Documenting Programming Projects

OCR A Level Computer Science H446

D Hillyard
C Sargent

Published by
CRAIGNDAVE LTD
12 Tweenbrook Avenue
Gloucester
Gloucestershire
GL1 5JY
United Kingdom

admin@craigndave.co.uk
www.craigndave.org

Craig 'n' Dave

Text © D. Hillyard and C. Sargent 2020
Original illustrations © D. Hillyard and C. Sargent 2020

Edited by A. Fenn
Graphics and typesetting by CraignDave Ltd
Second edition 2020

ISBN:

About the authors

David Hillyard

David is a postgraduate-qualified teacher with a Bachelor of Science (Honours) in Computing with Business Information Technology. David has over twenty years' teaching experience in ICT and computing in four comprehensive schools and one grammar school in Cheltenham, Gloucestershire. He has been an assistant headteacher, head of department, chair of governors and founding member of a multi-academy trust of primary schools.

Formerly subject leader for the Gloucestershire Initial Teacher Education Partnership (GITEP) at the University of Gloucestershire, David has successfully led a team of PGCE/GTP mentors across the county. His industry experience includes programming for the Ministry of Defence. A self-taught programmer, he wrote his first computer program at ten years old.

Craig Sargent

Craig is a postgraduate-qualified teacher with a Bachelor of Science (Honours) in Computer Science with Geography. Craig has over fourteen years' teaching experience in ICT and computing in a large comprehensive school in Stroud, Gloucestershire, at a grammar school and as a private tutor. A head of department, examiner and moderator for awarding bodies in England, Craig has authored many teaching resources for major publishers.

He also previously held roles as a Computing at School (CAS) Master Teacher and regional coordinator for the Computing Network of Excellence. His industry experience includes freelance contracting for a large high street bank and programming for the Ministry of Defence. He also wrote his first computer program in primary school.

Foreword

We have compiled all the latest advice, tips and guidance documents from OCR, cross-referenced the clarification documents including moderator suggestions, examiner reports from past series and distilled all this information into one comprehensive guide for documenting your unit 3 programming project.

We also bring our many years of personal experience from our own classrooms to this document, we know what makes a successful project, we recognise the pitfalls and mistakes students make and as a result we know what advice to give our own students to make sure they are equipped to get the top marks.

Every year the projects we submit receive high praise from moderators. Consistently since the introduction of the 2015 specification, none of our marking has been adjusted and we have been commended for the approach our students have taken by using this guide.

In addition, make sure to watch our playlist titled, "A level: OCR Unit 3 Project Advice" on our CraignDave YouTube channel that contains a series of videos providing additional advice and guidance.

This document has been written specifically for those students who are producing programming projects that are not video games. If you are producing a computer game, please use our "Documenting Defold programming projects OCR H446" book instead. Examples in that book are based on Defold, but the guide is suitable for any language you might use.

Craig & Dave

Contents

INITIAL IDEAS

There are specific examination board requirements you should be aware of before you start your project. It is also important to scope your project to ensure it is manageable in the time you have available.

Choosing a project

One of the considerations when embarking on a programming project is ensuring you choose a project with sufficient scope to be of A Level standard; this is not easy to define, but it must be more advanced than typical GCSE work. As a minimum, it must include:

- a graphical user interface; and
- a substantial coded element.

Trivial problems, regardless of how well you solve them and write them up will not be able to provide you with the right evidence to meet the criteria in the top mark band across the whole project.

Consider the complexity of the algorithms you study on the A Level Computer Science course as a benchmark. These include linear search, binary search, hashing functions, bubble sort, insertion sort, merge sort, quick sort, PageRank, Dijkstra and A* algorithm. SQL as a data definition and manipulation language can also provide a good coded element. Also consider the data structures, arrays, lists, graphs, has tables and trees. A project that does not make use of at least one of these is unlikely to attract higher marks.

Small applications, teaching tools, simulations, a back-end database with a front-end website interfaced with a high-level language are good ideas for a project. You should avoid simple multiple-choice quizzes, simple data retrieval systems and projects written in Microsoft Access or Excel with VBA.

Approved programming languages

You also need to consider the programming language you intend to use. There are approved languages, and beyond these your teacher will need to seek permission from the examination board to use them. Approved languages include: Python, C family (C#, C++, Objective-C, Java, Visual Basic, PHP, Dephi, Robot-x, Monkey-x, Swift, NodeJS, Haskell, Unreal, Unity, Lua and JavaScript.

Remember you will only have a limited amount of time and therefore it is inadvisable to attempt to learn a new language while completing the project. It is better to use one you have learned or been taught prior to undertaking the work.

A* TOP TIP

You should take on a challenge rather than attempting a project that is too simplistic. It is acceptable not to complete every aspect of the work providing it is written up well.

Getting the scope right

Students must choose their own project individually. You cannot ask your teacher for ideas (although they can help you select from a list you have made) and although theoretically it could be a large group project with each student contributing a significant part, this is unlikely to be successful and is not recommended.

Your choice of program for your project is one of the most important choices you will have to make. The size of the problem you undertake is known as its 'scope'. Some features will fall within the scope and others outside. Essentially you can solve almost any problem you wish; however, it is important that you get the scope right. Too little scope and the project becomes easy and trivial. Too much scope and it becomes unmanageable and unachievable. Your teacher can help you scope the project once you present them with an idea. It is usually possible to take simple ideas and expand them through a series of iterations to arrive at an appropriate scope.

Many of the best projects manipulate data. Stock control systems, booking systems, expert systems, revision tools and simulations all make use of data sets. Programs that create, read, edit, write, and delete data, especially from files usually have a good scope.

Robotics projects and programs that interface with hardware such as sensors are likely to require the use of APIs or hardware-specific functions and can make excellent Computer Science projects. Using Raspberry Pi or Arduino computers can provide for interesting projects providing you can write plenty of original code and not use drag and drop script builders. Raspberry Pi uses Python (that's what the Pi means) and Arduino uses C, so both are approved.

Networking projects can also be extremely fun, making use of interaction across a network.

Examiner's advice

"The best projects tend to be more ambitious. A project that a candidate can complete without encountering any challenges will give them little to write about. Conversely, a candidate can still receive high marks for an ambitious project that does not achieve all its aims.

Questions to think about

- Is it easy for me to research the problem?
- Does my project include substantial algorithms?
- Will my project require me to use arrays, lists or other data structures beyond variables?
- Does my project use external data sets that must be researched or stored in either a database management system (DBMS), serial, CSV, XML, JSON files?
- Do I have enough experience using the language I intend to use?
- Will I be able to complete the work independently without asking for help?

Before you embark on your project it is important that you have had plenty of practice solving problems and writing algorithms. Although the coursework is only worth 20% of your final grade it is a significant undertaking that will require a lot of your time.

A* TOP TIP

Writing code is fun but it is only worth 15 out of the 70 marks available. Being good at programming helps a lot but it does not guarantee success. A good write-up does!

Stakeholders

In the past it was a requirement that Computer Science projects had a real user with a real-world problem that you were attempting to solve. Those were the days when there were few off-the-shelf and online apps, so bespoke solutions were the norm. Today, A level projects do not require you to have a single, identified user who has a need for your program. Instead, you will need to identify stakeholders. This is a target audience or target market. A group of people who would have an interest in using your product.

That said, it is extremely useful to have at least one person who can represent the stakeholders with whom you can have regular contact. For example, if you were writing a Chemistry revision program for 15-year olds then it would be useful to talk to both a Chemistry teacher and a student 15 years of age.

Your stakeholders will be able to provide you with additional success criteria at the analysis stage, insights at the design stage, provide feedback during the software development and an evaluation at the end. Not having suitable stakeholders will severely limit your discussion resulting in a weak project.

Programming paradigms

You may choose any programming paradigm including procedural, object-oriented, or functional that your chosen language supports. There is no requirement to use object-oriented techniques. It is a misconception that OOP provides substantial algorithms, it often makes coding simpler! However, it does usually indicate that a project has sufficient scope.

There may be language specific techniques you can make use of including external libraries to increase functionality. For example, the program must have a graphical user interface so if you are planning to use Python you may want to use Tkinter to provide this functionality.

There is no need to reinvent the wheel unnecessarily. If the language supports built in algorithms and data structures you should make use of them. Why write a sorting algorithm in Java when it already provides a Tim sort which is extremely efficient? You gain credit for using appropriate features.

Working on your project

It is sensible to structure your work into the four sections that match the mark scheme outlined in this guide, although evidence to support assessment can be found anywhere within your submission. For example, if you include something in the design that should really be included in the analysis you will not be penalised. The sub-sections in this guide provide you with a handy explanation of how marks are awarded but you should use your own sub-headings.

You are permitted to work on your project outside of lesson time, including for homework but your teacher will need to authenticate your work as your own and therefore will want to see a significant proportion being completed in class too.

Your project report should be a word-processed document with a suitable cover page including your name, candidate number, centre name, and centre number. Ensure each page is numbered so that your teacher can use this for cross-referencing when marking. You must also include a bibliography or page of references. There is no requirement to adhere to a specific referencing standard.

An initial proposal

Before embarking on your project, it is a good idea to submit an initial proposal to your teacher; this will enable them to assess the scope and complexity of your project before you do too much work.

It is important to choose a project you will be interested in developing; this is a substantial piece of work – you need to enjoy it!

Your project proposal should include:

- A title.
- A brief overview of your idea.
- An outline of what you want to achieve as an outcome.

You do not need to submit your proposal in your final project. However, your teacher may decide to submit it to the examination board before you seek their guidance on scope, complexity, and viability.

The role of your teacher

The Joint Council for Qualifications sets rigorous expectations on the completion of internally assessed work and clear guidelines for the role of the teachers assisting you.

Teachers **are** permitted to:

- Support you in choosing an appropriate project without giving you a specific idea. For example, your teacher could advise and help you choose from a list of your own ideas.
- Provide you with the mark scheme.
- Review your work and provide oral and written feedback at a general level.
- Allow you to redraft your work in response to this feedback.

Teachers are **not** permitted to:

- Provide you with writing frames or paragraph/section headings.
- Provide detailed, specific advice on how to improve drafts to meet the assessment criteria.*
- Give detailed feedback on errors and omissions.*
- Intervene to improve the presentation or content of your work.*
- Provisionally mark your work and then allow you to revise it.

**Unless they record the assistance and adjust marks accordingly.*

ANALYSIS

Outlining the genre of your project, identifying the player needs and exploring ideas to scope the success criteria.

In this section, you are discussing **what** problem you are solving.

10 marks

What to include

- An introduction to your project.
- Your stakeholders.
- Detailed research.
- Features to be included in your solution.
- Limitations of your development.
- Computational methods.
- Hardware and software requirements for development and deployment.
- Specific and measurable success criteria.

Project definition

You should start your project by providing a brief introduction to it. Set the scene so the examiner knows what your project is about. This is not detailed research but simply a brief overview limited to a few short paragraphs. It is not an essential section, but it will help give you the impetus to write more detailed research later. Getting started with the report is often the hardest part. Once you get going, it seems less daunting. It is important you do not use a template or simply copy the headings from another student. How you structure your writing is completely up to you. However, you should use a logical structure to make it easy for your teacher to mark and an examiner to moderate.

Things to consider:

- The background to the problem.
- What is the specific problem you are trying to solve?
- Why is the solution needed?
- Outline your intentions.
- Your next steps, outlining how you will conduct further research.

A* TOP TIP

Do not forget to include a references or bibliography page in your report and keep a note of all your sources. You can only claim credit for original material.

An example of a project definition

In a multitasking operating system, processes that are ready for execution are held in a ready queue awaiting availability of the central processing unit. When the CPU becomes available a process from the ready queue is executed. A number of factors determine which process is chosen, how much processing time it is given and what happens to it when it requires data in order to continue or an interrupt occurs.

Background

Although with modern processors and operating systems the process of scheduling is complex, in the past there were just five simple algorithms and today A level students still learn about how these operations work.

The problem

To assist students in understanding how these scheduling algorithms work I intend to create a simulation of the five algorithms known as first come first serve, shortest job first, round robin, shortest remaining time and multi-level feedback queues.

Intentions

It is much easier for teachers to explain and for students to understand if they can see a pictorial abstraction of process scheduling and what is happening at each stage.

Why the solution is needed

My program will enable the user to create new processes to see their impact on the simulation and it will generate data that can be used to plot the efficiency of the algorithm and allow comparisons to be made.

The next step is to research the five algorithms in depth to fully understand how they work and discuss with a teacher of Computer Science what is important for the simulation to achieve and what the scope of the development will be.

Next steps

Examiner's advice

"Well structured projects in a single document, with contents pages, page numbering and subheadings are much easier to navigate and are particularly helpful in ensuring nothing gets overlooked."

Identifying suitable stakeholders

Make sure you clearly identify all the stakeholders or users for your system. Each different stakeholder is likely to have a different set of needs. In the previous example we identified two broad stakeholders, the teacher, and the students.

While it is not necessary to name individual users, it will be important that you can have regular contact with someone representing each group of stakeholders. It is therefore advisable that you do identify a named individual who can work with you throughout the project.

The weakest projects are often those that do not engage sufficiently with a stakeholder. It is a mistake to be both the developer and the end-user of your system. While it is possible, it is much harder in this situation to look at the problem objectively and not to make assumptions when you conduct your research. Being "inside the problem" is not a good idea but it can also be easily overcome. For example, if you are the student who requires the solution, surely other members of your class do too and therefore you can use another member of your class to represent the stakeholder?

Describing your stakeholders and some of their requirements is essential for all projects. To gain higher marks you will need to explain how they will make use of your proposed solution. For the top marks you also need to explain why it is appropriate to their needs. Much of this detail will need to come after your research when you decide on the features of your proposed solution. It would be easy to forget this if your stakeholder is not an integral part of each stage of your project.

A* TOP TIP

Do not simply state who your stakeholders are. Explain how they make use of your proposed solution and explain why it is suitable for their needs.

A* TOP TIP

Suitable projects allow a student to make meaningful decisions in the development process using stakeholders to direct future iterations.

Researching the problem in-depth

As the examiner reads your project it should become increasingly more detailed. In your introduction you simply outlined the nature of your project and outlined the stakeholders. In this section you must forensically analyse the problem to derive a set of proposals. It is likely that this section of your analysis will be the most extensive. Many projects fail to achieve their potential because they lack detail in the research.

Things to consider:

- Underpinning knowledge and calculations.
- Existing systems.
- Stakeholder and user needs.
- Your own ideas and approaches.

Underpinning knowledge and calculations

If your project has a knowledge base that must be understood to design a solution, you should explain this in as much detail as you can. For example, in the example project definition we outlined that our project would simulate the five scheduling algorithms. Therefore, in this section it would be essential to explain what these five algorithms are and how they work in detail. Diagrams and illustrations should be used to make this easier to explain. Remember that your teacher and examiner will not necessarily have an underlying knowledge of the context of your project.

Another example may be a planetary orbit simulation project. In this example you would want to explain the Newtonian physics and calculations that would be necessary to design your solution later.

Existing systems

If there are existing systems already available, it would be useful to examine in detail how these work. What are the features, what are the issues? Why doesn't this system provide a solution to your problem? You may be able to think creatively about what constitutes an existing system. In our project example there may be videos on YouTube that explain the concept of scheduling algorithms or exercises that you completed in class that our solution aims to improve upon.

If there is documentation such as data sets, import files or hardcopy outputs you should collect these and document or scan them into your work.

Stakeholder and user needs

Having a detailed discussion with a stakeholder to ascertain their needs is a very good idea. Think in advance about the questions you will ask about all aspects of the system. There is no need to provide transcripts of these meetings, just an outline of the discussion and its outcomes. The discussion should focus on what the system needs to do rather than how it is going to do it. This is about gathering the requirements, not about the algorithms.

Typical questions you might ask include:

- What are the main problems we are trying to solve?
- What are the stages of the process?
- What happens at each stage?
- What input and output screens will be required?
- Are there any specific requirements for the graphical interface?
- How will the user interact with the product?
- What data needs to be input into the system and how will this happen?
- What data needs to be output from the system and in what format?

This is not an exhaustive list; it provides you with a framework to structure your discussion. You may also find that some of these questions are not appropriate to your project. What is important is that by the end of your analysis you should have included sufficient detail so that if you were to get an experienced programmer to read your work, there is nothing more they would need to ask before designing the solution. This is incredibly hard to achieve and there is an expectation that this will not be perfect. During the design and coding stages it is certain you will need to ask further questions as things you did not initially consider come to light. The really fine details may not be entirely known and will be picked up in the development process.

Examiner's advice

"Interviews are not essential. Typing out interview transcripts is an unnecessary use of candidates' time. A summary of the interview's key points is sufficient."

Your ideas and approaches

You will also have several ideas about what you would like to achieve. Your stakeholders are unlikely to think of everything and they may not know what is possible either. This is where you can bring your creativity and knowledge of computer science to the process by making suggestions. As with all aspects of the project, there is no set format your should be following in your report, providing you meet the requirements of the mark scheme. Therefore you could include your ideas within a discussion of existing systems or stakeholder needs.

Remember that as a developer you are making your solution for someone else. It is important that your user needs are met as well as your own!

As you consider the ideas, to gain marks in the top mark band you will want to outline some suitable approaches you could take to achieve the aims without providing too much of the design at this stage.

Identifying essential features of the proposed solution

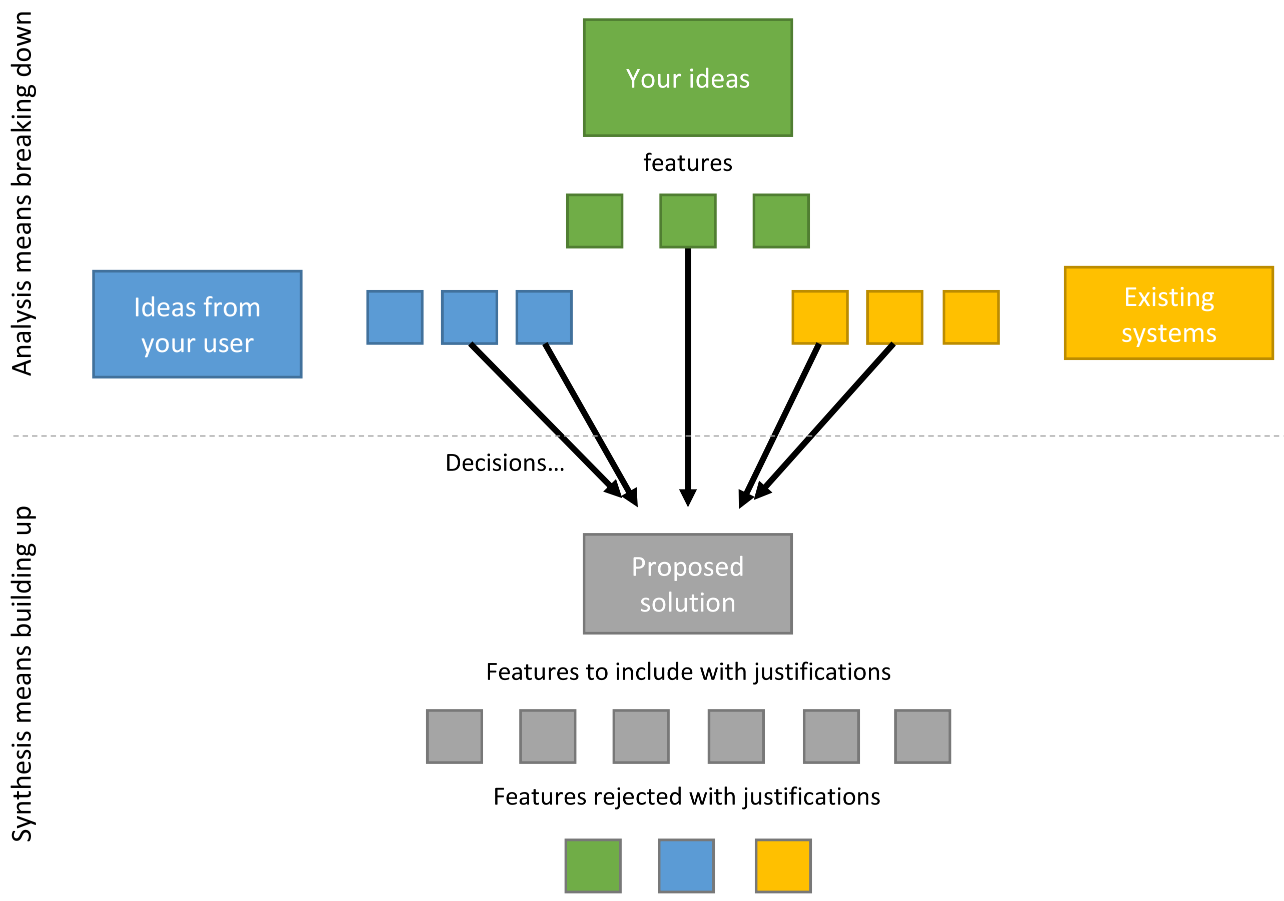
Having concluded your research there will be several features that you identified that must be implemented to solve the problem. There will also be some ideas that could be implemented, but perhaps these are unnecessary, too time consuming or too complex and therefore fall outside of the scope of the solution. There will also be ideas that need to be rejected for these reasons too.

Look back carefully through your research and provide a summary of each feature of your proposed solution, explaining and justify your choice for including it in your project.

A* TOP TIP

For the highest marks you cannot simply list the features of your proposed solution, you must provide an explanation and justification of each one.

This diagram is a useful way to visualise what you are trying to achieve when identifying essential features:


Analysis means breaking down
Synthesis means building up
Your ideas
features
Ideas from your user
Existing systems
Decisions…
Proposed solution
Features to include with justifications
Features rejected with justifications

Identifying and explaining any limitations

Although the project should be a challenge and it is acceptable not to fully complete the work, it is often better to have a smaller number of features implemented well than a large number unfinished. This demonstrates you can scope a solution well. You should identify the limitations imposed on the project with a justification for not including these features.

For example, you may also be limited by your hardware and available libraries. You may have to simulate the input and output of your target device during development, especially if you are targeting the tablet or mobile market. There may be limitations on the accuracy of your model due to the complexity of the calculations. For example, in a planetary orbit simulation it would be more accurate if the sun or a planet wobbled on its orbit due to the barycentre. However, this may be considered an acceptable limitation because it has little bearing on the purpose of the simulator.

Computational methods

A problem is defined as being amenable to a computational approach if there are algorithms that can solve it within a finite number of steps. Problems exist that can theoretically be solved in a finite number of steps but would take far too long with today's technology to solve, so they are also not considered amenable to computation. Cracking 256-bit encryption is an example. It is possible but would take over 3×10^{52} years! Given infinite computing power and infinite time you cannot detect all metamorphic malware. The halting problem is a perfect illustration of a problem that is not amenable to a computational approach. A computer can never know if it is stuck in an infinite loop.

In this section you must explain what the features of your problem are that make it suitable to be solved by a computer rather than other methods.

Some things to consider include:

- Computers are suited to problems with sequential, logical steps.
- Computers can perform operations extremely quickly compared to humans.
- Computers eliminate human error (other than logic errors!)
- Computers are suited to storing and retrieving large amounts of data.
- Computers are suited to finding patterns in data.

Other considerations might include:

- Abstraction allows the removal of unnecessary details to reduce the complexity of the problem.
- Backtracking allows an algorithm to explore other possibilities to arrive at a solution.
- Heuristics allow a best fit rather than an absolute result to be acceptable.
- Pipelining is the result of one process being the input to another.
- Concurrency allows more than one process to happen at the same time.
- Visualisation allows us to view the problem from a different perspective.
- Caching allows for the storage of previously calculated results.

A* TOP TIP

It is important that this does not become a written account of the theory from component 2 of the course. You must apply the principles of what you have learned to your project.

Specifying hardware and software requirements

You need to discuss the hardware and software requirements for both development and deployment.

- State what IDE and programming language you will be using and the key features that make it suitable for your development. E.g. event driven, object-oriented, runs in a browser etc.
- State any additional libraries you intend to use and discuss why they are needed.
- State which operating system are you using – e.g., Windows, macOS or Linux.
- State the hardware required. This may simply be a keyboard, mouse, monitor and secondary storage or for robotics projects may also include actuators and sensors.

Examiner's advice

"It is recognised that some projects lend themselves more to discussion of hardware and software requirements. As such, some candidates will have little to write on this aspect."

A* TOP TIP

Remember it is important to justify your choices for top marks. Why is the language suitable? Why do you need to include library functions?

Identifying and justifying measurable success criteria

In the concluding section of the analysis, you are summarising what you intend to achieve. It is important your criteria are specific, measurable, and numbered for easy identification.

The success criteria serve three purposes:

1. An agreement about the scope of the solution – exactly what will be included in the program.
2. A framework to test the solution against.
3. A reference for the evaluation.

Statements such as "must be easy to use" are too subjective. For it to be a proper success criterion, you need to detail how it would be achieved rather than writing broad statements.

Examiner's advice

"Where candidates only have vague objectives like, "My interface must be attractive," they tend to not only drop marks in the analysis but also later in the project. Time spent on good-quality, measurable objectives pays dividends further down the line, as they form the backbone of strong testing and evaluation sections."

Analysts consider *volumetrics*, which means "how many and how often." Using numbers makes your objectives specific and easily measurable. For example, the simulation will only include four process states: ready, running, blocked and finished.

There is no need to state what you will not include in your program. You considered this in the limitations section. The success criteria are a simplistic reference point for the rest of the project.

You should briefly justify each of your choices. You are not completely rewriting the features of the proposed solution section but summarising the key reason for including the feature in the scope.

A good approach is to number each of your criteria so you can use the same number system later when discussing them in the testing and evaluation section. Aim for between 12 and 30 criteria. Too few, and you probably do not have sufficient scope and complexity for an A Level project. Too many, and you risk giving yourself too much work to do in the time available.

The example below is presented in a table, simply to illustrate the numbering of the criteria and how they can be justified more easily. There is no requirement for you to present your criteria in this way.

An example of success criteria for *the scheduling simulator*

No.	Criteria	Justification
1	Allow the user to select between five scheduling algorithms: FCFS, SJF, RR, SRT & MLFQ.	The data produced by the program should allow the fair comparison between the algorithms using the same simulation. These are the algorithms studied on the course.
2	Show the user a visual representation of four process states: ready, running, blocked and finished.	Understanding how the algorithms work requires the user to see how a process is managed by the operating system.
3	Allow the user to manually enter processes into the ready queue including their time to execute between 1 and 100 cycles.	This allows the simulation to be set up in a particular way by the teacher to simulate certain scenarios.
4	The user can choose an automatic simulation where processes are added to the ready queue randomly with a random time to execute between 1 and 20 cycles.	This will provide larger, more interesting data sets for comparisons to be made that would be too time consuming for the user to generate manually.
5	For automatic simulations, a seed can be saved by the user.	This allows the same random sequence to be used again in the future.
6	The number CPU cycles required for the currently executing process is updated during execution and displayed.	This allows the student to more easily see what happens with time-slicing.
7	The simulation can be stopped at any time.	This allows the user to restart the simulation very easily which is useful if the simulation is continuing for too long.
8	Processes move from the ready to the running state according to the rules of each scheduling algorithm.	This allows the simulation to accurately illustrate the algorithm.

9	Processes move from the running to the blocked state randomly, but in a predetermined way using the seed.	This allows the simulation to accurately illustrate the algorithm.
10	Processes move from the blocked state to the ready state after between 2 and 5 cycles, but in a predetermined way using the seed.	This allows the simulation to accurately illustrate the algorithm.
11	Processes move from the running state to the ready state if the time slice expires when simulating the RR algorithm.	This allows the simulation to accurately illustrate the algorithm.
12	Processes move from the running state to the ready state if a new process requires fewer cycles of execution than the running process when simulating the SRT algorithm.	This allows the simulation to accurately illustrate the algorithm.
13	The user can set the time slice for processes when the RR simulation is chosen.	This allows the user to see the effect of different time slices on the throughput.
14	The user can speed up, slow down and pause the simulation.	This allows the teacher to explain what is happening more easily.
15	The simulation should output a CSV data set: algorithm, number of cycles, number of finished processes & average wait time.	This allows comparisons between the algorithms and different data sets to be made.

Examiner's advice

"The requirements and success criteria are part of the key to a good project. An ideal set of requirements should be detailed enough to pass onto a third party to design the system."

SECTION		DO's		DON'Ts
Project definition	1	Set the scene for the examiner	☐	Assume the examiner will understand your problem.
	2	Outline why the program is needed	☐	
Identifying suitable stakeholders	3	Identify the user groups or individuals why will use your program.	☐	Be both the developer and user of your product. Make sure you have someone else representing the target market to give you objective feedback.
	4	Involve a representative from your stakeholders throughout your project using a RAD approach to development.	☐	
Research	5	Provide detailed research into existing solutions to similar problems.	☐	Rely on your own input only for the solution to your problem.
	6	Include any related knowledge and calculations necessary to understand the problem.	☐	Rely on an interview with and end user for all your research into the problem.
	7	Include a summary of the discussion with a stakeholder.	☐	Write up interviews as a Q&A session.
	8	Include a references or bibliography at the end of your project report.	☐	Forget you need a bibliography.
Features of the proposed solution	9	Identify the features you are going to include in your program.	☐	Attempt to solve a problem that is too complex in the time allowed.
	10	Identify any limitations of your proposed solution.	☐	Settle for a problem that is not going to have some challenges you can discuss later.
	11	Be realistic about what can be achieved in the time allowed.	☐	
Computational methods	12	Explain why the problem is suited to being solved by a computer.	☐	Explain the theory of computational methods from unit 2.
Hardware and software requirements	13	Specify the hardware requirements for your solution.	☐	Forget there are requirements for both development and deployment.
	14	Specify the software requirements for your solution including the operating system.	☐	Simply identify the programming language you are using without justification.
	15	Identify any additional utilities or libraries that will be required to implement your solution.	☐	
Success criteria	16	Specify the measurable requirements of what your program must do.	☐	State vague subjective criteria that are difficult to measure.
	17	Justify why the requirement is necessary.	☐	

DESIGN

The plan for your development.

In this section, you are discussing **how** you are **going** to solve the problem.

15 marks.

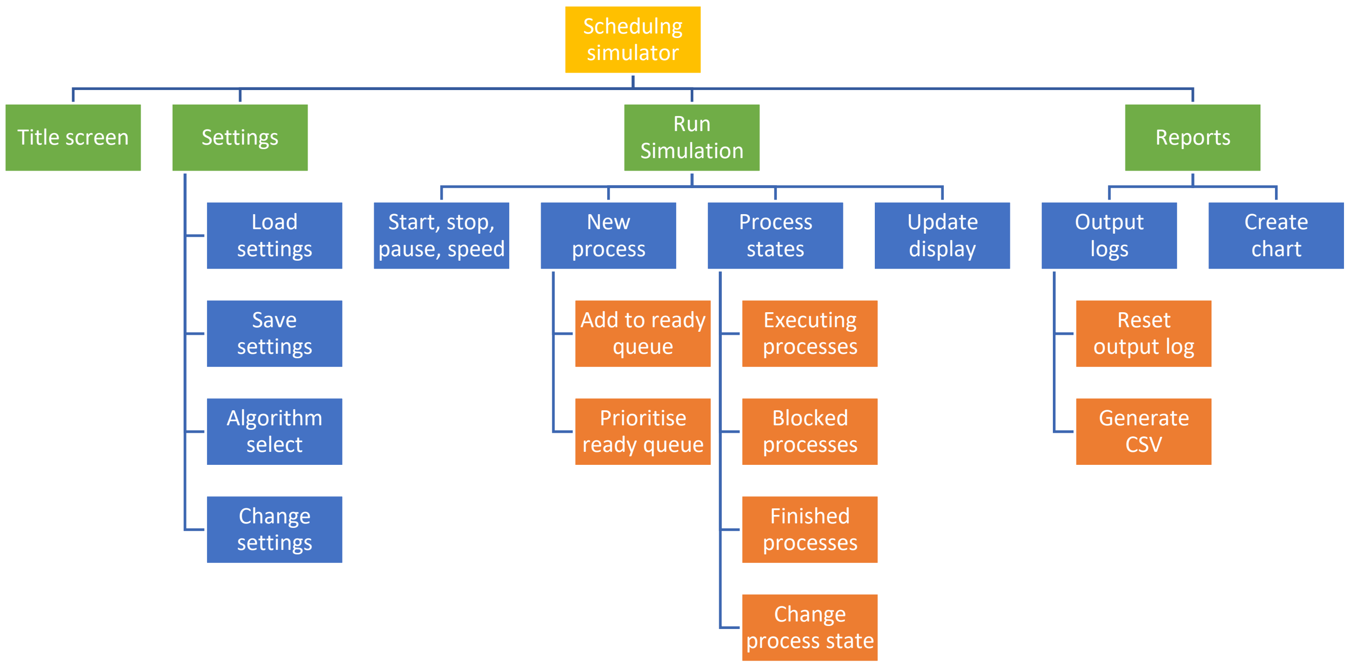
What to include

- A structure diagram illustrating the problem decomposition.
- Explanation of the components of the solution.
- Algorithms in pseudocode.
- Usability features.
- Data structures to be used.
- Input validation.
- Test data to be used during the development of the coded solution.
- Test data to be used post-development.

Breaking the problem down systematically

This section of your report explains the structure of your intended solution. In the analysis section, you explored ideas and described your intentions; this was about *what* you were going to do. In the design section, you are writing about *how* you are going to realise these intentions.

Start with a structure diagram, which will illustrate the subroutines you intend to develop. Below is an example of a structure diagram for *the scheduling simulator*.



Notice how the number of boxes underneath the ones above increase; this shows the problem being broken down into increasingly smaller parts. There is no right answer to producing this diagram – it depends on how you visualise the program you are going to make.

A* TOP TIP

You need to balance the level of detail you decide to include on your structure diagram. If it does not fit on the page or is unreadable, you have included too much detail.

- The top yellow box is the name of your program.
- The green boxes underneath the name of the program identify the key interface screens the user will see.
- The blue boxes identify the main modules or components of the solution, each associated with an interface.
- The orange boxes identify the processes and subroutines for a module. These are the routines you are likely to need to pseudocode later in the design section.

The colours are not important. Choose your own style, providing it is easy to read. Using SmartArt hierarchy objects in your word processing application is acceptable. Avoid using 3D shapes, as these will be difficult to read.

Top down design diagrams, JSP diagrams and UML diagrams are all good examples of how you can demonstrate problem decomposition. You do not need to adhere to a particular standard use the diagram that best fits your type of project.

This diagram becomes a handy reference because the examiner should be able to follow each box to later sections of your report – e.g., algorithms, coded solution and testing.

A* TOP TIP

Although it is possible to combine the design and code sections of your report in an iterative way, make sure your work still meets all the mark scheme criteria for design.

Defining the structure of the solution to be developed

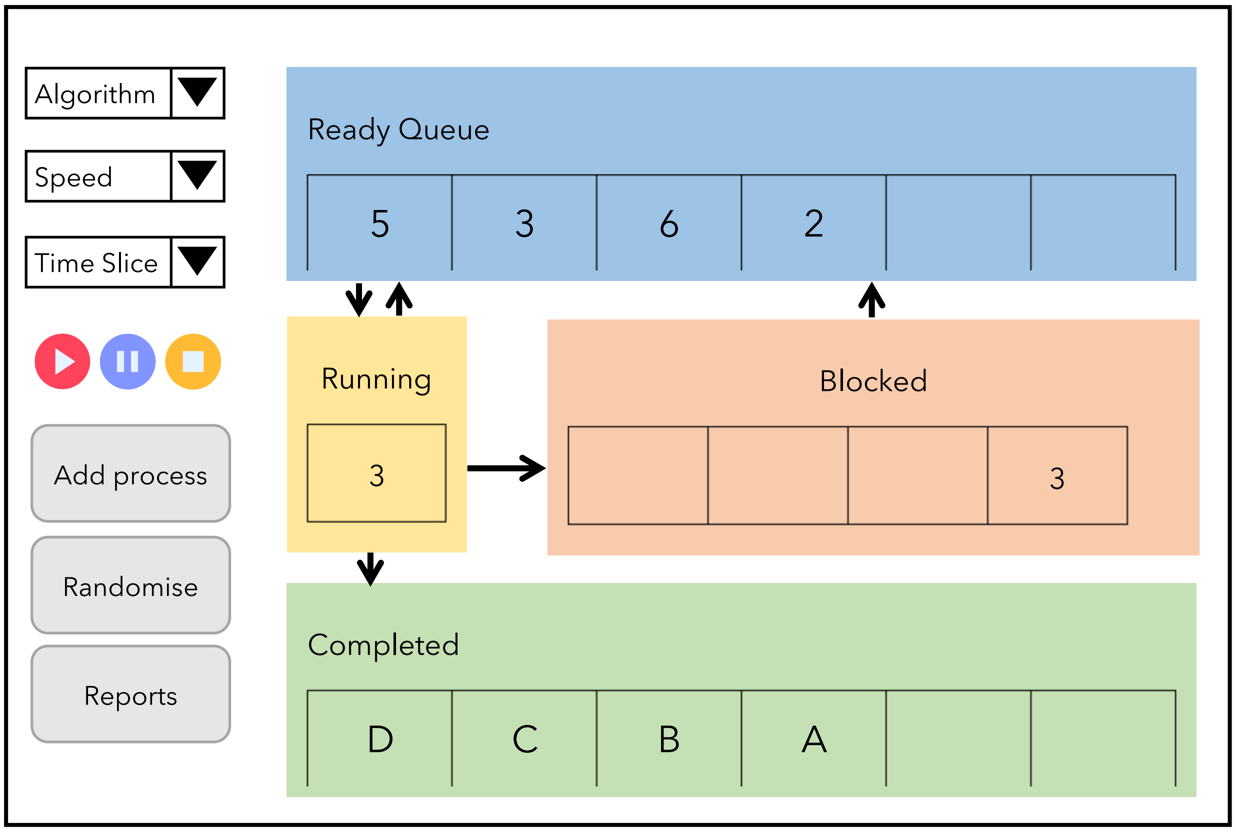
Take each box from your structure diagram and explain what is happening at each stage.

Interfaces

You should draw a technical sketch of what your interfaces will look like. You can either hand draw this with a pencil and ruler and scan or photograph it into your work, or use the shapes provided by your word processing software. While it is possible to also create a non-working form design in your chosen IDE and crop it into this section, care must be taken that this does not look like a screen shot from your coded solution. The examiner must be in no doubt this is a design.

Providing annotations on your proposed interface designs will be useful to highlight key features and you should discuss the usability features you have included explaining why they are necessary.

An example of an interface design



It is obvious that this is not a screen shot from the actual development, but a concept design. You can see how a few additional annotations would be helpful to explain aspects of the interface, and usability conventions such as the play, pause and stop buttons. This is unlikely to be the final design that is implemented, and it is acceptable to change your mind later when coding the solution without updating your design sketches.

Validation of inputs

If your interface includes text entry boxes, combo boxes, list boxes, file selection boxes or sliders you should explain the validation you are applying to these. Typical validation checks include:

Validation check	**How it works**	**Example**
Format check	The data is the correct data type and follows an expected pattern.	If an integer is expected, then characters should be rejected. Date of birth is dd/mm/yyyy Ensuring an @ is in an email address.
Length check	The number of characters isn't too long or too short.	A password that needs to be eight characters or more. A credit card is 16 numbers.
Presence check	Data has been entered into a field.	In most situations it is unlikely that no input is a valid input. Username must be entered before continuing.
Range check	Data must be within a valid range.	The month cannot be less than one or more than twelve.
Lookup table	Combo boxes and list boxes restrict the input to valid inputs.	Choosing from a list of options.
Check digit / checksum	Checks the input is an acceptable entry before attempting to look up in a database.	Credit card number is valid. This doesn't mean it exists, just that it is a valid number.

Giving the user buttons to click instead of commands to enter not only makes the interface more user friendly, but it also ensures only valid actions can be taken.

Sanitisation is the process of taking what otherwise might be an invalid input and converting it into a valid input before it is processed. For example, if the data entry must be in uppercase, instead of rejecting lowercase characters you could simply convert them after they are input. Data sanitisation provides a better user experience and more for you to write about.

Modules and processes

If the box on your structure diagram is a module or process you should describe the purpose of the routine and explain what the code in that section will achieve. Remember to justify the approaches you are taking. You may not need to describe every box on the diagram if many routines on the same branch can be discussed together.

An example of a paragraph for this section

New process

The user can add a new process to the ready queue at any time by clicking the 'add process' button. This will show a simple pop-up box asking the user how many cycles the process will require. Once selected the process will be added to the queue.

Explaining what happens in detail

If the randomise option has been selected the program will create 2-6 random processes for the queue before the simulation starts using a for loop. The number of cycles to execute will also be randomised between 1 and 20. While the simulation is running, random processes may also be added to the queue no more than one per cycle. Although the simulation will be random from the user perspective, the numbers generated will be based on a seed generated from another random number between 1 and 10000 to ensure they are deterministic. This will allow the user to save the seed in a text file run the program again in the future with the same settings to generate the same output. The processes in the ready queue will be stored in an array and will largely operate as a queue as this data structure is the best fit for the operation here.

Justification

Depending on the scheduling algorithm chosen, the queue may need to be prioritised. For example, for SJF and SRT algorithms. In these situations, I will use an insertion sort to sort the processes by the number of cycles remaining in the array. The insertion sort is easy to program and given the data set is small there is no need for it to be more efficient.

Justification

It is important to remember the design section is not as much about *what* you are going to do (this was discussed in the analysis) but more about *how* you are going to do it. Be careful not to repeat what you have written in the analysis too much. If you get this section right, you should be able to write basic pseudocode from it in the next section of your report.

For top marks, you will need to justify your design decisions. These are not justifications of the inclusion of features – you did this in the analysis section. You are justifying how these features will be implemented.

Algorithms

This section of your report is not the actual code but, instead, the algorithms you intend to write. It is a decomposition of the subroutines you identified in the systems diagram for the modules.

Although you can use flowcharts, they get unwieldy very quickly when algorithms become more complex and are time-consuming to produce. It is advised that you use pseudocode in this section.

Algorithms are represented with pseudocode so they can be interpreted by programmers, irrespective of the programming languages they are familiar with. Pseudocode, as the name suggests, is a false code or a representation of code that can be understood by most people.

The differences between algorithms, pseudocode and source code

Algorithm: A logical sequence of the actions of a process. A programmer implements an algorithm using source code. They then express it in a language-independent way using pseudocode.

Pseudocode: Has no defined syntax, so it cannot be compiled or interpreted by the computer. However, it is written in a way that makes it easy to produce real code for most languages.

Plain English	Pseudocode	Source code	Machine code
Describing your solution (the previous part of your report)	Proposed algorithms for procedures/functions/methods (this part of your report)	The real programming code (developing a coded solution)	Binary the computer executes

At this stage, you might feel you do not have the confidence to write your algorithms. Perhaps you're not entirely sure what you are going to do – in which case, your analysis and the first part of your project may be weak – or you're not sure how to write the code because of insufficient experience with your chosen language. If you hit a brick wall here, go back and make sure you have been decisive about what your program needs to achieve in the previous section.

Once you have done that, remember – pseudocode is not real code. Have a go at identifying the logical steps, invent commands as you need to and use variables as you write the commands.

Pseudocode

There is no requirement to use the examination board reference language. These are for writing examination questions only.

You should use indenting with pseudocode, just like with real code.

It may not be obvious why you have approached a specific routine in the way you have. There are often hundreds of different ways to write an algorithm, each delivering the same output. Therefore, you should justify your approach where appropriate. Consider space and time complexity, readability and maintainability of the code, and thinking ahead as areas that could lead to good justifications.

Example pseudocode

Pseudocode

```
FUNCTION sort_processes ()

  FOR index1 = 1 TO queue.LENGTH
    index2 = index1 – 1
    WHILE index2 > 0 AND queue[index2 -1] > queue[index1]
      queue[index2] = queue[index2 -1]
      index2 = index2 -1
    END WHILE
    queue[index2] = queue[index1]
  END FOR
  RETURN queue

END FUNCTION
```

Justification

This routine sorts the ready queue for the SJF and SRT algorithms.

This is an insertion sort which is efficient with a small data set of less than 64 items.

It will be needed to prioritise processes in the ready queue.

Note how this routine is not written in any particular programming language, although it has clear similarities with BASIC. This allows you to use commands that may not exist in your target language to make it easier to produce the pseudocode.

You do not need to adhere to the standard illustrated, use a pseudocode approach you are familiar with.

Students usually struggle at this point of the project so if you feel daunted do not worry. Consider everything you have learned about programming and using the list of simple commands on the next page, just have a go. It is acceptable for your real code not to match this algorithm design. If it looks like you have copied and pasted real code retrospectively you will not gain any credit in this section.

Commands you could use in pseudocode

Command	Purpose

`FUNCTION` *`name (parameter1, parameter2)`*
 	
`RETURN` *`value`*
 	
`END FUNCTION`
	Functions create a reusable program component that accepts parameters and returns a value or structure. They are used to avoid the duplication of code that may be used multiple times.
	
	Code should be indented inside functions.

`SUBROUTINE` *`name (parameter1, parameter2)`*
 	
`END SUBROUTINE`
	Subroutines structure your program into logical sections. Use a subroutine for each major part of your code so that you do not have one long code listing.
	
	Code should be indented inside subroutines.

`variable` `=` *`value`*	Assign a variable a value. This could be the value of another variable, a constant or result of a function.

`IF` *`variable operator value`* `THEN`
 	
`ELSE IF`
 	
`ELSE`
 	
`ENDIF`
	IF is a selection statement that compares the value of a variable. Remember the operators:
	
	= equals
	< less than
	<= less than or equal to
	> greater than
	>= greater than or equal to
	DIV integer division
	MOD modulus
	
	Code should be indented inside selections.

`FOR variable = 0 TO` *`value`*
 	
`END FOR`
	FOR is used to repeat lines of code.
	
	Use a for loop when you know how many iterations need to be performed.

`WHILE variable operator value`
 	
`END WHILE`
	WHILE is used to repeat lines of code.
	
	Use a while loop when the number of iterations could be variable. For example, with validation when you don't know if the user will enter a correct value, or when you want a code routine to end before it is finished.
	
	Many students use IF when they should be using WHILE, and use WHILE when they should be using FOR!

`OPEN FILE filename`
 	
`WHILE NOT filename.EOF`
 	
`  READLINE string`
 	
`END WHILE`
 	
`CLOSE FILE`
	Files are used to read and write data from secondary storage. All files need to be opened and closed before data can be read and written to them.
	
	It is common to use a while the end of file is not reached loop to read data until the end of the file although this may not be necessary.

The linear search iterating over an array is one of the most common algorithms:

```
value = "what we are looking for"
found = FALSE
index = 0

WHILE NOT found and index < array.LENGTH -1

  IF array(index) = value THEN
    found = TRUE
  ELSE
    index = index + 1

END WHILE
```

Note you can use either parenthesis () or square brackets [] for indexes.

Remember you can invent commands in pseudocode, so for example, if you were really stuck writing this algorithm you could simplify it to: `IF value IN array THEN`

Try to avoid simplifying your algorithms too much because you want to gain some credit for your logical thinking, but do so when necessary.

You should have a pseudocode routine for each major module or algorithm in your program. There is no need to design trivial algorithms used for pressing buttons on GUIs. For example, you may have an interface where the user clicks a button to open another user interface. The pseudocode to achieve this could be:

```
SUBROUTINE BUTTON on_click()
  OPENFORM "frm_Scheduler"
END SUBROUTINE
```

It is not necessary to include these minor algorithms. Equally, if you are likely to have many repeating code elements, it would be appropriate to show these only once – but remember, if this is the case, you would be better off using a subroutine anyway.

A* TOP TIP

Remember, justifications are the key to achieving the highest grade. Justify why you need the algorithms you have designed.

It is not essential to justify every line of code. It is acceptable to justify blocks of code instead. For example, if you decide to implement a linear search, hashing or a sorting algorithm, it would be acceptable to justify the algorithm itself and why alternatives are not suitable rather than justifying individual lines of code.

You will know you have achieved a good report if the examiner can "follow the thread" of small aspects of your project, from the initial proposals, to success criteria, to a description of the modular nature of the solution to the actual algorithm design. Everything needs to be coherent and easy to follow throughout your project report.

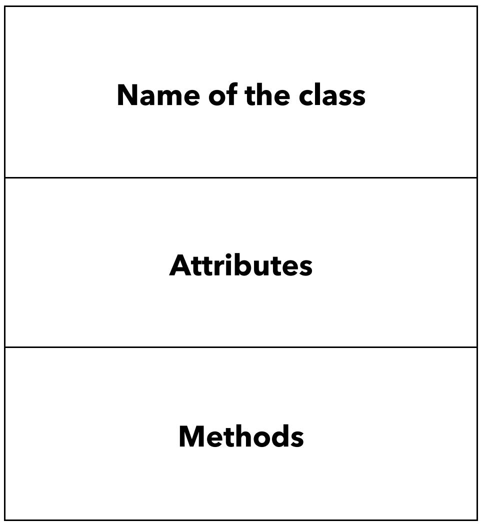
Examiner's advice

"Reverse-engineered code is given no credit."

Identifying the data structures and validation

Now that you have written your algorithms, you should be able to identify all the constants, variables, data structures and files you need. If you are taking an object-oriented approach you will need to include class diagrams.

Class diagrams



There are three rectangular sections to each class diagram.

1. The name of the class
2. Attributes: the variables and constants.
3. Methods: the subroutines.

You will have several small class diagrams. If you are using inheritance remember to illustrate this with an arrow between the classes. Child classes point to their parent class.

Remember, to achieve the top mark band, you need to justify your choices. If you are using an array, perhaps this is because you want to iterate through the structure with a loop or maybe you have chosen a list because it is dynamic and items can easily be added and removed at run-time.

Boolean data types have two states, making them an ideal choice for when situations need to be stored as true or false, on or off. In modern developments, programmers often avoid using a Boolean so a third and fourth state can be introduced later if a future iteration of the project requires it.

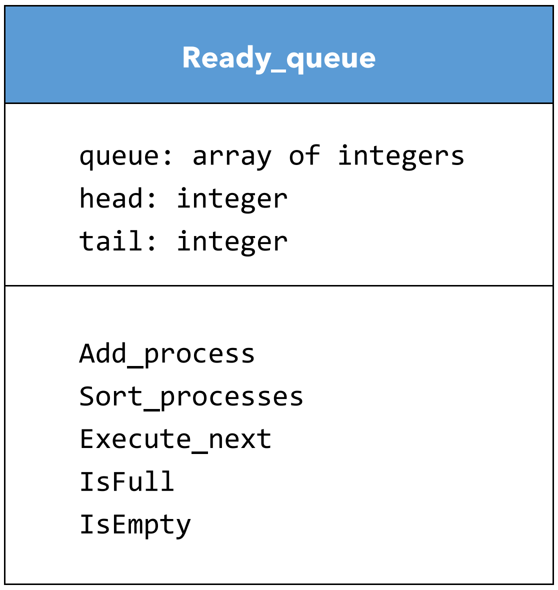
Remember, if your program's values do not change, using constants is more efficient, as the compiler can use immediate instead of relative addressing, speeding up the execution of your program.

In addition to the classic data structures, you should consider the need for higher-order data structures such as graphs and trees.

Examiner's advice

"Candidates who opted to take an object-oriented approach tended to fare particularly well in the design section, having a clear idea as to how their project can be decomposed."

An example class diagram



If you have adequately described each subroutine and produced pseudocode for it, the purpose of the methods should be clear.

Each variable, constant and attribute should now be identified. A table is a useful way to do this.

An example of a variable table with justifications

Variable	Data type	Purpose, justification and validation
`queue`	Array of Integers	Stores the processes in the ready queue. As the size of the ready queue is fixed in my implementation an array is suitable. The element holds the number of cycles the process needs to execute.
`head`	Integer	The next process to enter the running state is at this index position in the queue array.
`tail`	Integer	The position of the last process in the ready queue.

Do not forget to justify any decisions you make about the data structures here.

A* TOP TIP

It may seem odd to justify your variables, especially if you name them appropriately. However, justifications are a key requirement of achieving marks in the top mark band.

As programmers we are making lots of decisions about how to code the algorithms. For example, should you use an array, or would a list be better? Do you need to store all the data in memory, or would it be better to use a file? This is why justifications are important to achieve a mark in the top mark band because it demonstrates your thinking process.

Files

If your program uses external files, it is important to illustrate the structure of how the data is being stored because it will be an abstraction – i.e., the relationship between the data and the program will not be immediately obvious.

```
7318523
4
120
```

For example, consider the data file to the right, named “sim1.txt”. It is not obvious what the values represent. What is the number `7318523`? Why is it needed?

Using structured files like JSON can help alleviate this. However, if you are using files to load and save data, you should describe one record of the structure.

An example of an annotated file

sim1.txt	

`7318523`
`4`
`120`
	The seed used to generate the random number sequence.
	The time slice used for the RR algorithm.
	The number of cycles to run the simulation for.

Encoding

Using characters or values to represent string data is common in programming. For example, in our scheduling algorithms program we could choose the four characters "FCFS" to represent "first come, first serve" or simply assign it the number 1, with the shortest job first algorithm being option 2. You must explain all the encoding you are using so it is obvious what any values in your code represent.

Identifying test data to be used during iterative development

At this point, it is worth reflecting on your systems diagram and algorithms. From this, identify six to ten key milestones you will need to achieve. You will use these later in the coded solution.

Example milestones

1. Creating the main interfaces
2. Adding items to the ready queue
3. Moving items from the ready queue to the running state
4. Handling executing processes in the running state
5. Handling completed processes
6. Handling blocked processes
7. Saving and loading results and simulation data
8. UI features: pause, stop, reset, speed up, slow down
9. Charting results data

For each of these key milestones, identify the test conditions you will need for that milestone and present them in a table.

A development test example

Milestone 2: Adding items to the ready queue		
Test number	**What is being tested and inputs**	**Expected output**
1	The user can select the "add process" button and a pop-up box prompts them to enter the number of cycles. Test -5, 0, 1, 12, 100, 230, H, #	A number between 1 and 100 is accepted. Clicking cancel aborts the operation. All other inputs are rejected. The process is added to the ready queue.
2	If the FCFS or RR algorithm is selected the new process is put at the back of the queue. Test queue: 5, 12,8,9 - input 4.	The processes are in the correct order in which they were entered. Queue contains: 5, 12, 8, 9, 4.
3	New random processes are added at the correct position in the queue for FCFS and RR.	New processes are always added to the back of the queue.
4	If the SJR or SRT algorithm is selected the queue is sorted with the process with the lowest number of cycles being placed at the front. Test queue: 6, 8, 9, 12 - input 8.	The processes are in the correct order numerically by cycles. Queue contains: 6, 8, 8, 9, 12.
5	New random processes are added at the correct position in the queue for SJF and SRT.	New processes always cause the ready queue to be sorted.

Note how it is necessary to test valid inputs, invalid, inputs and boundary inputs. The test data should consider a wide range of input possibilities. The test data here is far from comprehensive even for this small part of the program. In addition to this, developers also perform white-box tests where each of the program paths are explored. For example, an IF statement has two possible outcomes. It is entirely possible that the test data chosen does not explore this sufficiently and therefore further tests are required when you know how the algorithms are working.

Identifying further data to be used post-development

Post-development testing includes alpha, beta and integration testing.

Alpha testing includes looking at the success criteria to determine if the solution works as intended before releasing the software to the stakeholders.

Remember to justify the tests you are going to perform later to achieve top marks.

A* TOP TIP

You may also decide to offer your stakeholders a beta test opportunity so they can suggest any final refinements and test it meets their needs.

In the post-development phase you are able to test that all the individual modules work together. This is known as integration testing. You should prepare scenarios in advance to be used post-development to check the output from the entire program matches expectations.

A post-development test example

Post-development test	**Justification**
The FCFS algorithm with the processes 5, 12, 8, 4 should execute in 29 cycles with the processes being output to a finished state in the same order they arrived. The average wait time should be calculated as 11.75: (5+17+25) / 4	The program should correctly illustrate a process moving from the ready queue to running to finished states and calculate the average wait time for the processes.

SECTION		DO's		DON'Ts
Structure of the solution	18	Provide an overview of the structure of your solution using a diagram to illustrate breaking the problem down into smaller parts.	☐	Be concerned about using a JSP/UML standard.
Breaking the problem down systematically	19	Provide evidence of problem decomposition by explaining each component of your intended program.	☐	Reverse engineer your completed program. Your code will likely deviate from these initial design ideas. Include screen shots from your finished program. Spend too much time on graphics and colourful illustrations.
	20	Include sketches of interface designs.	☐	
	21	Explain the validation and input sanitisation that will be necessary.	☐	
	22	Consider the usability features of your program and expected UI conventions.	☐	
	23	Justify your approaches to all the above.	☐	
Algorithms	24	Provide a set of language independent algorithms to describe each of the subroutines and methods.	☐	Use flowcharts, they are too cumbersome for large problems. Provide code or reverse engineered code as an algorithm.
	25	Use your own pseudocode standard if you need to use commands you are not familiar with.	☐	
Data structures	26	Provide class diagrams if you are taking an object-oriented approach.	☐	Forget to include descriptions of any encoding. Forget to include a file structure reference.
	27	Explain and justify the variables and data structures that you will use.	☐	
	28	Justify your choice of data structures.	☐	
Test data for development	29	Identify 6-10 milestones for the development.	☐	Do this retrospectively. Test data should be drawn up in advance of development. Forget that good testing includes a variety of valid, invalid and boundary tests.
	30	Identify the tests that need to be performed for each milestone before it can be considered complete.	☐	
Test data for post-development	31	Identify and justify testing that needs to be performed when all the modules have been developed.	☐	Avoid issues where your program fails. These are good to talk about even if they are not resolved. Forget that post development includes integration, alpha and beta testing.
	32	Identify data that is designed to test the robustness of the solution; good testing should attempt to break the program.	☐	

DEVELOPING A CODED SOLUTION

The iterative development process.

In this section, you are discussing **how** you have made a **solution** to the problem.

15 marks	Coded solution
10 marks	Testing to inform development

What to include

- Evidence of developing each milestone.
- Explanations of what you did and why.
- Annotated code with suitably named variables and routines.
- Evidence of input validation.
- Review of each milestone.

An agile approach to development

At the end of the design section, you identified six to ten key milestones. In this section, you need to provide evidence of developing each of those milestones. The development must be an iterative process. That means you cannot just provide a code listing of your finished product. Instead, you must document the story of how the development evolved as it was being coded.

It is expected that your development will change from your initial design ideas. Rarely do programs become exactly what was initially proposed. When you do change your mind, you do not need to go back and change your analysis and design. Instead, document your changes and the reasons behind them in this section of your report. If the change is so fundamental that the design no longer resembles the development, you may want to go back and make your report more coherent, but this should rarely be necessary unless your analysis was very weak.

In this section of your report, you will be including parts of your code. If it is practical to do so, it is helpful to include a full code listing in an appendix at the end of your project. In this section, you will provide only parts of the code that are relevant to each milestone.

When developing a milestone, you will frequently need to return to previous aspects of the code to make changes or add new routines. Do not include code sections you have already included in your report previously – just the new parts.

Examiner's advice

"At the top end, candidates have clear and discrete iterations. Each iteration has stated objectives, which were tested and evaluated at the end of the iteration."

Documenting the milestones

For *each* milestone, include the following:

1. **Objective of the milestone:** In one or two sentences, describe what this milestone achieves.
2. **Code listing:** Provide a listing of the code for this milestone. Code must be commented with appropriate variable and data structure names.
3. **Explanation of the code:** Explain what the code section does and justify algorithms or approaches. It might not be immediately obvious to an examiner how the code works or why problems have been solved in the way they have. Remember, the examiner may not be familiar with the language you are using. If input validation is required, explain that too.
4. **Module testing:** Explain the tests you performed for this section of code and the outcome of those tests. You must provide evidence of this process. A few simple screen shots for each milestone will be enough.
5. **User feedback:** Your user should tell you what they like and dislike about this aspect of the development so far, with ideas for future development. Your user must be an integral part of the iterative development process.
6. **Reflections:** As a result of testing and user feedback, discuss what needs to be changed.

A* TOP TIP

The very best projects tell the story of the development. You do not need to include commentary of resolving every single error, but you should explain the challenges you faced.

Rapid application development (RAD) where a series of evolutionary prototypes are developed, shown to the user, and further developed until eventually the final product is delivered is how your project is intended to be documented. Try to avoid a simplistic life-cycle approach and instead describe your journey of development. This should be more like a dev-diary and less like a linear explanation of achievements.

A* TOP TIP

Do not forget to justify the approaches you have taken in the explanation of the code – especially if they are different to your original design intentions.

Example of part of one milestone

Adding items to the ready queue – sorting the queue for SJF/SRT.

Code I wrote to achieve this milestone:

```
Sub sort()
  'Sort the ready queue using an insertion sort
  Dim n As Integer = queue.Count
  Dim index, index2, current As Integer
  'Iterate over all the processes in the ready queue
  For index = 1 To n - 1
    current = queue(index)
    index2 = index
      'Find the place for the process
      While index2 > 0 AndAlso queue(index2 - 1) > current
        queue(index2) = queue(index2 - 1)
        index2 = index2 – 1
        'Insert the process in the correct place
        queue(index2) = current
      End While
  Next
End Sub
```

Examiner's advice

"Care must be taken that code shown in screenshots is big enough and of a high enough resolution. Ideally, syntax highlighting should be preserved. It is the responsibility of candidates to ensure that their final PDF or printed copy is readable. Credit cannot be given for code that cannot be read."

Explanation of the code

When an item is added to the ready queue the tail pointer is increments and the new item is appended to the end of the array at the position pointed to by the tail pointer. I could have inserted the new process directly into its correct position using a linear search to find the position and then moving all the other items up one index before inserting it. This would work fine, but I wanted to create a routine that would sort the ready queue no matter what state it was in. For example, processes that arrived back from a blocked state and new processes can be handled at the same time. This would enable the simulation to show the ready queue being evaluated before a process is chosen for execution which seemed better for teaching how the algorithms work.

I did consider using a few sorting algorithms because they would all work, but I knew how to code an insertion sort and the number of items to sort is very small, so efficiency wasn't a concern.

When I ran the code, it would fail in the line that finds the correct position for the process.

```
While index2 > 0 And queue(index2 - 1) > current
```

This is because index2 – 1 could be reading past the end of the bounds of the array. I needed the execution of the condition to stop if index > 2 without evaluating the second part of the expression. I noticed that this seemed to happen automatically in other languages I had used.

After researching the while loop I found that you could use AndAlso to prevent the second part of the statement being executed if the first part was not met. This solved my problem.

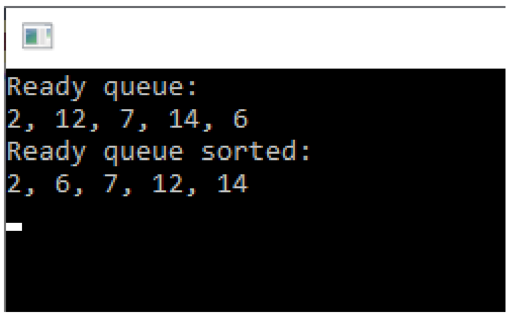
Module testing

I manually added some processes to the array before executing the routine to check it was working. I tested the following data sets:

- 2, 12, 7, 14, 6 – a random set of data
- 2, 2, 3, 4, 6, 8 – an already sorted set of data with duplicate values
- 15, 7, 4 – a reverse sorted set of data.
- 6 – only one item in the array and zero items in the array.

Once the program is written I will need to test this routine again without using manual inputs to check it still functions.

Evidence


```
Ready queue:
2, 12, 7, 14, 6
Ready queue sorted:
2, 6, 7, 12, 14
```

Examiner's advice

"There is no need to exhaustively demonstrate each test, but there should be enough evidence to convince the reader that substantial testing took place. The best projects focus on the more unusual problems that cropped up during developmental testing (getting libraries to work, nuances of programming language, issues with file types) rather than trivial syntax errors."

User feedback

I showed one of my stakeholders the implementation of this algorithm on the interface form. She commented that the ordering happened so quickly that it was hard to tell what had actually happened. She suggested that maybe a log feature would be useful to output what was happening at each stage but also agreed that the simulation speed slider to be implemented later may also help.

Reflections

I decided it was important to research exactly how and when the order of new processes was handled in scheduling algorithms to decide how important the visual aspect of this was to understand the scheduling process. Whether this happened dynamically as soon as a process arrives at the queue or if it happens before the operating system chooses the process to execute matters.

It is expected that your coded solution will deviate from your design. There is no need to rewrite the design section – this gives you plenty to write about when documenting a milestone.

A* TOP TIP

SECTION		DO's		DON'Ts
Agile development	33	Provide evidence of developing each milestone using a development diary.	☐	Provide one code listing as evidence of the coded solution without a discussion of the development process. Go back and change your design section.
	34	Include a code listing for the milestone and highlight interesting routines.	☐	
	35	Explain how the milestone was implemented, discussing approaches that were taken and how they deviate from the original design.	☐	
Code	36	Produce modular code using procedures, functions and/or methods.	☐	Assume that a data structure or approach you used is obvious to the examiner. Create code for the sake of it if there are suitable library functions you can use. Copy and paste code without references.
	37	Use comments at the beginning of routines and for selection and iterations as a minimum.	☐	
	38	Use sensible variable and data structure names.	☐	
Development testing	39	Test each module for function including the possible different program paths.	☐	Forget the full range of validation techniques you could use. Forget that input sanitisation is an appropriate technique. Forget that drop-down boxes, list boxes, and sliders are validation techniques too.
	40	Test a wide range of valid and invalid inputs and situations.		
	41	Provide evidence of the testing, perhaps using a limited number of screen shots.	☐	
Stakeholders	42	Show your completed milestone to a stakeholder for review and capture their comments.	☐	Forget to include your stakeholder in the development process. Remember this is an agile approach, not software development life cycle.
Reflection	43	Review each milestone and discuss what needs to be changed because of feedback and testing.	☐	Be subjective in your review. Hide weaknesses in your code, discuss them instead.
	44	Explain any changes required and any modifications to the design of the solution that result from the testing.	☐	

EVALUATION

Reviewing the project.

In this section, you are discussing **how effective** your project was.

5 marks	Testing to inform the evaluation
15 marks	Evaluation

What to include

- Evidence of your program being tested for both function and robustness, ideally with a video.
- Discussion of each of the success criteria and whether they were full, partially or not met including how unmet criteria could be addressed.
- Discussion of the usability features and whether they were successful, a partial success or a failure including how any issues could be addressed.
- Consideration of the limitations and improvements that could be made.

Testing to inform the evaluation

In the coded solution section, you performed what is known as "bottom-up" testing. Each module or milestone was tested independently for functionality as it was being developed, but at that stage, you did not test the program as a complete product. Towards the end of a project, a post-development testing phase takes place before the product is shipped.

- Integration testing checks all the individual components work together as a whole.
- Alpha testing assesses to what extent the final product achieves the success criteria.
- Beta testing allows the user to test the program for usability before release.

Your testing to inform the evaluation should consider all these types of tests.

A* TOP TIP

You must include both integration testing by checking all aspects of your program work together and usability testing to test the interfaces and user experience are appropriate.

Evidence

It is important to provide evidence of fully testing your program beyond the simple screenshots you have produced to evidence the completion of a milestone. The examiner will not run your project code. The only evidence they will see of your program being used is what you present here. The best way to evidence your solution is to create a video of your program being used.

Windows 10 has a built-in feature called Game Bar that will allow you to record your screen. Go to Settings > Gaming on your PC to control the settings for the Game Bar and see the keyboard shortcuts. Alternatively, there are plenty of other products you can use too.

It is important your video evidence is methodical and demonstrates:

- **Function:** The program being used with a comprehensive selection of its features.
- **Robustness:** Demonstrating that the majority of syntax, logic and run-time errors are handled. This includes validation checks and error messages.
- **Usability:** That the product can be used with the interfaces designed and that the on-screen text and messages are useful.

A good approach is to copy your test tables from the design section and add two extra columns to the end of each table; one to record a timestamp of where the test is performed in the video and a second to indicate whether the test was passed or failed. Your product does not need to pass every test. What matters more is that you have good evidence of tests being carried out.

A* TOP TIP

The best evidence covers many eventualities. Just because an algorithm works for one data set does not mean it works for others!

Providing a voice-over that explains what is being tested will help the examiner follow what they are being shown in your video. There is no need to include a script in your project.

Although this section is only worth five marks, it is the evidence of testing for the coded solution and the basis of the evaluation, so its importance should not be under-estimated.

Example of a test table

Milestone 1: Move the player			Evidence	
Test no.	**What is being tested and inputs**	**Expected output**	**Timestamp**	**Pass/Fail**
1	The user can select the "add process" button and a pop-up box prompts them to enter the number of cycles. Test -5, 0, 1, 12, 100, 230, H, #	A number between 1 and 100 is accepted. Clicking cancel aborts the operation. All other inputs are rejected. The process is added to the ready queue.	0:30	Pass
2	If the FCFS or RR algorithm is selected the new process is put at the back of the queue. Test queue: 5, 12,8,9 - input 4.	The processes are in the correct order in which they were entered. Queue contains: 5, 12, 8, 9, 4.	0:31-1:20	Pass
3	New random processes are added at the correct position in the queue for FCFS and RR.	New processes are always added to the back of the queue.	1:21-1:40	Pass

Examiner's advice

"It is important that evidence is provided of tests. Where there are several similar tests, it is not necessary to provide a screenshot of all of them, but there should be sufficient evidence to demonstrate the extent to which the system functions.

There is no "correct" number of tests you should perform; it will vary hugely from project to project. Your testing needs to be comprehensive. For example, you should be testing valid, invalid and boundary data at the very least, but invalid data can also include invalid characters, no data and data out of range. Consider some of the more unusual actions that could also crash your program. These are known as extreme tests.

Examiner's advice

"Many candidates provide video evidence of their testing, giving appropriate time codes for each test in their documentation; this worked extremely well and left the moderator in no doubt that the system worked as claimed."

Remember, as is the case with all sections of the project, how you evidence your work is up to you. There is no requirement to produce a video. You could choose to use a series of screenshots instead, but that makes it more difficult to illustrate some of the tests you are performing.

A* TOP TIP

For top marks, you should explain how your evidence shows that the success criteria have been met.

Examiner's advice

"Candidates should aim to take each objective listed in their analysis, cross-reference it with the tests they have carried out and discuss how successfully the objective has been met."

Evaluating your program

It is very easy to overlook the importance of the evaluation. Remember, it is worth more marks than the analysis, equal to the design and, combined with the testing, almost as much as the coded solution. Therefore, you need to leave yourself enough time to do this section justice.

At this point, all your development and testing work is complete. It may not be 100% how you envisaged it at the analysis and design stage, but the program is ready to ship! In the software development industry, there are very tight deadlines to meet to get a product to market. Sometimes, features need to be held back to be released later in patches and updates.

Consider asking a stakeholder to provide you with a review. That will often give you more to write about because they will see your project from a different perspective, and it can be hard to be truly objective about your project when you have poured your heart and soul into its development.

A review from your user is not essential but, if done properly, can provide you with a lot to write about in this section to achieve higher marks.

A* TOP TIP

You should copy your success criteria from the analysis section and provide commentary about how successful you were in achieving each of those goals. For each criterion, state whether it was fully met, partially met or not met at all. Also, explain how well the criteria was achieved in each case.

Consider the criticisms from your stakeholders. How could you address these if you had more time? What approaches would be required?

Consider the usability and interface of your product. How could you make the user experience even better? Ideas might include enabling a responsive form design on a variety of device resolutions. Would your product be better as an app or as a web-app to make better use of that platform or to increase interoperability?

Include a discussion about how the usability of the program could be improved.

A* TOP TIP

Limitations

You should discuss your limitations in detail. In addition to the time limitations you have already discussed that impact on the features you can include, there may also be limitations of the language you are using. For example, in the scheduling simulator we might conclude that it is a reasonable compromise to output the data to a text file and allow a spreadsheet to produce the charts because there is only limited charting support in the programming language chosen. Had we developed our program in JavaScript we could have made use of JS Charts to achieve this.

It is acceptable not to have met criteria and even to state that you do not know how to approach coding something that proved to be too difficult.

Example of a small section of the evaluation

Criteria		Success	Evaluation
1	Allow the user to select between five scheduling algorithms: FCFS, SJF, RR, SRT & MLFQ.	Partially met	The user can select between FCFS, SJF, RR and SRT. The program accurately simulates these algorithms with a wide range of data sets. MLFQ was not implemented because of the requirement to have multiple queues. Since I implemented the ready queue as a class it would be easy to create multiple instances of the class to overcome this.
2	Show the user a visual representation of four process states: ready, running, blocked and finished.	Fully met	The interface shows the four process states on the screen in four different coloured areas. The user can see the processes changing between these states.
14	The user can speed up, slow down and pause the simulation.	Not met	The user can stop and reset the simulation, but they cannot speed it up, slow it down and pause it. Pause would be easy to implement with a Boolean flag. Having a condition that the next stage in the process is not executed until the pause flag is false. I could slow the simulation down by having a for loop causing the simulation to wait until the loop finishes, however this will execute at a different speed on different machines so it is not ideal.

SECTION		DO's		DON'Ts
Post development testing	45	Provide evidence of testing against the test plans you created in the design section.	☐	Produce pages and pages of screenshots. Find a more efficient way of presenting your evidence. Forget that robustness is as important as function. Forget to cross reference to your success criteria, this is known as alpha testing.
	46	Use efficient ways of evidencing the testing. A video is an ideal alternative to screen shots.	☐	
	47	Provide evidence that the system is robust and does not crash too easily.	☐	
	48	Consider using a beta test and asking your stakeholder for a review.	☐	
	49	Cross-reference the test evidence against the success criteria from the analysis section to evaluate how well the solution meets these criteria.	☐	
Usability features	50	Show how the usability features have been tested to make sure they meet the stakeholder needs.	☐	Use evaluative comments that are subjective.
Evaluation	51	Provide a commentary on how well the solution matches each of the success criteria.	☐	Include a simple tick list to state each criterion was met. Make comments about how much you enjoyed the project. Forget to say how you might address any of the weaknesses identified in your evaluation.
	52	Comment on any unmet criteria and how these might be approached.	☐	
	53	Comment on any limitations and insurmountable problems you faced.	☐	