Why the structure behind the Final Year Project?

There are two high level goals behind the FYP:

1. Fostering Innovation and a Better Student Experience. The FYP should not be about risk-aversion and marks-chasing; it should be about exploring innovative solutions to address interesting problems. We don't want you to be stressed out by the FYP; we want you to be challenged, inspired, creative and productive. We believe the iterative structure of the FYP removes the "feast or famine" dynamic that can often cause panic, and will enable all of you to demonstrate your capabilities to yourselves, to the department, and to future employers.
2. Fostering Integrity and Maximizing Transparency. IS professionals continually engage in collaborative and integrative knowledge work. Correctly managing and documenting third party IP, whether in the form of source code, components, templates, or even instruction and advice received is critical and challenging. The structure of the FYP is designed to make the development process and product completely transparent, so that we can identify IP and academic integrity issues as they emerge and provide students with the resources needed to correct these issues before they become problems.

Course Objectives

The objectives of the FYP are to:

Analyse an organisation's/individual's/group's problem

Follow a development methodology to design, develop, and test a technology solution to the problem.

Demonstrate the solution

Identifying a Problem to be Solved through Technology

The first step is to identify a problem. This could be a problem that an organisation has, an individual has, or a group has. As an example:

A sports club have a problem in that they don't know how much money they are losing (or making) selling club gear and equipment. Coaches have gear and equipment in their car boots that they can sell to parents or players at training, but money often gets lost or forgotten about. The club has a rule that broken equipment will be replaced for free within one year of purchase but they rely on the good will of people to tell them when they bought it. It is suspected that the club are losing a lot of money because they have no way of tracking what was bought, when it was bought, by who, who sold it, and if the money was given to the club treasurer. The coaches all have mobile phones which they could use to access any proposed system but the club say that it would be a problem if the sales was only stored on one phone. The information has to available to all coaches.

As an example, this is the potential starting point for an FYP. There is a problem that needs to be addressed and the club need a technology solution to solve it. The key to the start of the project is a problem. It is a common mistake to start an FYP with a solution. As an example, "I know about technology X and want to apply it". This is not a problem to be solved; this is a solution looking for a problem.

Sometimes the problem solved by an FYP is more abstract than the example above (these projects tend to be called 'proof of concept' projects). In these cases, students do not address the concrete problem of a specific user, but rather a conceptual problem for a class of user. For example, social media allows new graduates to network effectively in pursuit of employment, but also exposes more of an individual's private life than would typically be shared with a potential employer. Thus, expressing and bounding the 'identity management challenge' of social media for graduate job seekers would be a potential starting point for an FYP.

Writing a Project Proposal

Once a problem has been identified, and agreed with your supervisor, the next step is to write a project proposal. The project proposal is an overview of what you think the project will look like and evolve over the duration of the project. It is not a low-level technical design; it is a high level overview. You are proposing a project to your supervisor.

The project proposal will detail what the project aims to achieve and why the system is required. This will include details of discussions with real or potential users/customers of the system and/or other subject matter experts and stakeholders. Your proposal should also be presented in the form of a business model canvas (we will give you more details on this at the start of term)

High Level Design

Once you have identified a problem to solve, and documented it in the project proposal, the next step is the high level design of your solution.

Your high level design will be presented as visually as possible and you should use techniques such as ERD's, data value maps (see http://datavaluemap.com), user personas, etc. (Again, we will provide more detail on this at the beginning of term, although you can do some research on it yourself over the summer).

Next, a list of all features of the system will be created, and listed in numbered priority order, with 1 being the highest priority. The features will be delivered according to this priority list. It is acknowledged that this list will change as the project progresses and that not all features may be delivered in the project's timescale. In fact, it will be rare that all the features in the initial proposal will be delivered and in the order initially proposed. The reality of software development projects in industry (and the FYP) is that requirements change throughout the project.

User Stories

The features will be written as user stories. A user story is a description of a single feature of the system that is required to solve part of the problem identified. They are usually written from the perspective of a user of the system. In some cases a user story can be written from the perspective of the developer, but these should be kept to a minimum. An example of a developer user story would be where it is necessary to create a small database to store data. A customer/client would rarely specifically ask for a database. A database is a solution and not a requirement. A customer would talk about accessing data or storing data; it is the developer who needs a database as a feature to solve the customers need for data storage and retrieval.

User stories are written using the following format:

As a …

I want to ….

So that I can ….

An example of a user story from a customer would be:

As a trainer

I want to be able to search for a player's name to find the date they purchased their equipment

So that I can see if they are due a free replacement within a year of purchase

It is important to note that this is not a technical description. It states a requirement not the solution. You as the developer will determine how to implement the solution.

Product Backlog List

You will have a list (called a product backlog list) which contains all the user stories you will need to solve the problem identified in the FYP. This list is originally created in the proposal, but will most likely change as your FYP progresses. Each user story will have a priority number assigned. This number gives the order in which the user stories will be developed. It is likely that the priority order will change and that you may not develop all the user stories by the end of the FYP. This is normal in development projects.

As an example of a product backlog list (each of these is the title of a user story)

1) Create a new purchase
2) Assign seller to purchases
3) Search purchase for free replacement
4) Edit incorrect purchase records
5) …
   …
   …
   …

20) Login screen for mobile connection

This list is stating the importance of each user story, and the order in which they will be developed. A major benefit of this is that if you never get to user story 20, the other user stories together will still be a fully developed system that you can present as your final project.

It might seem unusual after years of learning programming that a login screen would be a low priority, especially since its one of the things you have probably learned to do in each programing language you've studied. From the customer/clients viewpoint though, a login screen will probably be useful but it is not the most important thing they want. In the example used, would the sports club rather be able to add a new equipment sales record or to have a login screen? The best way to think about priorities is: "If my project stopped now, what would the customer want to see working" (or for the FYP, "If my project stopped now, what would go the furthest towards solving my chosen problem and demonstrating my system's potential"). That's what you deliver first.

Environment Set-up

After your user stories are submitted and before development begins, you have a two week period where you set up the development environment for your FYP. This could be gathering and installing any software needs you have, setting up hardware, etc. In the example for the sports club, you could look at ways for an app on a mobile phone to access a centralised database. You would need to set up a simple database and look at ways a mobile phone could connect to it, and if any additional software/apps/phone settings are required. Obviously, in this example, your final system would have multiple mobile phones accessing the system, but it the environment phase you are just checking the feasibility and determining what you would need for a single phone to connect. You are not developing here; you are examining if there are software or hardware requirements you will need. A simple example of environment needs would be a database and a mobile phone development platform.

Component Re-Use/Licensing Issues

During environment set-up and throughout the development process, it is critical that you correctly identify and manage IP that is not your own. In particular, during the environment set up phase it is important to identify any potential IP/licensing issues you may have.

In any development project (in the FYP or in industry) it is a common and extremely useful practice to re-use components, including open source components, publicly available software libraries, frameworks, templates, and other forms of reusable functionality, as part of your system.

There is no issue with using such third-party components in the FYP as long as:

1. The component is publically available, not a bespoke component contracted by you. ANY code developed by a third-party for hire (whether or not money changes hands) is strictly prohibited in the FYP. It is acceptable to get generic help from others on your FYP. For example you can get a tutorial on how to connect to databases in Java. What is not acceptable is if the help you receive is on how to connect to your database in your FYP.
2. You have the legal right to use the code (i.e. the component's licence allows its use for your purpose).
3. You leave intact any licensing/copyright text that is part of component code and you state clearly that you are using the component as part of your system documentation (typically via a README.txt file and code comments). You must also "cite" the source of any code snippets used in code (e.g. templates, code skeletons, tutorial material, etc.) using comments, and also describe them in your iteration documentation.
4. You add value through your use of the component! Remember the purpose of using a third-party component is to accelerate your development process, not to pad out your FYP. If your FYP goal is to build a calendar-sensitive automation tool of some sort, no one expects you build basic calendar functionality from the ground up; there are many good calendar APIs available. We DO expect you to do the work of integrating the component with your system, and to build on the functionality it provides to achieve your goal.

In the first week of term we will provide you with a detailed briefing and document (which you will sign) regarding UCC's plagiarism and academic integrity policies and penalties, and on how they relate specifically to the FYP in terms of third party code reuse, licensing/IP management, use of online and offline tutorials, and so on.

Developing in Iterations

An important thing about user stories is that they must be small enough that they can be developed in two weeks. They must also be independent of the development of other user stories. You should not have an entire system that can't be developed or fails to run because of a dependency between user stories.

In the FYP, each development iteration includes designing a solution for one or more user stories, writing the code (or creating a media object or piece of hardware or whatever), and testing the solution. All of these have to be completed in two weeks (that is two weeks of term time when you obviously have classes/labs/etc. happening at the same time). When a user story is shown to your supervisor it must be complete. It is not acceptable to say that it is "nearly complete" or "has most of what is needed". A user story is either 100% complete or not complete – there is nothing in between. The reason for the two week boundary on development is that the FYP will be created through iterations. Any user story that you think will take more than two weeks to develop must be divided into smaller user stories.

An iteration can be considered as a mini-project. Rather than planning a project for the entire two semesters, you will plan one iteration at a time. As you can guess from the 2 week boundary on user story development, each iteration lasts for two weeks.

At the beginning of each iteration, you will mark as complete all finished user stories in the product backlog list. You will then look at your prioritised list of remaining user stories and decide **how many** of them you can deliver in the next two week iteration (remember deliver means plan, develop, and test). You will not be deciding **which** ones to do – you will always do the top priorities – just how many. In some iterations, you may only plan to deliver one user story; in other iterations you may deliver more. It depends on how much work you think is involved in each user story. Occasionally, you may realise that you need to reorder the priority list; this is perfectly normal but must be done at the beginning of the iteration. It is bad practice (and will cause you problems and potentially lose marks) if you try to reorder the list mid-iteration.

During the iteration, you will design, develop, and test the additions to the system. At **the end of each iteration**, the student will present all relevant documentation and code to their mentor. This documentation includes:

- A process review/post-mortem on the iteration just completed.
- A list of the user stories that were delivered in the iteration.
- The results of testing of any code developed in the iteration.
- User documentation for the stories developed in the iteration.
- A development plan for the **next** iteration with a list of the user stories that will be delivered. This will include the product backlog list that is updated each iteration, with changes made as required.
- A test plan for the next iteration.

It is the student's responsibility to decide what will be worked on in each iteration. The output of an iteration must be a working system, although in the early iterations all functionality will obviously not be present. If the code does not compile and work as described then this is not a working system.

Version Control

All code and documentation must be under version control. Each document and submitted element of code must be given a version number, based on the iteration, and all versions saved. At the end of each iteration, the current version of documents and code must be submitted to the mentor. It is the student's responsibility to ensure that all versions of documents and code are saved and backed up as any version can be asked for at any time, and marks lost based on what is missing.

It is vital that you are continually developing across the two semesters. You will be marked for what you do in each iteration and how you do it.

Audits

Each week throughout the two semesters, students will be selected at random for an audit of their evolving code and documentation. Version control as discussed above is important, as the auditor can also examine any previous iteration of your system development as part of the audit. The auditor may examine, for example, your demonstrated understanding of the code and how it works; if your code, documentation, and plan matches the work that was done in a particular iteration; if the current version compiles and runs, and so on. Marks for the relevant iteration will be lost if the student does not attend the audit (without appropriate documented reason) and may be lost if the audit identifies anomalies in what was presented by the student and the student's understanding of what was presented, the accuracy of documentation, or related issues.

Completion and Demonstration

After the final iteration, you will present and demo your full system to your supervisor.

Separately, you will present and demo your system to supervisors other than your own. Your own supervisor will not be present at this. You are therefore demoing your system to people who have not seen it before. You will also provide a video which you will use in the demo to explain the purpose of your system. The video is two minutes at an absolute maximum and can be considered as a sales pitch. Try to make it more interesting than just you standing and reading from a script. Example videos will be provided to you. The video must be in a standard format such as avi, mov, or mp4.

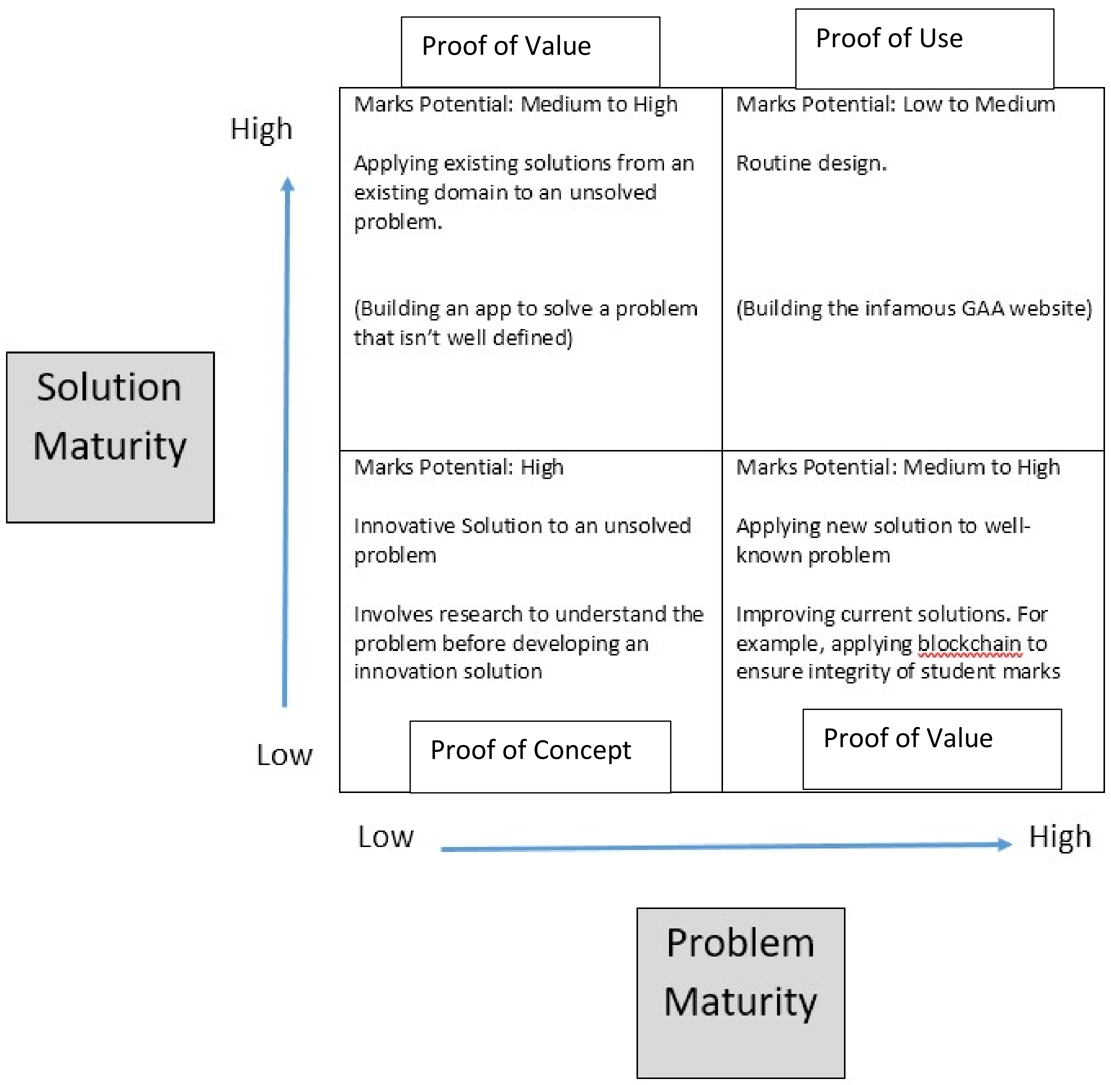
Mark Breakdown and Timetable

The mark breakdown for the FYP is as follows:

- 5% for the project proposal
- 7.5% for the High Level Design
- 7.5% for the development environment including documentation describing the development environment and licensing
- 30% for the system delivered in each two week iteration (5% for each iteration)
- 15% for the documentation and adherence to the development process in each two week iteration (2.5% for each iteration)
- 10% for final system: working functionality (marked by supervisor)
- 25% for final demo of system functionality and value (marked by supervisors of other FYP groups). 5% of these marks will be for the video.

It can be seen from the breakdown that it is vital that development take place throughout the two semesters with 65% of the marks being awarded for continuous delivery throughout the academic year.

As a guide to how the systems will be marked, below is a diagram showing the potential marks for the different types of fyp.


Proof of Value
Proof of Use
High
Marks Potential: Medium to High
Applying existing solutions from an existing domain to an unsolved problem.
(Building an app to solve a problem that isn't well defined)
Marks Potential: Low to Medium
Routine design.
(Building the infamous GAA website)
Solution Maturity
Marks Potential: High
Innovative Solution to an unsolved problem
Involves research to understand the problem before developing an innovation solution
Marks Potential: Medium to High
Applying new solution to well-known problem
Improving current solutions. For example, applying blockchain to ensure integrity of student marks
Low
Proof of Concept
Proof of Value
Low
High
Problem Maturity

The timetable is as follows: (

Date	Deliverable
week of 23/09/2019	Proposal
week of 07/10/2019	High Level Design
week of 21/10/2019	Environment
week of 04/11/2019	Iteration 1
week of 18/11/2019	Iteration 2
week of 20/01/2020	Iteration 3
week of 03/02/2020	Iteration 4
week of 17/02/2020	Iteration 5
by the 06/03/2020	Iteration 6
on the 11 and 12 /03/2020	Demo/Final System