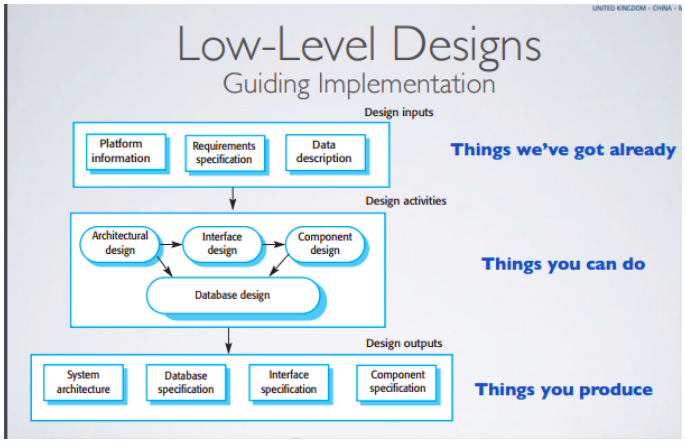
The purpose of models

- Derive the requirements for a system
- During the design process to describe the system to engineers implementing the system
- After implementation to document the systems structure and operation

Low level designs

- *Architecture design*
  - identify the overall structure of the system, the principle components, their relations and how they are distributed.
- *Interface design* define
  - define the interfaces between system components. Specifications should be unambiguous, so that they can be build independently.
  - Designing APIs for exchanging information between components
- *Component Design*
  - Take each system component and design how it will operate. The design model may be used to automatically generate an implementation.
- *Database Design*
  - Design the system data structures and how these are to be represented in a database



Architecture Designs

Specifies all the different components you are going to build, how will they relate,connect. Is critical in explaining how the components fit into the big picture.

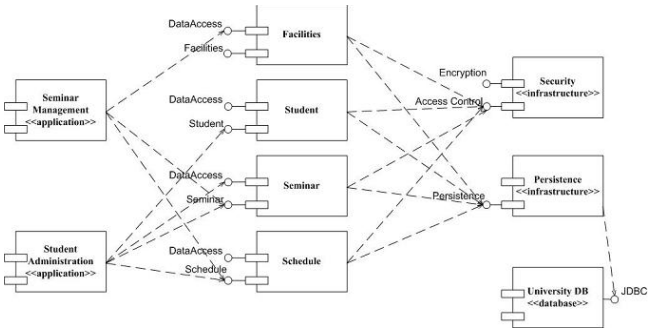
Interface Designs

Final prototypes help define the "User" interface (not just how it looks, how it interacts as well). It identifies all the other interfaces (between the mobile apps and the server,what API calls are needed, what type request or push? Etc.)

Sometimes interfaces are prescribed for you : authentication protocols or if an existing system already makes information available in a particular format etc.

Component Designs

- Class Diagrams
  - And lass Descriptions
- Class Hierarchy diagrams
  - Show relationships between similar classes
- State diagrams
  - Show different states of classes and cases of change
- Sequence diagrams
  - Showing low-level detail
  - Interactions between classes



http://agilemodeling.com/images/models/componentDiagramUML1.jpg

Test plans and Testing

Testing is designed to show that the software does what it is supposed to and discover program defects before it is put to use

Is a way of determining an end point. Used a lot in *test-driven* development. In test plan we have to :

- Design how it will work
- Specify exactly how functions will work
- You can therefore specify a test to check it
- Development test plans
  - Let us prove that a class functions correctly
    - who will do it
    - with what data
    - on which platform
- System/Integration test plans
  - Let us prove that the software meets the specifications
    - Test that cause one class to use another etc.
    - Tests that check that components interface correctly
- Acceptance test plans
  - Shows that the software meets the requirements
  - Often done with client – so they “accept” the software

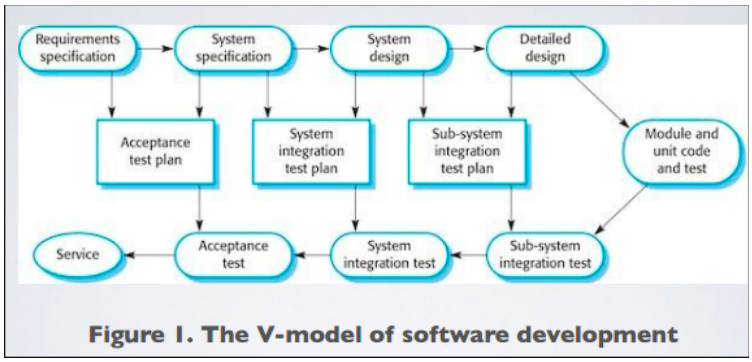


Figure 1. The V-model of software development

Reasons to build test plans

- Helps to define what counts as finished
  - How each stage back up the V is tested
  - Where when by who etc.
- Developers use Tps (Test plans) to test code before delivering it
- Managers using Tps can estimate testing workload and schedule it, which helps to manage budget
- Testing documentation servers as evidence to clients or as evidence of proper Software engineering.
- Testing process – description of approach taken.
- Requirements Traceability – links between reqs. And tests
- Tested Items – list of things to be tested.
- Testing schedule – schedule in relation to overall project
- Test recording procedures – how will the results will be recorded.
- Hardware/Software requirements – for testing machines
- Constraints – number of people,machines,etc. needed
- System tests – list of all the exact test cases that will be tested.

Each test case should include

- A statement of what is being tested
- Specifications of the inputs to the test
- What’s the expected output from the system is
- Specify the steps needed to carry out the test.

Validation Testing - tests that show the software produces the right answer - when you give it all types of correct data
Defect Testing - tests that show the software doesn't break - when you give it all types of incorrect data.

Writing tests for **Validation only**, is a common **mistake**

Acceptance test document

- Inputs and outputs are Use Cases
- The expected outcomes would be achieving Use Cases
- You can still document pass/fail/dates/reasons

Making effective test plans

- Focus on examples of correct data and examples of incorrect data
- Comprehensively cover all “types” of correct/incorrect data/actions
  - Test opposites of expected input (negative – positive,float -int etc.)
- Make tests that work and **create errors**

White vs Black box testing

- White box testing (transparent,glass,clearbox testing).
  - You know how the code is supposed to work. Your tests test for how it is supposed to work.
- Black box testing.
  - Tester doesn’t know how the code works, they just know what is supposed to be the output for an input. This is for release testing and user acceptance testing.