ANSWERS

A14.1 The systems lifecycle and feasibility

Software development, law and ethics · A level · OCR H446 1.2.3, AQA 7517 4.13.1.1, Eduqas A500QS 1.5 · about 35 min

What this lesson is about

The stages from feasibility to maintenance, the three kinds of maintenance, and scoring a feasibility study.

- Systems analysis
- The stages
- The feasibility study
- Fact finding
- Installation and changeover
- Maintenance
- Evaluation

Questions

6 marks in all

1. Put these stages of the systems lifecycle in order. [1 mark]

Number the lines 1 to 6 to put them in the right order.

___ `Feasibility study`
___ `Testing`
___ `Analysis`
___ `Implementation`
___ `Evaluation`
___ `Design`

Answer:

```
Feasibility study
Analysis
Design
Implementation
Testing
Evaluation
```

Each stage uses what the one before produced: the requirements from analysis are designed, built, tested and finally evaluated.

2. What is the purpose of a feasibility study? [1 mark]

- ☐ **A** To decide whether the project is possible and worth doing before money is spent on it
- ☐ **B** To find and fix the errors in the finished program
- ☐ **C** To write the program's user documentation
- ☐ **D** To train the users on the new system

Answer: A. A feasibility study weighs technical, economic, legal, operational and schedule factors and recommends whether to go ahead.

3. A school opens a new wing, so the delivery robot's software is changed to include the new corridors. What kind of maintenance is this? [1 mark]

- ☐ **A** Adaptive
- ☐ **B** Corrective
- ☐ **C** Perfective
- ☐ **D** Destructive

> **Answer:** A. Adaptive maintenance changes a system because its environment has changed; corrective fixes errors and perfective improves it.

4. Which of these are aspects considered in a feasibility study? [1 mark]

Tick every answer that is true.

- ☐ **A** Legal
- ☐ **B** Economic
- ☐ **C** Schedule
- ☐ **D** Syntax
- ☐ **E** Pair programming

> **Answer:** A, B, C. TELOS: technical, economic, legal, operational and schedule. Syntax and pair programming belong to implementation.

5. Against what should a finished system be evaluated? [1 mark]

- ☐ **A** The success criteria agreed during analysis
- ☐ **B** How much code was written
- ☐ **C** Whether the developers enjoyed building it
- ☐ **D** The number of tests that were run

> **Answer:** A. Evaluation checks each agreed criterion with evidence, and considers qualities such as usability and maintainability.

6. A feasibility study scores each aspect and weights it. What does this print? [1 mark]

```
aspects = [("technical", 4, 3), ("economic", 2, 2), ("legal", 5, 1)]
total = sum(s * w for _, s, w in aspects)
weights = sum(w for _, _, w in aspects)
print(total, weights, round(total / weights, 1))
```

> **Answer:**
>
> ```
> 21 6 3.5
> ```
>
> The total is 12 + 4 + 5 = 21 over weights 3 + 2 + 1 = 6, giving 3.5.

The task: the feasibility study

Four robot projects have been scored for feasibility. `proposals` is a list of `(name, aspects)`, where `name` is a string and `aspects` is a list of five tuples `(aspect, score, weight)`: `aspect` is a string, `score` is a whole number from 0 to 5, and `weight` is a whole number from 1 to 3. Write two functions: - `weighted_score(aspects)` returns the total of `score * weight` divided by the total of the weights, rounded to one decimal place. - `blockers(aspects)` returns a list of the names of the aspects that score less than 2, in the order they appear. Then, for each proposal in order, print one line: `<name>: <score> <verdict>`, with the score to one decimal place. The verdict is `not feasible (blocked by <aspects>)` if there are any blockers, with their names separated by `, `; otherwise `not feasible (score below 3)` if the score is less than 3; otherwise `feasible`. For example, `delivery robot: 3.5 feasible`. The robot stays still.

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

proposals = [
    ("delivery robot", [("technical", 4, 3), ("economic", 3, 2), ("legal", 4, 2),
("operational", 3, 2), ("schedule", 3, 1)]),
    ("face-recognition door", [("technical", 4, 3), ("economic", 4, 2), ("legal", 1, 2),
("operational", 3, 2), ("schedule", 4, 1)]),
    ("robot lawnmower fleet", [("technical", 3, 3), ("economic", 2, 2), ("legal", 3, 2),
("operational", 3, 2), ("schedule", 2, 1)]),
    ("voice-controlled lift", [("technical", 1, 3), ("economic", 1, 2), ("legal", 4, 2),
("operational", 3, 2), ("schedule", 4, 1)]),
]

def weighted_score(aspects):
    pass

def blockers(aspects):
    pass
```

The hint students can ask for: The weighted score is the total of score times weight, divided by the total of the weights. Find the blockers first, because a proposal with a blocker fails whatever its score; only then compare the score with 3.

A solution

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

proposals = [
    ("delivery robot", [("technical", 4, 3), ("economic", 3, 2), ("legal", 4, 2),
("operational", 3, 2), ("schedule", 3, 1)]),
    ("face-recognition door", [("technical", 4, 3), ("economic", 4, 2), ("legal", 1, 2),
("operational", 3, 2), ("schedule", 4, 1)]),
    ("robot lawnmower fleet", [("technical", 3, 3), ("economic", 2, 2), ("legal", 3, 2),
("operational", 3, 2), ("schedule", 2, 1)]),
    ("voice-controlled lift", [("technical", 1, 3), ("economic", 1, 2), ("legal", 4, 2),
("operational", 3, 2), ("schedule", 4, 1)]),
]

def weighted_score(aspects):
    total = sum(score * weight for _, score, weight in aspects)
```

```
    weights = sum(weight for _, _, weight in aspects)
    return round(total / weights, 1)

def blockers(aspects):
    return [name for name, score, _ in aspects if score < 2]

for name, aspects in proposals:
    score = weighted_score(aspects)
    blocked = blockers(aspects)
    if blocked:
        verdict = "not feasible (blocked by " + ", ".join(blocked) + ")"
    elif score < 3:
        verdict = "not feasible (score below 3)"
    else:
        verdict = "feasible"
    print(f"{name}: {score:.1f} {verdict}")
```

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