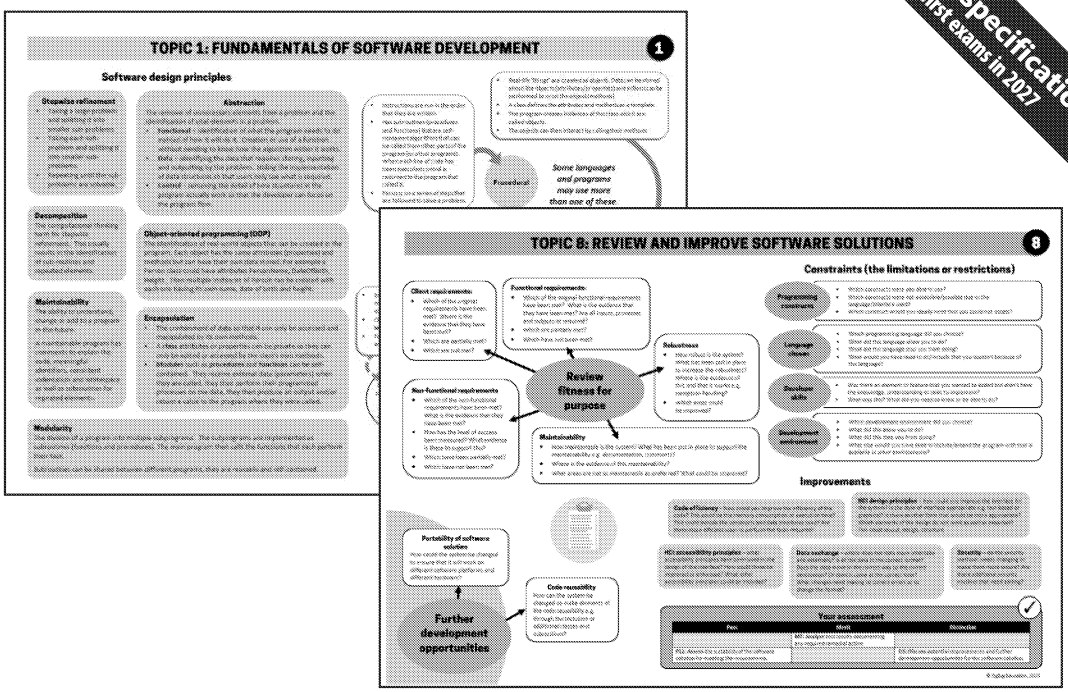
2025 specification first exams in 2027



Topic on a Page

for Unit F166: Software Development

Cambridge Advanced National (Extended Certificate)

zigzageducation.co.uk

POD 12852

Contents

All posters are provided in both A3 and A4 formats

Teacher's Introduction

This resource is intended for use by students studying the **OCR Level 3 Alternative Academic Qualification: Cambridge Advanced National in Computing: Application Development: Unit F166: Software development**. This is an optional external unit for this qualification and is assessed by an assignment.

It is important to always check the exam board website for any new information, including changes to the specification and sample assessment material.

The intention of this resource is to provide a condensed 'Topic on a Page' which provides an overview of the content of each topic area, which will enable students to review their learning and apply it to the supplied activity sheets.

How to use this resource

The resource consists of:

- 8 A3 posters covering the topics as listed below, labelled: 1 — 8
- 8 A3 activity posters which are partially completed and provide opportunities for students to fill in gaps to show their understanding of the topics and key terms, or as a planning tool to make notes for what they will do in their assessment task. These are labelled: 1 — 8

Opportunities for use:

- Printed out and displayed on classroom walls
- Individual copies to be given to students as the topic area is delivered
- Activity sheets can be given out at the end of topic delivery to check understanding
- Used as a planning or note-making tool for the assessment task

Topic Area 1: Fundamentals of software development

1

1.1 Software design principles
1.2 Programming languages

Topic Area 2: Design software solutions

2

2.1 Tools and techniques to design software solutions
2.1.1 Software Design Specifications (SDS)

3

2.1.2 Software Design Documentation (SDD)

Topic Area 3: Create software solutions

4

3.1 Programming techniques to develop software solutions
3.1.1 Variables and constants
3.1.2 Operators
3.1.3 Selection
3.1.4 Iteration

5

3.1.5 Encapsulation
3.1.6 File manipulation
3.1.7 Data structures

6

3.1.8 Other constructs and error handling
3.2 Technical skills to create software solutions

Topic Area 4: Test software solutions

7

4.1 Software solution testing

Topic Area 5: Review and improve software solutions

8

5.1 Techniques to review the effectiveness of software solutions
5.2 Improvements to, and further developments for, software solutions
5.2.1 Constraints and improvements
5.2.2 Further development opportunities

July 2025

FUNDAMENTALS OF SOFTWARE D

Software design principles

Stepwise refinement

- Taking a large problem and splitting it into smaller sub-problems.
- Taking each sub-problem and splitting it into smaller sub-problems.
- Repeating until sub-problems are solvable.

Abstraction

The removal of unnecessary elements from a problem and the identification of vital elements in a problem.

- **Functional** – identifying what the program needs to do instead of how. When creation or use of a function without knowing how the algorithm within it works.
- **Data** – identifying the data that requires storing, inputting and outputting by the problem. Hiding the implementation of data structures so that users only see what is required.
- **Control** – removing the detail of how structures in the program actually work so that the developer can focus on the program flow.

Decomposition

The computational thinking term for stepwise refinement. This usually results in the identification of subroutines and repeated elements.

Object-oriented programming (OOP)

The identification of real-world objects that can be created in the program. Each object has the same attributes (properties) and methods but can have its own data stored. For example, a Person class could have attributes PersonName, DateOfBirth, Height. Then multiple instances of Person can be created with each one having its own name, date of birth and height.

Maintainability

The ability to understand, change or add to a program in the future.

A maintainable program has comments to explain the code, meaningful identifiers, consistent indentation and whitespace as well as subroutines and repeated elements.

Encapsulation

- The containment of data so that it can only be accessed and manipulated by its own methods.
- **Class** attributes or properties are private so they can only be edited or accessed by the class' own methods.
- **Modules** such as **procedures** and **functions** can be self-contained; they receive external data (parameters) when they are called; they then perform their programmed processes on the data; they then produce an output and/or return a value to the program where they were called.

Modularity

The division of a program into multiple subprograms. The subprograms are implemented as subroutines (functions and procedures). The main program then calls the functions that each perform their task.

Subroutines can be shared between different programs, they are reusable and self-contained.

- Instructions
that they are
- Has subrou
and functio
contained a
be called fr
program (o
When each
been execut
returned to
that called i
- Focus is on
are followed

Fu

- Instructions are
mathematical
executed one a
- Closely related
declarative lan
- Focus is on wh
and is often use

Declara
Programmer
program shou
doesn't decl
Results are d
declared. The
answer

DESIGN SOFTWARE SOLU

Software Design Specifications (SDS)

Documentation that describes everything the software needs to do.

The **format** can be written documentation, an interactive document (e.g. HTML p... or ...les c...inked documents.

Layout needs to be clearly structured, understandable and easy to navi...t... e.g. ...ink ...within the document to jump to key ele...

Templates exist that are set up with the structure and hea...g... pe... t... complete.

Client – what does the client need or not need?

Organisational policies – what policies of the client's organisation need to be followed and met? What other organisations need to be interacted with and what rules do these have?

Technical requirements – what hardware and software does it need to be compatible with? What response times are required? What security is required?

Language – what type of programming language needs to be used? What can this language do and not do?

Industry standards ...gulations does the system ne... ...llow? What hardware and software does the system need to interact with?

Solution overview
A description of the scenario and the proposed solution, covering the main elements such as purpose, core requirements, target audience (who is going to be using the system) and key processes.

Client requirements
What exactly does the client need from the system? Measurable statements that the final solution can be compared to and you can ask 'Has this requirement been met?'

Functional requirements
What does the system need to do? What inputs does it need to take? What needs to happen to these inputs (the processes)? What outputs does it need to ...uce? What format are these inputs and outp...

Non-functional ...ren...
What el... ...s ti... ...n need to do or have? Does it n... ...thetic? Does it need to be appropriate ...a ...udience? Does the system need to produce a ...sponse within a set time constraint? Does the system need to have a specific level of security?

Constraints
What restrictions are there or limits on what the system has to do or where it has to operate?

Tim... syst... crea...

Har... syst... out...

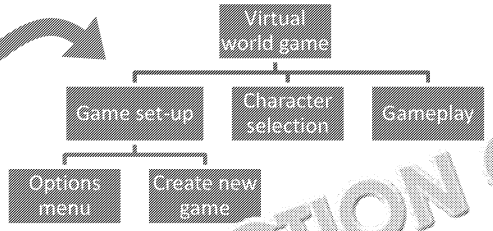
Aud... syst... Lang...

Bud... mon... spe... and...

Ext... wha... the... soft... lim...

SOFTWARE DESIGN DOCUMENT

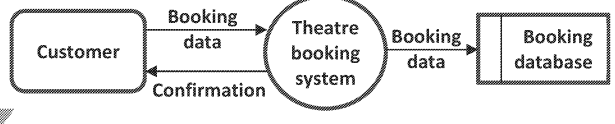
Data structure diagram
How is the system broken down? What are the sub-problems?



Architectural design
- Externa[l] components of the syste[m] s[ys]t[em] that will interact with the [s]
- [C]omponent-level design (What are [the] parts/modules/subroutines within [the] What do these do?)
- Component interfaces (Which part[s] need to communicate, and by wha[t] communicate? How is data transfe[rred]
- Module/component interactions (W[hen] different parts of the system comm[unicate] do they communicate?)

Data flow diagram
What data goes into the system? Where does this come from? Where does it go? Where is data stored? Where is data accessed from?

Level 0 (context)
System in the middle, external entities

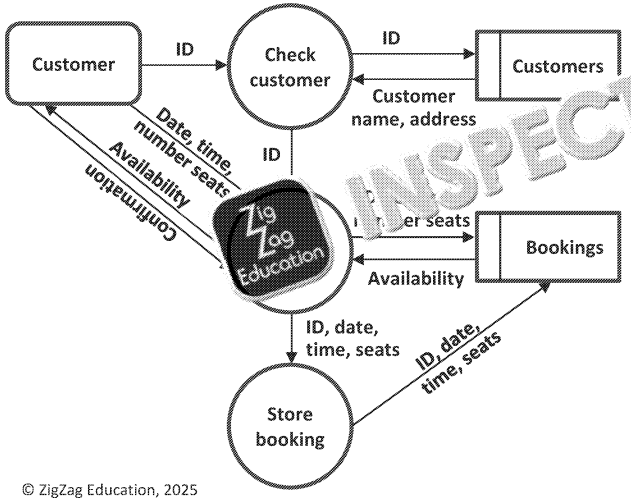


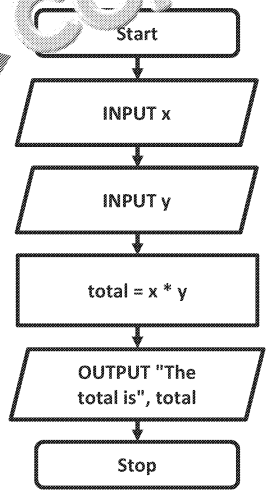
Level 1
Expands the system to include the processes and how these interact

Algorithm designs
- What **inputs** will be put into the system? From the us[er]
- What **processes** will be carried out? Calculations? Ope[rations]
- What **storage** is required? External files? Database? H[ow is it] structured? How are they accessed?
- What **outputs** will be produced? How are these create[d]? format? When are they created?

Flowchart
- Design the algorithms behind the system, the processes.
- Shows the inputs, processes and output[s] [at] each stage.

Pseudocode
'Fake' [code] to pl[an] work[ing] outp[ut] synta[x]






```
INPUT x
INPUT y
total =
OUTPUT("
```

PROGRAMMING TECHNIQ

Variables and constants

Variable: space in memory, with an identifier, that can store an item of data. The data **can** be changed while the program is running.

Constant: space in memory, with an identifier, that can store an item of data. The data **cannot** be changed while the program is running.

Identifier naming conventions: the name given to variables / mo... ...ata structures.

Identifiers must be meaningful, e.g. say what they ... ...tes what is being stored, `xyz` does not state what is be... ...

Kebab case:
`each_word_is_in_l... ...and_separated_by_an_underscore`

Camel case:
`EachWordStartsWithA...ppercaseLetterAndNoSpaces`

Data types

Data type	Description	E...
Integer	...hole number	-100 0 9954
F... point (real / float / double)	A decimal number	-100.2 0.052 35.2
String	A sequence of characters (letters, numbers or symbols)	"Yes" "22.6" "Pa55w...
Boolean	True or false	TRUE FALSE

Casting: converting from one data type to another.
Use a function that casts, e.g. here the function `int()` ... string `"123"` into the integer `123`

```
theNumber = int("123")
```

Selection

A condition is checked. The result of the check determines which code runs (and doesn't run).

IF *condition* **THEN**
Run this code when the condition is true
ELSE
Run this code when the condition is false
ENDIF
(You can have an IF without an ELSE)

IF *condition1* **THEN**
Run this code when condition1 is true
ELSEIF *condition* **THEN**
Run this code when condition1 is false and condition
ENDIF
(You can have multiple ELSEIFs. You can also have an ELSE at the end that runs when all conditions are false)

SWITCH *variable*
CASE *Condition1*:
Run this code when co... ...e
CASE *Condition2*:
Run this code when condition2 is true
CASE DEFAULT:
Run this code when all conditions are false
ENDSWITCH
(CASE conditions are usually one value, e.g. CASE 10.
You can have as many cases as required.
Default always comes at the end)

Iteration

Code runs repeatedly either a set number of times or based on a condition.

Fixed loop: runs a set number of times, usually a FOR loop.

FOR *variable = start value* **TO** *end value*
Code that repeats
NEXT *variable*

Variable starts at *start ... ...nd ... ...ses* by 1 each itera...on u...l it m... ...d value.*

STE... ... ...o change the amount by ... th... a...ble increases (or decreases)

Post-condition: the loop runs until a condition is true. The condition is tested after the code in the loop runs. Usually a repeat-until loop.

REPEAT
Code that repeats
UNTIL *condition*

The program runs the code that repeats. The program then tests the condition. If the condition is false the code that repeats runs again.

Pre-condition: the ... a condition is true. Th... tested before the loop ... a while loop.

WHILE *condition*
Code that repeats
ENDWHILE

The program checks t... If the condition is true ... repeats runs. Then th... checked again. The co... may never run.

P6: C... soft...	
P7: C... soft...	
P8: U... impl... itera...	
P10: ... and ...	

PROGRAMMING TECHNIQ

Encapsulation

All data and processes are contained within one class/module. Data can only be accessed and manipulated using that class/module.

Module: a self-contained section of the program that has its own subroutines and data.

Procedure: a self-contained section of code that c... called from other parts of the program or oth... g... The code executes and **does** ... retur... a ... to ... de that called it.

Library: pre-written, ... d compiled subroutines. Can be imported and cal... ...ner programs.

Function: a self-contained section of code that can be called from other parts of the program or other programs. The code executes and **returns one or more** values to the program that called it.

Parameter: a value that is sent to a subroutine and can then be used within the subroutine.
Passing **by reference** (byref): any changes to the variable will also change the original value.
Passing **by value** (byval): creates a copy of the variable, changes are not applied to the original.

Class: a set of **properties** (data) and **methods** (functions and procedures) about a specific object (an instance of a real-life item). A class is a template for an object. Multiple instances of the class can be declared; they all have the same properties but with different values.

Getter: a function method in a class. This returns a property from the class.

Setter: a function/procedure method in a c... h... assi... or changes a property in t... The ... p... vate properties where the ... be ... se... and changed directly – only through ... d setters.

File manipulation

External files store data so it remains after a program ends.

To **read** data from a fil... e is ... ed.
The data is ther... ead ... ither ... ore or one line at a time).
The fi... ...ed.

```
...hel... ...open("Filename.txt", "r")
... NOT EOF
    dataRead[x] = theFile.readLine()
    x++
endwhile
theFile.close()
```

Data ...

Stack: a data structure where the last data item added is the first to be accessed (LIFO – last in, first out). Items are added to the **top** (end) of the stack. Items are accessed from the **top** of the stack.

- A **pointer** stores the position of the top of the stack.
- A function **push()** stores data on the stack.
- A function **pop()** accesses data from the stack.

Queue: a data structure where data is accessed in the order it is added (FIFO – first in, first out). Items are added to the **tail** (end) of the queue. Items are accessed from the **head** (start) of the queue.

- A **pointer** stores the position of the head and another **pointer** stores the position of the tail.
- A function **enqueue()** stores data on t... ...ueue.
- A function **dequeue()** acce... ...or ... he queue.

Linked li... ... da... ...stores a pointer to the next data item.
D... ...n o... ...essed by moving through the linked list starting from ... ...ead node).

Y...

Pass	
P6: Create a user interface for the software solution.	**M4: Use** programm... appropriate file m...
	M5: Use programm... appropriate data s...
P9: Use source code comments, indentation and version control to make the software solution maintainable.	
P10: Use appropriate naming conventions and data types in the software solution.	

PROGRAMMING TECHNIQ

Constructs

User input – taking data from the user, often by keyboard, e.g. `INPUT number`

Output to file – storing data in an external file (see file handling)

File input – reading data from an external file (see file handling)

Output to user – giving data to th... user, ... h... screen but co... al... audio or m... ove... g an actu... e.g. `OUTPUT ("Hello")`

... pr... ure or module – passing data to another part of a program as a parameter

Error handling

Try – except: stops runtime errors from crashing the program. If a runtime error occurs, the except code runs instead.

The program runs the code within the try.
The except has code to run if the error does happen.

e.g.

```
try
  theFile = open("File.txt")
  while NOT EOF
    print(theFile.readLine())
  endwhile
except
  print("Error reading from file")
endtry
```

Validation: checks that data is reasonable, without bou... and acceptable.

Does not check data is correct.

Validation	Des...
Presence	Chec... as not been left blank
Length	Check... r of characters
Range	Checks ...eric data is within two values
Format	Checks that data has specific characters, e.g. a postcode has the correct number of letters and numbers
Existence or lookup	Checks that data is the same as previously stored data, e.g. data entered must be either Yes or No

Searchi...: ch... ...n it... in a data s...uctu... find a specific value. ...t searching algorithms work in different ways. Two examples are linear and binary.

Line... chec... elem... time... with...

Binary search: search a set of ordered data. Compare the mi... If the data is less than the middle value, repeat with the first e... If the data is greater than the middle value, repeat with the e... the last element.

Technical skills

Development environment

Used to write, test and run the program.

Editor

Allows you to write the code. Provides other features, e.g.:

- **Autocomplete** – detects the command you are typing and fills in th...
- **Keyword highlighting** – changes the colour of a language's keywor...
- **Syntax checking** – checks the syntax as you are writing and highligh... code that is syntactically incorrect
- **Autocorrect** – detects erro... ...anges them for you, e.g. missp...

Debuggi... to...s

...elp ... ...nd errors to correct them.

- **...reakpoint** – set a point in the code where the code will stop running. You can then check the flow of the program and variable contents.
- **Memory inspector** – view the content of the variables and data structures as the program runs, or when you stop it using a breakpoint.
- **Stepping** – run one line of code then pause until you tell the program to run the next. Allows you to check the flow of the program and the content of data structures.

Runtime environment

Lets you run the program. Takes input from the user and produces the output for the user.

TEST SOFTWARE SOLUT

Requirements testing:
Test every measurable requirement that was identified in the analysis and design stage.
Which requirements have been met? Which have been partially met? Which have not been met?

Component testing:
Test each individual component part of the system. These may be the modules/classes/subroutines that were developed, or the separate parts of the pro ram

Test plan: A document that identi
identifies the different elements tha
e test data that needs to be
nd utputs that will all be tested a

Integrati g:
Ind ne a system that
est are combined and then
ogether.
Subroutines may work in isolation but may not work in combination.

Iterative:
you make a
Test with di
small chang

Testing types

System testing:
Testing the system as a whole once all parts are together.
Testing it in different ways, different run-throughs, performing actions in different orders with different data.

Elements to test?

Inputs – what data needs to be input? What range of data can be input? What are the bounds of acceptable data? What does invalid data do?

Outputs – what outputs should be produced? What forma are they in? How should ts differ
de nd the puts?

Navigation – how does a user move throu W order can they move through different menus and options? Does each option/button/i ta e user to the correct outcome? Can the system be exited?

Error handling – are errors detected and handled without crashing the program? What happens when invalid data is entered or an invalid option is selected? Are error messages appropriate and meaningful?

Data storage – is all data stored correctly? Is it in the correct location/file? Is the data in the correct format? Is data read from the correct locations? Is the data split or processed correctly when it is accessed?

REVIEW AND IMPROVE SOFTWAR

Client requirements

- Which of the original requirements have been met? Where is the evidence that they have been met?
- Which are partially met?
- Which are not met?

Functional requirements

- Which of the original functional requirements have been met? What is the evidence that they have been met? Are all inputs, process and outputs as required?
- Which are partially met?
- Which have not been met?

Review fitness for purpose

Robustness

- How robust is the system? What has been put in place to increase the robustness? Where is the evidence of this and that it works, e.g. exception handling?
- Which areas could be improved?

Non-functional requirements

- Which of the non-functional requirements have been met? What is the evidence that they have been met?
- How has the level of success been measured? What evidence is there to support this?
- Which have been partially met?
- Which have not been met?

Maintainability

- How maintainable is the system? What has been put in place to support the maintainability, e.g. documentation, comments?
- Where is the evidence of this maintainability?
- What areas are not as maintainable as preferred? What could be improved?

Code efficiency – how could you improve the code? Could be the memory consumption. Include the constructs and data structures. Are there more efficient ways to perform the tasks?

HCI accessibility principles – what accessibility principles have been used in the design of the interface? How could these be improved or enhanced? What other accessibility principles could be included?

Portability of software solution
How could the system be changed to ensure that it will work on different software platforms and different hardware?

Code reusability
How can the system be changed to make elements of the code reusable, e.g. through the inclusion of additional classes and subroutines?

Further development opportunities

Pass
P12: Assess the suitability of the software solution for meeting the requirements.

ACTIVITY POSTER

FUNDAMENTALS OF SOFTWARE D

Software design principles

Stepwise refinement

Abstraction

The removal of unnecessary elements fro[illegible] and the identification of vital elements in [illegible]

- Functional –
- [illegible]
- Control –

Decomposition

Object-oriented programming (OOP)

Maintainability

Encapsulation

Modularity

Fu[illegible]

Complete the des[illegible] design principle[illegible] types of progra[illegible]

ACTIVITY POSTER

DESIGN SOFTWARE SOLU

Software Design Specifications (SDS)

Documentation that describes everything the software needs to do.

The **format** can be written documentation, an interactive document (e.g. HTML p... or ...les c...inked documents.

Layout needs to be clearly structured, understandable and easy to navi...t... e.g. ink... within the document to jump to key ele...

Templates exist that are set up with the structure and hea... ... pe... ... complete.

Describe ... ...s to ... ...tem. ▶

Client

Organisational policies

Technical requirements

Language –

Industry standards

Solution overview

Client requirements

Functional requirements

Non-fu... ...ements

Constraints
What restrictions are there or limits on what the system has to do or where it has to operate?

Com... the ...

Tim...

Har...

Aud...

Bud...

Ext...

ACTIVITY POSTER

SOFTWARE DESIGN DOCUMENT

Data structure diagram
How is the system broken down? What are the sub-problems?

Data flow diagram
What data goes into the system? Where does this come from? W… does it go? Where is … stored? Where is d… accessed from?

… (context)
System in the middle, external entities

Level 1
Expands the system to include the processes and how these interact

Draw an example of each of th…

Architectural design
- External components of the syste… syst… hat will interact with the …
- …po… nt-level design (What are … …s/modules/subroutines within … What do these do?)
- Component interfaces (Which part… need to communicate, and by wha… communicate? How is data transfe…
- Module/component interactions (… different parts of the system comm… do they communicate?)

Algorithm designs
- What **inputs** will be put into the system? From the us…
- What **processes** will be carried out? Calculations? Op…
- What **storage** is required? External files? Database? H… structured? How are they accessed?
- What **outputs** will be produced? How are these creat… format? When are they created?

Flowchart
- Design the algorithms behind the system, the processes.
- Shows the inputs, processes and outputs at each stage.

Pse…
'Fak… to pl… work… outp… synt…

P3: … inte… solu…

P4: … algo… solu…

ACTIVITY POSTER

PROGRAMMING TECHNIQ

Variables and constants

Variable:

Constant:

Identifier naming conventions: the name given to variables / modules / ... structures.

Identifiers must be meaningful, e.g. say what they are stori... ...l stat... ...at is being stored, `xyz` does not state what is being st...

Kebab case:

Camel case:

Data types

Data type	Description	E...
Integer		
D...ating ... real / ... double)		
String		
Boolean		

Casting: converting from one data type to another. Use a function that casts, e.g. here the function `int()` converts the string `"123"` into the integer `123`

```
theNumber = int("123")
```

C... d... e... diff... and...

Write examples of each programming construct.

Selection

A condition is checked. The result of the check determines which code runs (and doesn't run).

IF-THEN-ELSE

ELSEIF

SWITCH-CASE

Iteration

Code runs repeatedly either a set number of times or based on a condition.

Fixed loop:

Pre-condition:

Post-condition:

P6: C... soft...

P7: C... soft...

P8: U... impl... iter...

P10: U... and ...

ACTIVITY POSTER

PROGRAMMING TECHNIQ

Encapsulation

All data and processes are contained within one class/module. Data can only be accessed and manipulated using that class/module.

Module:

Procedure:

Library:

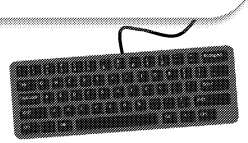
Function:

Parameter:

Class:

Getter:

Setter:



File manipulation

External files store data so it remains after a program ends.

Reading from a file

Data

Stack:

Queue:

Linked lis

Pass

P6: Create a user interface for the software solution.	**M4: Use** programm appropriate file ma
	M5: Use programm appropriate data st
P9: Use source code comments, indentation and version control to make the software solution maintainable.	
P10: Use appropriate naming conventions and data types in the software solution.	

ACTIVITY POSTER

PROGRAMMING TECHNIQ

Constructs

User input

Output to file

File input

Output to user

pr ure or module

Types of algorithm

Searchi

ch h it n in a data s uctu find a specific value. D t searching algorithms work in different ways. Two examples are linear and binary.

Lin

Binary search:

Another searchin

Technical skills

Development environment

Complete th description example the cons

Editor

Debuggi ta s

Runtime environment

Error handling

Validation: checks that data is reasonable, without boun and acceptable.

Does not check data is corr

Validation	Des ti
Presence	
Length	
Range	
Format	
Existence or lookup	

ACTIVITY POSTER

TEST SOFTWARE SOLUT

Requirements testing:

Component testing:

Test plan:

Integration testi

Iterative:

◀ Complete the descriptions of testing types and methods. ▶

Testing types

System testing:

Write how you will test these elements in your system. ▼

Elements to test?

Inputs

Outputs

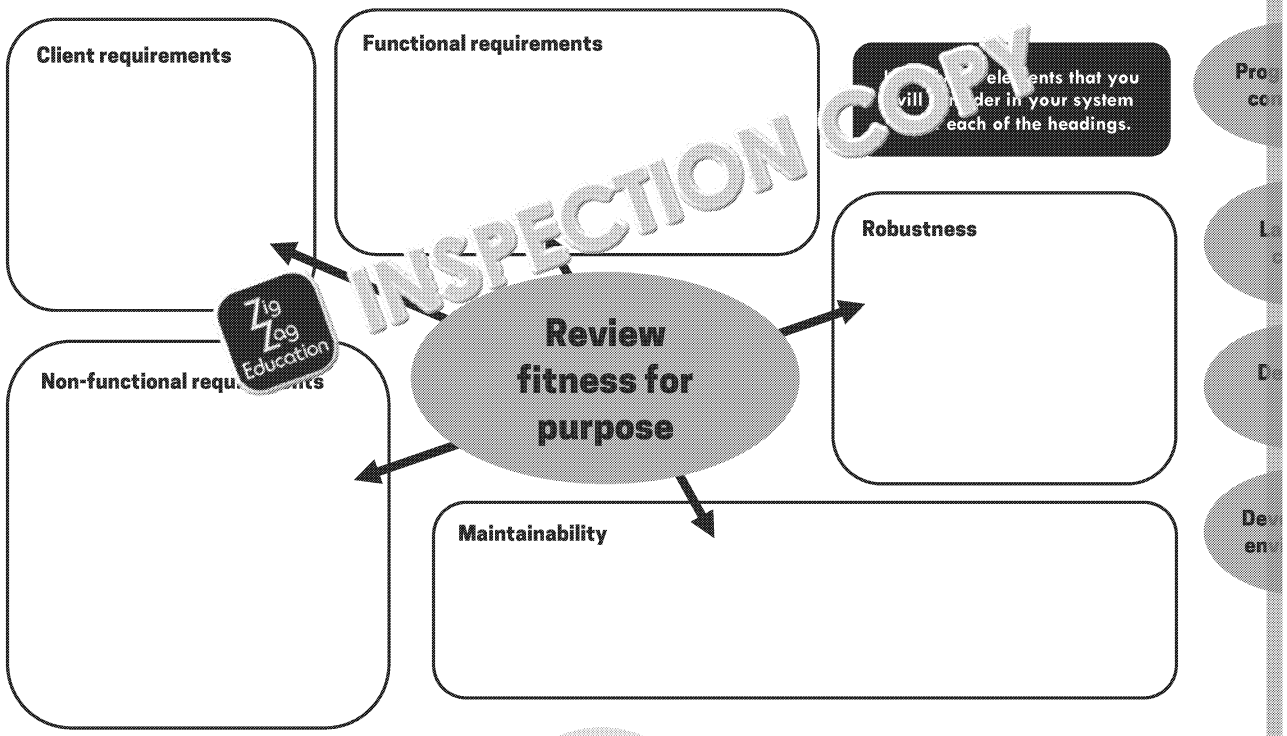
Navigation

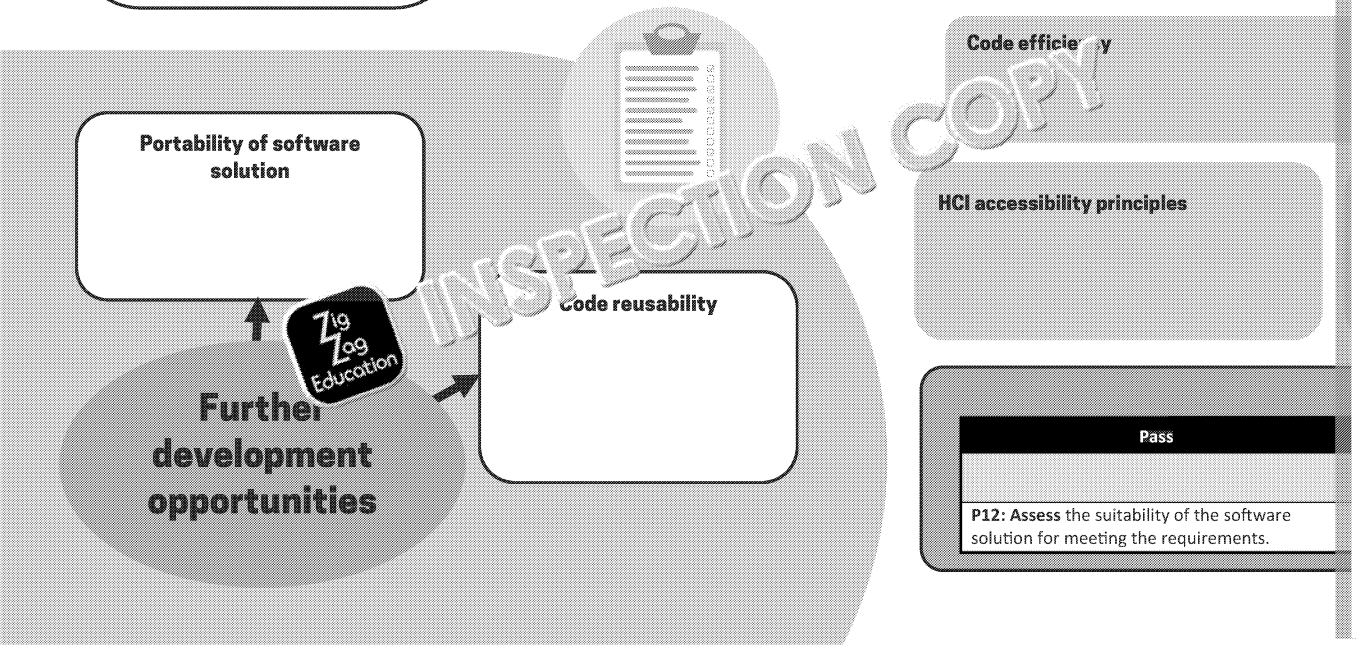
Data handling

Data storage

ACTIVITY POSTER

REVIEW AND IMPROVE SOFTWAR





Pass
P12: Assess the suitability of the software solution for meeting the requirements.