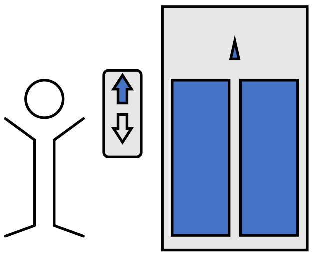
Sample Test-Planning Document

- *This is a skeleton sample of the sort of paper you could use to support LO2. It has quite a bit of commentary to explain what is going on – you don't have to explain the process.*
- *The purpose of this sample is to give you an idea of the sort of evidence you need to produce.*
- *The evidence does not need to be presented this format:*
  - *it could be spread across various files in a repository.*
  - *The evidence could be some subsection of another document produced for the project.*
  - *There is no prescriptive format you need to follow.*
  - *The assessment criteria for LO 2 want an indication that you can identify and use information about the requirements and tests you develop and evaluate a plan for the testing.*

Suppose we consider the development of the lift controller illustrated in the Chapter 2 slides:



Then we might consider two overall requirements. There are many other requirements a real lift system would involve but these are good enough to illustrate. The first is a safety property and the second is a requirement on a measurable attribute of the system.

- R1: It is not the case that the door on floor *k* is open, and the lift is not at floor *k*.
- R2: If the lift is in normal operating mode, then the mean time between pressing the call button on floor *k* and the lift arriving on floor *k* travelling in the correct direction is 20 seconds.

R1 and R2 are both system level requirements. It may be that lower-level requirements arise as you consider various aspects of the requirements. Here we will only consider R1 and R2 and avoid detail to keep the presentation easily readable. In real-life projects that are many requirements that are described in detail and managing the whole collection is a considerable task. You have a fixed amount of resource to devote to the whole collection of requirements so you should carefully monitor your use of resource throughout this process.

Priority and Pre-requisites

[At this stage the key resources for constructing this section are:

- Chapter 2 – the notions of validation and verification since you are aiming to be appropriately confident that your requirements are valid and have been verified. In addition the pyramid diagram with "optimistic inaccuracy" etc are useful.
- Chapter 3 – the six engineering principles are useful in helping to suggest the sorts of effort we want to take in our analysis and testing.]

Now consider each requirement in turn and provide a short assessment of it's A&T needs:

- R1: This is a safety requirement which probably has regulatory requirements associated with it. So:

        - This suggests we devote a reasonably high level of resource to ensuring we meet the requirement.
        - The principles of Chapter 3 indicate we should consider at least two different T&A approaches if the requirement has high priority.
        - Chapter 4 suggests that early detection of issues will lead to reduced requirement for T&A effort so we should think of early approaches to V&V.
        - The partition principle suggests that to help ensure safety we should decompose the requirement into the control component that directs the door to open at appropriate points, but this door open signal is protected by an *interlocking* that ensures R1 is maintained at each floor[1]. The interlocking will be very simple, and it will override any attempt to open the door when the lift is not at the relevant floor. There will be an interlocking for each floor so we describe a generic one that can be used for each floor. This is not a full specification of the sensors and actuators for a full interlocking it just eliminates the possibility that the door opens when the lift is not at the floor or is moving. We'll return to the lift when we consider models
        - We need to know what the inputs and outputs of the interlocking are:
            - Inputs:
                - Lift_moving: true if the lift is moving, false otherwise
                - At_floor: true if the lift is at this floor, false otherwise
                - Open_door: this is true if the controller is requesting that the door is open and false if the controller is requesting that the door is closed.
            - Outputs:
                - Door_control: if this is true the door is open or opening and false if the door is closed or closing
            - Specification:
                - Door_control is only true when Lift_moving is false, At_floor is true and Open_door is also true
        - The principles of CH03 of Y&P suggest we decompose the spec to get an interlocking spec and a controller specification and the interlocking compose with the controlled guarantees R1. Focussing on R1 we can see we get **two different tasks** we need to schedule in the plan:
            - Some sort of inspection of the interlocking to ensure the requirement is properly implemented by specific code and that there is no extraneous code that cannot be seen as implementing the requirement.
            - A later exhaustive test that checks every possible combination of inputs and checks the result conforms to the specification.
- R2: This is a lower priority requirement so we will probably devote less effort to this requirement. This is a measurable attribute of the code, so we need the means to measure this:

[1] Systems like the lift controller are usually implemented by a polling system where the controller "polls" each sensor in the system for its state, then calculates the next state and sends message to each of the actuators to adopt a particular condition and that is the new state. This is repeated and the whole system keeps sensing the environment and then changing to reflect what the sensors detected.

    - We can only really test this for the completed system so this is a system level test that probably occurs late.
    - There are validation and verification issues with this.
    - To verify we will need some sort of synthetic data and the means to run tests on this.
    - To validate (and verify) this will require some sort of logging of performance of the system
    - This suggests the **following tasks** need to be scheduled into the testing:
        - Generating synthetic data to test the controller.
        - Building some sort of scaffolding to simulate the hardware of the lift so early tests are possible on synthetic data.
        - Designing the logging system to capture real performance
        - Designing and implementing the analytics intended for the lift data
        - Feeding collected data back into the early testing

Scaffolding and Instrumentation

Here we describe what scaffolding and implementation are needed in order to carry out the given tasks (and this may give result in more tasks to build scaffolding and instrumentation). For our requirements:

- R1: the inspection will not require any scaffolding or instrumentation but if we plan a later exhaustive test (perhaps exercising the code many times) we will need to build scaffolding that presents randomly generated inputs and checks the output meets the spec, Building the scaffolding is another task to be scheduled.
- R2: this is more extensive we probably need:
    - Some sort of simulator for the hardware so it is possible to test the software. This is scaffolding and will need to be scheduled early.
    - Having data for the simulator may involve some effort that needs to be scheduled.
    - The system test will include combining the simulator and the controller and using synthetic data to observe its response
    - You will need to schedule designing the instrumentation to log activity in the lift and tools for analysis.

So, after the first two stages there will be some identification of tasks that need to be done. The final stage schedules them into a particular lifecycle.

Process and Risk

In this section you think about where to place the tasks you have identified in the lifecycle process. To do this you really need to estimate how long the task will take (we won't go into that here – it is a big topic). And what is necessary for the task to be possible: resources, code, data, results from earlier tasks. Considering R2 we can see that building the simulator and generating synthetic data should probably be done early and can be done concurrently the other activities come later closer to deployment and during operation.

Chapter 20 considers a range of risks and once you have positioned your test tasks withing your chosen lifecycle you need to consider risk related to the project design and scheduling. For example: in R2 there what if the synthetic data is unrepresentative – could that mean the design may perform poorly? How can we mitigate this – it is clearly a potentially serious issue BUT is it likely? I few believe it is sufficiently likely we may need to attempt some validation activity to give us confidence that we have a good simulator