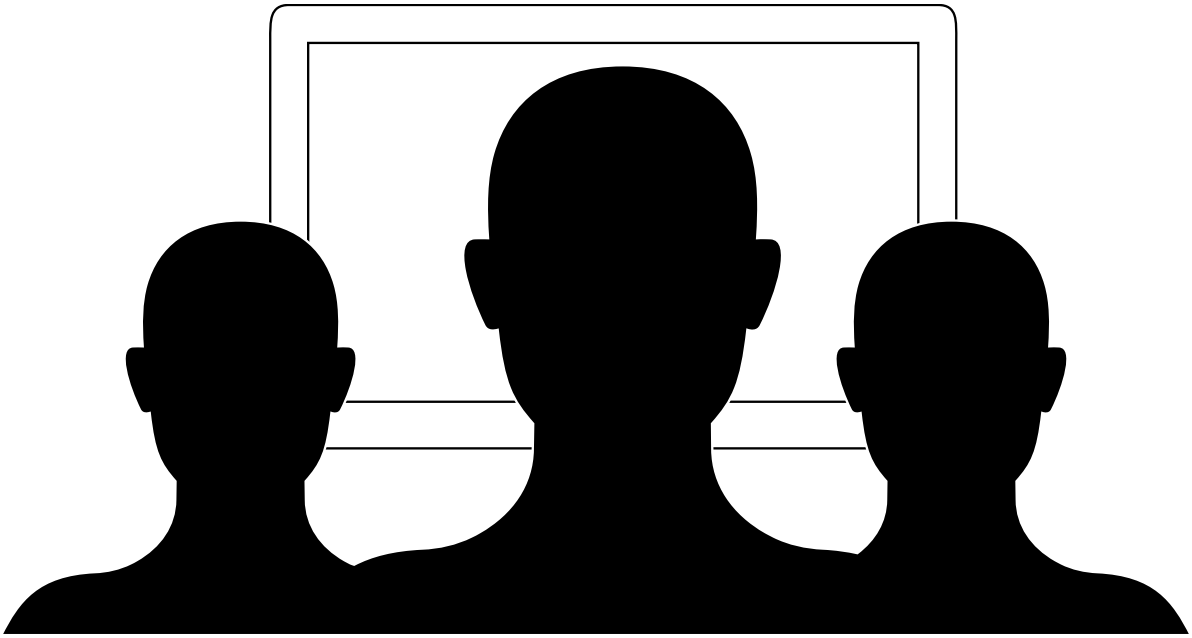
The guide to

Software Engineering & Professional Practice



Michael B. Gale
m.gale@bham.ac.uk

2024/25

CONTENTS

λ.1

The Module

For many of you, a degree in Computer Science will be the stepping stone to a career in software engineering. At the same time, most of your experience building software has been to write some Java code on your own, compile it directly with the Java compiler, and run it on your own machine. Real software engineering is very different.

In this module, we will explore the gap between the programming you will have done in your first year and the real software engineering you are likely to do in your careers. You will be introduced to the different contexts that software engineering takes place in and the different processes that organisations may employ surrounding it. Furthermore, you will gain foundational knowledge of relevant tools and technical skills.

This guide serves as a companion to the module by giving you an overview of all the major components. It will be a living document that will evolve to include more content as the module progresses.

1.1 Recommended reading

You are not required to purchase any books for this module as there are many free resources available online. As we focus on covering a breadth of topics, rather than a single topic in great depth, I will make individual recommendations for each week.

1.2 Timeline

This module is comprised of approximately 22 lectures and 3 pieces of coursework. This section contains a chronological schedule of all of these components. Note that

the schedule may be subject to changes due to *e.g.* staff illness or other unforeseen circumstances. Each lecture aims to answer a specific question, which is shown in the timeline. You can test your understanding by asking yourself that question after each lecture and checking that you can answer it.

Lecture 1: Introduction

2 October — *What is this module about?*

Module overview, different settings for software engineering

Lecture 2: Classic software engineering techniques

2 October — *What are traditional software engineering techniques?*

Waterfall, Spiral, Agile, Prototyping, UML, ...

Lecture 3: Technical overview of a software project

How is a software project organised?

9 October

Introductions to software architectures, code repositories, build tools, dependency management, continuous integration, deployments, ...

Lecture 4: How engineering teams work

How do engineering teams work?

9 October

Engineering design reviews (EDRs), Architecture Design Records (ADRs), units of work, Directly Responsible Individuals, retrospectives, ...

Lecture 5: The humans

16 October — *What are human aspects that engineers need to be aware of?*

Users, customers, sales teams, field teams, accessibility, internationalisation, ...

Lecture 6: Effective technical writing

16 October — *How can we communicate technical information effectively?*

Good style, common mistakes, examples of good and bad writing, ...

Lecture 7: Observability

23 October — *How do you know what's going on with your software?*

Logging, metrics, alerting, ...

23 October	**Lecture 8: Data analytics for Software Engineers** *How can we use data to inform planning decisions and review impact?* Where and why to leverage data, engineering with data, ...
Week 5	**Deadline: Coursework I**
30 October	**Lecture 9: Stop calling it `_Version14ReallyFinal.docx`** *How can we use version control effectively?* Version control systems, Git basics, undoing things, ...
30 October	**Lecture 10: Version control for teams** *How can teams leverage version control to collaborate?* Common workflows, branches, merging, rebasing, conflicts, ...
6 November	**Lecture 11: Build tools** *How is real software built?* Build tools, automating builds, managing dependencies, build caching, reproducible builds, ...
6 November	**Lecture 12: Continuous integration** *What is continuous integration and how does it help us?* Continuous integration systems, examples, deployments, ...
13 November	**Lecture 13: Basic testing** *What are some essential techniques for testing code?* Testing frameworks, unit tests, integration tests, ...
13 November	**Lecture 14: Not-so-basic testing** *What are some more sophisticated testing techniques?* Property-based testing, proofs, mocking, designing implementations for testing, ...
20 November	**Lecture 15: Containerisation** *What is containerisation and how is it useful?* Docker-style containerisation, examples, ...

20 November

Lecture 16: Orchestration

Given a bunch of containers and a bunch of servers, how do we run them?

Kubernetes and friends.

27 November

Lecture 17: Threat modelling

How do we incorporate security into the software engineering process?

What is threat modelling, when to threat-model, different strategies, ...

27 November

Lecture 18: Supply-chain security

How do we ensure none of our dependencies have security vulnerabilities?

Vendoring dependencies, security aduits, vulnerability databases, automatic dependency updates, ...

4 December

Lecture 19: Static analysis

How can we check source code for security vulnerabilities?

What static analysis tools are, how they work, and examples of one in action.

4 December

Lecture 20: Generative AI as a tool

Is generative AI going to replace all of us?

Examples of how generative AI can assist in the software engineering process.

Week 11

Deadlines: Coursework II & III

11 December

Lecture 21: Open-source

What considerations are there for using or contributing to open-source projects?

Licensing, contributing, benefits, ...

11 December

Lecture 22: Conclusions

What have we learnt about software engineering?

Summary of the module and other general information.

1.3 Coursework

There are three pieces of coursework which you will have to complete. You will receive feedback for the first coursework before the second and third coursework are due.

1.3.1 The Engineering Design Review (40%)

Description You get to write an *Engineering Design Review* (EDR). An EDR is a document, written by a software engineer, which discusses a proposed design for new functionality in a software system. The document is primarily aimed at other engineers, but also engineering or product managers. An EDR is an opportunity to propose a solution to a problem, discuss your chosen design, explain why you chose it over alternatives, and propose how work on implementing it should be structured.

Aims This coursework is designed to develop your ability to research a technical problem, propose a solution to it, think about the context in a wider project and organisation, and exercise your technical writing skills.

1.3.2 The Prototype (40%)

Description As a small group of 3-4, you will implement a *prototype* based on *one* of your EDRs from the first coursework. A prototype is an as-simple-as-possible implementation of your design that can demonstrate that your design is feasible. In other words, while the goal of your EDR is to describe a solution that makes sense to you and other engineers in theory, the goal of the prototype is to demonstrate that the design holds up once implemented, and no more.

Aims This tests your ability to collaborate as a small team, choose appropriate implementation tools for your prototype, implement a design, and simulate interfaces and dependencies that your prototype relies on, but which do not actually exist.

1.3.3 The Reflections (20%)

Description On your own, you will write a short document reflecting on your participation in the module. By "reflecting", we mean "thinking constructively about". Reflecting on your development as an individual is a useful skill in any career path. Additionally, organisations may, in one way or another, also ask you to reflect on your development in the context of your employment. This can either

be used to help you work towards your career goals, or factor into formal rewards conversations (conversations about salary increases, bonuses, etc.). In addition to reflecting about your own development and work, you may also reflect about colleagues, your team, the organisation you work for, etc.

Aims Software engineering requires many different skills and, naturally, we are better at some than others. By reflecting on your work, you can figure out which skills you can leverage more strongly to benefit from, and which skills you need to work on going forward. This coursework tests your ability to think constructively about your work, development, and factors that contribute to it.

$\lambda.2$

COURSEWORK

2.1 Your organisation

To frame the tasks you will be carrying out for this module's assessments, you will choose one organisation from the following options. You will *pretend* to work for this organisation and the assessments represent tasks that you are carrying out for that organisation. The choice is entirely yours, but you will stick with the same organisation *for all three assessments*. In particular, note that the second coursework is carried out as a group. While you will be able to choose your own group, you can only team up with others *in the same organisation*. You may wish to take this into consideration when choosing an organisation, as well as your own personal interests.

The descriptions of the companies and their products are merely supposed to give you an idea of what sort of companies they are, what sort of products they make, and how they broadly work. For your coursework, you are free to make further assumptions about other products not described here, features not described here that you rely on but exceed the scope of your work, or implementation aspects not mentioned here. You should however document any such assumptions you make.

2.1.1 Student Smart Homes (SSH)

Smart home technology is increasingly popular in homes throughout the UK. Student Smart Homes (SSH) is a start-up which specialises in producing hardware and software specifically aimed at student houses. The products are marketed either directly at students or providers of student accommodation. For students to be able to use the products, they must be suitable for use in rented accommodation. In other words, they must not require any permanent modifications to the accommodation.

In either case, the products must be suitable for multiple users, who may not have the same level of trust for each other as members of the same family.

Some of their existing products are outlined in the following sections.

The SSH Hub - First Class

The *SSH Hub - First Class* is Student Smart Homes' most recent smart home hub product. The device is USB-powered and connects other SSH products in the student home (via Bluetooth or WiFi) to the SSH Cloud (via WiFi). It runs software which detect SSH products in range, communicates with them, and propagates information between the SSH Cloud and those devices.

The SSH Cloud

The *SSH Cloud* encompasses all of Student Smart Homes (SSH)' online services. It includes both a web-based interface that users can access to manage their accounts and remote-control their devices, as well as services that exchange information with the SSH Hubs.

The SSH App

The *SSH App* is an iOS and Android app which allows users to access and control their SSH devices from their smartphones. The SSH App communicates with the SSH Cloud, which in turn communicates with the user's SSH Hub - First Class, which in turn communicates with the SSH devices.

The SSH Camera

The *SSH Camera* is a smart camera which can be positioned almost anywhere thanks to versatile mounting options, including a roll of sticky tape that comes included for free in the box. The camera can be used for multiple different scenarios thanks to its AI features:

- By pointing the SSH Camera at the front door, it can detect who enters and leaves the student house. The SSH Console Table (sold separately) will then show a beautiful overview of who is or has been in the house and when.
- By pointing the SSH Camera at the sink, it can detect who just dumps their dirty dishes in it and never washes up. The SSH Cloud (subscription required) will then automatically send notifications to the SSH App to remind those house mates to clean up.

- By pointing the SSH Camera at the fridge, it can detect who takes what from the fridge. The SSH Cloud (subscription required) will then automatically bill users for items they have taken, but were't theirs.

The SSH Console Table

The *SSH Console Table* is a console table made out of beautiful walnut veneer with a touch screen as the table's top surface. The screen allows multiple users to connect to their SSH accounts via an SSH Hub - First Class (sold separately) and provides a central point for students to access the SSH ecosystem in their home. Aside from controlling SSH devices, such as the SSH Camera, the SSH Console Table allows students:

- To view a unified timetable to help plan joint events
- To see who is currently in the house and when people have come and gone
- Arrange rotas, such as for who cleans the dishes or takes out the bins
- View rankings for who has washed the most dishes or washed them the fastest (SSH Camera required, sold separately)
- Make purchases from third-party providers and automatically split the bill

2.1.2 Higher Education Software Solutions Plc

Universities depend on a large number of industry-specific software systems, such as for managing student records, detecting plagiarism, managing learning content, generating timetables, and so on. That is on top of extensive needs for more general-purpose software, such as office programs, single sign-on solutions, content management systems for websites, and so on.

Higher Education Software Solutions Plc is the market leader in providing comprehensive solutions for these software needs to universities.

Some of their existing products are outlined in the following sections.

CSRS

CSRS (pronounced *courses*) is Higher Education Software Solutions' core product offering. It combines student records, timetabling, and learning management, as well as many other features in one package. Universities can either host CSRS on-premise in their own data centres, or Higher Education Software Solutions offer CSRS as a

managed service that they host for universities. Since all universities are different, CSRS provides many different configuration options that universities, departments, module organisers, and individual users can change. For more technically-savy users, CSRS provides an API which can be used to automate many parts of CSRS.

RewriteItIn

RewriteItIn is Higher Education Software Solutions' state-of-the-art anti-plagiarism solution. It takes a coursework specification, produces variations of the specification using several, popular LLMs. For each LLMs variants of the coursework specification, as well as the original, each LLM is then asked to produce a solution to the specification. All of the resulting LLM-generated reports are then combined with the pool of student submissions in a traditional plagiarism checker. RewriteItIn is available to universities as a separate SaaS solution with a monthly or annual subscription. For universities that use CSRS, RewriteItIn can easily be integrated with assignments.

2.2 Coursework I: The Engineering Design Review

When joining an organisation, it is unlikely that you will start an entirely new piece of software from scratch. Instead, you will be contributing to an existing software system. Depending on the organisation you work for, the process by which work is planned and allocated to you may vary considerably. However, it is likely that the process is primarily driven by business needs in one way or another. Some examples of common processes include:

- Your organisation's leadership decide what to work on, based on their thoughts on what will allow the organisation to make (more) money. Your management chain may then divide the necessary work up and allocate some to you.
- The leadership team has identified broad priorities for the organisation which product managers work to design the product around. Product managers will then coordinate with engineering managers to plan how the product they designed can be engineered, and some of the resulting work is then allocated to you.
- Some organisations may give total freedom to engineers to work on features that they consider to be important.

Regardless of the process used, software engineers like you will be required to come up with solutions for the features that are a business priority. For example, you may be told that some feature is deemed necessary and that you should work on it, but it may be up to you to figure out how exactly it should work and integrate with other parts of the software.

Where there is a design space – in other words, there are multiple different options for how a given feature could be implemented – it makes sense to describe the approach that you think is best in a document for an Engineering Design Review (EDR) and compare it with other alternatives that you considered. The EDR is an opportunity for other engineers on your team or other teams who would be affected by your implementation to give you feedback on the proposed approach.

Sometimes, EDRs can also be a means by which engineers can propose some feature that they think is worth working on. In that case, it is likely that engineering and product managers will also review the document to decide whether the feature is indeed worth allocating time for and fits in with the overall product direction.

2.2.1 Task

Choose one of the organisations introduced in Section 2.1. You will stick with this organisation for all assessments in this module.

Your task for the first coursework is to write an Engineering Design Review (EDR) for one project in that organisation. See Section 2.2.3 for guidance on projects.

2.2.2 Structure & Assessment

Your EDR should be a document of roughly 5 pages that you submit as a PDF. The structure of your EDR should be as outlined below.

For each section in the outlined structure below, a number in brackets indicates roughly how much of your EDR should be dedicated to that section, as a percentage. Your EDR will be marked on the *technical content* as well as the *quality of the technical writing*. In marking each part of your EDR, equal weight is given to the quality of the technical content and how well it is communicated to the intended audiences. For your feedback, you will get a mark for each section, as well as overall suggestions for how the EDR could be improved.

Introduction (20%) Establish what problem your EDR is addressing, how the EDR relates to existing systems in the organisation, and why solving the problem should matter to the organisation.

To write this part, you should read through your chosen organisation's description in Section 2.1 and think about the goals of your chosen organisation as well as how the products described may be broken down further into smaller systems. You may make any reasonable, technical assumptions about how the products work internally. Indeed, part of your work here is to think about how the described products may work.

Goals and non-goals (2%) Using quantifiable metrics, list what the design proposed by your EDR aims to accomplish and what it explicitly does not aim to accomplish. In particular, think about what "success" means for the project and how it can be measured.

Design overview (30%) Describe your chosen design in sufficient technical detail that other engineers can decide whether the design is sensible and the best option for the problem at hand. You do not need to describe every little aspect of the implementation, but just enough to communicate how you suggest solving the problem at hand in a way that other software engineers can understand and review.

Alternatives (20%) Give a brief overview of other design options that you considered. Discuss their advantages and disadvantages compared to the design you chose. Most problems have many possible solutions. You do not need to describe

every possible solution, just some other ones that could reasonably chosen but you rule out for the reasons given. If you truly think that there is no alternative to the design you proposed, you should explain why that is the case.

Milestones (10%) Propose a plan for how work on the design will proceed if the EDR is accepted (in other words, if other engineers think it is a reasonable design and managers allocate time to work on it). Typically, most projects are broken down into a few, numbered milestones that allow for evaluation at each step once they have been reached to determine whether work should proceed on subsequent milestones. This should cover different phases of implementation, testing, and deployment.

Dependencies (4%) List which other teams in your organisation are affected by the changes proposed in your design. Since your chosen organisation does not really exist, you should make reasonable assumptions about how your chosen organisation may be organised into different engineering teams that are responsible for different aspects of the products.

Cost (2%) Ignoring staffing costs for the project, would the proposed design affect the upkeep of the system for the business or make the product more expensive? For example, are you anticipating changed server hardware requirements or cloud resources?

Privacy and security concerns (5%) Think about whether your design involves collecting any new personal data, or processes existing personal data in a new way that users may not have consented to. If so, discuss this here, how the data will be handled, and what permissions must be obtained from users. Additionally, discuss any security concerns that may apply to the design.

Risks (5%) In a table, describe the main risks of your design, the impact that they would have, and any mitigations that already exist or you propose.

Supporting material (2%) Link to any relevant material, such as documentation for tools, frameworks, etc. that you referred to while writing the EDR or that reviewers may wish to refer to in addition to what you wrote.

2.2.3 Suggested projects

For each of the available organisations, there are suggested projects that you could tackle with your EDR. You may also think of your own project to write an EDR for,

but should ensure that it is comparable in scope to the suggestions. In particular, note that for the second coursework you will, as a group, *prototype* one of your group members' EDRs. If in doubt about whether your own idea is suitable, feel free to ask.

Student Smart Homes (SSH)

1. In some student houses, students use grocery delivery services to have groceries delivered to their house. To save on delivery fees, all participating students in the house may place orders together. This project is to extend SSH Console Table and SSH App with the ability to add items to a shared order for the next delivery. Student Smart Homes has partnered with several supermarkets that provide information about available products along with their current prices that can be searched and added to the order. Prices may change over time as promotions come and go or the base price of a product changes. The students should be able to view the items in the order at any time, including who added what, what the total cost is, and the total cost of all items for each individual member of the house is.

2. The SSH Camera can provide detailed information about the contents of the fridge, including quantities. This project is to extend SSH Cloud with a recipe suggestion function that, using the data about available ingredients from an SSH Camera, and a database of recipes, suggests possible recipes using those ingredients. It may also suggest recipes for which most ingredients are available and only some need to be purchased.

3. The SSH Camera can also provide information about who is at home at any given point. Using occupancy trends over time that are available from data collected by the SSH Camera, this project is about adding a feature to the SSH ecosystem that automatically suggests times during the week which may be convenient for a given student to clean their dishes or take out the bins because they are typically at home. Conversely, the system may suggest making use of the TV or a gaming console because all or most other members of the household are typically away.

Higher Education Software Solutions Plc

1. To produce good feedback for students, it is helpful to have *custom feedback forms* for assessments that markers can fill in. Currently, markers are only given two inputs per student submission in CSRS: a numeric input for the overall mark, and a free-form text field for feedback. This project is to design a feature that allows module organisers to specify custom feedback forms,

which may comprise different sections corresponding to different aspects of the assignment:

- Instructions for each section.
- For each component of the assignment, an input for a component mark.
- Multiple free-form text fields for different aspects of the feedback.
- The ability to label submissions with tags (e.g. "1st SSH project") to allow markers to quickly filter all of their assigned submissions by a given tag.

Module organisers should have a convenient way to specify such custom feedback forms for each assignment, markers should then see the form for each submission that they can fill in, the data should be savable for each submission, and students should later see the feedback in the form.

2. For assessments in Computer Science (and some other sciences), there are steps which can be automated once a student has submitted some work. For example, for a programming assignment, compiling the code, running tests, and comparing the student's code to the skeleton code to see what changes have been made. Currently, markers for such assignments need to download the submission from CSRS and perform all of these steps manually on their own machines. This project is to implement a feature which lets module organisers configure a sequence of steps that should be performed automatically once a student submits something. The results should then be made available to markers alongside the actual submission.

3. Student numbers have consistently been increasing for many years. This means that marking coursework for large modules typically requires several people to mark all submissions. For a module organiser to ensure that each marker is consistent is difficult, even with detailed mark schemes. This project is about trying to *detect differences in marks across markers*, and *suggest mitigations*. This may take place once all submissions have been marked and all students' marks are available along with the information about which marker marked which submission. Alternatively, it could take place sooner once there is enough data available. The detections made by this system may indicate that one or more markers are harsher than others, or more generous, allowing the module organiser to intervene. If useful, you may assume that you also have access to students' marks from previous assignments across different modules and years, as well as all other records about each student that are held by a university.

2.3 Coursework II: The Prototype

Once an Engineering Design Review document has been reviewed by all interested parties, improved by the author based on feedback received, and management has decided to allocate engineering time for the proposed work, the engineer(s) assigned to the work can get started.

2.3.1 Task

Form a group with 3-4 members, ensuring that everyone in the group wrote an EDR in the context of the same organisation for the first coursework. Then choose one of your team's EDRs from the first coursework and implement a prototype for it. The emphasis here is solely on demonstrating that the design proposed in the EDR is viable and you should only build what is necessary for that purpose.

You may use any programming language, libraries, tools, etc. that you want for this. However, you should make use of software development techniques that have been covered in the module. This should include version control, appropriate workflows for a team, testing, build tools, dependencies on existing libraries where useful, continuous integration, etc. You can read the assessment criteria below for more information on what is expected.

The design proposed in your EDR forms part of a larger system, which does not really exist, and you are not expected to build all the systems your design depends on. Instead, you should "simulate" other parts of the overall system to the extent that your design depends on in whatever way is most convenient. This may be accomplished by, for example, hard-coding values received from other systems or parts of the same system, building extremely simple versions of other systems that just return test data, randomly generating data, replacing networking with direct user input, terminal output instead of network responses, and so on. If you are not sure how to simulate a part of the overall system that you need, feel free to ask for suggestions!

2.3.2 Working as a group

Every group of engineers works differently and different engineers bring different qualities. In a real workplace, some engineers will be more experienced and work faster than others. Regardless, everyone has their unique strengths and weaknesses. In your group, make the most of each others' strengths and work around your weaknesses. If you are in a group with someone who you feel is not contributing as much as others, try to identify what strengths they could tap into. Alternatively, it is

common for more experienced engineers to mentor less experiences ones. Figuring out how to make your group work effectively and help each other is as much part of the coursework as the technical aspects.

2.3.3 Submission

Your submission should include an archive of your code repository, including its full history. It should also include a short report that briefly summarises (no more than 200 words) what was accomplished and discusses in detail how it was accomplished (no more than 800 words). This report may include any supporting material that demonstrates the software engineering techniques you employed that is not already included as part of the archive of your code repository. For example, screenshots of key issues demonstrating how you planned and provided updates on work.

Furthermore, each member of the group should *individually* write around half a page of *reflections* on every other group member. The reflections should be *constructive* in nature and you should be happy to share them with the individual they are about. Assume that, rather than being students at a university, you are colleagues at a workplace and that the end of the module or group project is not the end of your working relationship. What would you suggest to your "colleague" so that working with them will continue to be productive in the future or become productive? You can read more about what reflections are in the specification for the third coursework.

2.4 Coursework III: The Reflections

Regardless of what career you choose to pursue after completing your degree, a good skill to have is to routinely reflect on the following:

- The work you did since your last reflections.
- What your work accomplished.
- What new skills you acquired from doing the work.
- Where things could have gone better and what you would do differently if you did the same work again.
- What you will focus on accomplishing until the next reflections.

Doing this may help you understand what weaknesses you need to work on, what strengths you can utilise, and shape the direction your career progresses in. Some organisations may also employ a reflections process between an employee and their manager, either as part of a rewards process or to allow managers to support employees with their careers.

2.4.1 Task

Write a document of *no more than two pages* reflecting on your participation in this module. You should answer all of the following questions:

1. Think of your three biggest accomplishments during this module: describe what they are and how will they affect you going forward.
2. Was there an occasion where something you did related to the module affected someone else positively or negatively? What did you learn from it?
3. Think of one occasion where you did something that you would do differently now under the same circumstances.
4. Identify one topic that we have covered or one skill you needed that you think is a particular strength of yours. How are you planning to make use of this strength going forward?
5. Identify one topic that we have covered or one skill you needed that you think is a particular weakness of yours. How are you planning to work on it going forward?

Remember that we are not looking for you to talk about how great everything has been. Instead, we are looking for honest and thoughtful reflections that provide a constructive assessment of yourself. You can gain full marks for this coursework even if things did not go so well, as long as you write about what happened, what you learnt from it, and what you will do differently in the future. On the other hand, you may not get many marks if you have nothing constructive to write about yourself.

2.4.2 Assessment

Your answer to each of the five questions carries a weight of 20% for this coursework. Each answer will be marked according to the following scale:

20% An answer which shows an excellent level of critical reflection and is communicated clearly.

10% An answer which shows a reasonable level of critical reflection, or a reasonable answer that is not communicated clearly.

0% No answer, an answer which shows little evidence of critical reflection, or an incomprehensible answer.