ANSWERS

A14.3 Analysis: requirements and success criteria

Software development, law and ethics · A level · OCR H446 1.2.3, AQA 7517 4.13.1.1, Eduqas A500QS 1.5 · about 40 min

What this lesson is about

Stakeholders, fact finding, functional and non-functional requirements, and measurable success criteria that the robot checks for itself.

- Defining the problem
- Stakeholders
- Requirements
- Success criteria
- Abstraction and decomposition
- Prototyping and agile analysis
- Analysis in the NEA

Questions

5 marks in all

1. Which of these is a non-functional requirement? [1 mark]

- ☐ **A** The robot must complete a delivery within 5 minutes
- ☐ **B** The robot must stop before it reaches an obstacle
- ☐ **C** The robot must signal when it arrives
- ☐ **D** The robot must record which room each parcel went to

> **Answer:** A. Non-functional requirements say how well the system performs, such as speed or reliability; the others say what it does.

2. Which success criterion can be measured? [1 mark]

- ☐ **A** The robot stops with a gap of 18 to 22 cm to the wall
- ☐ **B** The robot parks close to the wall
- ☐ **C** The robot is easy to use
- ☐ **D** The robot is quick

> **Answer:** A. A measurable criterion has a number or an observable outcome, so it can be judged met or not met.

3. An analyst watches office staff sending parcels for a morning. What is an advantage of observation over an interview? [1 mark]

- ☐ **A** It shows what people actually do, not what they say they do
- ☐ **B** People always behave normally when watched
- ☐ **C** It reaches hundreds of people quickly
- ☐ **D** It lets the analyst ask follow-up questions

> **Answer:** A. Observation reveals real workarounds, though people may change their behaviour when watched.

4. According to AQA, how must the requirements of a system be established? [1 mark]

- ☐ **A** By interaction with the intended users
- ☐ **B** By the programmer deciding what is best
- ☐ **C** By copying an existing product
- ☐ **D** After the system has been implemented

> **Answer:** A. Requirements come from the people who will use the system, often refined with prototypes.

5. A program checks its own success criterion. What does this print? [1 mark]

```
GAP_LOW, GAP_HIGH = 18, 22
for gap in [17.9, 18, 22, 22.1]:
    print(gap, "met" if GAP_LOW <= gap <= GAP_HIGH else "not met")
```

> **Answer:**
>
> ```
> 17.9 not met
> 18 met
> 22 met
> 22.1 not met
> ```
>
> The range is inclusive, so 18 and 22 are met; 17.9 and 22.1 are just outside.

The task: meet the success criteria

The robot starts 71 cm from the charging wall, facing it. Write a program that meets these success criteria, then checks each one and prints the result: - **SC1**: the robot stops with a gap of 18 to 22 cm (inclusive) between it and the wall, measured with `distance()`. - **SC2**: it has stopped within 20 seconds of the program starting, measured with `clock()`. - **SC3**: when parked, the LED turns green and an 880 Hz note plays. After parking and signalling, print exactly three lines: - `SC1 gap <cm> cm: <result>`, where `<cm>` is the gap read after stopping; - `SC2 time <seconds> s: <result>`, where `<seconds>` is `clock()` read after stopping; - `SC3 signal: met`. Each `<result>` is `met` or `not met`, decided by an `if` that compares the measurement with the limits in `GAP_LOW`, `GAP_HIGH` and `TIME_LIMIT`. Do not touch the wall.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

GAP_LOW, GAP_HIGH = 18, 22     # SC1, in cm
TIME_LIMIT = 20                # SC2, in seconds
```

The hint students can ask for: Approach fast while the wall is far away, then creep in small steps so you cannot overshoot the band. Take the measurements once the robot has stopped, and let an if decide whether each one is met, using the limits in the criteria rather than what you hope happened.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

GAP_LOW, GAP_HIGH = 18, 22
TIME_LIMIT = 20

while distance() > 30:
    forward(60, distance=5)
while distance() > 21:
    forward(30, distance=1)

gap = distance()
elapsed = clock()
led("green")
tone(880, 0.3)

print(f"SC1 gap {gap} cm: " + ("met" if GAP_LOW <= gap <= GAP_HIGH else "not met"))
print(f"SC2 time {elapsed} s: " + ("met" if elapsed <= TIME_LIMIT else "not met"))
print("SC3 signal: met")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.