Guidelines for Final Year Project Reports in Computer Science at UCC

Department of Computer Science
University College Cork

February 2020

Abstract

The report is a key component of the final year project since it is the lens through which others view it and judge it. A hastily- or poorly-written report will not do justice to what has been accomplished, and so it is vital that the report be approached with the same care and attention as the core technical aspects of the project. This document describes the various elements of the report and how they should be organized. It also provides some guidance on the how to approach the writing process.

Contents

1 Introduction

The final year project is the single most important element of your degree, and a well-written and well-presented report is key to how you communicate what you have achieved to others, especially those who will evaluate it. The report represents the only window some readers have into what you have accomplished, so they should be able to determine what has been done, to appreciate its technical merit and to assess the overall quality of the work from the report alone.

Your job as a writer is to convince the reader that the project represents a significant, non-trivial piece of technical work carried out in a competent, professional manner. Even the most polished report will not be able to camouflage the deficiencies of a poor or insubstantial project, but equally a poor report can obscure the true worth of an otherwise impressive body of work. Treat the report as an integral and vital component of the project, not as an afterthought to be cobbled together hastily at the last minute.

There is no artificial upper or lower limit on the length of the report: it should be as long as required to describe the work in appropriate detail but without pointless wordiness. However, if your work can be adequately described in ten pages, say, then it is very unlikely to have enough substance to impress anyone.

Later sections of this document describe various aspects of the report in greater detail. Section 3 clarifies what is expected of the substance of the project itself and of the report that describes it. Section 4 maps out the broad themes to be addressed in the report and a recommended breakdown into sections that reflects those themes. Section 6 contains advice on various aspects of the writing process.

The work presented and the report that describes it must be the fruits of your own individual effort. It is acceptable to rely on the work of others, but only if you acknowledge that debt and clearly delineate the boundary between your work and theirs. If you stand on the shoulders of giants, it seems shabby, even dishonest, not to give them credit for their help. Section 5 describes the convention to be adopted in citing exiting material (quoted text, figures, tables, images and so on) that your project relies upon while Appendix A describes the policy in regard to academic integrity as it relates to final year projects.

Writing always seems to take longer than planned, even for experienced writers, so it is important that you allocate sufficient time to allow for this.

Your supervisor will guide you through the writing process, but do not expect him to work miracles if you leave everything to the last minute.

The details such as relevant deadlines and submission procedures will be communicated in due course. Typically, the project open day is held in the last week of the second semester with the project report due about two weeks later. The submission deadline will be strictly enforced.

2 General guidance on technical writing

Subsection 2.1 provides some general guidance on the tone of the report. Subsection 2.2 summarizes some of the practical details.

2.1 General advice

The report should be a coherent, objective account of the work: what the objectives were, how the task was approached, what was accomplished and how *etc.*. It should not be (a) a narrative account of how you spent the year, (b) a guided tour of your code or (c) a user manual.

The report should be written in the third person ("he", "she", "it", "they") and not as a narrative written in the first person ("I", "me"). The tone of the report should be formal and not conversational, but without being pompous or stilted. It should be written mainly in the present tense, though the past tense may be appropriate when describing past work *e.g.* "Smith developed an $O(n^2)$ algorithm for the *Bandersnatch Problem*."

Clarity and readability are essential for successful technical writing. Keep your prose clear, direct and concise. Avoid superfluous verbiage, convoluted sentence structures and waffle. Muddled or unintelligible prose is guaranteed to frustrate any reader, so rephrase clumsy, contorted or obscure passages, so that the meaning is clear. The objective is not merely to be understood, but to express yourself in way that cannot easily be misunderstood.

Always keep your intended readership in mind when writing. You should assume that the reader is technically knowledgeable and familiar with most mainstream computer science concepts (*e.g.* a good CS graduate), but not necessarily with every arcane detail of every language, package or technology. You should pitch the discussion so that the flow of ideas is clear, but without belabouring the obvious or the straightforward. There is no need to dwell on standard ideas such as binary trees, or recursion or Java's for-loop syntax,

but if you rely on the Java Reflection API, say, then a brief summary may be in order to remind the user of its capabilities.

Overall, try to make the reader's task as rewarding and painless as possible. The less of a chore the reading of your report, the more favourable an impression it will leave in the mind of the reader.

It goes without saying that a report littered with spelling, grammatical or punctuation errors creates a very poor impression. Make sure you proofread your text carefully and use a spelling checker as you write.

2.2 Practical details

Document-preparation systems You may use whatever document-preparation system you like, as long as it is capable of meeting the requirements laid out in this document and is capable of generating PDF. LaTeX [13] is popular, as is Word (and its various clones such as LibreOffice [6] *etc.*). Choose whichever system you feel most comfortable with, but LaTeX has better support for integrating material such as references, mathematical content, code listings, figures, graphs and so on. There is a wealth of explanatory material on all aspects of technical writing using LaTeX [5].

Numbering Pages should be numbered at the bottom of the page using Arabic numerals beginning with the first page of the Introduction as Page 1. Note that material appearing before the first page of the Introduction need not be numbered.[1]

Dimensions The document should use A4 paper with wide margins and 12-point type. The hard-copy version should be printed single-sided.

Submission The project will be submitted both as hard-copy and as a soft-copy PDF. You will also be expected to submit a soft copy of your code. The details of the submission process will be communicated in due course.

[1] For the purposes of the project, you may leave all the front matter unnumbered, though the standard publishing convention is to number these elements using lowercase Roman numerals.

3 Expectations

Most projects involve the development of some piece of software, though there are other possibilities as listed below. Expectations and the assessment yardsticks applied vary between different types of projects, reflecting the underlying diversity of the projects themselves. Clearly what may be appropriate in the case of a software development project may be pointless and irrelevant in the case of an algorithm design project.

Software development project The most common category of project involves the design and implementation of some substantial piece of software or perhaps a hardware-software system.

Research project (empirical) Other projects typically involve the experimental evaluation of some idea (algorithm, heuristic *etc.*). This may involve the bench-marking of some software package or a rigorous comparison of a number of competing algorithms or protocols.

Research project (investigative) Projects in this category might involve the development of a new algorithm for some problem or an enhancement of an existing algorithm.

Some projects will mingle elements of all three types. Most of the discussion that follows focuses on software development projects (by far the most common variety), but the other types are also touched on.

Given the diversity of projects, it is impossible to give a precise, rigid formula for what constitutes a good or a bad project. Ultimately, the evaluation of each project is a matter of judgment for the examiners, however the paragraphs that follow give some insight into what examiners look for when reviewing a project.

Every project is expected to demonstrate evidence of ability to tackle a substantial task of some complexity and evidence of technical competence and appropriate professional methodology in the approach to that task. "Substantial" means requiring sustained effort and technical skill over an extended period, not something that could be completed in a matter of days. The work should demonstrate technical skills appropriate to a would-be graduate in computer science, not something, say, that is within the reach of an average first or second year student.

The work should be conducted in a professional manner using methodology appropriate to the task. For example, a software development project would be expected to demonstrate a professional approach to analysis, design and implementation, the finished product should be completed to a good standard. Every project should include an appropriate evaluation (*e.g.* user survey where appropriate).

Examiners will assess projects in part on the basis of the overall technical merit of what has been accomplished and partly by their judgment on how well the body of work rates on the various qualitative criteria mentioned above (how "substantial", how "complex", how "competent", how "professional" and so on). Each project is assessed by at least two examiners and the assessment process is conducted under the oversight of the External Examiner.[2]

Not every project achieves everything that was envisaged or hoped for at the beginning, but a project can still receive a respectable mark if what was completed represents an acceptable technical accomplishment.

4 Organization and structure

This section specifies the the standard organization of a project report. It focuses mainly on software development projects but also touches the other types. Subsection 4.1 discuses the heart of the report and the sections into which it should be subdivided. Subsection 4.2 specifies the format and sequencing of the other elements (title page, table of contents and so on).

4.1 Main sections

The report should have the following elements and they should appear in the order shown. The starred elements are optional *i.e.* may not be relevant or appropriate in every case. The body of the report comprises the six sections

[2]The External Examiner ("Extern" for short) is a senior academic from outside the University appointed to oversee the assessment arrangements for the degree programme and to ensure that standards being applied are comparable to those of peer institutions at home and abroad. The Extern visits UCC in the summer to review arrangements such as exam papers and exam marking. Externs historically take a keen interest in project reports.

in the middle segment (from "Introduction" to "Conclusions"),[3] which are discussed in greater detail later in this sub-section. The material appearing before the first dotted line ("Title" to "Table of Contents") is referred to as *front matter*. The format of these elements will be discussed in Sub-subsection 4.2. The material after the second dotted line is referred to as *back matter* and these are dealt with in Sub-subsection 4.2.1.

Title Page

Abstract

Declaration

Acknowledgements*

Table of Contents*

Introduction

Analysis

Design

Implementation

Evaluation

Conclusions

Appendix*

References

[3]You may vary this six-section structure if, having discussed the matter with your supervisor, he agrees that a different structure provides a better, clearer presentation of the work. For example, where two prototypes were developed, there might be one chapter per prototype each containing analysis, design and implementation subsections rather than a single chapter each for analysis *etc.* as suggested below.

The substantive sections in the body of the report should be broken onto subsections and sub-subsections as appropriate. Done properly, this can enhance the readability of the document. Not only do the section and subsection headings provide a "map" of the structure, but this partition permits the reader to choose which aspects to read in detail and which to skim over.

4.1.1 Introduction

The Introduction should provide a brief non-technical overview of the project. It should indicate clearly what the goal was and what was achieved and also how, in broad terms, the project was conducted. It may include a thumbnail sketch of the main concepts or technologies used, but without any of the technical details.

The idea is to present a summary of the work that is accessible to as wide an audience as possible. It should try to enthuse the reader to read the rest of the report.

4.1.2 Analysis

This section should describe both the *objectives* of the project and any relevant *context*. It should present a clear and comprehensive picture of what the project was intended to achieve.

The project goals should be spelled out in detail, the expected deliverables listed and their requirements detailed. In the case of a software development project, a rigorous and comprehensive requirements analysis should be carried out and the detailed specifications documented thoroughly.

This section should also describe any background material that may be necessary to understand the project or to understand any considerations that shaped any of the major technical decisions taken in the course of the project.

If your project builds on existing work, it is appropriate to summarize that work here. Similarly, if some existing work tackles a related or similar problem, then it is appropriate to summarize that work and state what distinguishes it from your own.

For an empirical research project, this section should describe in detail the concepts (*e.g.* algorithms, heuristics *etc.*) that are to be investigated in the report. The report should present a comprehensive literature review

that surveys what is known and not known about the topic, the strength and weaknesses of existing approaches and so on. This section should also clearly spell out what it is hoped the project will contribute to the state of knowledge in this area.

In the case of an investigative research project, this section should include a precise formulation of the problem to be investigated and any terminology to be used should be introduced. It should also include a comprehensive literature review as above.

4.1.3 Design

This section should provide a high-level overview of the "architecture" of the system but without any code-level details. It should describe the major components of the system, the role that each plays and the interrelationships between them. It should also describe any tools, technologies or packages or libraries that your system depends upon and describe how they fit into the overall design. Every significant high-level technical decision reflected in the design should be justified: the reader should be convinced that the proposed design is both reasonable and realizable.

In the case of an empirical research project, this section should outline the design of the experiments to be conducted and the experimental methodology underpinning them. It should describe the test data sets to be used and it should justify the choice.

In the case of a investigative project, say to develop an algorithm, this section should present the algorithm itself and illustrate of its operation perhaps by means of an example or two. Technical details such as proofs of correctness and performance analysis should appear in the Implementation section.

4.1.4 Implementation

This section should describe the actual implementation but with a focus on the technical challenges rather than the mundane coding details. There no need to describe straightforward aspects in great detail or to quote large blocks of code, but any non-obvious aspects of the implementation should be explained. For example, a brief summary of the main data structures employed might be included and the role that each plays in the overall scheme of things.

The objective is that a competent programmer should be able to understand how the software was constructed based only on your description of it. Do not give a blow by blow narrative account of your coding efforts, but it is appropriate to indicate any significant or unexpected technical challenges encountered along the way (flaky behaviour of third-party software, misleading or inadequate documentation and so on) and how they were resolved or overcome.

Where the implementation is incomplete, you should state clearly what has and has not been done. It is appropriate to sketch how uncompleted elements of the the system might be implemented, as long as you are honest in describing the status of uncompleted work.

For an empirical research project, this section should also describe how the experiments were conducted. For example, it might describe any software "apparatus" used to run the experiments and harvest raw data. It might also describe the details of how the raw data was analyzed, how any graphs or tables are constructed from the raw data and so on.

For investigative projects, this section should include any detailed technical work such as proofs of correctness and performance analysis. It should also describe any experimental work undertaken to validate the algorithm as above.

An Efficient Algorithm for the Jabberwocky Problem

Hugh Gigeaux

Final-Year Project– BSc in Computer Science
Supervisor: Dr Anne Other

Department of Computer Science
University College Cork

April 2016

Figure 1: Sample title page

4.1.5 Evaluation

The report should include evidence that the system was tested rigorously (unit testing, system testing and so on) to convince the reader that the software is fit for purpose. Any known errors in the software should be described.

A detailed and honest appraisal of how well the completed system meets the initial project objectives should be included. As stated earlier, it is not unusual for a project diverge from the original plan, which is acceptable as long as what was completed as constitutes a respectable body of work.

For a software development project, this section should discuss whether

the system is for purpose and to what degree. How featured is it? How well-tested and reliable is it? How user friendly is it? In some circumstances it will be appropriate to of human evaluation (*e.g.* user experience survey). Externs tend to be keen on this.

In the case of an empirical research project, this section should present the experimental results obtained, making appropriate use of tables, graphs and charts. The results should be analyzed and explained in detail. Any conclusions to be drawn from these results should be presented here and carefully justified with reference to the data.

For investigative research projects this section should include a detailed comparison with any existing or competing algorithms. Some experimental performance results based on a concrete implementation may also be appropriate.

4.1.6 Conclusions

This section shall be a summary of the project and what has been accomplished. You should also outline what conclusions can be drawn from your work: what does your project contribute to the sum of human knowledge? You should also sketch any future developments or lines of enquiry suggested by your work.

In particular, you should include

- A **reflection** subsection. What you have learnt during the course of the project? Which new skills did you acquire? Did you identify any shortcomings in your skills and methodology (e.g. time planning)?

- A **summary of how the project was conducted**. This can be very beneficial for the planning and time management of future projects. For example, you could insert here a graphical timeline where the various phases of your project undertaking are marked. Critical areas (e.g. too short or overlapping) should be highlighted here.

  A well-known format of such a timeline is known as Gantt chart.

4.2 Support elements

The previous section discussed the body of the report. In this section we describe the support materials. This includes the front matter (title page and so on) and the back matter (references and possibly appendices, if any).

Title page
The Title material should appear on a page of its own at the beginning of the report and include the following elements.

- The full title of the project
- The full name of the author
- The qualification for which the report is submitted ("Final-Year Project– BSc in Computer Science")
- Name of the supervisor of the project
- The department name ("Department of Computer Science")
- The name of the institution ("University College Cork")
- The Month and Year of submission

The title should be short and succinct and describe the project as accurately as possible.

Abstract Each report must contain an abstract that crisply summarizes the project in a single paragraph or so. Properly worded abstracts are vital in any form of technical writing. Most potential readers will use the abstract to assess whether the document itself is worth reading.

- The abstract should appear on a page of its own
- It should be positioned vertically on the page beginning roughly one third of the page from the top
- It should be labelled with the word "Abstract" (bold face)
- It should be between 50 and 200 words in length

Declaration The report must contain a signed declaration that the work submitted is the fruits of your own individual effort. This should appear on a page of its own and use the wording and layout provided in Appendix B. A report that omits this declaration or that is unsigned will not be accepted.

Acknowledgements You may include a brief paragraph of acknowledgements if you wish to record your thanks to those who have supported or helped you (either at a personal level or technically) during the course of the project.

It should appear in a page of its own and headed with the word "Acknowledgements (bold face) in a format akin to that used for the Declaration page.

4.2.1 Back matter

Back matter refers to the material at the back of the report after the substantive sections in the report body. Mainly this relates to the References section, discussed in more detail in Section 5, and possibly appendices (if any).

Appendices typically are used to include material that is secondary, whose inclusion within the main body of the report would disrupt the flow of ideas. For example, lengthy but straightforward mathematical proofs or derivations might be placed there (with a brief summary and pointer in the main body). It is also common to place bulky experimental data or graphs in an appendix. In such cases the body should include a summary and a pointer to the appendix. Other examples of material suitable for inclusion in an appendix include detailed API descriptions, summaries of software packages used, software user manuals and so on. If there is more than one appendix, they should be labelled alphabetically ("Appendix A", "Appendix B" and so on).

A list of all published material referred to in the document should be included at the end of the document in the References section. See the back of this document example and the section that follows.

5 Acknowledging sources

You must acknowledge the sources the any published material that you make use of in your report. Failure to do so can have very adverse consequences [10]. The details of each source (author name, title of the work in question, publisher and so on) should be recorded in the References section at the back of the report. This section should be arranged alphabetically by

author name (first author in the case of multi-author works)[4] and labelled numerically. Use the references at the back of this document for models of how books, articles or web resources are detailed.

Where you rely on some pre-existing technical result (theorem *etc.*) or other material, you should cite an authoritative source that presents that result or material.

Citation Within the text of the report you indicate the source by including its (numerical) label enclosed in square brackets *e.g.*

> The *Travelling Salesman Problem (TSP)* is NP-complete [8].

Reference Include an entry that includes the publication details of the work in question in the References section:

> [8] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1979. ISBN: 0716710447.

Here we are are relying on the Garey and Johnson book to substantiate the statement about the NP-completeness of the TSP.

You should also use references to provide pointers to background material which some readers may not be familiar with. If, say, your project employs some dynamic programming techniques, it may be appropriate to include a reference to one of the standard algorithm textbooks that discusses this concept [3] in case the reader needs to refresh his/her memory.

As quality control on the Web is uneven, printed sources are generally regarded as more authoritative than web pages. Exceptions include situations were websites make soft-copies of published material available [4] or websites such as `python.org` [7] that is the definitive source for information on the Python programming language.

[4]When referring to websites where there is no author listed, treat the web page name or document name as the "author" of the purposes of alphabetical listing.

6 Writing advice

This section includes some guidance with various aspects of the writing process. Appendex C contains some further helpful tips.

6.1 Drafting and revision

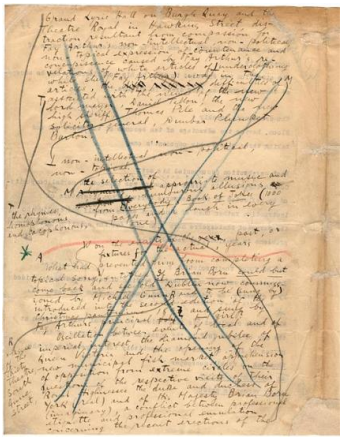

Figure 2: Manuscript fragment of the Joyce's *Ulysses* reflecting multiple revisions by the author.
SOURCE: Morgan Library and Museum, New York.

Very few can write perfect, fluent, well-organized prose at the first attempt. Most writers, even the most gifted ones, go through multiple iterations of revision and re-drafting before completing a document. It can be an exhausting and exasperating experience, but once you accept that the text will evolve through multiple drafts, it takes the pressure off the first draft as there is no expectation that it will be perfect.

Do not be afraid to discard material and rewrite it from scratch. Often a complete redrafting of a paragraph or section will result in a dramatically better text and is also usually less painful than you might think. Approach each revision with an open and critical frame of mind. The objective is not to find the minimal set of alterations that create an acceptable text, but to improve the clarity, readability and accuracy to create the best impression possible in the mind of the reader. Root out passages that are vague or inaccurate or open to misinterpretation and rework them so the the meaning is crystal clear. Do not wander off the point: readers have little patience for lengthy digressions on side issues of dubious relevance. Keep your prose crisp and direct. Find a form of words that expresses what you have to say clearly and succinctly and coherently. Avoid waffle. Above all never forget the reader: if he or she is bored, frustrated or mystified, he is unlikely to have a favourable impression of your work.

One approach that many find helpful is to start with an outline and then tackle each section one by one. The outline is a one- or two-page overview that maps out the main sections and includes a brief thumbnail sketch (a couple of paragraphs) of what will appear in each section. Most people find the "first bits" (Introduction and Abstract) hardest to write and so leave

them until last and start in the "middle" (Analysis, Design *etc.*).

6.2 Proofreading

Every piece of writing needs to be thoroughly checked for mistakes in spelling, punctuation and grammar and so on. You need to make proof reading an integral part of your writing regime. Try to enlist someone to proofread your document: fresh eyes will see problems that you can no longer see.

6.3 Quoting the words of others

If you quote any text (even a sentence) you must indicate the source as discussed in Section 5. In technical writing, it is rarely appropriate (or effective) to cut and paste lengthy passages of text even with attribution. If you rely on some author's description of some idea, then paraphrase instead in your own words (and indicate the source).

If you wish to include some short memorable quotation or aphorism, indicate the source either as above or by name, as illustrated below.

> "Testing shows the presence, not the absence of bugs."
>
> –*Edsger Dijkstra*

6.4 Figures

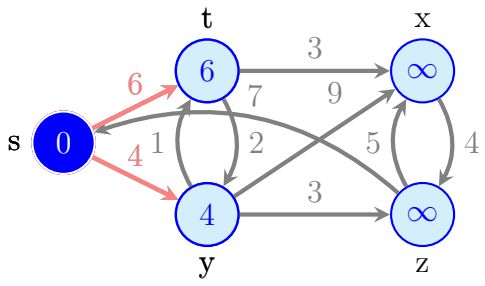

Figure 3: A weighted directed graph

Figures such as images, graphs, diagrams can often and idea more effectively than prose. You must, of course, indicate the source of any such material that is not your own work. It is also important that the figure is clear (not blurry or smudged) and legible. If you are presenting a graph of results from some experiment, make sure that the graph is presented at a scale that the reader can make sense of it (not too small) and especially that any text labels on the axes *etc.* are easily readable. While figures can be invaluable, it is

possible to overdo it. By all means include a screen-shot of your system's GUI, but one or two should suffice.

Think carefully about how you use colour. All graphs and figures should we intelligible when printed in black and white.

Every figure should have a caption that describes it, such as you see in the figures shown above. It is almost always necessary to include some textual material that introduces or explains each figure. (We omit this requirement here as the figures included in this document are not intended to be integrated with the text, whereas those in your report should be.) For most figures, *e.g.* a graph of experimental results, there will be some discussion that interprets it and justifies any conclusions to be drawn from it.

6.5 Code

Organizing your report around a guided tour of the code is very poor practice. Source code is rarely the most effective way to express an idea and code should be quoted sparingly, if at all, in the body of the report. When discussing your implementation for example, brief snippets of code may be appropriate to explain how some technical details were handled but you should avoid including lengthy passages of code. Try to explain your ideas, not the arcane coding details that were employed to implement them.

Figure 4: An overview of Java
SOURCE: Petrus Bertius. Tabularum geographicarum contractarum. Amsterdam, 1618.

An exception to the above rule is appropriate however when presenting an algorithm, where pseudo-code (or code) generally *is* the clearest way of presenting the ideas. It is necessary to accompany any pseudo-code with some explanatory text to help the reader interpret it. This may a summary of its operation (*e.g.* by means of an example) and some discussion/proof to demonstrate its correctness.

6.6 Some useful resources

For sound advice on (non-technical) writing in general, Strunk and White's slim classic *Elements of Style* [12], a stalwart on US campuses for generations, is well worth reading.

For a reliable source on spelling and related matters, you should refer to *The Oxford English Dictionary* [1]. There are copies in the library and the on-line version is quite useful. For guidance on punctuation and grammar, the authoritative *Fowler's Dictionary of Modern English Usage* [9] is a good source. There is also a (limited) on-line version.

The IEEE Computer Society has a useful style guide [11]. It is aimed at authors intending to publish in the organization's technical journals and offers guidance on technical (specifically computer-related) writing.

Appendices

A Academic integrity policy

All students are bound by UCC's Plagiarism Policy [2] and are expected to familiarize themselves with the provisions (and penalties) of that policy. The principles set out below interpret this policy in the context of final year projects in Computer Science.

The overarching principle is that the presentation of the project, encompassing both the project report and the underlying body of technical work, should be accurate and honest in describing what has been accomplished and in specifying the precise individual contribution of the examinee.

Report plagiarism The project report must be written entirely and exclusively by the examinee. The source of any material included in the report that was not created by the examinee must be carefully acknowledged using the convention specified elsewhere in this document. This provision applies not only to any quotation of passages of text, but also any other material, including but not limited to: tables, figures or images, code or pseudo-code fragments, mathematical material such as proofs or equations and experimental results. It is not sufficient to list the source among the references at the end of the report, the use of the material must be flagged in the text at the point of use.

Code plagiarism The project implementation must be the be result of the examinee's individual effort. Unacknowledged use of source code from any quarter (printed, online or human) is unacceptable. The above provision applies specifically to source code; building on existing publicly available software tools (packages, libraries *etc.*), is acceptable provided the use is properly acknowledged.

Source code, other than the examinee's own, may be incorporated into the project only under the following strict conditions: (i) that the express permission of the project supervisor has been granted for such use, (ii) that its use is necessary and appropriate *e.g.* where the project requires the adaptation of some existing open-source code, (iii) that the source is properly acknowledged and (iv) that the boundary between the examinee's work and the original code body has been clearly spelled

out in detail both in the source code (using comments) and in the body of the report.

Data plagiarism and falsification The source of any sample data or benchmarks used to derive experimental results must be acknowledged. The source of any third-party results quoted in the report must be acknowledged and all results must be reported honestly without selective bias or distortion or misrepresentation.

Misrepresentation The report should not misrepresent the work in any way. It must not exaggerate the extent of the work (*e.g.* what has been completed and what not) or its quality (*e.g.* the reliability or capability of any software produced) nor should it make any claims that misrepresent the novelty or contribution of the work.

Unethical behaviour The examinee is expected to respect the terms of use attached to any technical and other resources used in the project. This includes any conditions stipulated by the creators or owners of any software employed, any confidentiality or disclosure agreements entered into and the acceptable usage policy laid down by the University.

Unauthorized and unacknowledged assistance The project and the report must be the work of the examinee alone. The examinee may not accept help from any quarter to conduct any portion of the project work or to write any portion of the project report.

It is acceptable get others to critique or proofread the text of the report to suggest improvements, but the text submitted must be entirely of the examinee's creation and his/hers alone. It is acceptable to discuss general programming principles with others but all of the code submitted must be entirely of the examinee's creation and his/hers alone. Receiving tutorial help mastering any of the technical tools (language, package *etc.*) required for the project is acceptable as long as the assistance is of a general nature and not specific to the application of that tool to the project.

Complicity Assisting another in violating this policy is itself a violation of this policy.

Students who have any questions about any aspect of this policy should discuss these either with their project supervisor or the project coordinator.

B Declaration of originality

Declaration of Originality

In signing this declaration, you are conforming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:

- this is all my own work, unless clearly indicated otherwise, with full and proper accreditation;
- with respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;
- with respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Signed:.....

Date:.....

C Miscellaneous tips

1. Favour active over passive voice, e.g. "Kieran wrote this document." rather than "This document was written by Kieran." This is not an absolute rule, but overuse of the passive voice sounds stilted and awkward.

2. Be consistent in your use of tenses and do not flip back and forth between tenses.

3. If you adopt a rule on some issue *e.g.* on how to capitalize section headings, be consistent in how you apply it. For example in this document words like organize or plagiarize use the -ize suffix consistently.

4. Names of programming languages and software packages are proper nouns and should be capitalized e.g. `Java` and not `java`.

5. Numbers of ten or less should be spelled out in English, numerals should be used for larger values *e.g.* "We conducted six experiments with 200 repetitions of each."

6. Consider any awkward page breaks particularly where they result in a widow or orphan: a one-line fragment separated by a page break from the material with which it should naturally be connected. For example, a section heading at the bottom of a page where the text of the section begins at the top of the next page.

7. Do not confuse `there`, `their` and `they're` or `its` and `it's` or `to`, `two` and `too`.

8. Explain every term, acronym or abbreviation the first time it appears in the text unless its usage is universally understood.

9. Avoid contractions such as `isn't`, `doesn't`, `can't` and so on.

10. Avoid slang and colloquialisms such as "My application was like totally banjaxed so I fired up the debugger thingy to have a look-see to try to belt it back into correctitude LOL.".

11. Do not misuse apostrophes. Learn to distinguish between `cats`, `cat's` and `cats'`.

12. The following is a list of words that are commonly misspelled in technical writing.

across	divisable
accommodate	feasible
acknowledgment	independent
complement	implement
consistent	occurrence
dependent	preceding
descendant	referring
discrete	separate

13. Some handy rules[5]:

    - Watch out for prepositions that sentences end with.
    - When dangling, consider your participles.
    - About them sentence fragments.
    - Make each pronoun agree with their antecedent.
    - Don't use commas, which aren't necessary.
    - Try to never split infinitives.

[5]From *Mathematical Writing* by Donald E. Knuth, Tracy Larrabee and Paul M. Roberts. Course Notes for CS209, Stanford University, 1989.

References

[1] Jeremy Butterfield, ed. *Fowler's Dictionary of Modern English Usage*. 4th. Oxford University Press, 2015. ISBN: 978-0199661350.

[2] University College Cork. *UCC Plagiarism Policy*. URL: `www.ucc.ie/en/exams/procedures-regulations/`.

[3] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. McGraw-Hill Higher Education, 2001. ISBN: 0070131511.

[4] Charles Darwin. *On the origin of species*. (Available online: `www.gutenberg.org/ebooks/22764`). London: John Murray, 1859.

[5] Marc van Dongen. *LaTeXand Friends*. URL: `csweb.ucc.ie/~dongen/LAF/LAF.html`.

[6] Document Foundation. *LibreOffice documentation*. URL: `www.libreoffice.org`.

[7] Python Software Foundation. *Python website*. URL: `www.python.org`.

[8] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979. ISBN: 0716710447.

[9] *Oxford English Dictionary*. 3rd. (Available online: `http://www.oed.com`). Oxford University Press, 2010. ISBN: 978-0199571123.

[10] Helen Pidd. "German defence minister resigns in PhD plagiarism row." In: *The Guardian* (1 Mar. 2011). URL: `www.theguardian.com/world/2011/mar/01/german-defence-minister-resigns-plagiarism`.

[11] IEEE Computer Society. *IEEE Computer Society Style Guide*. URL: `http://www.computer.org/cms/Computer.org/Publications/docs/2015CSStyleGuide.pdf`.

[12] William Strunk Jr. and E. B. White. *The Elements of Style*. 4th. Longman, 1999. ISBN: 978-0205309023.

[13] Latex Project Team. *LaTeX- a document preparation system*. URL: `www.latex-project.org`.