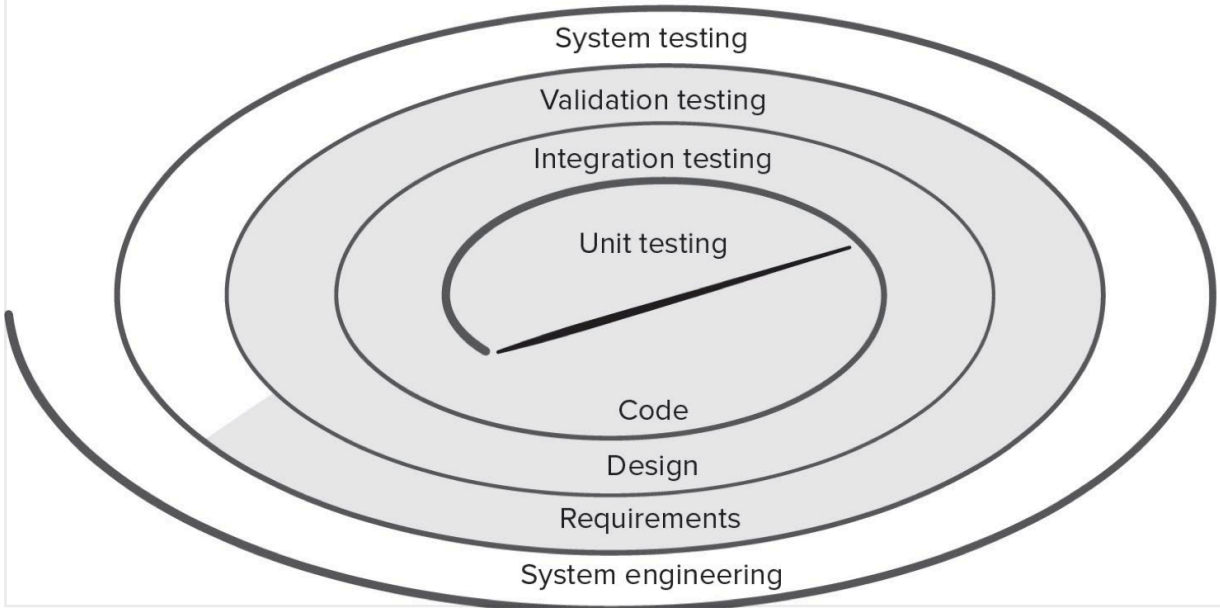
10 Software Component Testing

- Strategic Approach to Testing
  - Conduct effective technical reviews as this can eliminate many errors before testing begins.
  - Testing begins at the component level and works “outward” toward the integration of the entire system.
  - Different testing techniques are appropriate for different software engineering approaches and at different points in time.

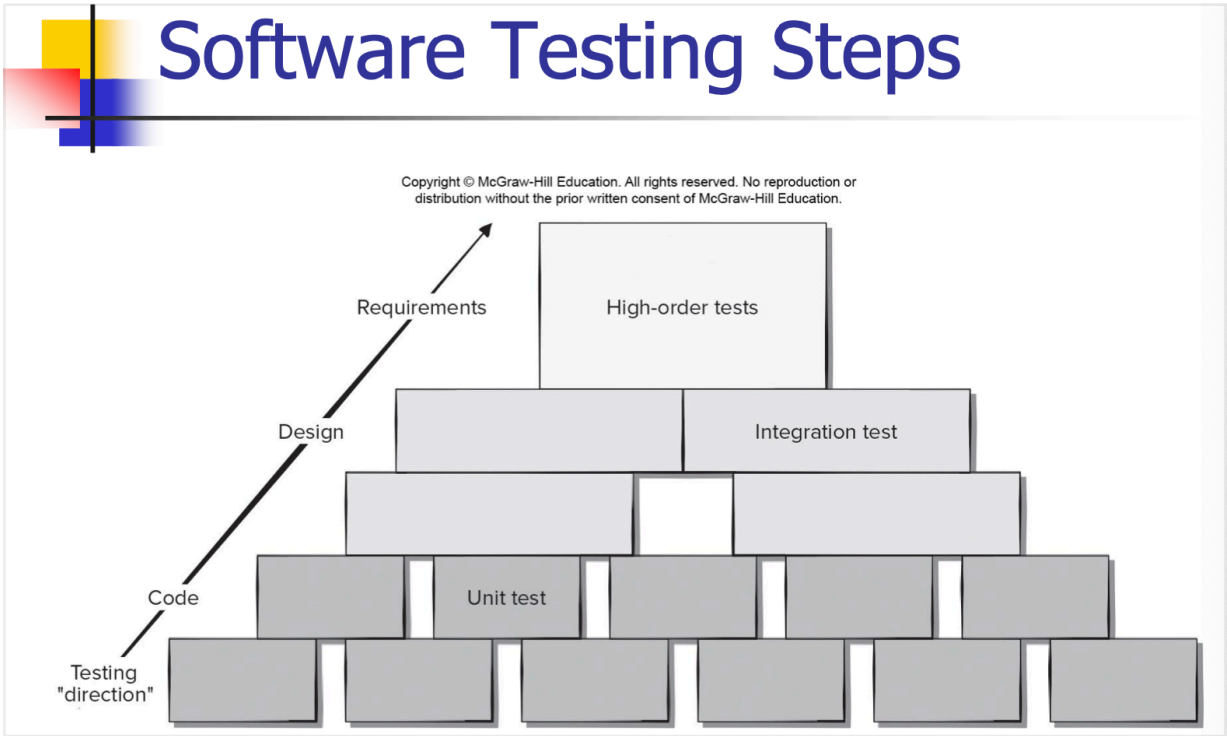
- Verification and validation:
  - Verification refers to the set of tasks that ensure that software correctly implements a specific function, are we building the product right?
  - Validation refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements, are we building the right product?

- Organizing for Testing:
  - Software developers are always responsible for testing individual program components and ensuring that each performs its designed function or behaviour.
  - Only after the software architecture is complete does an independent test group become involved.
  - The role of an independent test group(ITG) is to remove the inherent problems associated with letting the builder test the thing that has been built
  - ITG personnel are paid to find errors.

Testing Strategy



- Unit testing begins at the centre of the spiral and concentrates on each unit as they are implemented in source code.
- Testing progresses to integration testing, where the focus is on design and the construction of the forward architecture. Taking another turn outward on the spiral
- Validation testing, is where requirements established as part of requirements modelling are validated against the software that has been constructed.
- In system testing, the software and other system elements are tested as a whole



Criteria for Done:

- You're never done testing; the burden simply shifts from the software engineer to the end user (wrong)
- You're done testing when you run out of time or you run out of money. (Wrong)
- The statistical quality assurance approach suggests executing tests derived from a statistical sample of all possible program executions by all targeted users.
- By collecting metrics during software testing and making use of existing statistical models, it is possible to develop meaningful guidelines for answering the question: "when are we done testing?"

Test Planning:

1. Specify product requirements in a quantifiable manner long before testing commences.
2. State testing objectives explicitly.
3. Understand the users of the software and develop a profile or each user category.
4. Develop a testing plan that emphasizes "rapid cycle testing."
5. Build "robust" software that is designed to test itself.
6. Use effective technical reviews as a filter prior to testing.
7. Conduct technical reviews to assess the test strategy and test cases themselves.

8. Develop a continuous improvement approach for the testing process.

Role of Scaffolding:

- Components are not stand alone program some type of scaffolding is required to create a testing framework.
- As part of this framework, driver and/or stub software must often be developed for each unit test.
- A driver is nothing more than a ‘main program’ that accepts test-case data, passes such data to the component, and prints relevant results.
- Stubs (dummy subprogram) serve to replace modules invoked by the component to be tested.

Cost effective testing:

- Exhaustive testing requires every possible combination and ordering of input values be processed by the test component
- The return on exhaustive testing is often not worth the effort, since testing alone cannot be used to prove a component is correctly implemented.

Test case design: Design unit test cases before you develop code for a component to ensure that code that will pass the tests.

Error Handling: A good design anticipates error conditions and establishes error handling paths which must be tested.
Among the potential errors that should be tested when error handling is evaluated are:

- Error description is unintelligible
- Error noted does not correspond to error encountered.

White Box testing: you can derive test cases that

- Guarantee that all independent paths within a module have been exercised at least once
- Exercise all logical decisions on their true and false sides
- Execute all loops at their boundaries and within their operational bounds
- Exercise internal data structures to ensure their validity

Cyclomatic complexity:
$V(G) = E - N + 2$

Black Box Testing: testing attempts to find errors in the following categories

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access

- Behaviour or performance errors.

Unlike white box testing, which is performed early in the testing process, black box testing tends to be applied during later stages of testing.

Black Box - Interface Testing:
-Is used to check that a program component accepts information passed to it in the proper order and data types and returns information in proper order and data format.
-Components are not stand alone programs testing interfaces requires the use stubs and drivers.
-Stubs and drivers sometimes incorporate test cases to be passed to the component or accessed by the component
-Debugging code may need to be inserted inside the component to check that data passed was received correctly.

Object Oriented Testing:
To adequately test OO systems three things must be done

1. Definition of testing must be broaden to include error discovery techniques applied to object-oriented analysis and design models
2. Strategy for unit and integration testing must change significantly
3. Design of test cases must account for the unique characteristics of OO software
4.