing strategy for ‘Unit testing’, ‘Integration testing’ and ‘Systems testing’. As mentioned in the previous paragraph, there were a set of test cases (10 in total) that tested the processes involved in the system – test cases that tried to replicate typical functions the user may use the system for. For example – this could consist of logging in to the system, adding a patient to the database or searching for a specific record etc.

The majority of these tests were successful. Each part of the system had been thoroughly unit tested throughout its development, so problems were very unlikely to be found.

The group did however find a couple of problems with various ASP pages regarding test case 10, which was designed to check users could not access certain pages by typing in the address of the page in the browser address bar (if they were not logged in). For two ASP pages this was unsuccessful and a user was allowed access, (this was shown up by our testing activities). The problems simply related to minor mistypes in the code for these pages and so the problem could be solved and the pages re-tested to enable all the Core Functionality test cases to pass.

5.10.2 Reporting Functionality

All the system reports were tested on a pre-fabricated dataset. Each report was run from within the system and where extra parameters were needed, these were kept as similar as possible across all the reports in order to maintain a degree of standardisation. The expected output was then checked against the actual output to gain a test or failure status (See Proof of test Results 5.2.3 for evidence of each report test). Apart from a small number of reports, all successfully gave the correct output. Those that were unsuccessful were re-worked in Crystal (normally due to slight formulaic errors in the design of the report). Other errors that were identified involved naming errors of the reports, where the report name used to attempt to load the report was different from the actual name of the report.

5.10.3 Acceptance Testing

The group believed it to be important that the end user have some exposure to the system before the end of the project. The group arranged to meet with Chris Farmer (the consultant from the hospital and one of the primary users of the new system). This meeting was arranged in order to gain some initial feedback on how well the system satisfied the requirements of the hospital and to understand whether any changes were needed before its final delivery.

At the meeting, the user was prompted to use the system following several pre-prepared scenarios the group had in mind (that would attempt to mimic typical functionality of the system). The user was then asked if at the end of the process they could give some feedback on what they thought of the functionality provided by the system for each scenario and if there were any problems they had. See Acceptance Test 5.2.4 document for details of the results of this acceptance testing process. Also see Meeting Minutes 21-03-2006 6.8 document for a detailed
description of the minutes taken at this meeting.

The results of the meeting were very encouraging. The user commented that the system could be potentially very useful in attempting to reduce the time a patient spent on the transplant list. The user also commented that he would like to try installing the system on the hospital computers in order to see how it functioned.

There were some comments the user made about potential improvements for the system. Please refer to Testing Results 5.2.1 document section 4 for details of these suggested amendments.

Overall, the acceptance testing process was a largely successful one. The group were able to identify several possible modifications that could be made to the system to satisfy some of the suggestions made by the end user. Even without these changes though, the software should be in a mature enough state for it to be given a trial run on the hospital computers.

5.11 Remaining Problems

Although the system satisfies the majority of the major requirements, there are still some minor bugs/problems remaining.

There are some transplant related data fields that the group believe are currently missing from the database that could be added at some point. These extra fields would also allow some extra reporting functionality to be incorporated. The group attempted to obtain information about these fields from the end user but were unsuccessful with this request before the end of the project. Adding these extra fields though should not present much complexity in terms of the database implementation and the relevant reports could also be updated.

The log-in and Email functions do work satisfactorily on the development machines; however the log-in will not currently connect to the hospitals current authentication system and has been implemented as a stand alone process within our software. Extra code would be required at the hospital end to implement this successfully. The Email has also not been tested on the hospital system and may need additional code to interface with the Email protocols used by the hospital. We have also learnt that it may not be necessary to send an Email to UKT when a patient record is added, so this functionality should be removed in future.

It is also not ideal having NHS Number as the main search field for the database either as the customer informed us that these are often very long numbers, and as such could be very time consuming to enter into the system.

5.12 Evaluation

The group were generally pleased with the software we were able to produce, as it satisfied the large majority of its initial requirements.

Development time was longer than we had anticipated, due to the fact we had failed to fully take into account the time it would take to learn some aspects of the development tools and also the complexity involved in implementing some of the system functions (such as the display of reports and Email).

There are also some slight bugs still remaining. For instance, we found it difficult to conceive how we could design an Email system that would allow for the confirmation of received Emails. Some other aspects could also have been improved upon if there was further contact with the customer and a more specific requirements gathering mission completed (i.e. the group was unaware of the true scale of the task faced by the hospital in terms of auditing needs until near the end of the project – the customer had not fully specified this at previous meetings). This could also have led to the adding of additional required fields to the database.

On face value, it appears as if the system we created is relatively simple. However, none of the group members were familiar with ‘all’ of the development tools, and so a large amount of development time was spent learning how to use these tools effectively to satisfy our requirements.

A number of the tools required considerable customisation of options files and settings in order to get them working as we needed and a number of the features we had to implement needed much research to be carried out into them (mainly via internet help sites and the relevant literature [3], [8]) in order that we could learn how to implement them. For instance, the display of a report may seem like a trivial matter, but the fact none of us had done this in a web environment before meant extensive research had to be carried out to find ways of doing this and attempts made at trying different combinations of code to make it work. Often these processes involved extensive, time consuming trial and error to get
even simple things working the way we wanted.

The final system composes integrated work from 4 different tools. The fact that we were using 4 separate tools for the development of the system (Visual Web Developer, SQL Server 2005, Shalom HelpMaker and Crystal Reports) as opposed to one (which would often be the case with a Java development) meant that a lot of time had to be spent learning how to get these tools working in unison, before work even began into implementing the system.

The group would’ve liked to have followed an iterative approach to the software lifecycle model (this was originally the plan – See Project Planning 2.1.5 document – information regarding the previous version of the document) as this would’ve allowed for more evaluation at different stages of implementation and more customer feedback. However, our hand was forced somewhat (due to time issues mentioned in the second paragraph of this section) so we had to adapt our model in order to develop the software in one go (see Project Planning 2.1.5 document, section 2).

Further details regarding the evaluation process can be found in the Review 6.1.3 document.

6. Conclusions

It is difficult to fully ascertain how successful our final system was, due to the fact that at the time of writing it has not been installed on the end user hardware. The real test will come if the system goes live on the hospital system, working with real world data. We did however; manage to produce a system that met all the major requirements that were set out to be met at the beginning of the project. It would have been advantageous though to have had more end user input in order to make sure that these requirements were fully satisfied from their point of view. Initial user acceptance feedback though was very encouraging.

The group is confident that we produced a useful working system and the majority of functionality that was required has been implemented (See Requirements Specification 3.1.1 document). Data entry and manipulation, Email functionality, audit tracking of patient data over time and detailed reporting were all able to be integrated into the final system.

We also believe that we have all added worthwhile knowledge of tools and technologies that we were previously completely unfamiliar with, such as Visual Web Developer and SQL Server 2005. We found the use of Visual Web Developer of critical importance to the completion of the project (due to it including its own development server) and believe it to be a very powerful development tool (albeit one that did require a lot of customisation in order to get it fully working as we wished).

As far as we know, the system we created is the only one of its kind (apart from the original spreadsheet system that is used by the hospital at present). It is far superior to this original system in that it now has memory of events and is able to track/log changes made to data over time (including being able to tell the user who made the change and to what data they made the change to).

The ideas we came up with for the system are not particularly novel, but were suitable for the task in hand and were well implemented. We were however able to take the established idea of a database driven website with built in reporting functionality, and develop it to create a system that fit around the requirements of our end user. We do believe however, that we came up with a clever design for the database that provided the required audit logging and ‘memory’ capability that the customer wanted (See appendix A for a diagram of the final database schema.

We would advise groups attempting a similar project in the future to ensure that they have regular group contact and also regular ‘face-to-face’ contact with the end user/customer, as via Email it is very difficult to fully convey ideas and opinions to any acceptable degree of usability. We would also advise an early start of implementation activities but at the same time interspersing it with documentation (as our group did) instead of leaving all the documentation to the end of the project and then rushing it so it does not properly document a process that actually occurred months in the past.

There is scope for this project to be picked up by another team in the future. The Email functionality that we created has been tested and works on the development machines; however it has not been tested on the hospital configuration and so may need to be adapted in order that it integrates correctly. We would also like to improve the log-in functionality of the system so that it interfaced with the current hospital authentication process. This would mean any users of our system could use their current hospital log-in credentials to access the system as opposed to needing separate log-in credentials as they do
with our system as present. There are also some extra fields/columns that could be added to the database, that the customer was unable to provide us with by the project deadline. These would help to improve further the range of reports on offer to the user and fully satisfy the data storage requirements of the patient data. Simpler functionality might also be modified such as changing the primary system search key from NHS Number to patient surname or EKHT (East Kent Hospital number).

7. Acknowledgements

We would like to thank our project supervisor, Roger Cooley for his continued support and advice, and for making himself readily available and indeed stressing the need for regular group meetings.

We would also like to thank Toby Wheeler and Dr Chris Farmer from the Kent and Canterbury Hospital for all the information they have supplied and the guidance they have given relating to the project.

8. Bibliography

[1] UK Transplant
http://www.uktransplant.org.uk


http://msdn.microsoft.com/vstudio/express/vwd/

http://www.microsoft.com/sql/editions/express/default.mspx

[6] ASP.NET
http://www.asp.net/

[7] Internet Information Service (IIS)

[8] Microsoft Developer Network
http://msdn.microsoft.com/

[9] Crystal Reports
http://www.crystalreports.co.uk/


APPENDIX A
Database Relational Schema Diagram

APPENDIX B
Final System Screenshots

Diagram 1 – The log-in menu
Diagram 2 – The main menu
Diagram 3 – Search for a patient menu
Diagram 4 – Add a new patient menu
Diagram 5 – Change a patient status menu

Diagram 6 – Example report search menu

Diagram 7 – Modify a patient record menu

Diagram 8 – Report select menu

Diagram 9 – Example of report display
APPENDIX C
Simplified Project Gannt Chart

Gantt Chart

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Start</th>
<th>Finish</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initial Investigation</td>
<td>3/16/2005</td>
<td>11/10/2005</td>
<td>11d</td>
</tr>
<tr>
<td>2</td>
<td>Feasibility Study</td>
<td>11/10/2005</td>
<td>11/10/2005</td>
<td>11d</td>
</tr>
<tr>
<td>3</td>
<td>Existing System Review</td>
<td>11/10/2005</td>
<td>11/10/2005</td>
<td>11d</td>
</tr>
<tr>
<td>5</td>
<td>New System Design</td>
<td>1/12/2005</td>
<td>1/12/2005</td>
<td>11d</td>
</tr>
<tr>
<td>6</td>
<td>Implementation and testing planning</td>
<td>1/12/2005</td>
<td>1/12/2005</td>
<td>11d</td>
</tr>
<tr>
<td>7</td>
<td>Testing phase</td>
<td>1/12/2005</td>
<td>1/12/2005</td>
<td>11d</td>
</tr>
<tr>
<td>8</td>
<td>User Guide development</td>
<td>1/12/2005</td>
<td>1/12/2005</td>
<td>11d</td>
</tr>
</tbody>
</table>

APPENDIX D
Data Access Diagram

Data Access Diagram

Patients File

<table>
<thead>
<tr>
<th>NHS Number</th>
<th>EKHT Number</th>
<th>Current Status</th>
<th>Current Modality</th>
<th>Forename</th>
<th>Surname</th>
<th>Date of Birth</th>
<th>Gender</th>
<th>Blood Group</th>
<th>Tissue Type</th>
<th>Telephone Number</th>
</tr>
</thead>
</table>

Transplant Status File

<table>
<thead>
<tr>
<th>Status ID</th>
<th>NHS Number</th>
<th>Status</th>
<th>Status Start</th>
<th>Status End</th>
</tr>
</thead>
</table>

Modality File

<table>
<thead>
<tr>
<th>Modality ID</th>
<th>NHS Number</th>
<th>Modality</th>
<th>Modality Start</th>
<th>Modality End</th>
</tr>
</thead>
</table>
APPENDIX E
Logical Data Flow Diagram