# **Introduction**

**The** **Non Exam Assessment** is a practical computing exercise, the object of which is to produce a complete working solution to a problem. The project allows you to develop your practical skills in the context of solving a realistic problem or carrying out an investigation.

The project is intended to be as much a learning experience as a method of assessment; you have the opportunity to work independently on a problem of interest over an extended period, during which you can extend your programming skills and deepen your understanding of computer science.

The most important skill that should be assessed through the project is your ability to create a programmed solution to a problem or investigation. This is recognised by allocating 42 of the 75 available marks to the technical solution and a lower proportion of marks for supporting documentation to reflect the expectation that reporting of the problem, its analysis, the design of a solution or plan of an investigation and testing and evaluation will be concise.

# **The project report**

The project report should be presented in the following order:

|  |  |  |
| --- | --- | --- |
| **Section** |  | **Max mark** |
| 1 | Analysis | 9 |
| 2 | Documented design | 12 |
| 3 | Technical solution | 42 |
| 4 | Testing | 8 |
| 5 | Evaluation | 4 |
| **Total** |  | **75** |

This does not mean that the evidence must be produced in this order. Further detail on what should be included in each section is included later. The overall project is marked out of 75 with the following mark distribution above.

The project documentation should be clearly organised, including appropriate titles, page numbering and a contents page so that evidence can be easily found and referred to Items that are not directly required in the report, such as interview evidence and documents collected, should be included in appendices to the report.

**Other points:**

* The report should relate to the final version of the project
* Practice good project management, stick to stated deadlines
* Make regular backups of your project

# **Types of problem/investigation**

You are encouraged to choose a problem to solve or investigate that will be of interest to you and that relates to a field that you have some knowledge of. There are no restrictions on the types of problem/ investigation that can be submitted or the development tools (for example programming language) that can be used. The two key questions to ask when selecting a problem/investigation are:

* Do you have existing knowledge of the field, or are you in a position to find out about it?
* Is a solution to the problem/investigation likely to give you the opportunity to demonstrate the necessary degree of technical skill to achieve a mark that reflects your potential and goals?

Some examples of the types of problem to solve or investigate are:

* a simulation for example, of a business or scientific nature, or an investigation of a well-known problem such as the game of life
* a solution to a data processing problem for an organisation, such as membership systems
* the solution of an optimisation problem, such as production of a rota, shortest-path problems or route finding
* a computer game
* an application of artificial intelligence
* a control system, operated using a device such as an Arduino board
* a website with dynamic content, driven by a database back-end
* an app for a mobile phone or tablet
* an investigation into an area of computing, such as rendering a three-dimensional world on screen • investigating an area of data science using, for example, Twitter feed data or online public data sets
* investigating machine learning algorithms.

There is an expectation that within a college, the problems chosen by students to solve or investigate will be sufficiently different to avoid the work of one student informing the work of another because they are working on the same problem or investigation.

# **Technical Skills (Table 1)**

|  |  |  |
| --- | --- | --- |
| **Group** | **Model (including data model/structure)** | **Algorithms** |
| A | Complex data model in database (eg several interlinked tables)  Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standardFiles(s) organised for direct access    Complex scientific/mathematical/robotics/ control/business model      Complex user-defined use of objectorientated programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfacesComplex client-server model | Cross-table parameterised SQLAggregate SQL functionsUser/CASE-generated DDL scriptGraph/Tree TraversalList operationsLinked list maintenanceStack/Queue OperationsHashingAdvanced matrix operationsRecursive algorithmsComplex user-defined algorithms (eg optimisation, minimisation, scheduling, pattern matching) or equivalent difficulty Mergesort or similarly efficient sortDynamic generation of objects based on complex user-defined use of OOP modelServer-side scripting using request and response objects and server-side extensions for a complex client-server modelCalling parameterised Web service APIs and parsing JSON/XML to service a complex client-server model |

|  |  |  |
| --- | --- | --- |
| **Group** | **Model (including data model/structure)** | **Algorithms** |
| B | Simple data model in database (eg two or three interlinked tables)Multi-dimensional arraysDictionariesRecords Text filesFile(s) organised for sequential accessSimple scientific/mathematical /robotics/ control/business model   Simple OOP modelSimple client-server model | Single table or non-parameterised SQL Bubble sortBinary search   Writing and reading from files  Simple user defined algorithms (eg a range of mathematical/statistical calculations)Generation of objects based on simple OOP modelServer-side scripting using request and response objects and server-side extensions for a simple client-server modelCalling Web service APIs and parsing JSON/XML to service a simple client-server model |
| C | Single-dimensional arraysAppropriate choice of simple data typesSingle table database | Linear searchSimple mathematical calculations (eg average)Non-SQL table access |

Note that the contents of Table 1 are examples, selected to illustrate the level of demand of the technical skills that would be expected to be demonstrated in each group. The use of alternative algorithms and data models is encouraged. To achieve top marks you should be looking to carry out a project in group A.

Although the table above only affects the marks for your technical solution remember that the more technical skills will also give you more scope in the analysis and design sections to achieve the best marks.

# **Coding Style (Table 2)**

|  |  |
| --- | --- |
| **Style** | **Characteristic** |
| Excellent | Modules (subroutines) with appropriate interfacesLoosely coupled modules (subroutines) – module code interacts with other parts of the program through its interface onlyCohesive modules (subroutines) – module code does just one thingModules(collections of subroutines) – subroutines with common purpose groupedDefensive programmingGood exception handling |
| Good | Well-designed user interfaceModularisation of codeGood use of local variablesMinimal use of global variablesManaged casting of typesUse of constantsAppropriate indentationSelf-documenting codeConsistent style throughoutFile paths parameterised |
| Basic | Meaningful identifier namesAnnotation used effectively where required |

The quality of your code will also be important to the overall mark you achieve, marks within each band are allocated according the skills. You must use excellent coding style to achieve top marks.

The descriptions in Table 2 are cumulative, ie for a program to be classified as excellent it would be expected to exhibit the characteristics listed as excellent, good and basic not just those listed as excellent.

Project Development Calendar 2017/2018



The dates stated below are approximate. Specific dates set for completion of each stage will be given later.

It is vital that all deadlines during the development are met. These will be closely monitored.

|  |  |  |
| --- | --- | --- |
| Stage | **Start** | **To be completed** |
| Initial idea / Background | March 2017 | March 2016 |
| Analysis – investigation of current system | March 2017 | April 2017 |
| Analysis – specifying the new system | April 2017 | May 2017 |
| Design | May 2017 | July 2017 |
| Implementation | September 2017 | January 2018 |
| Testing | January 2018 | February2018 |
| Appraisal | March 2018 | March 2018 |

The exam board recommend to devote at least 50 hours of lesson time to the project overall, as a student you will be expected to also work on your project in your own time. We are focusing on the project in the final weeks of your first year (around 25 hours), you will then have one lesson a week next year dedicated to your project.

Some of the lessons in the first year will look at important skills such has modelling and databases. The rest of the time will be for you to investigate your project ideas and to complete your analysis.

The aim of this schedule is to ensure the project is completed on time. It will give you enough time to produce a good piece of coursework yet will ensure that it will be out of the way before the final exam period begins.

**Personal Log Book**

Write the date in the relevant box as you start and complete each section. You will be told the deadline for each section. There will be an opportunity to have a draft of each section marked and feedback given before the final version is submitted.

**Analysis** **Deadline** May 2017

**Date started** March 2017 **Date completed**

You are expected to:

* produce a clear statement that describes the problem area and specific problem that is being solved/investigated
* outline how they researched the problem
* state for whom the problem is being solved/investigated
* provide background in sufficient detail for a third party to understand the problem being solved/ investigated
* produce a numbered list of measurable, "appropriate" specific objectives, covering all required functionality of the solution or areas of investigation (Appropriate means that the specific objectives are single purpose and at a level of detail that is without ambiguity)
* report any modelling of the problem that will inform the Design stage, for example a graph/network model of Facebook connections or an E-R model.

A fully scoped analysis is one that has:

* researched the problem thoroughly
* has clearly defined the problem being solved/investigated
* omitted nothing that is relevant to subsequent stages
* statements of objectives which clearly and unambiguously identify the scope of the project
* modelled the problem for the Design stage where this is possible and necessary.

**Progress Checker**

|  |  |  |
| --- | --- | --- |
| **Task** | **Draft** | **Final** |
| Background to/identification of problem  |  |  |
| The prospective user(s) |  |  |
| Investigation methods |  |  |
| Interview questions and transcript of responses |  |  |
| Relevant documents |  |  |
| Description of current system  |  |  |
| Problems with current system |  |  |
| Additional user requirements/limitations |  |  |
| Context diagram (Data Sources and Destinations) |  |  |
| Data Flow Diagram(s) for existing system |  |  |
| Analysis Data Dictionary (User Data Requirements) |  |  |
| Data volumes |  |  |
| Purpose of the new system |  |  |
| Specific objectives of the proposed system |  |  |
| Limitations of the proposed system  |  |  |
| Constraints placed on new system |  |  |
| Appraisal of the feasibility of potential solutions |  |  |
| Proposed solution justified |  |  |

If appropriate:

|  |  |  |
| --- | --- | --- |
| E-R Models for database projects |  |  |
| Identification of Objects and Object analysis diagrams for Object-oriented programmed solutions |  |  |

**Design** **Deadline** July 2017

**Date started** May 2017 **Date completed**

Students are expected to articulate their design in a manner appropriate to the task and with sufficient clarity for a third party to understand how the key aspects of the solution/investigation are structured and on what the design will rely, eg use of numerical and scientific package libraries, data visualisation package library, particular relational database and/or web design framework.

The emphasis is on communicating the design; therefore it is acceptable to provide a description of the design in a combination of diagrams and prose as appropriate, as well as a description of algorithms, SQL, data structures, database relations as appropriate, and using relevant technical description languages, such as pseudo-code. Where design of a user interface is relevant, screen shots of actual screens are acceptable.

**Progress Checker**

|  |  |  |
| --- | --- | --- |
| **Task** | **Draft** | **Final** |
| ***Outline system design:*** |  |  |
| Overall description of system |  |  |
| Structure diagram/ Module descriptions |  |  |
| IPSO chart |  |  |
| ***User interface design (HCI):*** |  |  |
| HCI rationale |  |  |
| Sample of planned data capture and entry designs |  |  |
| Sample of planned valid output |  |  |
| ***Data requirements for programmed solution:*** |  |  |
| Description of Record Structure |  |  |
| Validation Required |  |  |
| File organisation and processing method  |  |  |
| ***Data requirements for database solution:*** |  |  |
| Normalised tables and revised E-R diagram |  |  |
| Identification of appropriate storage media |  |  |
| ***Processing:*** |  |  |
| Processing required/ System flowcharts |  |  |
| Algorithms required for data transformation |  |  |
| ***or for object oriented solution:*** |  |  |
| Class/object diagrams/definitions |  |  |
| Object behaviours and methods |  |  |
| ***Other aspects:*** |  |  |
| System security / Security and integrity of data |  |  |
| Overall test strategy |  |  |

**Implementation** **Deadline** January 2018

**Date started** September 2017 **Date completed**

Students should provide program listing(s) that demostrate their technical skill. The program listing(s) should be appropriately annotated and self-documenting (an approach that uses meaningful identifiers, with well structured code that minimises instances where program comments are necessary).

Students should present their work in a way that will enable a third party to discern the quality and purpose of the coding. This could take the form of:

* an overview guide which amongst other things includes the names of entities such as executables, data filenames/urls, database names, pathnames so that a third party can, if they so desire, run the solution/investigation
* explanations of particularly difficult-to-understand code sections; a careful division of the presentation of the code listing into appropriately labelled sections to make navigation as easy as possible for a third party reading the code listing.

**Progress Checker**

|  |  |  |
| --- | --- | --- |
| **Task** | **Date/Time** | **Done** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Testing** **Deadline** February 2018

**Date started** January 2018 **Date completed**

Students must provide and present in a structured way for example in tabular form, clear evidence of testing. This should take the form of carefully selected and representative samples, which demonstrate the robustness of the complete, or nearly complete, solution/thoroughness of investigation and which demonstrate that the requirements of the solution/investigation have been achieved. The emphasis should be on producing a representative sample in a balanced way and not on recording every possible test and test outcome. Students should explain the tests carried out alongside the evidence for them. This could take the form of:

* an introduction and overview
* the test performed
* its purpose if not self-evident
* the test data
* the expected test outcome
* the actual outcome with a sample of the evidence, for example screen shots of before and after the test, etc, sampled in order to limit volume.

**Progress Checker**

|  |  |  |
| --- | --- | --- |
| **Task** | **Draft** | **Final** |
| *Module test plan with individual tests using a minimal set of test data including:* |  |  |
| expected results for clearly defined typical data |  |  |
| expected results for clearly defined erroneous data |  |  |
| expected results for clearly defined boundary data |  |  |
| samples of annotated hard copy of actual test runs |  |  |
| *System test plan with tests showing the system working:* |  |  |
| Plan showing typical sequence of actions  |  |  |
| annotated hard copy showing the system working  |  |  |

**Appraisal** **Deadline** March 2018

**Date started** March 2018 **Date completed**

Students should consider and assess how well the outcome meets its requirements. Students should obtain independent feedback on how well the outcome meets its requirements and discuss this feedback. Some of this feedback could be generated during prototyping. If so, this feedback, and how/ why it was taken account must be presented and referenced so it can be found easily.

Students should also consider and discuss how the outcome could be improved more realistically if the problem/investigation were to be revisited.

**Progress Checker**

|  |  |  |
| --- | --- | --- |
| **Task** | **Draft** | **Final** |
| Comparison of performance against objectives |  |  |
| Clear evidence of user feedback |  |  |
| Analysis of user feedback |  |  |
| Suggested improvements based on user feedback |  |  |