A Web Application for the School of Computer Science and Statistics

Brian Marron
B.A.(Mod.) Business and Computing
Final Year Project May 2014
Supervisor: Dr. Jeremy Jones

School of Computer Science and Statistics

O’Reilly Institute, Trinity College, Dublin 2, Ireland
Abstract

The foundation for this project was formed from discussion my project supervisor Dr. Jeremy Jones, other lecturers and fellow students.

Trinity College has a large number of Student Information Systems at work. The college has struggled to provide one integrated system which makes use of the information available on offer. Existing systems do not communicate with each other and there is a heavy reliance on administration staff which is problematic for both lecturers and students alike.

The purpose of this project is to create an exploratory prototype to facilitate the timely and convenient communication between students and the lecturers whose modules they take. The selected approach was to reach users across multiple platforms through a responsive website and enable lecturers to email and text the students taking their modules.

Overall, the aims of the prototype were achieved, however it is just that, functionality common amongst existing student information systems would need to be added if the site was to be viably introduced and potentially add features if it was to be accepted throughout the college, unlike its predecessors.
Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

__________________________________________  _________________
Name                                            Date
Acknowledgements

I would like to thank my supervisor Dr. Jeremy Jones for his continued support and guidance throughout this project. I would also like to thank my family and friends, especially Diarmuid, James and Sakinah for their help and encouragement.
Contents Page

Introduction 6
Background 6
Motivation 7

System Design 10
Selected Technologies 10
Database Design 12
Use Case Diagrams 14
User Interface 15

Implementation 19
Front-End 19
Responsive Web Design 19
Back-End 20
Registration 21
Login 21
Security 22
Follow 25
Manage Modules 25
Email 28
SMS 29
Conclusion 30
Evaluation 30
Difficulties Faced 30
Further Work 32

Bibliography 35

Code [CD]
Introduction

Background

In recent years, there has been a large number of Student Information Systems introduced at Trinity College. IT systems linking students with lectures, administration staff and other college services are poor. One contributing factor has been the lack of an integrated system. Different Departments used different individual systems, despite each one offering similar if not the same services. Another has been the heavy reliance of each on administrators. Some lecturers have opted for their own directory to distribute their lecture materials. Other, less IT literate member of staff however, rely on the various information systems. WebCT, MyTcd Portal, MYZONE, TCDBlackboard, TCD Moodle, Webexone and Turnitin.com are all examples of information systems that have been used in Trinity College in recent years. None of the above offer a unique selling point and many are cluttered with features that add no value to either staff or students. As well as staff constantly moving from system to system, students face the problem of keeping track of which lecturer is using which portal to distribute their material.

Another considerable challenge to the college systems is its heavy reliance on administration staff. Lecturers depend on lists of students provided by the department’s administration. Administration staff members from each department are responsible for admitting students to online systems such as WebCT or Blackboard. This addition of a third party to the system makes the process more cumbersome. Students who need to access course material have limited power to do so. The college allows for students to switch modules at the beginning of term, however, this is not taken into account by the system. As well as students struggling to gain access to material, there is also constant confusion surrounding the inaccuracy of class lists provided by the department at the beginning of the year. The breakdown in
communication has a greater impact on Erasmus and foreign exchange students who may only be spending a year or single semester at college. These students are unfamiliar with various structures and systems in Trinity and often have English as a second.

The College has also failed to adapt to the increasing use of smart phones by the student population. Over 1.6 million people use smartphones in Ireland today (Eircom household survey, 2013) with the highest penetration in the group between 18 - 27, the prime college going age group. Due to increasing penetration of smartphones, students have become accustomed to an immediacy of information. The College however still relies heavily on email or updates on blackboard or portal to inform students. Information between the college and students as well as lecturers and students occasionally needs to be delivered quickly. For example a lecturer cancelling a class at short notice must depend on a class email list to email students. Often this is not adequate to inform all students affected as it counts on students checking their college email before they depart for class. With access to college Wifi recently expanded to accommodate portable devices, demand for student services across multiple platforms has increased.

Motivation

The aim of this project was to develop a simple exploratory prototype that would allow students to ‘follow’ a module, and enable lecturers to manage their own modules and communicate with students that were currently following their module in a timely manner. By allowing users to manage their own information and storing it in one integrated database, the accuracy, value, and accessibility of the information is vastly increased. The system should improve convenience and efficiency for both types of user.
Requirements

The following is a list of **functional** requirements for the system:

- Enable users to register themselves
- Allow students to follow unfollow modules as they wish
- Enable staff to manage their own modules
- Enable staff to communicate with the correct students in a timely manner.

The following is a list of **non-functional** requirements for the system:

- To be secure- Information should only be accessible to those who are logged in as
  users of the system. Sensitive information should be handled appropriately. Only
  member of college staff should be able to manage modules.
- To be accessible- to all staff/ students across a variety of different platforms. No
  assumptions are made about the technical ability of the user other than their
  ability to use the device through which they access the site.
- Efficient- the site should improve on the efficiency of existing systems

Methodology

Sommerville (2007) proposed three possible software development lifecycles which were
considered to guide the construction of this project namely: Waterfall, Iterative Development,
and Component-Based Software Engineering. The selected approach was inspired by The
Waterfall Model (Royce, 1970). Despite often being criticised as overly simplistic, outdated and
risky, it was adequate for this project. The model outlines isolated steps [Figure 1] which flow
together in a prescribed order.
The rationale behind Waterfall is to ‘measure twice and cut once’. It is argued that time spent in earlier phases validating requirements and design saves time (and in the real world money) in the long run. It was important to identify obstacles at an early stage due to the time constraints. Each step begins only when the preceding step has been completed and progresses the design towards the end goal. In theory it is a sequential process, however, in practice it can often be iterative in nature. Verganti (1999) claims that it is not sufficient to be overly anticipative nor reactive in the development process. Instead, he proposes a balance between the
two- Planned Flexibility, whereby early decisions anticipate future opportunities or difficulties that may arise, but facilitate flexibility at late stages in the development process. Because of the authors inexperience on such a project it was important to constantly endeavour to anticipate, however, it was important not to overly commit to any uninformed decisions.

**System Design**

**Selected Technologies**

The two options of communicating with students via handheld devices were either to develop an app or to develop a website. An app would have limited the system to handheld devices and either iOS or Android, whereas a website would not. It is for this reason that a website was the selected approach for the purposes of this project.

Mobile handheld devices have a viewport that is a considerably more narrow than a desktop browser window. This is a problem for websites designed exclusively for desktop browsers. This requires the users to scroll, pan and zoom in on particular areas of the site. One possibility to combat this problem is to create a separate (m.) mobile site. This however, is widely considered to be bad practice. Instead, the selected approach was to aim to create a responsive website which functions fully across all platforms. “Responsive Web Design is” is a name given to the set of techniques applied at the style and layout of a website that allows it to adapt itself to any device or width of screen. Doyle (2011) claims that “Adaptive Design” is a more fitting description as the “website really adapts to the device, rather than continuously to changes in its environment”. To build such a website, the author explored three possible architectures:

1. .Net Framework
2. Java

3. Hypertext Markup Language (HTML), Javascript, Cascading Style Sheets (CSS) and PHP Hypertext Preprocessor (PHP)

After evaluating all of the above, HTML, Javascript, CSS and PHP were selected. One of the main reasons for their selection was that the author had no experience with them and they presented a learning opportunity. WAMP (Windows, Apache, MySQL and PHP) was the solution stack chosen. It is suitable for building heavy-duty dynamic websites and is composed entirely of free and open source software. Though developed separately, each component works and scales well together and can be adapted to meet requirements. HTML is the standard markup language used to create web pages. It describes the structure of the website semantically along with cues for the browser to present it. The browser does not display the HTML tags themselves. It allows images and objects to be embedded and can be used to create interactive forms. Web browsers can also refer to CSS to define the design of web pages. CSS was introduced together with HTML 4. It can be added to HTML in the following ways:

- Inline- using the style attribute in HTML elements
- Internal- using the <style> element in the head section
- External- using an external CSS file.

XAMPP, an open source web server solution stack was used for the purpose of creating and designing this web application. It consists of an Apache HTTP Server, a MySQL/ SQLite database and interpreters for scripts written in PHP and Perl. It was selected as it is a valuable development tool that enabled work on a local machine without an internet connection. Javascript makes HTML pages more dynamic and interactive. The HTML <noscript> tag is used to provide an alternate content for users that have disabled scripts in their browser or have a
browser that does not support client side scripting. The <noscript> element can contain all other HTML elements. The contents within a <noscript> element will only be displayed if scripts are not supported, or are disabled.

Windows was selected as it was the operating system running on the authors local machine. Apache is the most widely used web server and supports a variety of features. The School of Computer Science and Statistics provide students with access to their Apache Server (s-web) which was used to test email and to enable access to the site from other machines e.g., phones, tables, various screen sizes to test its responsive features. Students also have access to a MySQL database. Again this is a common database management system which is secure.

Database Design

![Entity Relationship Diagram](image)

[Figure 2: Entity Relationship Diagram]

The above entity relationship diagram shows the data model selected and how the various actors relate to each other. At an early stage, the decision was made to explicitly distinguish a difference between the two different types of user. This was done because a module may be associated with one or more students (ie those that ‘follow’ it), however, it is only ever associated with the one staff member that created it. It also eliminated the null columns that would have resulted had a data model been selected with just one user containing both types. All data used in the application was stored, retrieved and updated from the MySQL database:
**Student**

The ‘Student’ table stores data concerning students. All values are inserted to the table when the student registers. `student_id` is the primary key used to uniquely identify a row and it is automatically incremented.

**Staff**

The ‘Staff’ table stores data concerning staff members. All values are inserted to the table when a staff member registers. `staff_id` is the primary key used to uniquely identify a row and it is automatically incremented.
Module
The ‘Module’ table stores data concerning modules associated with staff members and followed by students. It enables one staff member to be associated with multiple modules. All values are inserted into this table when a staff member creates a module. *module_id* is the primary key used to uniquely identify rows and *staff_id* is a foreign key of the ‘Staff’ table, relating to the unique staff member associated with the module.

Following
The ‘Following’ table stores data concerning which ‘Students’ follow which ‘Modules’. This table prevents duplication of redundant data and the need for unnecessary null values. It enables a student to follow as many modules as he/she wishes to follow, provided they are eligible to do so. Both columns in this table, *student_id* and *module_id* are foreign keys. Together, they are used to uniquely identify rows.

Use Case Diagrams
Use case diagrams are a form of behavioural UML (Unified Modelling Language). They are a graphical representation used to analyse the intended behaviour of a system and how it interacts with users and other systems at a high level of abstraction. They map generic tasks or activities that must be carried out by a system if it is to meet requirements (Kawabata and Kasah, 2007).
User Interface

If the user cannot use a system, it is broken. Many systems are broken by design. The objective was to reduce human error by creating a simple and consistent user friendly interface. The system would be used by people with a diverse technical ability. A hierarchical task analysis was carried out to divide the goals of the users into tasks and subtasks. Stanton and Diaper (2003) argue that this gives a greater understanding into what the user wants to do. Then, by carrying out a cognitive task analysis, we gain insight into how the user carries out these tasks. It enables us to tailor the UI accordingly to enhance the users experience. Examples include shortening the most common tasks and identify possibilities for error (human as opposed to technical).
The only high impact error that was identified was if a lecturer accidentally deleted a module. All information regarding the module would be lost and all students following that module would be deleted from the ‘Following’ table. To counter this, it was decided an alert would pop up to prompt the user that they had attempted to delete the module and ask for confirmation of the intended action. Other possible errors included users inputting incorrect data, a lecturer texting the wrong class and students following the wrong module. To avoid such slips, text input should be kept to a minimum. Radio buttons and drop down menus should be used where possible. Lecturers and students should be presented with a list of modules to choose from when texting, emailing, following etc. They should receive feedback to confirm any actions taken or reasons why actions could not be taken. Where possible, actions should be easily reversible.

The most common tasks were registering/ logging in, following a module, or texting/ emailing a class. The use of radio buttons and drop down menus also helps to speed up this task. When choosing to follow a module, a student should only ever be presented with those which they are eligible to follow. To avoid the user having to sort through this information, it is handled at the back-end. Once the lecturer selects the class he/ she wants to communicate with, all they have to do is enter the message and press send. Again, all of the queries are carried out in the back-end to save time and effort, thus increasing satisfaction for the users.

As the system would also be used across multiple platforms so design would have to consider layout for desktop, laptop. With a limited screen space it was important not to clutter the interface with functions that would not add value. Design of the user interface began with low fidelity prototypes. Paper sketches and storyboards for all three platforms were used to complement scenarios. They were a fast and cheap means of identifying problems.
A side-navigation was avoided because of the reduced horizontal real estate on the screen of mobile phones. A horizontal menu was viable across all platforms. As a front-end design and the structure of the website’s content began to form, Balsamiq was used as a tool to design wireframes that represented the user interface. Examples of these mock ups can be seen below in Figures 6 and 7. A decision was made to form a template for each page of the site. Every page would consist of a header, which contained the title of the site and the menu, a main, which contained the body of content being accessed on a particular page, an aside which contains various widgets depending on the state of the system, and a footer, which would be a placeholder for various links and information if the site was to be developed further. This consistent layout not only shortened the task of developing the site, but would also make it more coherent for users. A consistent layout would reduce time required by users to familiarise themselves with the site making it easier to interact with.
[Figure 6: Student Registration Wireframe]

[Figure 7: Staff Homepage Wireframe]
Implementation

Front-End

The frontend is the interface between the user and the backend. It is responsible for collecting the users inputs in various forms and processing them to conform to backend use. HTML was used to structure the documents content and CSS was used to structure the documents presentation. Javascript was also used to interact with the user client-side. The majority of user input is handled using forms. They can contain input elements such as text fields, checkboxes, radio-buttons, submit buttons and more. They pass the data input by the user on to the server by means of the PHP superglobals $_GET and $_POST. Both create an associative array whose keys are the names of the form controls and whose values are the input data from the user. Because they are superglobals, they are always accessible, regardless of scope.

Objectives regarding the styling of the user interface highlighted above were achieved by linking to an external CSS style sheet in the head of each HTML document:

<link href="CSS/layout.css" rel="stylesheet" type="text/css">

Basic styles were designed to familiar to make the website more attractive for users. The same styles were applied to each page on the site.

Responsive Web Design

Responsive web design makes no assumptions about the dimensions of the viewport, but rather adapts to devices that have portrait and landscape modes. Instead of tailoring disconnected designs to each of an ever-increasing number of devices, they are treated as facets of the same experience. As part of the CSS3 specification, W3C created ‘media queries’ which improved
upon the promise of its earlier ‘media types’ to target not only device classes, but also to query the physical characteristics of the device. Media queries identify the width of the viewport without the necessity of running Javascript. The CSS even responds in real time to changes in the size of the viewport, so if users resize their window, the CSS automatically adjusts and enforces the changes.

All containers have their widths defined in percentages meaning that they are completely based on the viewport rather than the containing block they would have been in a traditional fixed grid layout. The containers do however have a minimum width beyond which they will not resize. Media Queries evaluate to true/ false. If true, the associated values are applied. The width percentage of the fluid grid is recalculated relative to the width of the viewport at various breakpoints which are enforced by media queries within the CSS file:

```css
@media only screen and (min-width: 481px) {
    /* Styles here */
}
```

The above line is the media query to target ‘tablets’ with a viewport of between 481 pixels and 769 pixels. Styles within the query are applied to windows that fit this criteria.

```css
@media only screen and (min-width: 769px) {
    /* Styles here */
}
```

The above line is the media query to target ‘desktops’ with a viewport of greater than 769 pixels. Styles within the query are applied to windows that fit this criterion. Anything below 481 pixels is considered to be a mobile phone. If the viewport, or indeed the windows moves below this breakpoint, the site moves from being two columns in width to single columns stacked on top of each other and the ‘aside’ container shifts below the main container. Tempting as it is, the aim of responsive design is not to target specific devices, but rather to provide the optimum experience at any size.
Back-end

When carrying out the hierarchical task analysis, it became apparent that although each individual task of the users was different, each one required the sub task of viewing a list of modules, and then action upon those modules. Identifying similar subtasks at an early stage in the development process proved to be a valuable observation. Each task would direct the user to the list_modules.php page. Although the content would be presented in the same format for each task, the actual content that was presented and the actions available to take on the list of modules would be tailored based on a GET method.

As well as helping the development process, this also adds consistency and coherence to the user interface and reduces the time required to familiarize oneself with the system.

Registration

To register, both students and staff alike were presented with a HTML form. Once all the required fields were filled in with valid data that did not already exist in the database, the user could submitted the form and the data would then be inserted into the relevant table in the MySQL database. Error handling was initially done using Javascript, however, it was changed to PHP upon the discovery that some browsers do not support, and some users disable client side scripting. The author has since been made aware of the <noscript> tag, so this decision could easily be revered.

Login

The Login widget compared the input values to values in the database. If they did not match, the user was presented with and error message telling them why they were unable to log
in. If they did match, the user was logged in and a session was created. Sessions store information about the user currently visiting a website. Information regarding sessions is stored on the server and unavailable to the user. Sessions are a lot more secure than cookies. They are quicker and easier to use. One example of how sessions were used is that the users type (student/teacher) was stored. As mentioned above, this was used to protect the privacy of staff pages. It was also used to return a different menu and different widget. This enabled the same design to be used for both students and staff, while tailoring the content to them.

Security

One of the most obvious security requirements was the need to prevent anyone from registering as a member of staff and abusing the service. When the website was hosted on the SCSS web server (s-web), a .HTACCESS file could be used to restrict access to the staff registration page:

```plaintext
AuthType Basic
AuthUserFile /var/apache/etc/users
AuthGroupFile /var/apache/etc/group
AuthName “Brian Marron Final Year Project Restricted Access”
require group scss-staff
```

This meant if anyone attempted to access the staff registration page, they were prompted to enter their TCD credentials and only SCSS staff were granted access.
It was also important to protect pages from being viewed if a user was not logged in. To do so, the function `protect()` is placed at the top of each page to redirect a user if they are not logged in to the page. Protected php:

![protected.php](image)

Likewise, no one other than staff should be able to view staff pages. If someone is logged with a session type other than staff attempts to access a staff page, the `block()` function redirects them to the page `blocked.php`:
$_GET is an array of variables passed to the current script via the URL parameters. 

$_POST is an array of variables passed to the current script via the HTTP POST method. 

Information sent from a GET method is visible to everyone. This meant it was only sufficient for sending non-sensitive data. However, information send with a POST method is not visible to others as all values are embedded within the body of the HTTP request. Such a level of security was required and used when sending sensitive form data such as passwords. Once the data was sent, in order to make the database itself more secure, md5 encryption was used with passwords to protect them. 

Many of the HTML forms required values which were not actually input by the user. To protect this sensitive information, these input types were changed to hidden which meant that they were still input once the form was submitted but that they would not be seen by the user. 

Following on from testing, it was decided to protect students phone numbers when a staff
member was texting a class. As staff members do not have access to students phone numbers at present, and there is nothing to be gained from providing them with access to them, the input tag which contains the phone numbers selected is hidden. However, lecturers do have access to student email addresses at present, so a conscious decision was made to echo the values into an input field when a lecturer is emailing a class.

The htmlspecialchars( ) was another function which helped to prevent cross-site scripting attacks on code (where attackers exploit the code by injecting HTML or Javascript code).

Follow

Other than registering and logging in, following a module is the only other task currently undertaken by a student using the system. To do so, a student was presented with a table of modules that he/she was eligible to follow (i.e., from their year of study). Once the student clicked the “Follow” button, a record was added to the Follow table of the MySQL database containing the student’s ‘student_id’ and the module’s ‘module_id’.

Manage Modules (Add Edit Delete)

In computer programming, Create, Read, Update and Delete (CRUD) are the four basic functions of persistent storage. The hierarchical task analysis identified the lecturers goal of managing their modules. These could be broken into the tasks of CRUD, or in the language of the user, Add, Edit, and Delete a module. The modules a lecturer taught were viewed on their homepage. If they wanted to make a change to this list they clicked the “Manage Modules” button on the staff menu and were presented with the following scaffold:
• Add a module presented the lecturer with a form. When the form was completed and submitted, the input values were inserted into the ‘Module’ table of the MySQL database. The `staff_id` of the lecturer was input to relate the module to the staff member.

• Delete a module redirected the lecturer to `list_modules.php`. A GET method was used to return a table of modules that the lecturer was currently teaching and offer the ability to delete each module individually:
If the link is clicked to delete a module, the lecturer is presented with all information concerning the module and asked to confirm that they do wish to delete the module. This was done to reduce human error and any slips that may have accidentally occurred. Dependencies between the tables are enforced by the database. For example if a module is deleted from the ‘Module’ table, all records of students following that module are automatically deleted from the ‘Following table.’

- **Edit a module** also points to a `list_modules.php` which displays a table of modules that the lecturer teaches only this time, the button offers the option of editing a module. This is achieved by use of a GET method. If a lecturer clicks to edit a module, they are presented with a form which is pre-populated with the information regarding the module from the database using echo.
Once they are satisfied with the new changes, they click the edit module button and the new values are updated for the old values in the database.

Once a module has been added, edited or deleted, the lecturer is returned to the page list_modules.php where they receive confirmation of the action they have just taken and they can view an updated list of modules that they teach.

**Email**

To send an email in PHP requires an Simple Mail Transfer Protocol (SMTP) server. Although the SCSS web server automatically had one set up, it needed to be set up on the local machine to work with XAMPP. Initially, this was done by re-configuring the php.ini and sendmail.php files in XAMPP. Though this also worked, emails were marked as spam. An email
sending library called “PHPMailer” was downloaded from: https://github.com/PHPMailer. This library allowed the use of Google’s smtp server by submitting a valid Google email address and password for the address the mail was to be sent from. Again, for security reasons, all input fields other than the message and subject were hidden from the user. Another composite MySQL query was used to find the students in the class that the lecturer wanted to email, and then to find their email addresses. The result of the compound MySQL was echoed into the input field of the form that require the email addresses that the mail was to be sent to.

SMS

To send a text message or SMS (Short Message Service) from within PHP script requires the use of an SMS gateway. A direct-to-mobile gateway is a device which ha built-in wireless GSM (Global System for Mobile) connectivity. It allows SMS messages to be sent and/ or received by email, from web pages, or from other software applications by acquiring a unique identifier from the SIM (Subscriber Identity Module). There is a large quantity of SMS Gateways who facilitate sending text messages from a HTTP. One such company which was selected during this project was iFlow. www.iflow.net has been in operation since 1998 and claims to have over 10,000 users from over 7,000 organisations.

When a member of staff clicks on the option to text a class, they are presented with a table of classes that they were currently teaching. After selecting the class they want to send a text message to, they are prompted to type the message and press send. The form data requires of additional data to the numbers selected and the message that is to be sent. These values are handled using hidden inputs. On submission, the form is posted to: http://www.siflow.net/api/sendsms.api and the message is sent to the intended recipients.
Conclusion

Evaluation

Overall, the objectives of the prototype were achieved. It was successful at carrying out all requirements (both functional and non-functional). It is proof of concept that it is possible to provide users with the ability to manage their own information, however, it is but a proof of concept. If it was to be introduced to Trinity College, additional work is required. Features that are available on existing solutions would need to be added if it was to be of value to users. Additional features would also need to be added to avoid it being introduced, used for a short period of time, and then replaced like so many of its predecessors.

The prototype proves that if stored correctly and utilized effectively, the information available can improve processes currently in place. When adding future features, extensive testing would need to be carried out. It is imperative not to make the same mistake as so many before and offer numerous features that offer no value to users. Interview, surveys, observations

[Figure 13: Using an SMS Gateway]
etc could all be carried out on individuals and groups from around the college to get a greater understanding of the needs of the user.

**Difficulties Faced**

As already mentioned, the author had no experience of any of the markup or programming languages before undertaking this project. It was also stated that one of the reasons for selecting the languages that were chosen was the abundance of materials available to aid the learning process. One difficulty experienced was to sort quality sources from poor sources. It was not desirable to pick up bad habits or practices at any stage during this project. When making decisions, numerous sources often had to be considered and weighed against each other.

One such example of when poor advice was followed was the selection of a development environment. Notepad++ was used to write all of the code for the entire project. Now at the end of the project, having manually debugged throughout, it is now appreciated how much time and effort an integrated development environment such as Eclipse or Net Beans would have been. Software such as Dreamweaver could also have been used during the design process to avail of its split screen which enables developers to view code and an open a webpage simultaneously. Source code repositories such as Github, Google Code, Launchpad or Subversion which is available to TCD students would also have proved a useful resource to help track the various versions as the project progressed through development.

Another difficulty was the order in which development was carried out and the prioritising of certain aspects. A lot of time spent early on perfecting the CSS for an attractive website. Despite being unnecessary for a prototype, the idea had to be scrapped because background images for the various different pages were too high in resolution and took too long
to load. They also did not appear well on mobile/ tablet as the landscape picture was squeezed into a portrait viewport. Time would have been better spent focusing on the back end functionality of the PHP.

This could be linked to the tendency to add functionality as it was learned even though it was not included in the requirements. Functionality was often added because the technology allowed it. Later it would be deleted because it offered little or no value to the users and possibly diluted the value of existing features.

Integrating a third party in to the website to enable the use of SMS functionality proved to be difficult. The first SMS gateway selected ceased servicing the Irish market. Documentation regarding the use of Iflow was unclear and communication what poor.
Further Work

If the site was to be developed further beyond a prototype, there is a number of measures that would need to be taken:

- The first would be to incorporate features of existing information systems which add value to the users but are not available on this prototype. This would include the ability for lecturers to upload documents such as module outlines, lecture slides and recommended readings. Another might be to enable staff to upload marks. However, rather than simply uploading a PDF document containing a list of the classes’ marks, an interface could be provided to allow the lecturer to award each mark to the student and the associated module. A table of ‘Marks’ would be created in the database with reference to both the student and the associated module.

- Functionality that would offer considerable value to staff would be the ability to reschedule a class. An integrated system would be able to identify a time when each student within the class is free, and an available room that can cater for the capacity of the class. This would save lecturers and administration staff a great deal of time and effort.

- As well as providing the functionality for a lecturer to email a class of students that are following a module, it may be of value to both students and staff to add the functionality to enable staff to email other groups of students such as particular courses, years or faculties.

- Testing could be carried out on the value of a news feed as a more efficient channel for a lecturer to contact a class than email. It would also offer the student the ability to follow a lecturer rather than just a module. This would promote communication concerning
subjects of interest, rather than confining it to that regarding the module and course work. If the application was to be used by the School of Computer Science and Statistics, or indeed the college as a whole, the cost of bulk text messages could quickly grow. Sending push notifications to a user’s phone would fulfill the same need for free. There is also the possibility to add more functionality for the student. A forum could be set up for class to share ideas or material relating to each module. By engaging the student, the system would promote participation. However, it would be necessary to test the demand for such features and not just introduce them because the technology allows it.

- More extensive CSS styles would need to be applied. To save time, it would be possible to implement a framework such as bootstrap, however, for the purpose of this project, it was sufficient and beneficial to hard code the CSS from scratch. The media queries embedded in style.css only touches on some of the media queries available in responsive design. Rather than alter the placement of content, it is possible to introduce entirely new layouts. Media queries are also not limited to layout changes. They offer a wide range of possible features and are supported by popular browsers such as Safari, Chrome, Firefox and Opera which all natively parse media queries. For example a different navigation scheme could have been offered to mobile users, however, this was considered to be out of the scope of the exploratory prototype for the purpose of this project. Another possibility would be to enable users to turn on geolocation to locate the rooms where lecturers were taking place. This simple feature would be of particular value to new and visiting students.

- Javascript libraries such as node.js, progress.js, jQuery which are growing in popularity could be utilised to make web pages more dynamic and attractive to users. An example of
this would be to use AJAX when a student clicks to follow/unfollow a module. Rather than the query going server side and forcing the page to reload, the query could be handled client side and the user would know straight away that the task was complete.

- Perhaps one oversight may have been not to verify the domain of email addresses when registering to make sure everyone that used the site had a @tcd.ie email address to prevent those from joining that were not TCD students. This could be done by inserting a simple Javascript function that would be called on submission of the form to verify the validity of the email address domain.
Bibliography

Doyle, M. (2011). Responsive Web Design Diversified. Available at:
http://www.elated.com/articles/responsive-web-design-demystified

Accessed on: January 15th 2014

Eircom. (April 2013) Eircom Home Sentiment Survey. Available at:


W3C: Specifications. Available at: http://www.w3.org/TR/ Accessed: April 1st 2014