1 - Analysis

NEA

Analysis

The first stage is to analyse the problem.

"What does the solution need?"

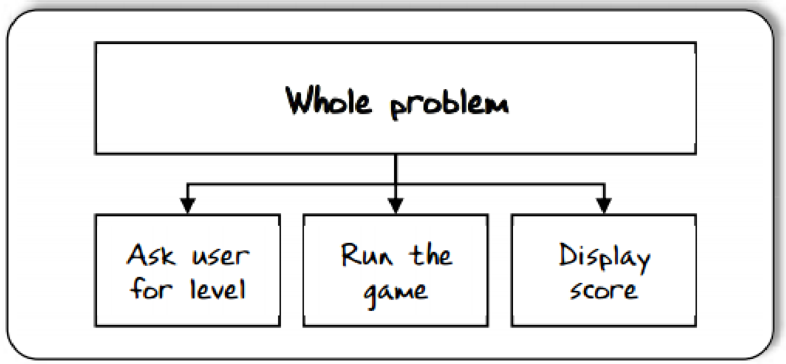
- Inputs?
- Processes?
- Outputs?
- Storage?

You could break each of these down further using "Who-What-When-Where-How" where appropriate.

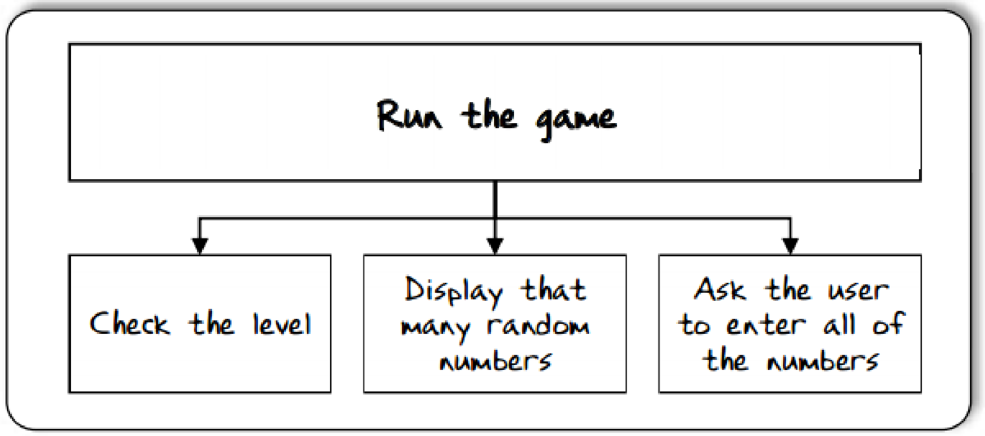
By doing this you can state your "Objectives".

Analysis – Decomposition

In the analysis section, you need to break the task into sub-tasks, a process known as problem decomposition. Using our memory game example, we might decompose the problem like this:



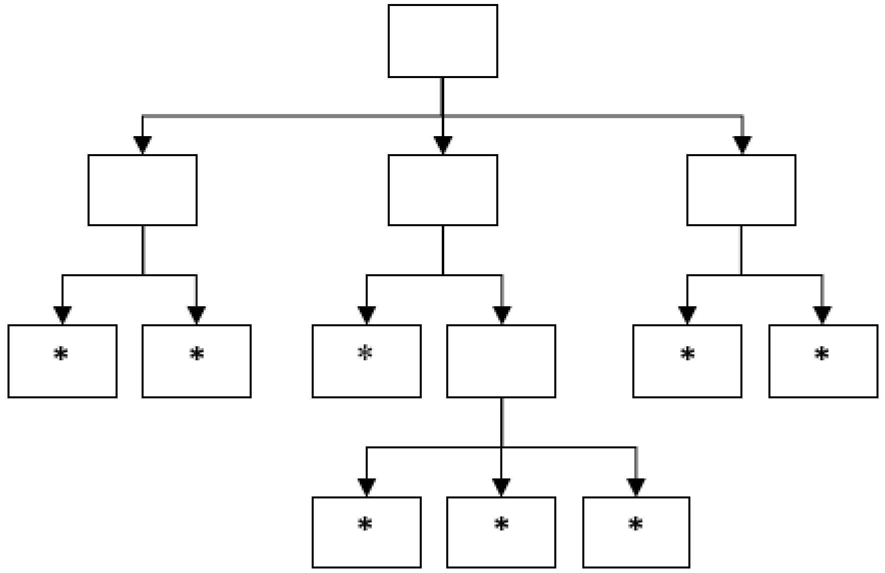
This is a top-level decomposition, but the job's not finished yet. Any one of these sub-tasks could be broken down into sub-tasks of their own. Let's take the 'run the game' part of the program


Run the game
Check the level
Display that many random numbers
Ask the user to enter all of the numbers

Analysis - Decomposition

Any sub-task that can be broken down further should be broken down further.

Once you have a series of tasks that are as decomposed as possible, you're ready to write your analysis, which could be in the form of a hierarchy diagram:

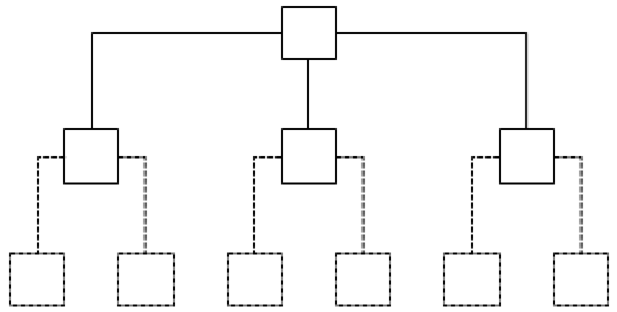


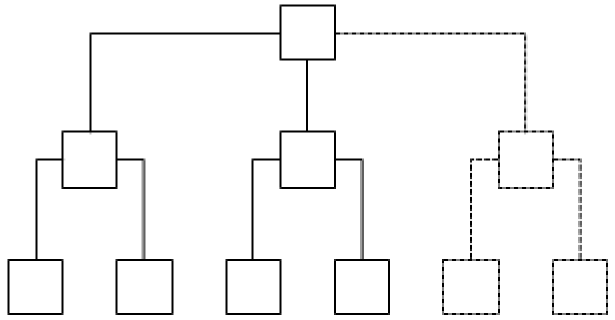
Any terminal node (indicated above with a * symbol) should contain a task that cannot be broken into smaller tasks. If you were to read these from left to right, you would read a detailed description of the program's tasks, in the order that they would happen.

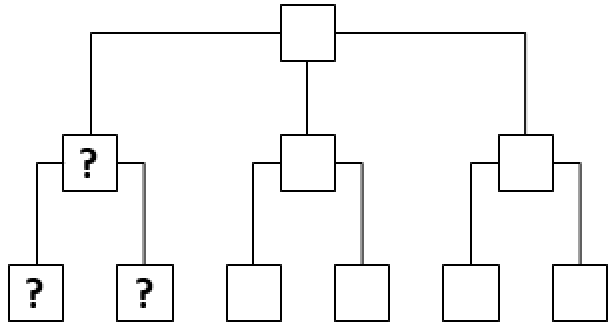
Analysis - Decomposition

In order to gain a mark from the middle band, you still need to break the problem down into pieces; you need to show that you understand the smaller pieces of the larger task that is being asked of you. Work from the middle band might demonstrate the following features:

The task is broken into pieces, but not broken down far enough:	Some tasks are broken into sub-tasks in a quite detailed way, but not all tasks have necessarily been identified:	Not all tasks identified are necessary for the problem you have been given:



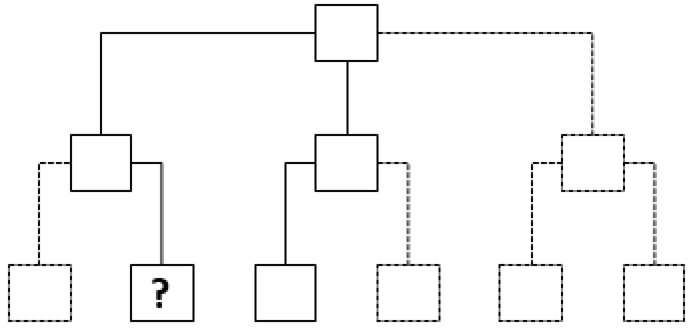




Analysis – Decomposition

‘Brief comments’ means you are describing either the problem or the solution, but not really making any progress in breaking it into sub-tasks.

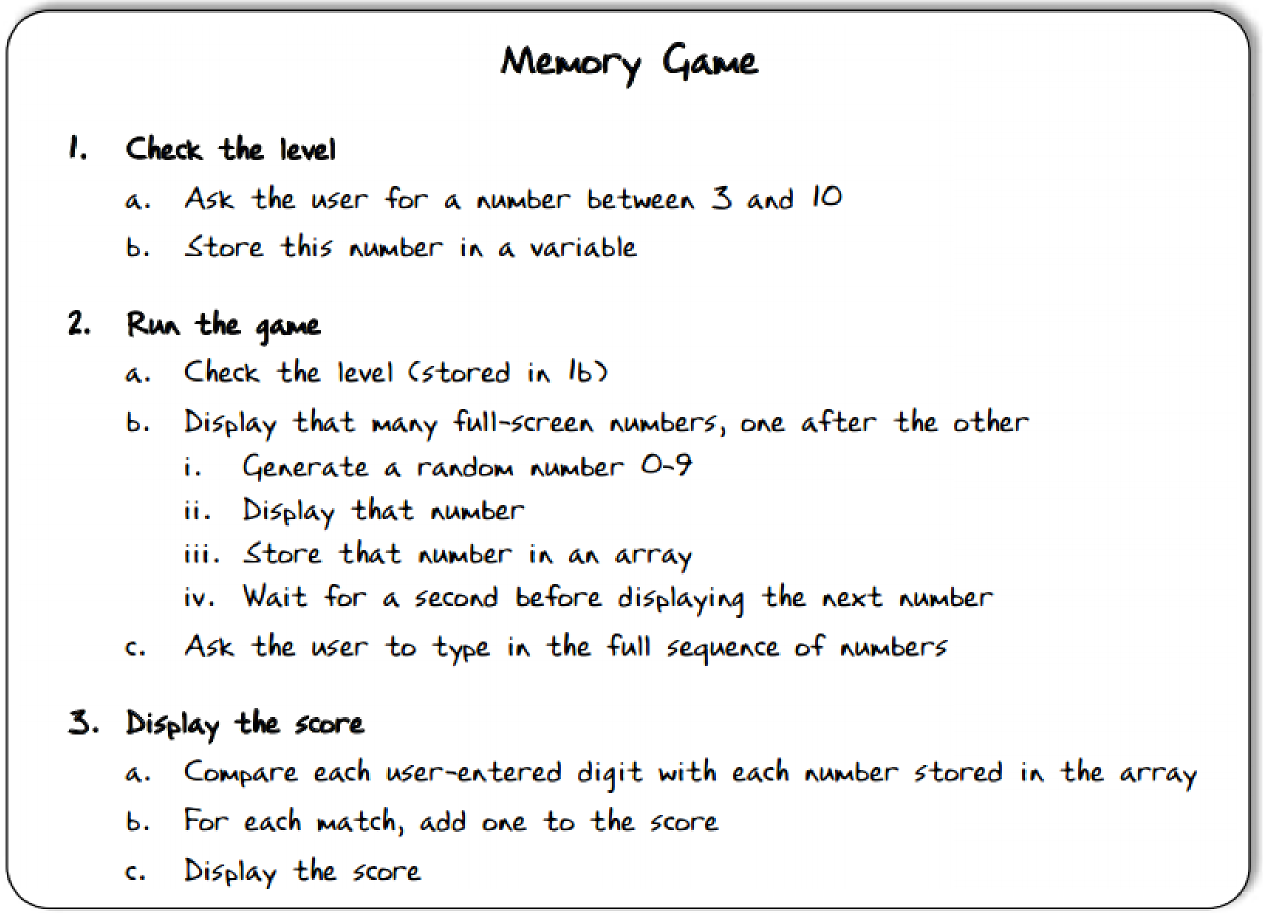
Typically, a piece of work in the bottom mark band demonstrates a combination of problems that exist in the middle band.



Analysis - Decomposition

An alternative to a hierarchy diagram is a series of nested bullet points, where each level of indentation signifies a task or sub-task that has been broken into smaller pieces.

Full marks are available by carrying out an analysis in this way; it's simply an alternative means of presentation:



Memory Game

1. Check the level
    a. Ask the user for a number between 3 and 10
    b. Store this number in a variable

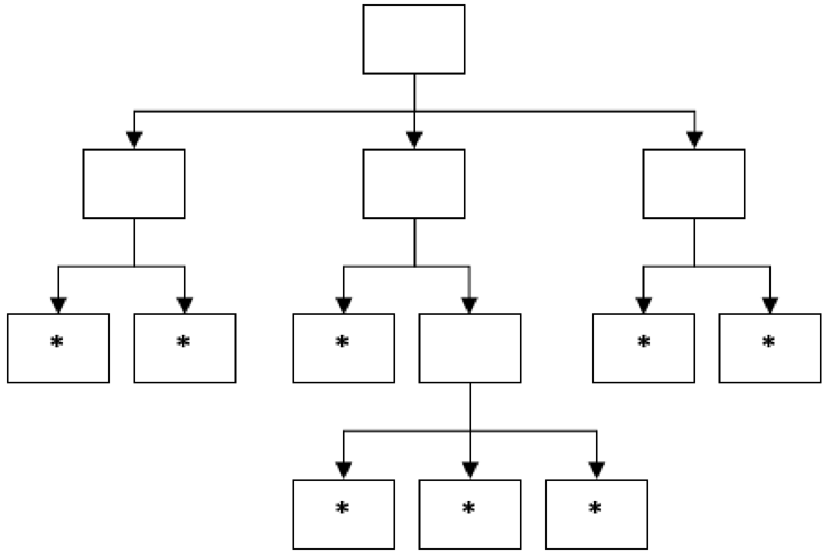
2. Run the game
    a. Check the level (stored in 1b)
    b. Display that many full-screen numbers, one after the other
        i. Generate a random number 0-9
        ii. Display that number
        iii. Store that number in an array
        iv. Wait for a second before displaying the next number
    c. Ask the user to type in the full sequence of numbers

3. Display the score
    a. Compare each user-entered digit with each number stored in the array
    b. For each match, add one to the score
    c. Display the score

Testing Plan

Testing does not take place until there is at least part of a program to test, but you are expected to know what you will be testing as part of the design.

This is the hierarchy diagram from the analysis section, and it makes a good template for designing your tests; each terminal node (*) indicates an area that should be tested, ideally separately from the rest of the program, for example:



“1. Ask user for level”

- You must test that the program asks for a number between 3 and 10
- You must test that that number is correctly stored in a variable

A good analysis section is a ready-made checklist for testing.

Testing Plan

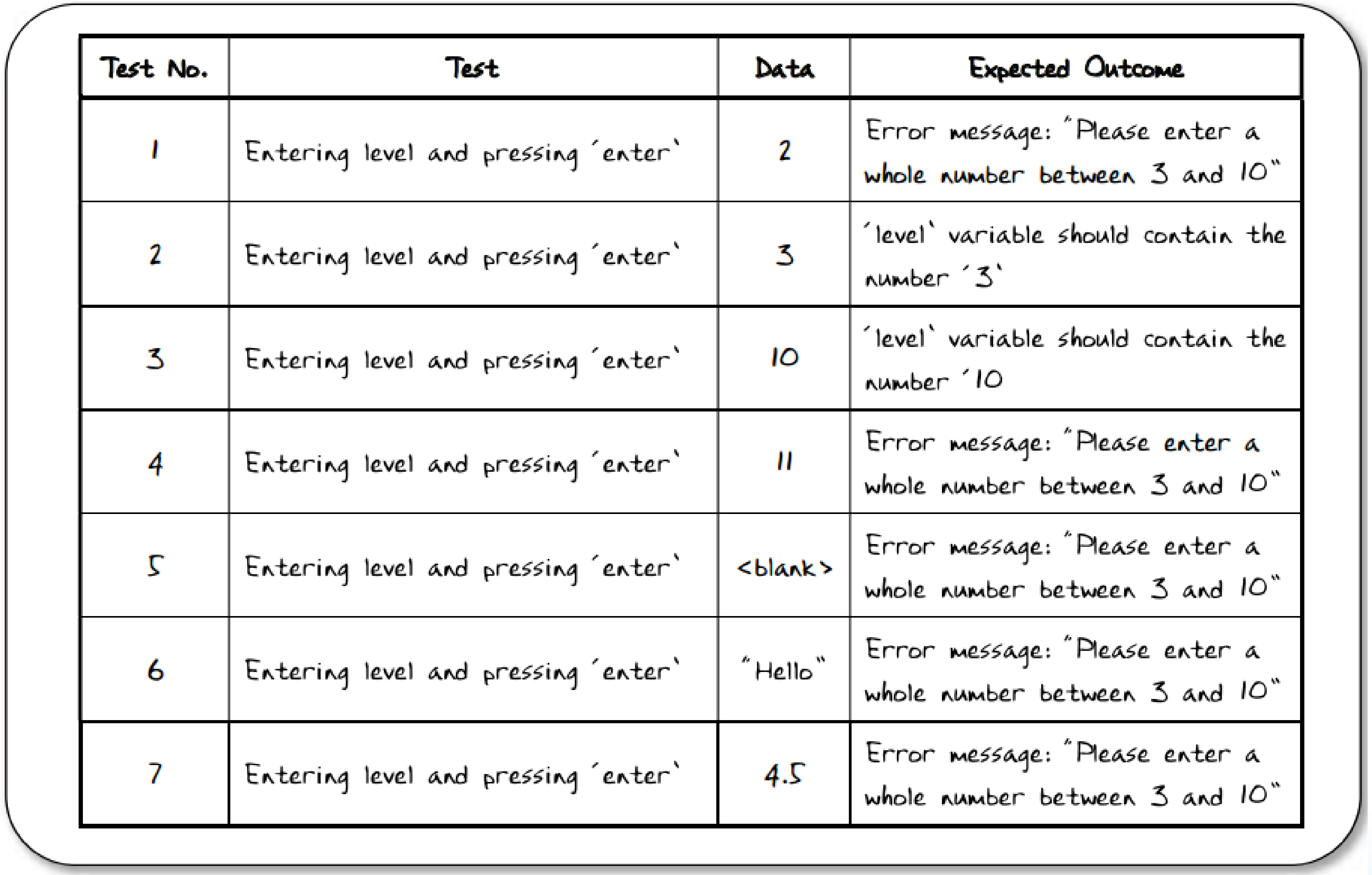
When documenting your planned tests, you'll need a table with four columns:

Test No.	Test	Data	Expected Outcome

- The 'test number' column simply labels each test 1, 2, 3, etc. so that you can refer back to them easily if you want to discuss individual tests in the evaluation.
- The 'test' column will contain a description of what exactly you are testing – the more detailed, the better. Ideally, someone else would be able to replicate your test by reading only this column and the data column.
- The 'data' column will contain the data that will be used in the test that you described in the 'test' column. If you were testing, say, a login form of a program, the actual values for 'username' and 'password' would be here.
- Expected outcome' means answering the question 'what is supposed to happen when this test is run?' Should the program display a number? Which number? Should it display an error message? What should it say?

Testing Plan

This is how the table might look for the "1. Ask user for level" module:



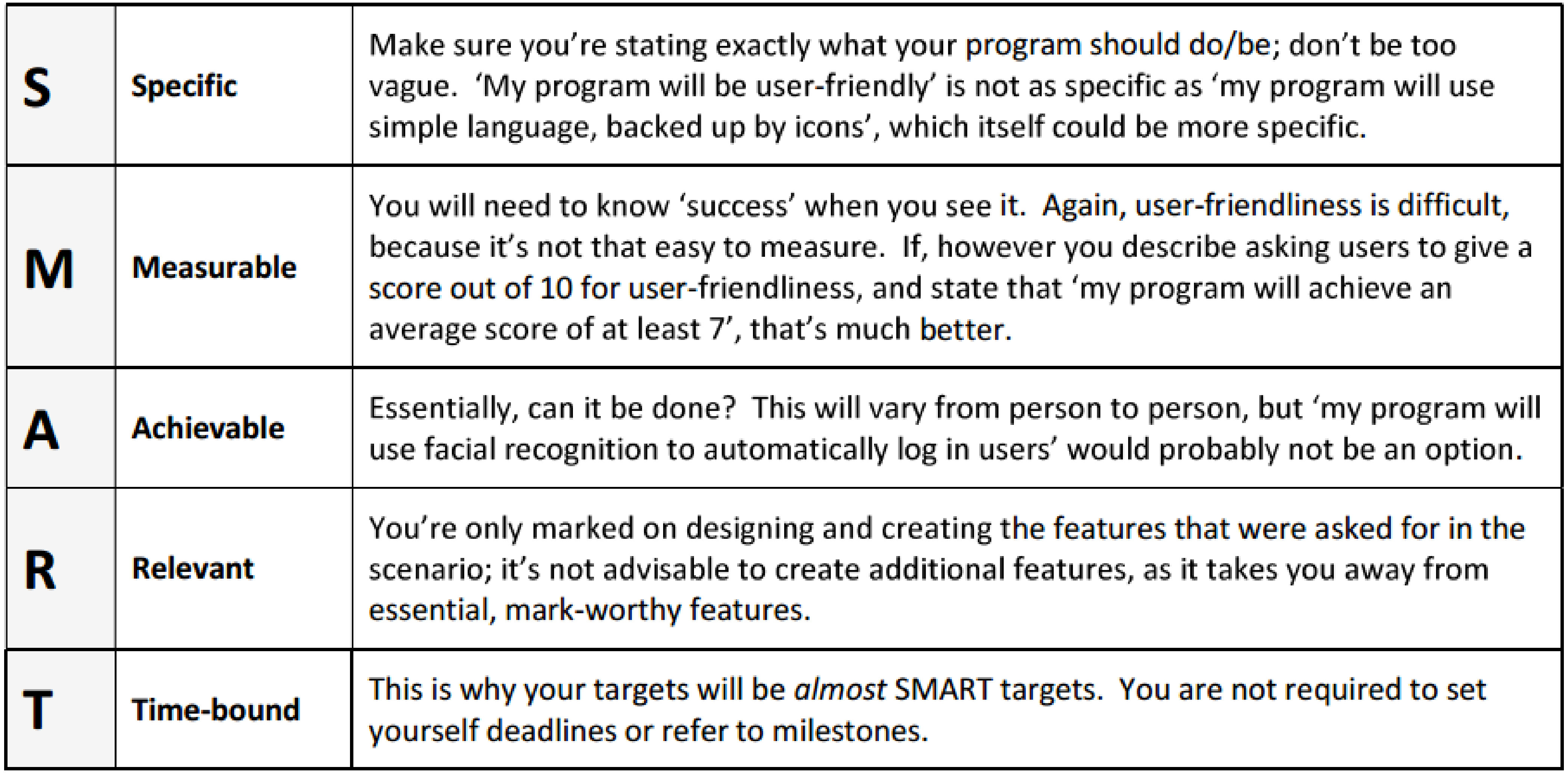
Test No.	Test	Data	Expected Outcome
1	Entering level and pressing 'enter'	2	Error message: "Please enter a whole number between 3 and 10"
2	Entering level and pressing 'enter'	3	'level' variable should contain the number '3'
3	Entering level and pressing 'enter'	10	'level' variable should contain the number '10
4	Entering level and pressing 'enter'	11	Error message: "Please enter a whole number between 3 and 10"
5	Entering level and pressing 'enter'	<blank>	Error message: "Please enter a whole number between 3 and 10"
6	Entering level and pressing 'enter'	"Hello"	Error message: "Please enter a whole number between 3 and 10"
7	Entering level and pressing 'enter'	4.5	Error message: "Please enter a whole number between 3 and 10"

The data tested should include:

- Typical Data
- Extreme Data
- Erroneous Data

Success Criteria

Your Success Criteria will be based on your Test Plan. Each success criterion will be almost a SMART target:



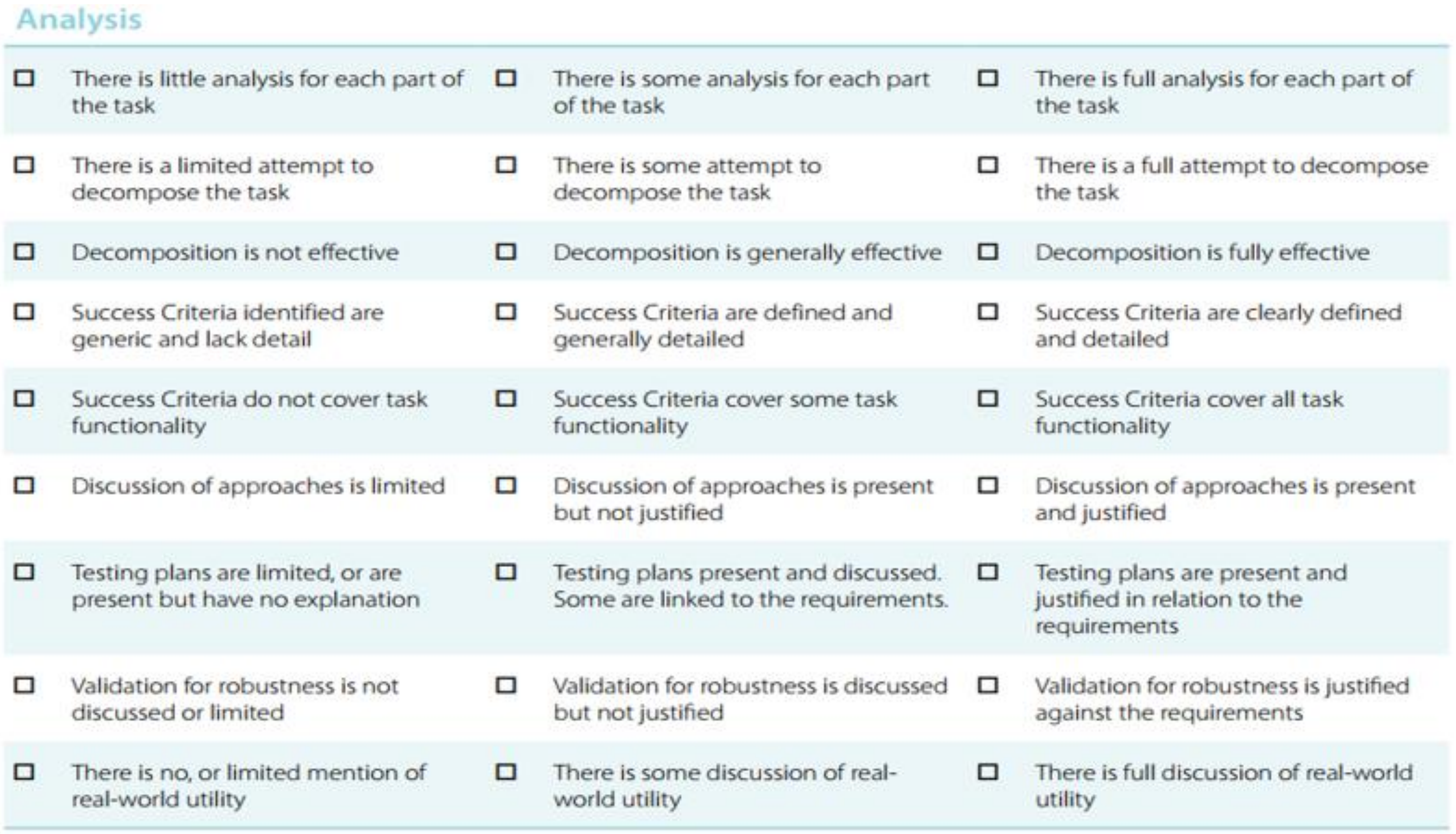
S	**Specific**	Make sure you're stating exactly what your program should do/be; don't be too vague. 'My program will be user-friendly' is not as specific as 'my program will use simple language, backed up by icons', which itself could be more specific.
M	**Measurable**	You will need to know 'success' when you see it. Again, user-friendliness is difficult, because it's not that easy to measure. If, however you describe asking users to give a score out of 10 for user-friendliness, and state that 'my program will achieve an average score of at least 7', that's much better.
A	**Achievable**	Essentially, can it be done? This will vary from person to person, but 'my program will use facial recognition to automatically log in users' would probably not be an option.
R	**Relevant**	You're only marked on designing and creating the features that were asked for in the scenario; it's not advisable to create additional features, as it takes you away from essential, mark-worthy features.
T	**Time-bound**	This is why your targets will be *almost* SMART targets. You are not required to set yourself deadlines or refer to milestones.

Testing & Success Criteria

Some success criteria for the memory test:

- My program will start by asking the user for a level between 3 and 10.
- No entry of invalid data will cause the program to crash. Instead, a helpful error message will be displayed and the user will be asked to re-enter the data.
- To aid user-friendliness, questions will always appear at the same point on the screen, in a large font, using simple language.

Analysis Checklist



Analysis

☐	There is little analysis for each part of the task	☐	There is some analysis for each part of the task	☐	There is full analysis for each part of the task
☐	There is a limited attempt to decompose the task	☐	There is some attempt to decompose the task	☐	There is a full attempt to decompose the task
☐	Decomposition is not effective	☐	Decomposition is generally effective	☐	Decomposition is fully effective
☐	Success Criteria identified are generic and lack detail	☐	Success Criteria are defined and generally detailed	☐	Success Criteria are clearly defined and detailed
☐	Success Criteria do not cover task functionality	☐	Success Criteria cover some task functionality	☐	Success Criteria cover all task functionality
☐	Discussion of approaches is limited	☐	Discussion of approaches is present but not justified	☐	Discussion of approaches is present and justified
☐	Testing plans are limited, or are present but have no explanation	☐	Testing plans present and discussed. Some are linked to the requirements.	☐	Testing plans are present and justified in relation to the requirements
☐	Validation for robustness is not discussed or limited	☐	Validation for robustness is discussed but not justified	☐	Validation for robustness is justified against the requirements
☐	There is no, or limited mention of real-world utility	☐	There is some discussion of real-world utility	☐	There is full discussion of real-world utility