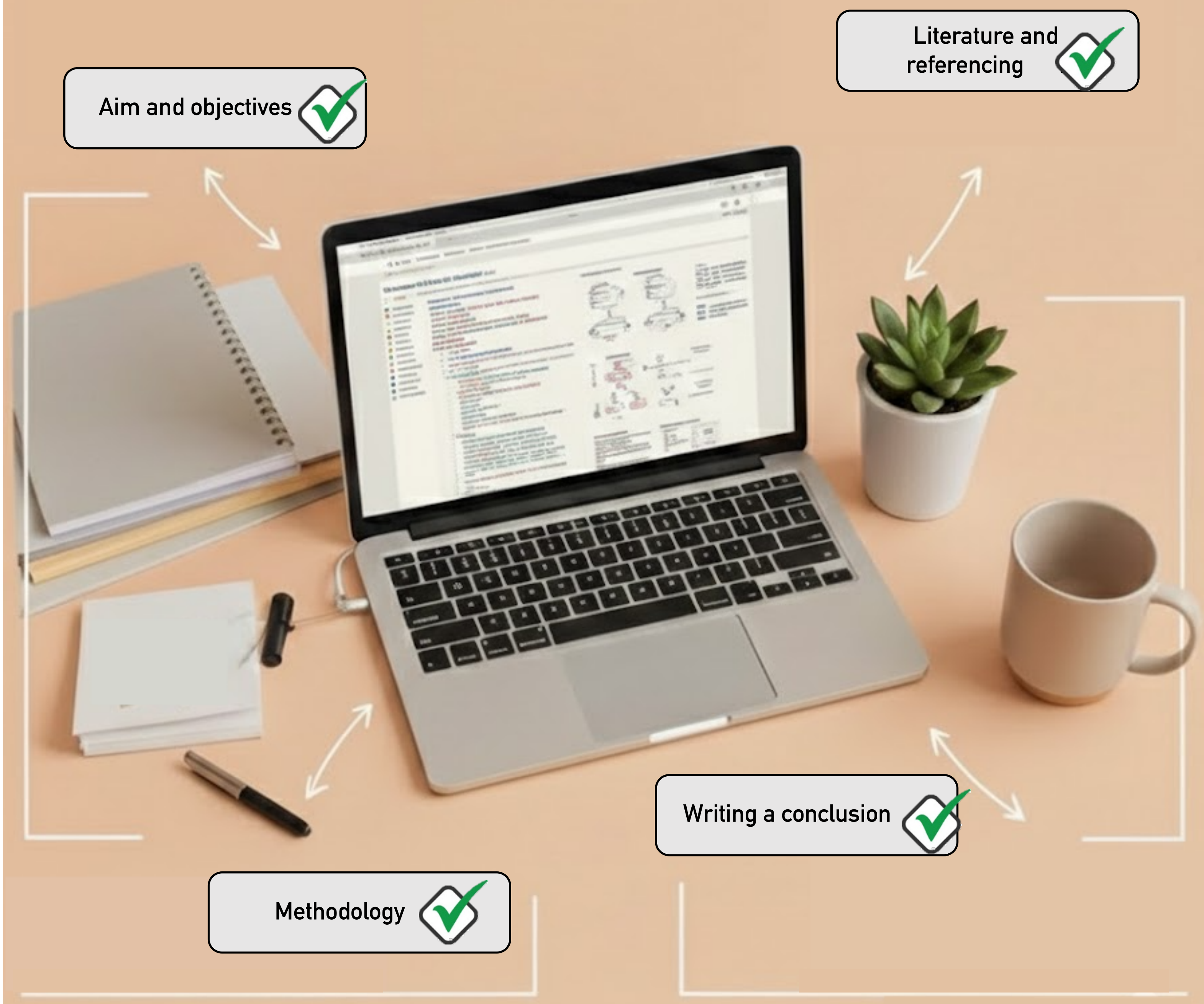
THE DISSERTATION



MARK PERRY, BRUNEL UNIVERSITY

FYP and MSc Dissertation advice for Computer Science students

Click link to access topic:

Note: this document is for guidance only. There are many ways of doing a project! This document presents an informed opinion, on a very limited number of issues. What may be correct in some contexts can be completely incorrect in others. It is therefore critical to seek the input of your supervisor into your own project on a regular basis to fill in the gaps and, perhaps, to get an alternative opinion on these.

The difference between the aim and objectives

The project aim (note that the word Aim is *singular* – so add one aim only) is your high-level intended achievement, while objectives are concrete, measurable steps required to meet the aim/s – they are specific quantifiable achievements.

Your aim and objectives *can* change over the course of the project, but try not to do this.

Writing the Aim

Your aim should usually be 1 sentence long, and outline, in as much detail as possible, what your project will attempt to do. Avoid specific details of the implementation process if possible, and especially if these aren't the point of your project, the methods. Try not to use terms like "Towards a …", "I will…", as these terms are too vague. Usually, the aim begins with something like "To develop a…". Try to think of your Aim as an extended version of the title of the project, where you have the space to actually say what you are planning to do.

Writing the Objectives

Objectives are actions that, if you can tick them off, will show that you have met your aim, and successfully completed your project. Sometimes these are similar to the chapters that you will write, but they may be linked to activities you want to achieve that cross chapters. They usually start with action verbs, for eg. compare, design, develop, evaluate, test. Something like 5-8 objectives is typical – fewer than this usually shows insufficient granularity in decomposing your project activities, while having too many indicates either a project that is too ambitious, or one that has over-specified the process before the project has begun.

Try to make these objectives 'SMART'. See for eg. https://www.wrike.com/project-management-guide/faq/what-is-smart-in-project-management/

It is OK (and expected) for project objectives not to state *how* they will be achieved. This is usually something that will be done during the completion of that objective. For example, it would be normal to state something like "Evaluate and compare suitable approaches to determine a software engineering method to do <x>", rather than "I will use <method y> to do <solution z> as an approach to develop my software".

Bear in mind that these set out the questions that you will be marked on! Describe them in ways that allow their topics to be assessed by the marker as being completed.

Reasons for doing background reading

Why? To get to know more about an area; to identify gaps in knowledge in the area; to get a sense of how researchers in the area approach problems; to help formulate research question(s). Much (although not all) of what you read will be written up into your literature review.

The literature review sets out the context of your project, and shows the ways in which the work presented in the project connects to work done by other researchers, developers, or significant thinkers. It is *not* a summary of everything that you have read, but should help the reader understand *your* dissertation problem in the light of what *other* people have done.

If you are doing a computer science project, the vast majority of the literature should be on the **computer science** aspects of what you are studying (with the same point for any other degree type). Thus, for example, if you are looking at the design of a computer-based platform to support climate change activism, I would not expect much of the literature to be about the problems of climate change. This might be relevant if you are studying a degree in climate science, but this is not your degree, and you will get very few marks for this.

What is a citation, and what is a reference? Why use both?

Citations are required 'in-text' at point of use (so in the main written part of the dissertation), and references are required to give full publication details to allow readers to find the cited work (ie. at the end in its own section). Their importance includes setting the background of what has been done before, acknowledging other work, avoiding plagiarism, and allowing the reader to follow up on specific work. Cited references should also connect back to the discussion chapter at the end of the dissertation to compare your work to what was done before.

You cannot cite, but not reference, or reference without citing. Both are required.

Some kinds of reference are more reliable than others

Poor selection of references will not support your arguments and may result in a low or failing grade. Unfortunately, AI tools, even when used just to help you do literature searches often identify very weak or poor references, so please take care in following up on these.

Look for an understanding of what it is for material to be peer reviewed, and the status of this. Magazines and newspapers are created by journalists, and industry white papers, marketing, or technical documents are typically generated to sell a product or service. So, rather than being researched independently by academics in a peer-review, you must treat them differently: ask yourself: *how much can you trust what they say?* It's fine to use these materials (in moderation), but you might want to identify or mention why a journalistic or industry citation is used, or the degree to which it can be believed.

Not all academic articles are good quality, and some are better than others. Some journals are 'predatory', and their publishers make money by publishing very poor-quality material. One way to start assessing this is to look at the journal and to see if it is indexed by Brunel's library.

Of course, Brunel Library does not carry every journal – these can be incredibly expensive! Other solutions to check quality are to check whether the journal is indexed in recognised databases (Web of Science, Scopus, PubMed, IEEE Xplore, ACM Digital Library, depending on the topic). You can also check a journal's reputation through its Impact Factor (Clarivate), CiteScore (Scopus), or by using discipline-specific reputation rankings. Google can help you find and make sense of this information!

What is the purpose of a reference list in a project report?

The reference list provides full publication details, which enables a reader to locate a referenced work. It needs to be in a form that is a recognised citation style (usually Harvard). Academics marking your work like to see these references correctly formatted, and you will pick up marks for this, or at least you will not lose marks. Only the items that are cited should be included in the reference list, not everything that you have read related to the topic. It helps avoid a claim of plagiarism against you, although it will not prevent it (especially if you do not cite the references in the text, or they are hallucinations from ChatGPT!).

How many references do I need to include?

There is no right or wrong answer to this. This is also discipline dependent: it is common for more technical projects to include relatively fewer references than once the orient to business or social science topics, and even within this, there is a huge variation.

Historically, most dissertations in computer science at Brunel range from 20-40 references (slightly more than this for Masters level projects), but depending on how these references are utilised, you may need more, or less than this. If you go for lower numbers of citations/references, it may be that you have not explored the design space of your project effectively. If you include too many references, this might be an indication that you have spent too long in researching the background to your project, and not enough time doing the other parts of the work, or that your application of these references in your dissertation is too shallow. Once you have a draft version of your literature review, it would be helpful to share and discuss this with your supervisor to see whether you have sufficient coverage of the existing literature in your background section.

How much should I write?

Although it is not a requirement, consider writing towards the upper end of the word limit. If you write less it will seem like you have less to say - which means that you've probably done too little work, or not thought about the topic enough. Discuss this with your supervisor.

Write concisely, and focus what you write to the most important topics – these are the things that will help bring you a good grade. Do not attempt to write about *everything* that you did during your project: focus your attention on the most critical aspects of your work. Try not to repeat yourself, so if you did two things that were similar (eg. elements of your code, research methods, software testing), you don't need to describe them both in detail.

Can I go over the word limit?

No. At least not normally. If you feel that you need to do this, discuss it with your supervisor. You may be penalised if you go over this word limit. Our experience of very long student reports is that these tend to include significant 'padding' that is not strictly relevant to the assessment, and rarely brings additional marks to what you have written. Most succinct reports naturally highlight the very best of your work, rather than burying the content that would get you the highest marks inside more mediocre contributions.

Contrary to rumour there is no rule that you can submit 10% over the word limit.

Methodology? What is this?

This section or chapter needs to show how you are going to approach the problem that you have set out in the Aim and Objectives. It usually combines the *Software Development Lifecycle* (SDLC) and *Project Management*. For solo development, the SDLC is the technical "how", while Project Management is the process of organising the "what and when".

<u>SDLC</u> focuses on the stages of the software process itself in meeting your aims: Requirement Analysis, Design, Implementation, Testing, and Iteration. It ensures the software is built correctly.

<u>Project Management</u> focuses on oversight, mapping your aims to the constraints surrounding the work: Time, Scope, Resources, Quality, and Risk. It ensures you are building the right thing and can finish it on time.

These are complementary but distinct elements, but for the purposes of the Dissertation, they are often presented together, and merged into a single, unified workflow. This chapter will be used to show your planning: it is important that you DO NOT write this section after completing the project, as this should lead, not follow your design, development, or testing/evaluation activities.

For simulation modelling, AI, usability-related, experimental or qualitative research, or other projects in which software development is *not* the largest part of your work, you will need to deploy an alternative to the SDLC or add more information about the approaches that you will use in your dissertation. Some of this would probably be covered in the other chapters, but if it is about the approaches that you plan to use, and how they drive the overall plans and activities of your dissertation, they should be explained and justified here.

When writing this chapter, you may wish to compare approaches to *justify* your final selection/s. The chapter will also be an opportunity for you to describe and justify the *Methods* and *Tools* that you will use as you follow the methodology.

However, this chapter is not an essay that just summarises approaches to methodology or methods. It is intended to show what ***you*** plan to do, and in enough detail so that a reader (ie. your assessors) will a) understand the reasons for your choices, b) what you intend to do, and c) how these steps are related to one another in meeting your project's aim and objectives.

It may be helpful to follow this kind of structure in your methodology chapter:

1. Explain your reasons for using a methodology
2. Compare potential methodologies
3. Select and justify a methodology
4. Describe how you will implement this methodology, including the steps you'll follow,
5. Describe your project management actives not covered in the previous points, for example, covering you time planning or risk analysis
6. Describe and justify the individual methods, software, data, or other tools you plan to use, and how these fit into the selected methodology

Your Choice of a Methodology – which one?

Usually, this will involve selection of a Software Development Lifecycle (see above!).

This selection will (almost always) need to be suitable for a solo project: this may present a problem for you, as most SDLC methodologies are optimised for groups. Selection of your approach must consider this. For example, SCRUM meetings might feel a bit lonely on your own if you are following an Agile approach!

This selection is also both resource dependent (mainly people and time) and the needs of the Dissertation (you don't usually need to do commissioning, deployment or maintenance). Commercial SDLCs may not be ideal or may need to be adapted. If you drop or add anything, you'll need to say what these are, and why you did this. You may also need to supplement anything that you take out with other elements that bolster the remaining components.

Some methods are generally (though not always) considered weak, even if they are in the lectures. The Waterfall method is the most obvious of these (just look online for reasons!), but also the spiral model, and other 'enterprise-level' approaches. Other methods are so lightweight that they can barely be classed as SDLCs, for example, Kanban, or Rapid Prototyping. If you do choose to use these, be aware that you will need to give strong justifications.

You should use the literature to help guide you in selecting a methodology that fits what you are planning to do. Cite this!

Finally, please make sure that you actually follow this methodology when you do the project. It should drive your activities, and not something that you abandon once completed.

Why is it important to present your results, and in what ways can they be presented?

You may need to collect and report on data when you describe setting up your initial system requirements, or in evaluating your designs as you build them.

It is possible to convey the results of a project or piece of research in different ways. For example, they can be presented visually, or they can be presented through descriptive text. However*, raw data* and *results* are **not** the same thing, and data needs to be *analysed* to come up with well justified conclusions.

Sometimes results can be put into an appendix; this may well depend on the kind of project you are doing (especially when the data showing results are very long, or similar/repetitive). Results can also be included in tables, diagrams and other figures.

For quantitative work, it is important to try to use statistics (usually linked to some form of hypothesis and experimental process) whenever possible to add significance to any claims that you make.

For qualitative research, you can't just make general claims without using empirical evidence (preferably embedding the research in some form of analytic process). You must demonstrate that you are not just picking data to suit your claims (what is known as using 'juicy quotes').

Please *always* follow a recognised set of data collection and analytic methods: use the existing literature on this, rather than making your own process up – you will get marks for reading and using published literature. Cite and reference them!

Do I need a user interface?

For most projects in computer science or business computing, where the primary reason for undertaking the project is to develop an algorithm or to design the underpinning code for a computer system, you do not need to design a well-constructed, highly usable user interface. This should not be something pinned onto your project as an additional feature if it is not required. It is important to recognise that you **do not** have to design or build a fully developed usable user interface for a dissertation. This will take time and effort away from your technical development work, and it is unlikely to that a quickly built, poorly developed, and badly evaluated user interface will gain you any additional marks.

For most projects where the focus of your primary work is on developing or modelling a complex system, the purpose of a user interface is to access the results of the computational process, or to input data so that the computer can use this as material to operate on. In these cases, your user interface can be extremely simple and need not be designed with the user in mind.

However, for the vast majority of projects where the user interface is the central part of the dissertation aim, and usability is the main criteria for project success, you **must** follow a recognised human-centred design process and spend a significant period of time working on iteration and evaluation of some form with users. This will require detailed ethical approval which you are advised to seek out as early as possible.

A dissertation is not a just collection of activities – it needs to tell a story

Sometimes it’s hard as a reader or a marker to understand *what* is being discussed or *why* it is relevant – you need to tell us!

Always introduce chapters, sections and subsections with a description of what this part is doing in the context of the thesis, or of the section it sits within. Without this context, your reader can’t see where it fits in, or why it is included. If your assessor can’t see its relevance, you will not be credited with (m)any marks for this part.

Similarly, don't just drop in a table or list or a figure without first explaining what you're doing with it and why you've done this. Without content or explanation, these are relatively meaningless and won't pick you up any marks. Always label figures and tables so we can understand what they are, and make sure that you refer to these in the text (eg. “… as shown in fig.4.4 …”). Without this, your readers may not know what the figure explains or how it supports your arguments.

Itemising things in bullet point list format can be helpful, but such lists can also hide your reasoning (which is assessed). Use bullet point lists carefully – remember that this is a report that explains what you did and why, not just a set of non-linear items or actions. Without writing these things into a structure, you may not be able to *describe, explain, justify, or evaluate* either in sufficient detail to understand their contents, or help show their connections in a sequentially connected form. While they may look clear and organised, bullet point lists will often not gain you marks.

How do I get a top grade?

Work hard. Plan ahead. Use your supervisor effectively and meet them regularly.

Identify where your marks are likely to be concentrated – if you’re developing software, it’s easy to convince yourself to spend ‘a little bit more time’ perfecting the code. Be aware that your code is not usually graded – but your written report is. It is better to have a slightly unfinished software solution add a finished report than the reverse (of course, you would need to describe what is incomplete, why not, and what you would need to do to complete it In the report).

One of the critical things that you will need to do is to **justify** your choices. It is this deep understanding of why you did things the way that you did them which distinguishes the kind of higher-level thought that will get you the top grades.

As a general piece of advice, do not just *say* that you did something in the report. This is not enough to pass. For almost everything important you write about *and want to get credit for*, we would expect you to: 1. explain the need for something, 2. describe what you did, 3. justify the reasons for this choice, 4. describe the outcomes of what you did, 5. evaluate the success of doing it, and 6. show how you would improve the situation with a better solution.

Testing and/or Evaluation?

DO NOT leave your planning or implementation of testing and evaluation to the end of the project.

This is an integral part of the project. It is a Learning Outcome, and needs to pass. Significant marks are allocated to this part. Normally, testing (ie. of code) and evaluation (ie. of usability) are critical parts of any software or design lifecycle, so that *not* considering it throughout the project means that you are not following the lifecycle that you say you are using – this is a major reason for failure.

What is testing? By testing, you should be assessing your software/code to see that what you are doing is what you think you are doing. Often this is characterised in terms of it's being Correct, Robust, Reliable, Trustworthy, and Efficient. Your project may require different methods at different stages of the software development cycle to assess these.

What is evaluation? Normally, this covers aspects of usability (in interaction design, user experience), and will involve some kind of assessment (with or without users) of whether the design of some aspect of software is useful and/or usable. You are encouraged to consider using multiple forms of usability evaluation, as different forms allow you to assess different aspects of the solution/s that you have created.

You need to ensure that any Testing or Evaluation planned fits into the design methodology or software lifecycle that you have chosen.

Do I need to do Testing *and* Evaluation? No. You might, or might not! But be aware that to get a good grade, you need to do good work. If you don't have time to do good work on both forms, you will not get a good mark, and it would be better to focus on one of these.

Treat how you document Software Testing and Usability Evaluation in the same way as any other part of your project. One or two pages of writing is almost always completely insufficient, and likely to fail.

What should you include in this chapter? Treat this section a bit like a science report – you might want to include the following (and add more where appropriate!): what is the question; justify your selection of methods to answer the question/s; Methods (in detail); Testing Environment; Results; Analysis; mini-Discussion of the results; Implications for changes to your initial design. You should consider doing this for *all* of the Software Testing and Usability Evaluation methods that you deploy, not just for one testing cycle.

Chapter structure

Typically, I recommend a 7 chapter structure, following something like the following format (of course, this will probably need to be adjusted depending on the type of project that you are doing):

1. Introduction. This will usually include subsections covering your motivation, aims and objectives, scope (what you will, and perhaps more importantly, will not cover), approach/methodology, and dissertation plan (perhaps with a diagram of the relationships between the different parts of the dissertation)
2. Literature review. I would not recommend titling it as a literature review – it would be better for the chapter title to tell me what the literature review is doing in the thesis. Focus on academic papers wherever possible. Most of your citations to references will be covered in this part (although not all).
3. Methodology (how will you approach the problem, what software development life cycles will you use, how will you spend your time [perhaps use a Gantt chart]?). Consider the difference between *Methodology* and *Methods* in this part. Use the literature and don't make your own methods up unnecessarily – you'll get marks for using the literature to find suitable existing approaches (please cite them!)
4. Design. This will cover how you plan to build the system (eg. abstractions, algorithms, UML notations, flow charts, pseudocode), and is often characterised as your systems architecture. Importantly, the section needs to include a text-based explanation for every part of your design – explaining what these designs or design notations do, and why they are suitable solutions for your research problem.
5. Implementation. This may include code, prototypes, or models (amongst other things). These might show one design, or compare different versions as you have refined and tested the solution. The most important things to write about here in addition to the key instances of your implementation are to explain what you have done and why. Do not try to show *everything* that you have done, especially the less interesting elements: use the space that you have to showcase your best work. It is important not to have diagrams or figures without some form of explanation. Recommend comments in any code as well as "callout arrows" to important elements in the screenshots and figures). This is likely to be a significant chapter with significant mark allocations.
6. Testing (there should be a significant chapter, not just a couple of pages as it receives a high proportion of your marks. It **should not** be something done right at the end, and it should follow the process described in your software development life-cycle.
7. Conclusion (summarise the project, uploading your key contributions/strengths, identify weaknesses, suggest future work, with perhaps a final personal observation).

Conclusion chapter: achievements, future work and potential weaknesses

The conclusion is much more than a bland project summary that just repeats what you already wrote about – it's purpose is to highlight your main achievements, perhaps to show what you didn't manage to do (and why), and to emphasise what you think are the most important elements of your work that the assessors (markers) should focus on.

It is really helpful to map your achievements against the initial objectives for your project. Effectively, this allows you to show that you have answered the questions that you set out yourself in chapter 1.

You may also wish to include how a future approach might differ from how you did things this time – knowing what you do now. Here, you would explain what you felt were the weaknesses in your current project, and how you would intend to improve them in the new project. Counterintuitively, showing your errors is likely to actually gain you marks because it shows that you understand the problem and the possible range of solutions better now.

The conclusion to your report should include discussion and reflective reviews of different aspects of the project, not just one or two. These may be chosen from the following list, although other aspects are acceptable too:

- Topic / area of work
- Defining aims and objectives
- Literature review
- Project plan
- Methodology or methods
- Project development
- Prototyping, Requirements, Design, or Coding undertaken
- Testing and evaluation
- Data analysis
- Writing the project report

About the author

Mark Perry is a (full) Professor in the Department of Computer Science at Brunel University. He has been supervising undergraduate, masters, and PhD projects for over three decades and acting as a tutor for around 350 dissertations over this time. He has also acted as an external examiner across a number of universities to assess several hundred more dissertations from across the UK, providing an excellent basis to help students produce well planned projects and clearly written dissertations that explain the challenges and successes undertaken.