Defensive design considerations:

Input validation = Check input data entered by the user meets the predefined rules/criteria before processing which helps prevent errors and ensures the program runs as expected

Input validation techniques:

- Type check (make sure it's in the correct data type)
- Range check (make sure input is in the correct range)
- Presence check (if the user has entered anything)
- Length check (input has correct number of characters required)
- Format check

Using input validation techniques allows programmers to make their program:

- More robust
- More user friendly
- Prevent further errors occurring later in the algorithm

Anticipating misuse:

- Division by 0
- Communication error

    - Connection may drop between client and host server
- Printer and other peripheral errors
    - Printer may have a jam
    - Programmer should always assume the print was not successful and give options to reprint
- Disk errors
    - Handle exceptions:
        - File/folder not found
        - Disk out of space
        - Data in file is corrupt
        - End of file reached

Authentication: Process of identifying the identity of a user or system to ensure that they can be giving access

To keep

- Username and passwords to access systems
- Encryption of data files

Use reCAPTCHA to verify it's a human

Programmers should be aware of SQL injection and hacks

<u>Maintainability:</u>

Use comments to:

- Explain purpose of program
- Explain sections of code
- Visually divide sections of program

Use whitespace to make it easier to see

Use indentation for every selection and iteration branch

Use descriptive identifiers

Use subroutines to:

- Structure code
- Eliminate duplicated code
- **In exam say put code into subroutine**

Constants should be declared at the top of the program

Constants:

- Location in memory
- Value cannot be changed

Variables:

- Memory location
- Temporarily stores data
- Value can be changed

<u>Syntax and logic errors:</u>

Syntax – The compiler/interpreter doesn't understand something because it doesn't follow the rules of the language (the rules/grammar of the language are broken so the program doesn't run)

Logic – The compiler/interpreter is able to run the program but it gives an unexpected output (the program run but it gives an unexpected output)

The purpose and types of testing:

Programs should be tested:

- Ensure there are no errors in the code
- Ensure program has acceptable performance and usability
- Ensure unauthorised access is prevented
- Check program meets requirements

Iterative testing:

- Program is tested while it is being developed
- Each new module is tested as it is written

- Any errors found in modules will be fixed and then retested

Final testing/terminal testing:

- Program is tested at the end of the development process
- Tests all modules work together (integration testing)
- Testing program produces required results

Normal data – Data which user is likely to input into the program and should be accepted

Boundary data – Data entered is at the limit of the accepted validation boundaries but should be accepted

Invalid data – Inputs with the correct datatype outside the accepted validation checks that should be rejected

Erroneous data – Data is in incorrect datatype and should be rejected

Refining algorithms:

Making programs more robust:

- Code which anticipates a range of possible invalid inputs
- Ensuring incorrect data doesn't crash the program
- Prompts should be user friendly
- Making sure only correct datatypes are entered

You can use exception handling