The candidate confirms that the work submitted is their own and the appropriate credit has been given
where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered
as plagiarism.

(Signature of student)________________________
Summary

The aim of this project is to provide the School with a distributed system to manage both the submission and marking of coursework. A database provides the storage mechanism for the submitted files. A three-tiered client-server architecture is used to allow students to submit files to the database for marking. A separate client application allows management of the database, including features to set courseworks and assign markers for the work. This database solution allows active rules to be written to handle workflow management of the marking process.
Acknowledgements

This report is dedicated to my dear wife Karen, who has kept us going over the last three years whilst I’ve been getting my career back on track. One love.

I’d like to thank Nick Efford, my project supervisor, for all the suggestions regarding the functionality of the system. Nick has acted as stakeholder in the project, highlighting features that would be useful. His input has been invaluable.

Thanks must also go to Steve King for making Red Hat 8 run on a 200MHz processor. No mean feat. It’s slow, but it works. Without this setup it is unlikely this report would even exist. Hopefully when I’ve graduated I’ll be able to afford some new kit.

Many thanks to my good friend Steph the Sub-Ed for the eleventh-hour proof-reading. Easy with that red pen!

Chris Gillespie was helpful in answering some of my questions about the current coursework submission system running under SIS. Hopefully, if the system I have developed is tidied up and used by the School, the problems he highlighted will be gone.

Mark Conmy and the support team have been a massive help via e-mail and the newsgroups in a number of areas, including the choice of DBMS (PostgreSQL beats MySQL hands down), and my initial forays into the Perl programming language.

Thanks to Stuart Roberts and Peter Mott for teaching me all about databases over the years. It is an area of computing that I find fascinating. Stuart checked over my E.R. model very early in the project. The model was correct. The database works wonderfully!

To all who completed the questionnaires to aid with my evaluation - the end-of-term pints are on me!

Thanks finally to Sarah Fores, who has been my personal tutor for the last three years. I hope I have not been too much trouble!
Contents

1 Coursework Submission and Management 1
   1.1 Overview ......................................................... 1
   1.2 Existing Solutions ............................................. 2
      1.2.1 E-mail attachments .................................. 2
      1.2.2 School Information System ............................. 3
      1.2.3 "submit" ................................................ 3
   1.3 The Proposed New System ..................................... 4
      1.3.1 Features of the new system .............................. 4
      1.3.2 Formal system requirements ............................. 4
         1.3.2.1 Functional requirements .......................... 5
         1.3.2.2 Non-functional requirements ...................... 5
      1.3.3 Methodology for the process ............................ 6

2 Project Management 7
   2.1 The Project Plan ............................................... 7
      2.1.1 Objectives ............................................... 7
      2.1.2 Minimum requirements ................................... 8
      2.1.3 Possible enhancements .................................. 8
      2.1.4 Deliverables ............................................. 9
      2.1.5 Project schedule ....................................... 9
   2.2 Revisions to the plan ........................................ 10

3 Building the Database 11
   3.1 Entity-Relationship Modelling ............................... 11
      3.1.1 Identification of entities and relationships .......... 11
      3.1.2 Entity attributes ...................................... 13
      3.1.3 The model ................................................. 13
   3.2 Table Schemata ................................................ 14
      3.2.1 Mapping the model to a set of logical schema ....... 14
      3.2.2 Derivation of PostgreSQL data definition statements 16
6.1 Workflow Management .................................................. 41
6.2 Archiving ......................................................................... 42
6.3 Synchronisation with SIS .................................................. 43

7 Evaluation 44
7.1 Satisfaction of System Requirements ................................. 44
  7.1.1 Functional requirements .............................................. 45
  7.1.2 Non-functional requirements ....................................... 47
7.2 Evaluation of Coursework Submission Application ............. 48
  7.2.1 Method ................................................................. 48
  7.2.2 Results ................................................................. 49
  7.2.3 Conclusions .......................................................... 49
7.3 Security Review ............................................................. 49

Bibliography 51
Chapter 1

Coursework Submission and Management

In this chapter, the area of electronic coursework submission is examined to identify the issues that arise. Previous systems used and developed within the School are examined and evaluated, and a proposal is set down for the new system that is developed in this project.

1.1 The Problem Domain

Many courseworks set by the School of Computing require electronic submission of files. Usually the files to be submitted in this way contain code that has been written by students as part of the coursework. There are a number of issues that must be addressed when developing a system to facilitate this process.

A suitable method for storing the submissions must be found. The files must be stored in such a way that they cannot be tampered with in any way by any person other than the student who submitted them. It is especially important that submitted files cannot be read by other students, as this would facilitate plagiarism, which has been a problem for the School in the past and will likely continue to be so. Markers must be able to access the files for marking. This access should be read-only, to prevent markers from accidentally altering the files before marking. It should be clear which files have been submitted by which students for the purposes of marking, so the markers will credit the correct student for submitted work. The storage mechanism can help with this, as files submitted by different students can be stored in different places.
There must also be some conduit for the process of transferring the files from students’ workstations to the storage area. Submission of files should certainly be possible from any of the School’s workstations, although a better solution would allow students to submit files from any machine with an Internet connection. It is important that files should not become corrupted en route to the storage area, as a corrupted file could cause a student’s work to be erroneously marked down. Students should be able to obtain some kind of receipt for the files they have submitted, in case some part of the system fails and the files do not reach the storage area uncorrupted. The receipt should identify the date and time at which the files were submitted, and, if possible, the content of the files. It should not be a simple task for other students to intercept files in transit, to prevent plagiarism.

Course leaders need to be able to manage certain parameters of the system to ensure that the submission and marking of files for a particular coursework are possible. There should be a method for setting up the storage for files that are to be submitted for a particular coursework, to ensure that students registered for the module are able to submit files. There should also be a method for assigning the people who are to mark the coursework, to ensure that the files can be accessed by these people (and only these people).

1.2 Existing Solutions

1.2.1 E-mail attachments

Some course leaders choose to accept electronic submissions as e-mail attachments. This method is not very sophisticated, but it meets some of the requirements highlighted above and, in certain circumstances, can provide a reasonably good solution to the problem. The submitted files are stored as attachments to e-mail messages in an e-mail account. The files can only be accessed by people who have the password to the e-mail account, so they are protected from other students and can be accessed by markers, who can simply be given the password to the e-mail account. The sender of each e-mail message identifies the student who has submitted the files. Use of e-mail as the transport mechanism means that files can be submitted from any location and are unlikely to become corrupted or be intercepted.

The major problem with this solution is with the issuing of receipts. It would be a simple task to set up an auto-reply message that responds to every e-mail received by the account. However, replying with a message that identifies the contents of the attached files would require some kind of script to be written, and, in practice, this is not usually done. This leaves the system open to unscrupulous students, who may submit a message with no attachment before the coursework deadline to buy more time. If they had a receipt for an e-mail message, they could claim they did attach files and blame the School’s e-mail server for the lack of attachments.

An ingenious method devised for SO21 coursework submissions uses e-mail as the transport mechanism, but requires that the e-mail be sent via an external program that generates the receipt. This is a
very good solution to the problem, as the receipt includes a checksum calculated across the attached files, which would settle any disputes between staff and students over submissions decisively.

1.2.2 School Information System

The most popular method employed for electronic submissions at the time of writing is the browser-based solution provided by the School Information System (SIS). Access to SIS is via the School’s intranet, which requires students to enter their SoC username and password. This identifies the student to the system. Access to the coursework submission facilities provided by SIS involves the student following the links Taught student records and View full details. This takes the student to a page containing their student records, which includes the link Submit coursework. The student can then choose the module for which they wish to submit files.

The system stores files in a directory in exactly the same form as they are submitted. To access the files for marking, the whole set of submissions for a particular coursework must be downloaded as a single gzipped tar file, which can then be unpacked, with each student’s submissions being held in a separate directory in the archive. These directories must then be manually divided between the markers, if indeed more than one person is to mark the coursework.

Course leaders set up the system to accept files for a particular coursework via a few HTML pages on SIS. This action can be performed only by the course leader. The system does not issue receipts for submitted files, which does lead to disputes between students and staff over whether files were submitted or not.

1.2.3 “submit”

The School used to employ a text-based program called “submit” that could be run from a terminal on the School’s Linux machines. This program was tied to the Linux machines, which meant that in order to submit files from other machines the user had to transfer files to their Linux directory then remote log-in to run the “submit” program.

Under this system, one person had control, and course leaders would e-mail their requirements to that person to set up the system to accept submissions for a particular coursework. Use of “submit” program has now been discontinued.
1.3 The Proposed New System

1.3.1 Features of the new system

The new system stores all the files and organisational data in a database. The database holds details of students’ module registrations, markers for modules and courseworks for modules as well as the submitted files. By centralising all this information, the database aims to provide a “one stop shop” for course leaders to set and manage courseworks, and for students to submit and manage files that relate to these courseworks. Active rules can be written for the database that distribute the files to markers and inform the markers when the files are ready to be marked.

The new system aims to provide a highly usable interface to enable students to submit and manage files from any machine connected to the Internet. Previous systems have concentrated only on the action of submitting the files. The new system allows students to download and delete submitted files. These additional features offer the student more freedom to manage their files. A student can ensure that the files they have submitted are correct by downloading them. The student could then confirm that the contents of the file held by the system are as they intended. If a student were to have a last-minute thought about how they might improve their submission, they could approach any computer, access their submitted file, download it, make the necessary alterations and replace the file, all through a simple user interface. The interface also allows students to download the specification for courseworks.

The new system also provides a simple interface for course leaders to give them freedom to set and alter courseworks as the need arises. Module registrations can be managed via this interface. The application allows the assignment of markers to modules, so the marking can be distributed when the deadline arrives. Courseworks can also be set through the user interface. Course leaders can upload the specification for a coursework and set the deadline. Markschemes can also be uploaded, for distribution with the submitted files when the deadline has passed. The application also allows course leaders to query the submissions, so they can see which students have submitted, which have not submitted and which submitted late. By making the system easy to use these features could save course leaders a great deal of time.

1.3.2 Formal system requirements

The requirements of the system were decided upon by examining the description of the problem domain, and extracting the necessary and desirable features of a coursework submission and management system. The major stakeholder for the project was Nick Efford, the project supervisor. Nick was able to describe features that would be useful to him as a course leader, and several of these features have been incorporated into the requirements.
1.3.2.1 Functional requirements

The functional requirements of a system specify the tasks that the system produced by the project will perform [4] (p 14). The functional requirements of the new system are listed below:

1. The system shall allow students to view a list of the files they have submitted for a particular coursework.
2. The system shall allow students to submit files for courseworks.
3. The system shall allow students to delete submitted files.
4. The system shall allow students to download submitted files.
5. The system shall allow students to download coursework specifications.
6. The system shall issue receipts for submitted files.
7. The system shall allow course leaders to add courseworks for modules.
8. The system shall allow course leaders to edit coursework details.
9. The system shall allow course leaders to remove courseworks.
10. The system shall allow course leaders to allocate markers to modules.
11. The system shall allow course leaders to allocate students to modules.
12. The system shall produce a list of students who have submitted files for a particular coursework.
13. The system shall produce a list of students who have not submitted files for a particular coursework.
14. The system shall produce a list of students who have submitted files late for a particular coursework.
15. The system shall distribute the submitted files to markers for marking.
16. The system shall inform markers when there are files ready to mark.

1.3.2.2 Non-functional requirements

The non-functional requirements, or quality requirements, specify other requirements of the system. These requirements do not specify tasks that the system will perform, but describe how the system will perform certain tasks [4] (p 14). Non-functional requirements include such considerations as security...
The non-functional requirements of the new system are listed below:

1. The system shall allow students to manage submissions from any computer running any operating system, provided that that computer has an Internet connection.

2. The system shall store the files in such a way that they can be accessed only by course leaders and the student who submitted them.

3. The system shall have sufficient capacity to store at least one semester’s electronic submissions.

4. The system shall be capable of handling concurrent access.

5. The system shall allow course leaders to manage submissions and marking from any computer running any operating system, provided that computer has an Internet connection.

6. The folders to which the files are copied for marking can only be accessed by the relevant markers.

1.3.3 Methodology for the process

The first task in the creation of the system is to build the database. It is essential that this is in place before development of the other applications, because these applications need to interface with the database. The database is designed using entity-relationship modelling and is implemented using the PostgreSQL 7.1 database management system, which is the version of PostgreSQL running on the School’s database server, csdb, at the time of writing. Chapter 3 documents the creation of the database.

Once the database is in place, development of the coursework submission application can begin. Chapter 4 documents the design and implementation of this application, which is a browser-based solution using server-side scripting written in Perl.

The next stage in the project is the development of the coursework management application. This application is written in Java, which has useful packages for database connection and graphical user interfaces. The process of creating this application is described in Chapter 5.
Chapter 2

Project Management

This chapter contains the project plan, as set out at the commencement of the project, and justifies changes to the plan.

2.1 The Project Plan

The project plan appears here as it appears in the mid-project report. The presentation of the plan is altered slightly, but the contents remain unchanged.

2.1.1 Objectives

To achieve the aim of the project, a set of objectives have been devised that will drive the project towards its aim. Accomplishment of the whole set of objectives would represent completion of the project.

1. Create a database to manage and store the submissions.
2. Create a client application to allow students to submit coursework solutions to the database.
3. Create a client application to allow staff and markers to access and manipulate the contents of the database as required.
4. Address issues of security to ensure that relevant content is only accessible by the correct users.
5. Add functionality to the database to assign marking tasks to markers.

6. Produce documentation for the applications to enable deployment of the system.

7. Test the system thoroughly to evaluate its correctness.

8. Evaluate the system as a whole to ascertain how it might improve the current process of coursework management.

2.1.2 Minimum requirements

The minimum requirements are designed to build a solid base to which functionality can be added. Completion of the minimum requirements alone would leave the project in a state where the product would not solve the problem, but would leave the project in a state where it could be expanded upon in the future. The following requirements are addressed as a priority:

1. There will be a central database, with fully justified design documentation.

2. The system will allow students to submit pieces of work to the database from remote sources.

3. The system will only allow trusted parties to access permitted subsets of the data.

4. There will be documentation for the application programs.

2.1.3 Possible enhancements

On completion of the minimum requirements, work will begin on the following enhancements:

1. Functionality will be added to the management application to enable course leaders to manage coursework details and assign markers to modules.

2. The application programs will be given graphical user interfaces.

3. Active rules will be implemented within the DBMS to manage the process of coursework marking. An example would be to send an e-mail to the markers of a particular coursework to inform them that work is ready to mark.

4. A solution will be found to the problem of archiving coursework submissions.

5. An investigation will be conducted into the practicalities of integrating the system with SIS records to maintain concurrency between the two systems.
2.1.4 Deliverables

To achieve the requirements and enhancements, it is necessary to have milestones, which, when reached, will demonstrate the completion of a requirement or enhancement. The following deliverables will be produced over the course of the project. Deliverables marked “*” are software deliverables, which will not be included in the report, although if they have been achieved, evidence of their accomplishment will be documented.

1. Logical database design documentation.
2. An implementation of the database. *
3. An application to allow students to submit coursework to the database. *
4. An application to allow staff and markers to make use of the system. *
5. A review of access issues which demonstrates the security of the system at various levels.
6. Documentation for the system.
7. Design documentation for a set of active rules to manage workflow.
8. Implementation of active rules within the database. *
10. A brief report investigating the possibilities of integration of the system with SIS.

Satisfaction of the minimum requirements is demonstrated by the production of deliverables 1, 2, 3, 5 and 6. Enhancements 1 and 2 will be satisfied on completion of the first 6 deliverables.

2.1.5 Project schedule

The following schedule shows the time frame for the production of deliverables. If this schedule can be adhered to, the project will be completed on time.
<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Description</th>
<th>Due date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Logical database design documentation</td>
<td>10 December 02</td>
</tr>
<tr>
<td>2</td>
<td>Database implementation</td>
<td>31 December 02</td>
</tr>
<tr>
<td>3</td>
<td>Client application program</td>
<td>31 January 03</td>
</tr>
<tr>
<td>4</td>
<td>Management application program</td>
<td>20 February 03</td>
</tr>
<tr>
<td>5</td>
<td>Security review</td>
<td>28 February 03</td>
</tr>
<tr>
<td>6</td>
<td>System documentation</td>
<td>7 March 03</td>
</tr>
<tr>
<td>7</td>
<td>Active rule design</td>
<td>21 March 03</td>
</tr>
<tr>
<td>8</td>
<td>Active rule implementation</td>
<td>28 March 03</td>
</tr>
<tr>
<td>9</td>
<td>Archiving solution</td>
<td>15 April 03</td>
</tr>
<tr>
<td>10</td>
<td>SIS synchronisation report</td>
<td>22 April 03</td>
</tr>
</tbody>
</table>

Writing-up of the report will progress throughout the report. The schedule leaves over a week before the deadline to tidy up the report and perform an evaluation of the solution.

### 2.2 Revisions to the plan

It became clear early in the project that the deliverables were running behind schedule. Implementation of the database was delayed until after Christmas, and technical difficulties caused completion of the client application to overrun by several weeks. For this reason, deliverables 7 and 8, concerning active rules to distribute the files to markers, were omitted from the project. The remaining deliverables were tackled in the order that they appear in the schedule. Lack of time has also led to a lack of formal testing for the use cases added to the coursework management application.
Chapter 3

Building the Database

In this chapter the process of creating the database to hold submissions is described in full, from high-level analysis of the universe of discourse, through to the data definition in PostgreSQL 7.1. The final section describes how the database can be set up using the files supplied with this report.

3.1 Entity-Relationship Modelling

The accepted method for relational database design is to use entity-relationship (ER) modelling [9] (p 23). The method to derive the model involves study of a text description of the universe of discourse (the environment in which the data to be stored resides). From this description, the main entities and the relationships between them are identified. The nature of the relationships are formalised, and an entity-relationship model can be produced. This model maps directly to a set of logical schema, from which the database schemata can be easily derived. This stage of the project is very important, as well-modelled databases are easier to query, maintain and adapt, should the need arise.

3.1.1 Identification of entities and relationships

The process begins with a text description of the universe of discourse. The candidate entities are the nouns in the description. The final set of entities in the model should be a subset of these. The universe of discourse can be described at high-level as follows:
Students are registered for modules. The modules have a number of courseworks. Students make attempts at the courseworks for registered modules. Each attempt will involve the submission of a number of files. Each module has a number of markers who mark the attempts at courseworks for that module.

Candidate entities: Student; Module; Coursework; Attempt; Submission; File; Marker.

By formalising the relationships between these entities, this set of candidate entities can be reduced to the definitive set of entities. The relationships identified are listed below:

Students are registered for modules. This is a many-many relationship, since students can be registered for more than one module, and modules can be taken by more than one student. Non-participation in the relationship can be allowed for either entity, to allow students and modules to be included in the database before module registration has been defined.

Modules have courseworks. This is a one-many relationship, as a module can have more than one coursework but each coursework belongs to a specific module. Non-participation can be allowed for courseworks to allow modules to exist in the database before courseworks have been decided, i.e. each module may have zero or more courseworks.

Courseworks have attempts. This is also a one-many relationship, because each attempt is at only one coursework, but each coursework may have many attempts. Each coursework may have zero or more attempts.

Students make attempts. This is another one-many relationship. Each attempt is made by a specific student, but each student will make more than one coursework attempt (there should be one for each coursework for the modules for which they are registered). Each student may make zero or more attempts.

Attempts have files. This is also a one-many relationship. An attempt must have at least one file, but may have a number of files. Each file relates to a single attempt.

Markers mark for modules. This is a many-many relationship. Modules may be marked for by more than one marker, and there is nothing to stop markers marking for more than one module. Non-participation is allowed for each entity.

Markers mark attempts. This is a one-many relationship. Markers will mark more than one attempt, but each attempt has a specific marker. Markers may be associated with zero or more attempts.

These relationships and the entities involved in them can be drawn up into an ER diagram. Since these relationships can be used to define all those described in the universe of discourse, only the entities
participating in these relationships need be retained. The candidate entity Submission can be discarded. The final set of entities is as follows: Student; Module; Coursework; Attempt; File; Marker.

3.1.2 Entity attributes

Each of the entities has a set of attributes which help to identify and differentiate between the different instances of the entities. It is helpful to decide on these attributes before the model is drawn up, as this makes the model more complete. It is intended that the set of attributes used in the model should be as minimal as possible whilst maintaining all the required functionality. It is not necessary to store, for example, a student’s address in the database, as this information is not pertinent to the process of submitting files for courseworks. The attributes to be stored for each entity are listed below:

- **Student** - username, name
- **Module** - module code, title
- **Coursework** - coursework number, deadline, specification, markscheme
- **Attempt** - mark
- **File** - filename, file, timestamp
- **Marker** - username, name

The entities Student, Module and Marker are strong entity types as they each contain a key that uniquely identifies each instance. Students and markers have a unique School of Computing username, and each module has an identifying module code. Coursework, Attempt and File are weak entity types. They do not contain a key which uniquely identifies them. Attempt, for instance, has only one attribute mark. Obviously it is entirely possible for more than one attempt to have the same mark. The mark only has real meaning when it is known which student owns the attempt, and which coursework the attempt belongs to. Weak entity instances have little meaning without ownership. Coursework is owned by Module, Attempt is owned by Coursework and Attempt, and File is owned by Attempt.

3.1.3 The model

The entities and relationships can now be drawn up into the final ER model (see figure 3.1). The “hen’s foot” notation used is a departure from the Chen notation found in most database texts. This notation was introduced in the DB11 course notes [6]. The model is complete and shows all entities (rectangles), relationships (arcs between entities) and attributes (ellipses).
Figure 3.1: The final ER model for the database.

3.2 Table Schemata

There are two steps in the creation of the database from the ER model. The first step is a mapping to a logical schema which could be implemented on any relational DBMS. The second step is a mapping from this logical schema to a set of DBMS-specific data definition statements which will define the tables and referential integrity constraints that make up the database.

3.2.1 Mapping the model to a set of logical schema

This is performed using ER-to-Relational mapping as described by Elmasri and Nevathe [3] (pp 290-295). There are seven steps in the process:

Step 1: Create a relation for each strong entity type including all its attributes. Choose one of the key attributes as the primary key for the relation (primary keys are underlined). This give us three relations:

Student (username, name)

Module (code, title)
Marker (username, name)

Step 2: Create a relation for each weak entity type, including all its attributes. Include as foreign key attributes all primary key(s) of the owner entities. The primary key is defined by the combination of foreign key attributes and the partial key of the type itself, if it has one. This step yields a further three relations:

Coursework (code, cwno, deadline, specification, markscheme)

Attempt (Student.username, code, cwno, mark)

File (Student.username, code, cwno, filename, timestamp, file)

Step 3: This step concerns binary 1:1 relationships. There are no 1:1 relationships in the ER model.

Step 4: Identify binary 1:N relationships in the model, and post the primary key from the 1 to the N, including it as a foreign key. This step adds the Marker.username attribute to the Attempt relation. All the other 1:N relationships are dealt with at step 2:

Attempt (Student.username, code, cwno, mark, Marker.username)

Step 5: Identify binary M:N relationships and create new relations to represent these relationships. Include as foreign key attributes the primary keys of the participating entities. The primary key is the combination of these two foreign keys. This step yields the final two relations.

Registered (Student.username, code)

Marks (Marker.username, code)

Step 6: This step concerns multivalued attributes, of which there are none in this model.

Step 7: This step concerns n-ary relations where n is greater than 2, and is not applicable for this model.

The complete logical schema of the database is listed below:

Student (username, name)

Module (code, title)

Marker (username, name)

Coursework (code, cwno, deadline, specification, markscheme)
3.2.2 Derivation of PostgreSQL data definition statements

To map the logical schema of the database to one which will satisfy the requirements, it is necessary to take into account certain features of the DBMS. Each attribute needs to be assigned a suitable data type offered by the DBMS that is capable of holding the information. Permissions must be set to allow only the correct subsets of data to be accessed or modified by certain users.

PostgreSQL 7.1 offers a VARCHAR data type, that can be used to hold variable length character strings up to the length supplied as the parameter. This is suitable for storing usernames, names, module codes and titles, coursework titles and filenames. The SMALLINT data type holds signed 2-byte integers, and can be used to store coursework numbers and marks. The DATETIME data type holds a date and time and is suitable for the storage of coursework deadlines and file submission times (timestamps). The BYTEA data type holds binary data of arbitrary length. This is ideal for the storage of file information, so it can be used to hold the submitted files and coursework specifications and markschemes.

The DBMS offers an alternative storage device for binary data - the Binary Large Object (BLOB). BLOBs are handled differently, and are stored in a separate table in a special format. The DBMS refers to these records using an Object Identifier (OID). The tables that hold BLOBs are not as secure as the main database tables, where BYTEAs are stored. For this reason, the BYTEA data type is chosen to store binary data.

PostgreSQL 7.1 offers only table-level and attribute-level granularity on permissions [7]. This has serious implications for this application, because in the logical schema all files are stored in the same relation. If permission were granted to all students on the File relation, they would be able to access all the files in the database, which is undesirable. Any student would be able to log directly into the database using psql at the command line and select any file. For this reason it is necessary to give each student their own table (filestore) in the database. The File relation is split into one relation per student, with identical attributes apart from the omission of Student.username, which is no longer required since all files in the relation belong to the student owning the relation. These new relations can be given the names of the students’ usernames, and represent the students’ filestores. The permission for each of these tables can then be set for access only by the relevant student.

Omission of the the Student.username attribute means that the relation can no longer reference the
Attempt relation, since Student.username is part of the foreign key from the Attempt table. Entries in the filestore relation must now reference the coursework relation, which has as its primary key code and cwno. This is an acceptable alteration, since there can only be one entry per student in the Attempt table for each coursework.

Some additional attributes must be added for the storage of files in the database. The coursework relation requires extra attributes to store the filename, timestamp and MIME type of the files containing the coursework specification and markscheme. The MIME type of the files must also be stored for each entry in the students’ filestore tables. These attributes are stored to identify the file contents and to allow them to be extracted correctly from the database.

The DBMS requires that all primary and foreign keys are defined for the tables so it can set up referential integrity checking. This is all the information that is required to write the data definition statements to build the database. These statements can be found in the file data_definition.sql, which can be found on the disk accompanying this report. The table ctxwjc is a test table under my own username for the modified File relation, and shows the format of all the filestore tables. The final statement tells the database to store dates in the form DD/MM/YYYY, as is the convention in Britain.

3.3 Normalisation

All tables in the database are in Boyce-Codd Normal Form (BCNF). Space in this report is at a premium so this is not demonstrated explicitly. The normalisation process did not require any alterations - all the relations were already in BCNF.

3.4 Integrity Constraints and Business Rules

3.4.1 Foreign key dependencies

All the primary keys and foreign keys are defined in the creation of the tables. PostgreSQL 7.1 takes care of maintaining referential integrity. The checks are set up automatically when the tables are defined.

3.4.2 Input validation

The applications that are to link to this database are both basically query builders. They take the module codes and coursework numbers from the database and allow the user to select from these. This means that input validation is performed by the client applications. Provided the students, modules and module registrations are set up correctly by the database administrator, the client applications should only allow
the user to select valid data. The two client programs therefore need to be designed in such a way that it is impossible to insert an invalid tuple to the database.

3.5 Building the Database

The files described in this section can be found on the disk accompanying this report.

To begin building the database, the file data_definition.sql is executed on the database. This sets up the student, module, marker, registered, marks, coursework and attempt tables. All the tables are unpopulated. The file module_pop.sql contains the SQL to add all current modules to the module table. This file may need editing.

A separate file student_template.sql (see Appendix B), is used to add the students. The file must be modified to include the student’s name, username and modules where indicated. The purpose of this file is to add the student to the student table, register the student for their modules, add a table to hold their files and set database permissions for that user. A script could be written to extract this information from the School Information System and generate the SQL to perform these insertions.

The file coursework_pop.sql adds three courseworks for each of the Level 1 modules. This is useful for testing the coursework submission application as the procedure to add courseworks is part of the coursework management application, which will be developed last. At this stage in the build, students should be able to upload, download and delete files for these dummy courseworks via the submission application.

N.B. All prospective users must be added to the DBMS as users. This enables permissions to be set for those users. Students must be added before using student_template.sql, as this file attempts to set table permissions for the user.
Chapter 4

Coursework Submission Application

This chapter describes the process of creating the coursework submission application, which allows students to submit files for marking, in addition to some other features.

4.1 Requirements

The requirements of the application are listed below:

1. The application shall enable management of submissions from any computer running any operating system, provided that that computer has an Internet connection.

2. The application shall allow users to view a list of the files they have submitted for any coursework for modules for which the user is registered.

3. The application shall allow users to upload files to their file store in the database.

4. The application shall allow users to delete files from their file store in the database.

5. The application shall allow users to download files from their file store in the database.

6. The application shall allow users to download coursework specifications.

7. The application shall ensure that receipts for submitted files are issued to the user.
These requirements are a subset of the system requirements from chapter 1. This application realises this subset. The requirements have been reworded slightly from those in chapter 1, as the method of storage for the files has been decided, which allows the nature of the requirements to be made more explicit.

4.2 Architecture

The application is built on a three-tiered client-server architecture. The presentation of the information is dealt with by the client machine’s browser. The application logic is handled by the scripts on the web server. The data manipulated by the scripts is held on the database server. This maps exactly to the PAD (Presentation, Application, Data) model for a three-tiered architecture, with each of the three layers residing on a separate machine.

The presentation layer communicates with the application layer via HTTP form data and cookies. Return communications are in the form of HTML documents that can be displayed in the browser. The application layer sends SQL statements to the data layer and result sets are returned (see figure 4.1).

![Figure 4.1: Deployment of the coursework submission application.](image)
4.3 Design

The decision to use a browser-based solution with server-side scripting helps to satisfy requirement 1. So long as the scripts are placed on an accessible web server, students will be able to use the application from any location.

To understand the design of the application, it is helpful to visualise the user’s modules, those modules’ coursework and each coursework’s files in a hierarchy. The user starts at the top of the hierarchy and may move down through it one level at a time following the edges between nodes. They may also move straight to any node in the same level provided it has the same parent node, e.g. if the user is in AR11 coursework 1 they may move directly to AR11 coursework 2. The user may begin again from the top of the hierarchy at any stage by selecting a different module.

A recursive server-side script enables the user to traverse this hierarchy. Pseudo-code for this script can be found in figure 4.2. By nesting the procedures to process the files (lines 14-30) inside the procedure to select a coursework (lines 9-31), and nesting this inside the procedure to select a module (lines 6-32), we can guarantee adhesion to these simple rules. The leaf nodes (files for a particular coursework) are displayed when the user selects a coursework number. This recursive script fulfils requirement 2.

From the page displaying the leaf nodes for a particular coursework, the user has a number of options, each of which is implemented by a separate server-side script. Firstly, they may upload a file to their file store in the database for this particular coursework (add a leaf node). The user browses for the file they wish to submit, then clicks a submit button to send the file to a script which handles its insertion into the database. This script satisfies requirement 3.

If the user has submitted files for the coursework, a “Delete” button appears next to each file. A click of this button triggers another script, which handles the deletion of the file from the database (remove a leaf node), fulfilling requirement 4. Another button marked “Download” appears by each file (copy a leaf node back to the client), a click of which triggers a script that sends the file back to the client machine (requirement 5). Another button triggers a similar script to handle the downloading of the coursework specification (requirement 6). A final button issues a receipt for the files currently held in the filestore (requirement 7).

4.4 Design Choices

4.4.1 Scripting language

There are a number of server-side scripting languages that could be used to implement this browser-based application, including Perl, Python and PHP. All these languages have modules that can be in-
Figure 4.2: Pseudo code for the main recursive script.

Get form data (username, password, code, cwno);
if (username && password) Set cookies;
else Retrieve cookies;
Connect to the database;
Get user’s modules; Display modules in a list;
  if (module selected at any time...)
    Post code back to this script;
if (code) {
  Get module’s courseworks;
  Display courseworks in a list;
    if (coursework selected at any time...)
      Post code and cwno back to this script;
    if (cwno) {
      Get files submitted for this coursework;
      for (each file returned) {
        Write a table row containing:
          filename, timestamp,
          button to delete,
            if (clicked at any time...)
              Post code, cwno, filename,
                timestamp to delete.cgi;
            button to download,
              if (clicked at any time...)
                Post code, cwno, filename,
                  timestamp to download.cgi;
        }
      Write file field to upload a file.
      if (file submitted at any time...)
        Post code, cwno, file to upload.cgi;
      Option to download coursework specification;
    }
}

cluded to facilitate tasks such as database connection and querying, generation of web pages, cookie handling and base64 encoding. The choice of scripting language, Perl, seems somewhat arbitrary, but it
has all the necessary features required to implement the application.

4.4.2 Cookies

State in the application is maintained by session-only cookies, which store the user’s username and password for the duration of the browser. The information contained in these cookies ensures that the script can connect to the database as the correct user, so the correct database records are displayed and edited. Storing this information client-side means that the username and password need not appear in the HTML source returned from the scripts, as the scripts can access this information from the cookie store on the client machine. In the initial implementation, the username and password appeared in the source as hidden form information. Session-only cookies are destroyed when the browser is closed, so, provided that users do not leave the browser open and unattended, their files are safe.

4.4.3 CGI

Perl’s CGI (Common Gateway Interface) module is designed for dynamic creation of web pages. It has many functions that help to generate fragments of HTML, and is particularly useful when generating tables and forms. The HTML could be generated using just \texttt{print} statements, but every single character of the HTML would have to be produced explicitly. The functions of the CGI module generate HTML tags automatically, which helps to hide the complexity of the script being written. The scripts in this application are written using a mixture of CGI function calls and explicit \texttt{print} statements.

4.4.4 DBI

Perl has a module called DBI which can be used to handle communications between the scripts and the database. A DBMS-specific DBI driver (DBD.pg) must also be included. This driver converts the generic DBI methods into native PostgreSQL commands. The DBI module makes querying the database from remote Perl scripts a simple task. DBI’s \texttt{connect()} function is passed the URL of the database and a username and password, returning a database handle object. The database handle has a \texttt{prepare()} function, which returns a statement handle object. The \texttt{execute()} function is called on the statement handle, and the rows of the query results can be accessed using the \texttt{fetchrow_array()} function.

4.4.5 base64 encoding

Binary data, such as that contained in the files that are to be sent to the database, should be encoded using base64 encoding or quoted-printable encoding [10] (p 654). The chosen method of encoding for this
application is base64 encoding, which translates binary data into a code which uses 64 characters: the 26 uppercase letters; the 26 lowercase characters; the ten digits; and the “+” and “/” characters. Using this method of encoding increases the size of the data to be sent, but guarantees that the information will be error-free after transmission. The limiting of the stored data to these 64 characters means that the file data can be passed through the scripts to the database as a single Perl scalar, because it is guaranteed not to include the “’” character, which would terminate the string of data being inserted into the database. It is important to remember that the data must be decoded once it has been extracted from the database to restore the it to its original form.

4.5 Implementation

The files described here are included on the disk accompanying this report.

4.5.1 HTML index page - index.html

The implementation begins with a simple index page written in HTML. The page contains a form prompting the user to enter their username and password for the system. The contents of this form are posted to connect.cgi that resides in the cgi-bin on the web server. A screen grab of the index page produced by this source is shown in figure 4.3.

Figure 4.3: Screen grab showing index page of coursework submission application.
4.5.2 Recursive Perl script - *connect.cgi*

The recursive script begins by looking for form information containing a username and password. If it finds this information, it knows that this is a new session, so cookies are sent to the client machine in the HTML header to hold this information. If it does not find this information, it knows it is dealing with an existing session, so the username and password cookies are retrieved from the client. A connection is then made to the database.

Provided the connection is successful, the user’s name is retrieved from the *student* table and the user is welcomed. The user’s modules are retrieved from the *registered* table and presented as a list. If the user selects from this list, the module code is posted back to this script, which begins again and generates a new page.

If the script has received a module code, it retrieves the coursework numbers for that module from the *coursework* table and presents them as a list. Selection from this list causes the module code and coursework number to be returned to this script.

If the script has received both a module code and a coursework number, it can retrieve the filenames and timestamps of any files that have already been submitted for this coursework. This information, if any is returned, is displayed in a table, with filename in the first column, timestamp in the second, a button to delete the file in the third, and a button to download the file in the fourth. Clicking either of these buttons will cause the module code, coursework number, filename and timestamp to be posted to *delete.cgi* or *download.cgi*, as appropriate. Following the table is a file field, where the user can browse for files to submit for this coursework. A click of the *Submit* button posts the selected file, module code and coursework number to *upload.cgi* as *multipart/form-data*. An option to download the coursework spec follows, which is not implemented due to time constraints. The receipt button does not appear for similar reasons.

Figure 4.4 shows how the page appears in the browser when a module and coursework number have been selected and two files have been submitted for the coursework.

4.5.3 Upload script - *upload.cgi*

The script retrieves the cookies containing the username and password from the client machine, and extracts the *multipart/form-data* sent by *connect.cgi*, which contains the module code, coursework number and the selected file. A connection is made to the database and a query discovers whether there is an entry in the *attempt* table for this user for this particular coursework. If this is the first file to be submitted for this coursework, an insertion is made to the *attempt* table containing the username, module code and coursework number. The filename and MIME type are extracted from the submitted file, and the file contents are read into the script and base64 encoded. An insertion is then made to the
user’s filestore containing the encoded file and its referential information. The value “now” is inserted to the timestamp field, which is translated to the current date and time by the DBMS.

4.5.4 Delete scripts - delete.cgi, deleted.cgi

The file delete.cgi retrieves the cookies from the client machine and gets the form data (module code, coursework number, filename and timestamp) which are sent from connect.cgi. Two buttons offer the user the choice of either deleting the file, which passes the form data on to deleted.cgi, or returning to the previous page, which is performed by posting the module code and coursework number.
back to connect.cgi. Figure 4.5 shows how this page appears to the user.

![Screen grab showing the options offered by delete.cgi.](image)

When deleted.cgi receives form data from delete.cgi, it retrieves the cookies, then deletes the appropriate file from the user’s filestore in the data. The user is then informed of the deletion, and offered a route back to connect.cgi.

### 4.5.5 Download script - download.cgi

This script selects the mime type and base64-encoded file information from the database. An HTTP header is constructed, and the decoded file information is returned.

### 4.6 Testing

The application is set up to run on the School’s web server, and can be accessed at the following address, at least until the end of June, 2003.

http://wwwdev.comp.leeds.ac.uk/ctxwjc/

The username ctxwjc and password kazwoz allow access to the database for testing.

The upload capabilities of the application has been extensively tested using many different file formats,
including source code (text files), executables (compiled source code), image files including .jpeg and .png files, and application files such as MS Word documents. It is able to store all these formats without a difficulty.

### 4.7 Documentation

One of the minimum requirements of the project is documentation for the client application. These do not appear in the report for this application, because the application is self-documenting. At every step in the application, all options available to the user are made explicit.

To set up the application, the index page should be placed on a webserver. The Perl files should be placed in a *cgi-bin* folder where they can be executed. The Perl files are hard-coded to connect to a development database (*ctxwjc* running on *csdb.leeds.ac.uk*). In a deployment of this software, the URL passed to the *connect()* function would have to be altered in the scripts to reference the real database. Paths to the other files in the application should also be checked.
Chapter 5

Coursework Management Application

This chapter describes the process of creating the coursework management application, which allows course leaders to manage registration of students, allocation of markers and assignment of courseworks.

5.1 Requirements

The requirements of the application are listed below:

1. The application shall allow course leaders to manage submissions and marking from any computer running any operating system, provided that that computer has an Internet connection.

2. The application shall allow course leaders to add courseworks for modules.

3. The application shall allow course leaders to edit existing courseworks.

4. The application shall allow course leaders to remove courseworks.

5. The application shall allow course leaders to manage allocation of markers to modules.

6. The application shall allow course leaders to manage registration of students for modules.

7. The application shall allow course leaders to view a list of students that have submitted for a particular coursework.
8. The application shall allow course leaders to view a list of students that have not submitted for a particular coursework.

9. The application shall allow course leaders to view a list of students that were late in submitting for a particular coursework.

These requirements are a subset of the system requirements from chapter 1. This application realises this subset.

5.2 Methodology

The application connects directly to the database on the database server. The presentation and application logic is all performed on the client-side, so it is a two-tier client-server architecture, with a fat client and thin server (see figure 5.1).

![Figure 5.1: Architecture of the management application.](image)

Since all the functionality of the system requires that the application be connected to the database, either to query or manipulate the data contained in it, the application must be capable of establishing this connection. To make this connection, a username and password must be provided, so the application must provide the user with a means of supplying this information. The requirements are all pertinent to
either a particular module or a particular coursework, so features allowing the module and coursework
to be selected are also required.

Once these features have been developed, the functionality of the system can be modelled as a set of
UML use cases. The use cases can be implemented one at a time, so each version of the application will
have more functionality than the last. Because the data is held in secondary storage (in the database),
the use cases do not interfere with each other. Each one either gets some data or manipulates some data,
then terminates.

Using this methodology means that the design and implementation of the application cross over to a
certain extent, since the design of the graphical user interface to accept username and password and
allow module and coursework selection requires a large amount of coding to be performed. These
features can be thoroughly tested to ensure their correctness. This will provide a robust platform for the
addition of the use cases, which can then be added to the application sequentially.

5.3 Use Cases

A use case is defined as a sequence of actions performed by a system that yields an observable result
of value to a particular actor [1] (p 19). The requirements of the application can be separated into four
main categories in the use case analysis - each category dealing with a separate aspect of coursework
management. These categories can be used to separate the use cases between packages. These packages
are summarised in figure 5.2.

![Figure 5.2: Packages available to the course leader.](image)

In the following subsections, the packages are broken down into the specific use cases that are con-
tained within. This type of layered analysis helps to group together related actions, which is useful
when designing the user interface. Use case descriptions are given for each use case, which aid the im-
plementation of the functions. The *preconditions* are the conditions that must be true in order to attempt to execute the use case. *Additional input* refers to any other input that is required during the execution of the use case. The *query* shows the SQL query that is executed on the database under the main flow of events. The *exception paths* show the circumstances under which the query is not executed on the database. Some of the use cases return *results*, which must be presented to the user. Note that the *only* user of the system is the course leader.

### 5.3.1 Managing module registrations

To manage module registrations, the course leader must be able to view the students who are currently registered, and have methods for adding and removing students from modules. Figure 5.3 summarises the use cases contained in the *Manage module registrations* package.

**Figure 5.3: Use cases contained in the Manage module registrations package.**

<table>
<thead>
<tr>
<th>Use case</th>
<th>List registered students.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition</strong></td>
<td>Module selected.</td>
</tr>
<tr>
<td><strong>Query</strong></td>
<td>SELECT r.username, s.name FROM registered r INNER JOIN student s ON r.username = s.username WHERE r.code = module ORDER BY r.username</td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td>Display on screen.</td>
</tr>
</tbody>
</table>
### Use case
Add student to module.  

**Precondition**: Module selected.  
**Additional input**: Student’s username.  
**Query**: 
```
INSERT INTO registered VALUES (username, module)
```
**Exception paths**: Student already registered; Student does not appear in database.

<table>
<thead>
<tr>
<th>Use case</th>
<th>Remove student from module.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition</strong></td>
<td>Module selected.</td>
</tr>
<tr>
<td><strong>Additional input</strong></td>
<td>Student’s username.</td>
</tr>
<tr>
<td><strong>Query</strong></td>
<td>DELETE FROM registered WHERE code = module AND username = username</td>
</tr>
<tr>
<td><strong>Exception paths</strong></td>
<td>Student not registered.</td>
</tr>
</tbody>
</table>

### 5.3.2 Managing module markers

Managing the markers for a module requires similar options to managing module registrations. The course leader must be able to view current markers, add markers and remove markers. These use cases are shown in figure 5.4.

![Use cases contained in the Manage module markers package.](image)

Figure 5.4: Use cases contained in the *Manage module markers* package.
### Use case
List module’s markers.

**Precondition**
Module selected.

**Query**
SELECT m.username, n.name FROM marks m INNER JOIN marker
ON m.username = n.username WHERE m.code = module ORDER
BY n.username

**Results**
Display on screen.

### Use case
Add marker for module.

**Precondition**
Module selected.

**Additional input**
Marker’s username.

**Query**
INSERT INTO marks VALUES (username, module)

**Exception paths**
Marker already marks for module; Marker does not appear in database.

### Use case
Remove marker from module.

**Precondition**
Module selected.

**Additional input**
Marker’s username.

**Query**
DELETE FROM marks WHERE code = module AND username =
username

**Exception path**
Marker does not mark for module.

### 5.3.3 Managing coursework

To manage courseworks held by the database, the course leader must be able to add and remove course-
works, view details of courseworks, and edit properties of the courseworks. Figure 5.5 shows these use
cases.

### Use case
Add a new coursework.

**Precondition**
Module selected.

**Additional input**
Coursework number.

**Query**
INSERT INTO coursework (code, cwno) VALUES (module,
coursework number)

**Exception paths**
Coursework with this number already exists; Value input is not a number.
Figure 5.5: Use cases contained in the *Manage coursework* package.

<table>
<thead>
<tr>
<th>Use case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View coursework details.</td>
<td>Use case: View coursework details. Preconditions: Module selected; coursework number selected. Query: SELECT deadline, specfilename, spectimestamp, markfilename, marktimestamp FROM coursework WHERE code = module AND coursework number = coursework number. Results: Display on screen.</td>
</tr>
<tr>
<td>Edit coursework number.</td>
<td>Use case: Edit coursework number. Preconditions: Module selected; coursework number selected. Additional input: New coursework number. Query: UPDATE coursework SET coursework number = coursework number = coursework number WHERE code = module AND coursework number = coursework number. Exception paths: Coursework with this number already exists; Value input is not a number; Students have already submitted files for this coursework.</td>
</tr>
<tr>
<td>Use case</td>
<td>Edit specification.</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Preconditions</td>
<td>Module selected; coursework number selected.</td>
</tr>
<tr>
<td>Additional input</td>
<td>New specification file.</td>
</tr>
<tr>
<td>Query</td>
<td>Uses JDBC’s <code>setBytes()</code> method to insert the new specification file to the database.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case</th>
<th>Edit markscheme.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>Module selected; coursework number selected.</td>
</tr>
<tr>
<td>Additional input</td>
<td>New markscheme file.</td>
</tr>
<tr>
<td>Query</td>
<td>Uses JDBC’s <code>setBytes()</code> method to insert the new markscheme file to the database.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case</th>
<th>Remove coursework.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>Module selected; coursework number selected.</td>
</tr>
<tr>
<td>Query</td>
<td><code>DELETE FROM coursework WHERE code = module AND cwno = coursework number</code></td>
</tr>
<tr>
<td>Exception path</td>
<td>Students have already submitted files for this coursework.</td>
</tr>
</tbody>
</table>

5.3.4 Querying submissions

The last three requirements are included in the Query submissions package. There are three simple options here: list those students who have submitted files for a coursework; list those students who have not submitted files; and list those students who submitted late. The contents of the Query submissions package are shown in figure 5.6.

<table>
<thead>
<tr>
<th>Use case</th>
<th>List students who have submitted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>Module selected; coursework number selected.</td>
</tr>
<tr>
<td>Query</td>
<td><code>SELECT a.username, s.name FROM attempt a INNER JOIN student s ON a.username = s.username WHERE a.code = module AND a.cwno = coursework number ORDER BY a.username</code></td>
</tr>
<tr>
<td>Results</td>
<td>Display on screen.</td>
</tr>
</tbody>
</table>
Figure 5.6: Use cases contained in the *Query submissions* package.

<table>
<thead>
<tr>
<th>Use case</th>
<th>List students who have not submitted.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preconditions</strong></td>
<td>Module selected; coursework number selected.</td>
</tr>
<tr>
<td><strong>Query</strong></td>
<td>SELECT r.username, s.name FROM registered r INNER JOIN student s ON r.username = s.username WHERE r.code = module AND r.username NOT IN (SELECT a.username FROM attempt a WHERE a.code = module and a.cwno = coursework number) ORDER BY r.username</td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td>Display on screen.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case</th>
<th>List students who submitted late.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preconditions</strong></td>
<td>Module selected; coursework number selected.</td>
</tr>
<tr>
<td><strong>Query</strong></td>
<td>Query to select the relevant students and the latest timestamp in the file-store for the selected coursework.</td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td>Display on screen.</td>
</tr>
<tr>
<td><strong>Exception path</strong></td>
<td>Deadline has not been reached.</td>
</tr>
</tbody>
</table>

5.4 Software Choices

Java is chosen as the language in which to implement this application. Compiled Java code will run on any platform via a Java Virtual Machine (JVM), which means that course leaders can run the program from Linux and MS Windows platforms without the need to recompile the code. The cost of this
portability is that programs written in Java do not generally run as efficiently as those written in a comparable language, such as C++. The instructions must be interpreted by the JVM, which translates them into native system calls. Java programs also tend to use more system resources, as the JVM must be loaded before the programs can run. For an application of this size, the pros of portability and ease of development outweigh the cons described here.

The Java programming language has many packages that can be imported to provide the user with fully implemented and tested objects designed to fulfill certain tasks. The `java.sql` package provides the objects required to use Java Database Connectivity (JDBC). JDBC uses a DBMS-specific driver to establish a connection to the database. SQL statements can be prepared and executed, and results from the database are returned to the client as `ResultSet` objects, from which the required data can be extracted.

The `java.awt` (Abstract Windowing Toolkit) and `javax.swing` packages enable the creation of Graphical User Interfaces (GUIs). `java.awt` provides the bridge between the native platform’s windowing system and the JVM, whilst the `javax.swing` package supplies GUI components such as buttons and text fields [2] (p 649).

5.5 Design of the GUI

The main panel of the GUI contains fields for the user to enter a username and password, using `JTextField` objects. An uneditable `JTextField` below these fields shows the current status of the application (Connected/Disconnected). JButton objects to connect and disconnect the application are positioned below. ActionListener interfaces are implemented for these buttons. The `ConnectHandler` takes the contents of the Username and Password fields and attempts to establish a connection to the remote database. If successful, the module codes are retrieved from the module table and displayed in `moduleList`, a `JList` object, which is situated to the right of the connection features. Selection from this list activates the `ModuleHandler` inner class, which implements the `ListSelectionListener` interface. The coursework numbers for the selected module are retrieved from the coursework table and displayed in another `JList`, `courseworkList`, and a variable `module` is assigned with the code of the selected module. Selection from `courseworkList` activates the `CourseworkHandler` class, which sets a variable coursework with the number of the selected coursework. The Disconnect button activates `DisconnectHandler`, which drops the database connection and clears the contents of the `JTextField` and `JList` elements. This part of the application provides the user with the means to connect to the database and select modules and coursework numbers. The layout of the components is shown in figure 5.7.Situated below the components described above is a large `JTextArea`, which is used to display query results. This has been cropped from the figure.

To avoid cluttering the GUI, the operations that can be performed on the data (the use cases) are placed
in JMenus. When the user clicks on the words in the menu bar at the top of the GUI, the use cases contained in that menu drop down in a list, as shown in figure 5.8. ActionListener interfaces are implemented for each of the use cases. Each menu represents one of the packages of use cases detailed above. This groups together related actions, which makes the application simple to use.

Figure 5.8: Screen grab showing JMenu containing Coursework options.

5.6 Implementation of Use Cases

The implementation of each of the use cases appearing in the drop-down menus is performed in the following way. An inner class that implements the ActionListener interface is added to Work.java after the existing inner classes. The actionPerformed() function of the ActionListener is implemented thus. An if statement tests the connection to the database. If the connection is null a JOptionPane informs the user that the application is not connected and the function ends. If the connection is not null, the preconditions of the use case are tested. If the preconditions are false, the user is informed via a JOptionPane, and the function terminates. If the preconditions are true, the function will gather any required additional information from the user, again using a JOptionPane. The database is queried to ensure that none of the exception paths of the use case are being followed. In exceptional circumstances, the user is informed, otherwise the query of the use case is executed on the database. Any results returned from the query are displayed in the GUI’s JTextArea.

Once implemented, each use case can be tested before moving on to the implementation of the next,
ensuring a smooth, incremental build, with each stable version having more functionality than the last. Not all the use cases described in this chapter have been implemented due to time constraints.

5.7 Using the application

Thorough testing of the implemented use case has not been possible due to time constraints. However the code to implement this application can be executed by compiling and running the file Work.java, which is included on the disk supplied with this report. The JDBC driver, jdbc7.1-1.2.jar, is also included on the disk. The classpath must be set to allow access to the classes in this archive.

The username and password required to connect to the database are ctxwjc and kazwoz, respectively.
Chapter 6

Additional features

This chapter investigates some features that are not addressed by the work done in this project, but would make the system more useful to the School.

6.1 Workflow Management

Workflow management has been suggested as an application for active databases [8]. An active database is one which allows for the specification active rules [3] (p 734). Active rules can be specified in PostgreSQL 7.1 using triggers. An active rule has three parts:

1. The event of the rule is the event that causes the rule to be triggered. This event can be an insertion, deletion or update to a table, or a temporal event, such as, for example, the arrival of a coursework deadline.

2. The condition of the rule is the condition that is tested to see whether the action of the rule is executed. This condition is optional. If no condition exists, the action of the rule is executed.

3. The action of the rule is the action taken if no condition is specified or the condition of the rule evaluates to true. The action can be a series of SQL statements, a database transaction, or the triggering of an external program.
An active rule could be written to be triggered in the event of a coursework deadline passing. The action of this rule could have three parts:

1. Divide the attempts at the coursework between the markers for the module by updating the entries in the attempt table to include the username of the marker as the *marker* attribute.

2. Trigger an external script to extract the files from the database and copy them into a folder with permissions set for access only by the relevant marker. The folder could contain one folder for each student’s files.

3. Send an e-mail to the markers for the module, informing them that the work is ready to mark.

The automation of this process would mean that course leaders would not have to perform any work to distribute the files for marking, other than ensuring that the correct markers are assigned for the module in the database, and the deadline for the coursework is set correctly. These parameters can be checked and amended via the coursework management application.

There is scope for implementation of further rules to extend the workflow management features of the system beyond this initial rule. If the markers were able to add the marks awarded for the work to the attempt tuples in the database, a further active rule could trigger a couple of weeks after the deadline. The condition of the rule would check if any of the attempt entries did not contain a mark. If the condition evaluated to true, the marker of that attempt could be sent an e-mail reminding them to get the work marked. Further rules that could be implemented are left to the imagination. This short section shows that workflow management is a viable application for active databases.

### 6.2 Archiving

It is impractical to simply allow the database to hold more and more files, semester after semester. The database need only hold the files for courseworks relating to modules taken in the current semester. If the database is allowed to grow unbounded, it will become less efficient, as more records will need to be checked to satisfy the queries.

Since the data is stored in a structured manner by the database, a possible solution for the archiving of the data is to simply replicate the database at the end of each semester. This copy of the database could then be saved as the archive of that semester’s submissions. The original database could then be cleared in readiness for the next semester’s submissions. This would involve removing any students who have left the School from the student table and dropping their filestore tables, deleting all entries from the registered table, the attempt table and the remaining filestore tables. Alternatively, a new database could be created from scratch to store the next semester’s work.
An alternative solution would be to write a script to extract the files from the database. The files could be copied into a directory structure, which could then be compressed to save disk space. Since the data is already stored in a structured manner, this solution is perhaps a little wasteful.

6.3 Synchronisation with SIS

At the beginning of each semester, it would be helpful if data could be extracted from the School Information System to populate parts of the database. Student's module registrations are stored in a database in SIS. A script could be written to read the username and module pairings from the SIS database and output a file containing a series of INSERT statements to execute against the coursework database. A more sophisticated solution could run at regular intervals, comparing the official module registration records from the SIS database with the coursework database. The coursework database could then be updated to be consistent with the SIS database. Since module registrations change very little during term-time, and the management application allows the user to manage module registrations, this solution would probably be overfitting the problem.
Chapter 7

Evaluation

Evaluation of the system is performed in a number of different ways. An analysis of the satisfaction of requirements is performed. The submission application is compared with the existing submission system running under SIS by means of a questionnaire issued to students. The final section of the evaluation is the security review, which is promised as a deliverable.

7.1 Satisfaction of System Requirements

The system requirements are listed in chapter 1. Satisfaction of all the requirements would clearly represent a successful solution to the problem. In this section, each of the requirements are addressed in turn, and an evaluation is made of how well the system satisfies that requirement, with a brief description of how the requirement is satisfied. Requirements may be completely satisfied, partially satisfied or not satisfied at all. Where requirements are not completely satisfied, a brief assessment is made of how much additional work would be required to see that requirement completely satisfied. The amount of work required is described as minimal if it could be implemented within a day, or significant if it is likely to take any longer than this.

Comparison of the final product to the initial requirements for a successful solution provides an objective measure of the success of the project.
7.1.1 Functional requirements

Requirement 1 states that the system shall allow students to view a list of the files they have submitted for a particular coursework. This requirement is completely satisfied. When a module and coursework number have been selected using the submission application the user is presented with a list of files, if any, that they have submitted.

Requirement 2 states that the system shall allow students to submit files for courseworks. This requirement is completely satisfied. The submission procedure of the submission application has been tested with many different file formats, as detailed in section 4.6, and the application is capable of dealing with all file types.

Requirement 3 states that the system shall allow students to delete submitted files. This requirement is completely satisfied. The procedure for deleting files is fully tested, and the corresponding entry in the attempt table is removed from the database should no further files remain for a particular coursework.

Requirement 4 states that the system shall allow students to download submitted files. This requirement is partially satisfied. There is a problem with the HTTP headers that requires attention, which causes the file to be downloaded as download.cgi, the name of the Perl script that extracts the file, from the database and returns it to the client machine. A minimal amount of work is required to completely satisfy this requirement.

Requirement 5 states that the system shall allow students to download coursework specifications. This requirement is not satisfied. However, the procedure to download coursework specifications from the database is virtually identical to the procedure to download submitted files, so once the problem with the HTTP headers required to completely satisfy requirement 4 has been resolved, complete satisfaction of requirement 5 would require a minimal amount of work.

Requirement 6 states that the system shall issue receipts for submitted files. This requirement is not satisfied. A suggested solution to the problem involves addition of a button to the HTML page generated by connect.cgi. On clicking the button, a checksum could be computed from the file information stored for the relevant coursework, which could be e-mailed to the user. Satisfaction of this requirement would require a significant amount of work, but the feature could be implemented under the existing system framework.

Requirement 7 states that the system shall allow course leaders to add courseworks for modules. This requirement is completely satisfied. The New coursework use case offered by the coursework management application allows course leaders to enter the number of the coursework they wish to add for a module. The result of this action is the creation of a new tuple in the coursework table. This use case is fully implemented and tested.

Requirement 8 states that the system shall allow course leaders to edit coursework details. This require-
ment is partially satisfied. Coursework numbers can be edited using the *Edit coursework number* use case offered by the management application. Three further use cases must be implemented in order to completely satisfy this requirement. This represents a minimal amount of work.

Requirement 9 states that the system shall allow course leaders to remove courseworks. This requirement is completely satisfied. The result of *Remove coursework* use case offered by the management application is to delete the corresponding tuple in the *coursework* table. This use case is fully implemented and tested.

Requirement 10 states that the system shall allow course leaders to allocate markers to modules. This requirement is completely satisfied. The *Marker* menu of the management application contains three fully implemented and tested use cases to manage the allocations. The *List markers for this module* use case allows the course leaders to view a module’s markers, displaying them on screen. The *Add marker for this module* use case allows course leaders to enter the username of the marker they wish to add. The result is to insert a tuple in the *marks* table, associating the marker with the module. The *Remove marker from this module* use case allows course leaders to enter the username of the marker they wish to remove. The result is to delete the appropriate tuple from the *marks* table.

Requirement 11 states that the system shall allow course leaders to allocate students to modules. This requirement is completely satisfied. The *Student* menu of the management application contains three fully implemented and tested use cases to manage the allocations. The *List registered students for this module* use case allows the course leaders to view a list of students registered for the module, displaying them on screen. The *Add student to this module* use case allows course leaders to enter the username of the student they wish to add. The result is to insert a tuple in the *registered* table, associating the student with the module. The *Remove student from this module* use case allows course leaders to enter the username of the student they wish to remove. The result is to delete the appropriate tuple from the *registered* table.

Requirement 12 states that the system shall produce a list of students who have submitted files for a particular coursework. This requirement is completely satisfied. The result of the *List submissions* use case offered by the management application is to display a list of students who have submitted files for the selected coursework to the screen.

Requirement 13 states that the system shall produce a list of students who have not submitted files for a particular coursework. This requirement is completely satisfied. The result of the *List non-submissions* use case offered by the management application is to display a list of students who have not submitted files for the selected coursework to the screen.

Requirement 14 states that the system shall produce a list of students who have submitted files late for a particular coursework. This requirement is not satisfied. The function that must be written has identical structure to the functions that implement requirements 12 and 13. The SQL query to extract
the information from the database requires is complex, but a single SQL query represents a minimal amount of work.

Requirement 15 states that the system shall distribute the submitted files to markers for marking. This requirement is not satisfied. A suggested solution to this problem is described in chapter 6, and involves an active rule in the database which, when the coursework deadline is reached, divides the attempts between the markers, then triggers a script to copy each student’s files into a folder that can be accessed by the relevant marker. A significant amount of work is required to satisfy this requirement.

Requirement 16 states that the system shall inform markers when there are files ready to mark. This requirement is not satisfied. The script that copies the files can be appended to send an e-mail to the markers for a module. Once requirement 15 is satisfied, this requirement can be satisfied by a minimal amount of work.

This analysis shows that of the 16 functional requirements specified for the system, nine are completely satisfied. Seven requirements remain unsatisfied, two of which are partially satisfied. Five of the unsatisfied requirements need only a minimal amount of work to be done before they could be said to be completely satisfied. The remaining two requirements need a significant amount of work.

7.1.2 Non-functional requirements

Requirement 1 states that the system shall allow students to manage submissions from any computer running any operating system, provided that computer has an Internet connection. This requirement is completely satisfied. The coursework submission application has been tested on both Windows and Linux machines using a number of different browsers including Netscape, Internet Explorer and Konqueror.

Requirement 2 states that the system shall store the files in such a way as they can be accessed only by course leaders and the student who submitted them. This requirement is completely satisfied. Permissions set for the filestore tables in the database prevent other users from accessing the files.

Requirement 3 states that the system shall have sufficient capacity to store at least one semester’s electronic submissions. This requirement is partially satisfied, in so far as the database that has been developed does not have the capacity to store this amount of data, but it would be a simple task to set up the database with increased capacity. This represents a minimal amount of work.

Requirement 4 states that the system shall be capable of handling concurrent access. This requirement is completely satisfied. There is no possibility of race conditions during the submission of files, as each student can operate on only one file at a time. PostgreSQL has an ingenious method of dealing with multiple connections. A Postmaster listens for connections. When a connection is established with the Postmaster, a new one is forked which continues to listen for further connections on the same port [5].
Requirement 5 states that the system shall allow course leaders to manage submissions and marking from any computer running any operating system, provided that computer has an Internet connection. This requirement is completely satisfied, with the proviso that the computer has a Java Virtual Machine capabilities and has the JDBC driver for PostgreSQL 7.1 installed with class paths set appropriately.

Requirement 6 states that the folders to which the files are copied for marking can only be accessed by the relevant markers. This requirement is not satisfied, because the workflow management features have not been implemented. Satisfaction of this non-functional requirement could be incorporated into the significant amount of work required to satisfy functional requirement 15. The additional work pertinent to this requirement would be minimal.

This analysis shows that four of the six non-functional requirements are completely satisfied. One of the requirements is partially satisfied. The remaining requirement is not satisfied. The work needed to completely satisfy these two requirement is minimal.

7.2 Evaluation of Coursework Submission Application

7.2.1 Method

Questionnaires were sent to a number of students when the submission application was completed. Students were set up as users in the DBMS and asked to submit a few files and explore the functionality of the system. They were then asked to answer a few quick questions about their experience of the software. The aim of the questions posed was to find out how easy the system was to use, whether the self-documentation offered is sufficient, and how the system compares to the existing SIS system for usability and functionality. The questions posed were as follows:

1. Were the on-screen instructions easy to follow? Did you ever feel ‘lost’ or confused about what was happening?
2. Did you find that the operation of the system was intuitive?
3. How do you think this system compares for usability with the current system running under SIS?
4. Do you consider the additional features to view and delete files to be useful?
5. Would you feel confident in using this system to submit coursework ’for real’?
7.2.2 Results

Seven students responded to the questionnaires. Their responses to the questions posed can be found in Appendix B.

The general consensus is that the on-screen instructions were easy to follow. A problem that is highlighted here is that the error-checking from the log-in page needs attention. Users found operation of the system to be intuitive, which was one of the main aims from the design of the application. All the users found that the system’s usability was at least as good as the existing system. Most thought it was better and a couple thought it was considerably better than the existing system. All the users thought that the feature to view the files was useful, which is to be expected, as it provides the user with instant feedback on their success or failure in completion of the task of submitting files. One user thought the delete feature should be disabled after the deadline, another thought that it should not be enabled until after the deadline. There is confusion here over the purpose of the delete feature. It is intended to give the user freedom to manage the files they have submitted. Most users found the delete feature to be useful. All users stated that they would feel confident using the system to submit coursework, with the proviso that the receipt feature was implemented.

7.2.3 Conclusions

The application is easy to use. The documentation offered by the application is sufficient. The front-end of the system (this application) compares very favourably to the existing system running under SIS for usability and functionality. Implementation of the receipt feature is a required improvement.

7.3 Security Review

The database provides a very good central storage mechanism for the data held by the system. Permissions set for different users of the database must allow access to the correct subsets of data, and should be protected from malicious attacks.

The location of the database is hidden from students when they submit files. Connection to the database is performed by the Perl scripts, so the URL of the database does not appear in the source code. It is important that the permissions for the script files are set to execute only, to prevent users discovering the URL. This is the first level of security offered by the system.

Should an individual discover the URL of the database, they require a username and password to access the contents. The system allows students to select from the student, module, registered, coursework and attempt tables. They could not discover any information here that would be of any use to them.
The system requires that students are able insert and delete tuples from the *attempt* table. There is potential for malicious attack here, as deletion of another student's tuple in the attempt table could cause the submitted files to be ignored when the deadline passes. Students can have select, insert and delete permissions for their own filestores. They can do what they like with the contents - if they do not access the database via the prescribed interface and make a mess of their filestores that is their own problem! An item that needs changing is to prevent students having select permission on the *Coursework.markscheme* attribute, for obvious reasons.

The use of cookies mean that a student’s username and password need not appear in the source code. Unfortunately, it appears that some browsers cache session-only cookies, which means that the password is not forgotten when the browser is closed. This presents a security risk. A student could open a browser while another student had gone away from their machine and go straight to `connect.cgi`, from where they could manipulate the unsuspecting student's files! This is clearly unacceptable.

As it is, the system requires each user to have a separate password for the database. Synchronising these passwords with students’ SoC passwords would be a difficult task. The security of the system could be improved by password protecting the whole application, so the student is required to enter their SoC username and password to access the index page, which could simply welcome the user (since it would know their identity) and ask them to click to enter the application. The students need not then be users of the database, which would solve the problems highlighted in this section. The Perl scripts could have full access to the students’ filestores, and could manipulate only the filestore of the student currently logged into the application.
Bibliography


Appendix A

Personal Reflection

The problem that I have attempted to tackle was a very challenging one which required the development of a new system from scratch. There was a lot of development to be performed and, since marks are awarded for the project write-up, I had to cease development before I would have liked in order to write up the work. Because I took five modules in the first semester, I did not do very much work on my project until after the Christmas holiday, despite the fact that I did work very hard in semester one. Had I begun development much sooner, the schedule for my project would have been more flexible to setbacks, which will inevitably occur when approaching a problem of this size. Despite the setbacks I did encounter, I do feel that the problem was a little too large to be dealt with in a single project. With luck, another student can build on this work next year.

I set myself a very demanding schedule to complete the minimum requirements and additional features after Christmas, which I considered to be achievable. One particular stage in the project, the development of the submission application, took much longer than expected due to a particular problem with the insertion of binary data into the database. This set the whole project back by several weeks. In retrospect, I wish I had sought more help from the staff in the School rather than trying to tackle that problem alone. Had I done this, the management application would probably have been completed and I may well have been able to build on the work that I have done to produce a working system that would be genuinely useful to the School. The project plan must be adhered to in order for the project to progress smoothly. This is the main lesson I have learned from this project. I would urge other students who find themselves in the situation where their project is getting worryingly behind schedule to go and get help, because universities are for sharing information, and the School is full of tremendously clever people.

The workflow management features to distribute the marking were ones which I really wanted to tackle.
Due to the nature of the system being developed, it was necessary for these features to be implemented towards the end of the project. The one major setback I encountered meant that this part of the problem is merely discussed as a suggested improvement rather than tackled. These were among the features that were intended to improve on the current system.

The nature of the problem I have tackled meant a very broad understanding across several different fields. I was not really sure whether my project was a database project or a software development project. I suppose really it is both. By trying to follow the published guidelines for these two types of project, I feel the write-up goes a bit too deep into certain areas. My project writeup is trying to be “all things to all people” in this respect. Better planning of the contents and page limits to be devoted to each section earlier in the project could well have prevented this. I think my chapters on the database and the submission application in particular are too deep. If I had devoted less time to this writing up I may well have been able to expand the product a little more, and write about some broader and more interesting topics.

Formal testing of many of the features of the system I have developed has been omitted from the project process. Testing is an important part of any software development process. If I had started my project earlier I would most certainly have found time to perform thorough and formal testing. The time taken to do this should not be underestimated.

I am pleased with much of the work I have done. The database is well-modelled, and following the recognised methodology through from the model to the creation of the database has produced a robust storage mechanism, which is both easy to interface with client applications and fulfils the requirements. Following prescribed methodologies for processes is important in the development of software systems.

I would urge other students, or anybody for that matter, to use Latex when compiling a report of this size. The learning curve is a little steeper than for WYSIWYG word processors, but the results are infinitely superior, and management of figures, references, citations and page numbering is a doddle. The Windows NT lab is absolute mayhem at the time of writing. Rows of Linux machines, with Latex installed, are available in the Linux lab.

It is my hope that this system is completed as a future project. The aim of this project report is to explain carefully all the steps that have been taken to bring the system up to its present state. I would suggest that, as a starting point, the database should be set up. The database works well as the central storage mechanism for the system, and should be reasonably easy to adapt should additional storage features be required. The submission application should then be installed on a web server, and the suggested improvements to that application should be made. The management application should then be completed and rigorously tested. The issues of extracting module registrations from SIS and writing the active rules and scripts required for workflow management should be addressed last. This would be a fun project for any student with an interest in databases, application programming and scripting languages. A thorough project could provide the School with a system that would solve the problem of
coursework submission and management for years to come.
Appendix B

Questionnaire Responses

The students who responded to the questionnaires were Diane Ashley (DA), Ian Atkinson (IA), Andy Gaw (AG), Steve King (SK), Doug Little (DL), Simon Lockyer (SL) and Rob Mason (RM). Their responses to the questions posed are listed below:

Were the on-screen instructions easy to follow? Did you ever feel ‘lost’ or confused about what was happening?

DA- Yes, easy to follow.

IA- Originally, I didn’t think that I needed to enter a password for test purposes (having only read the e-mail), so I just entered the username that I was given. When I clicked to log in, my browser didn’t do anything, and no ‘wrong password’ or ‘no password’ message was displayed. This was using Opera 6.1 for Linux. After I had submitted something for DB1 coursework 1, I decided to try for another coursework as well. When I clicked on the drop down menu, the options ‘2, 1, 2, 3’ were available. I was unsure why ‘2’ was available twice. In fact, since I was already on that screen, it would have made sense to me to only have the options ‘1, 3’ available. When I chose to download a file that I had submitted, I was not presented with a dialog box to choose a save location, but rather the file was displayed in my browser. The file was a HTML file. This also occurs with a text file. Because the file is ‘downloaded’ using a button, not a link, it’s not possible to right click a link and choose ’save target as’ to counter this behaviour.

AG- Instructions are fine, the whole interface is simple and clear, no problem at all.
SK- If you type the password in wrong you don’t get an error message, just a blank screen sort of thing.

DL- All on-screen instructions stated the obvious choice of commands to follow. They were easy to understand, follow and use.

SL- Yes. Once I became familiar with the format of WORK, making progress was easy to understand. At no point did I feel lost or confused as to where to make further progress.

RM- They were good. No problems understanding what to do next.

Did you find that the operation of the system was intuitive?

DA- Yes.

IA- The system is intuitive. I don’t think it’s necessary to tell the users to click the buttons; since they have appropriate wording on them it’s fairly obvious what they do. Simply having ‘Select module’ with the menu and button underneath would be fine, and similarly for the coursework number.

AG- Yes.

SK- Yes.

DL- Yes. a simple and easy to understand approach was found.

SL- Yes, because this version incorporates the same functionality as online submission systems I have used in the past as well as to make self-explanatory additions.

RM- Yes. Very clear as to what should be done next.

How do you think this system compares for usability with the current system running under SIS?

DA- This system is better, from what I can remember. It has more functionality, like being able to view what I have submitted.

IA- I find the system better than the current SIS system. The current coursework submission system is buried within the student records, whereas this is a separate system and so is more accessible. This system is also a lot less cluttered than the current SIS system, providing only the SOC module code which is all that is needed, instead of the University module code, level, semester, etc. With this system it is possible to use the browser’s back button to go to the previous page, where as with the current SIS system I often seem to have to go back to the beginning and then work my way forwards again.
AG- Tricky, since I haven’t used SIS submit since before Christmas, and none of my current modules use it, so I can’t have a look now. From what I remember your system is at least as good, and of course has additional features. The emailed confirmation is definitely a good thing.

SK- Much better! Email confirmation is great (if it works!), and the interface is much better than SIS’s.

DL- They are both easy to use systems - however this new system made the ability to switch between module/coursework number a lot easier.

SL- I would consider WORK to be an improvement on the current system in terms of usability. The system as a whole is straightforward and all of the features are self-explanatory.

RM- Considerable improvement! Just depends how east it is for the lecturers to set up.

Do you consider the additional features to view and delete files to be useful?

DA- Yes, for when you’re not sure what you’ve just sent - saves resubmitting.

IA - The view file button seems useful, but does not work properly. With files that the browser can display, such as HTML or txt, the file is shown in the browser and no save dialog is presented. With other file types, the filename presented in the save dialogue is ‘download.cgi’, not the original submitted file name. The delete file feature seems confusing, surely this shouldn’t be available until after the coursework deadline has passed? Why would I need to use this at all?

AG- I think they would be, yes. Particularly as it’s possible to upload duplicate copies of a single file - I’m not sure what the staff will make of that. Minor quibble/bug/feature - the default filename when downloading is one of your cgi scripts rather that the file selected.

SK- Yes, some first years haven’t been sure that they’ve submitted the correct SO13 work, and ended up submitting multiple times - the ability to view what you’ve submitted lets people like that sleep easy!

DL- Definitely.

SL- Yes, the view file function being particularly useful as it would allow me to confirm which version of the file I may have submitted last. I would want to know if the delete function is deactivated after the deadline has passed, and for how long after submission a user can still view the files they have sent.

RM- Yes. Sometimes when you’re not sure what you’ve submitted it’d be nice to be able to check,

Would you feel confident in using this system to submit coursework 'for real'?
DA- Yes.

IA- Yes.

AG- Yes.

SK- Yes.

DL- Yes.

SL- Yes, provided it would send an e-mail confirming submission.

RM- Yes, provided I actually got e-mail confirmation of submission.