








Syllabus Content:

4.4.1 Software development resources •

-  show understanding of the possible role of program generators and program libraries in the development process

4.4.2 Testing •

-  show awareness of why errors occur
-  show understanding of how testing can expose possible errors
-  appreciate the significance of testing throughout software development
-  show understanding of the methods of testing available: dry run, walkthrough, white-box, black-box, integration, alpha, beta, acceptance
-  show understanding of the need for a test strategy and test plan and their likely contents
-  choose appropriate test data (normal, abnormal and extreme/boundary) for a test plan

4.4.1: Program generators and program libraries

The first computers had to be programmed using machine code. This is a very tedious method for writing programs. Assemblers were invented to generate a computer program in machine code from assembly code instructions.

Later interpreters and compilers were invented to generate low-level code from high-level programs written by people. So you can see that program generators have been around for a very long time.

Development is ongoing to invent program generators that will take ever more abstract models and translate them into executable code.

Program generators

An integrated development environment (IDE) for a modern high-level language provides facilities for software development, such as a source code editor with intelligent code completion, build automation and a debugger. Some IDEs have more advanced forms of code generation. For example, programmers can design GUIs interactively or generate code from a wizard or template. Computer-aided software engineering (CASE) tools are also used to generate code.

Program libraries






In P2 we covered built-in functions. These are part of a program library

Program generator: a computer program that can be used to create other computer programs

Program library: a collection of pre-compiled routines or modules that a program can use

Why errors occur and how to find them:

Software may not perform as expected for a number of reasons:

-  The programmer has made a coding mistake.
-  The requirement specification was not drawn up correctly.
-  The software designer has made a design error.
-  The user interface is poorly designed and the user makes mistakes.
-  Computer hardware experiences failure.

How are errors found? The end user may report an error. This is not good for the reputation of the software developer. Testing software before it is released for general use is essential. Research has shown that the earlier an error can be found, the cheaper it is to fix it. It is very important that software is tested throughout its development. The purpose of testing is to discover errors.

Testing methods

We covered logic errors and run-time errors. We discussed black-box and white-box testing. We used debugging facilities in an IDE. We worked through program code by dry-running it and recording the steps in a trace table. Dry-running program code is also sometimes referred to as a 'walkthrough'.

These testing methods are used early on in software development, for example when individual modules are written. Sometimes programmers themselves use these testing methods. In larger software development organisations, separate software testers will be employed.

Integration testing:

Software often consists of many modules, sometimes written by different programmers. Each individual module may have passed all the tests, but when modules are joined together into one program, it is vital that the whole program is tested. This is known as integration testing. Integration testing is usually done incrementally. This means that a module at a time is added and further testing is carried out before the next module is added.

Alpha testing:

Software will be tested in-house by software testers before being released to customers. This type of testing is called alpha testing.

Acceptance testing:

Bespoke software (written for a specific customer), or **tailor-made software**, will then be released to the customer. The customer will check that it meets their requirements and works as expected. This stage is referred to as acceptance testing. It is generally part of the hand-over process. On successful acceptance testing, the customer will sign off the software.

Beta testing:

When software is not bespoke but produced for general sale, there is no specific customer to perform acceptance testing and sign off the software.

So, after alpha testing, a version is released to a limited audience of potential users, known as 'beta testers'. These beta testers will use the software and test it in their own environments. This early release version is called a beta version and the chosen users perform beta testing. During beta testing, the users will feed back to the software house any problems they have found, so that the software house can correct any reported faults.

KEY TERMS

Integration testing: individually tested modules are joined into one program and tested to ensure the modules interact correctly

Alpha testing: testing of software in-house by dedicated testers






Acceptance testing: testing of software by customers before sign-off

Beta testing: testing of software by a limited number of chosen users before general release

Test plans and test data

During the design stage of a software project, a suitable testing strategy must be worked out to ensure rigorous testing of the software from the very beginning. Consideration should be given to which testing methods are appropriate for the project in question. A carefully designed test plan has to be produced.

It is important to recognise that large programs cannot be exhaustively tested but it is important that systematic testing finds as many errors as possible. We therefore need a test plan. In the first instance, an outline plan is designed, for example:

-  Flow of control: does the user get appropriate choices and does the chosen option go to the correct module?
-  Validation of input: has all data been entered into the system correctly?
-  Do loops and decisions perform correctly?
-  Is data saved into the correct files?
-  Does the system produce the correct results?

Type of test data	Explanation
Normal (valid)	Typical data values that are valid
Abnormal (erroneous)	Data values that the system should not accept
Boundary (extreme)	Data values that are at a boundary or an extreme end of the range of normal data; test data should include values just within the boundary (that is, valid data) and just outside the boundary (that is, invalid data)

References:

-  AS & A level Course Book by Sylvia Langfield & Dave Duddell