ANSWERS

A14.6 Software engineering tools and version control

Software development, law and ethics · A level · OCR H446 1.2.3, AQA 7517 4.13.1.3, Eduqas A500QS 1.7 · about 40 min

What this lesson is about

IDEs, CASE tools, documentation, and version control: commits, branches, merges and finding the commit that broke the robot.

- The IDE
- CASE tools
- Documentation
- Version control
- Finding the commit that broke it

Questions

6 marks in all

1. Two developers need to work on different features of the robot's code at the same time. Which version control feature supports this? [1 mark]

- ☐ **A** Branches, later merged into the main line
- ☐ **B** Syntax highlighting
- ☐ **C** A watch window
- ☐ **D** A compiler

> **Answer:** A. Each feature is developed on its own branch, then merged; conflicts are flagged rather than work being overwritten.

2. Which IDE feature stops the program at a chosen line so its variables can be inspected? [1 mark]

- ☐ **A** A breakpoint
- ☐ **B** Auto-complete
- ☐ **C** Syntax highlighting
- ☐ **D** A merge

> **Answer:** A. Breakpoints, stepping and watches help find logic errors that give no error message.

3. What is a merge conflict? [1 mark]

- ☐ **A** Two branches changed the same lines differently, so a person must decide what to keep
- ☐ **B** Two files have the same name
- ☐ **C** A commit has no message
- ☐ **D** The program fails a test after a merge

> **Answer:** A. Version control cannot choose between two different edits to the same lines, so it asks a person.

4. Which of these belong in technical documentation rather than user documentation? [1 mark]

Tick every answer that is true.

- ☐ **A** The purpose and parameters of each subroutine
- ☐ **B** The data structures used
- ☐ **C** How to send a parcel with the robot
- ☐ **D** What each error message on the screen means

> **Answer:** A, B. Technical documentation is for programmers maintaining the system; user documentation explains how to use it.

5. What does this count of changed lines print? [1 mark]

```
old = "a\nb\nc".splitlines()
new = "a\nc\nd\nd".splitlines()
added = sum(max(0, new.count(x) - old.count(x)) for x in set(new))
removed = sum(max(0, old.count(x) - new.count(x)) for x in set(old))
print(f"+{added} -{removed}")
```

> **Answer:**
>
> ```
> +2 -1
> ```
>
> d appears twice in the new version and not at all in the old, so 2 lines were added; b was removed.

6. What does CASE stand for in CASE tools? [1 mark]

> **Answer:** computer-aided software engineering. CASE tools support the whole lifecycle: diagrams, code generation, automated testing and documentation.

The task: the commit log

`versions` is a list of commits in order. Each is a tuple `(message, text)`: `message` is a string, and `text` is the whole program at that commit, as one string with lines separated by `\n`. Write a function `changes(old, new)` that takes two such texts and returns a tuple `(added, removed)`. A line counts by its exact text, including its indentation. If a line appears more times in `new` than in `old`, the extra appearances count as added; if it appears more times in `old` than in `new`, the extra appearances count as removed. Compare each commit with the one before it; the first commit is compared with the empty string `""`. For each commit print `commit <n>: <message> (+<added> -<removed>)`, numbering from 1. Then find the **first** commit whose text contains the string in `BUG`, and print `bug introduced in commit <n>: <message>`. The robot stays still.

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

versions = [
    ("first drive", "from bugbot import *\nconnect()\nforward(50, distance=20)"),
    ("stop at the wall", "from bugbot import *\nconnect()\nwhile distance() > 20:\n
forward(50, distance=5)"),
    ("faster approach", "from bugbot import *\nconnect()\nwhile distance() > 20:\n
forward(100, distance=50)\nled(\"green\")"),
    ("beep when parked", "from bugbot import *\nconnect()\nwhile distance() > 20:\n
forward(100, distance=50)\nled(\"green\")\ntone(880, 0.3)"),
]
BUG = "forward(100, distance=50)"

def changes(old, new):
    pass
```

The hint students can ask for: Count how many times each line appears in the old version and in the new one; a line appearing more often in the new version was added that many times, and one appearing less often was removed. For the bug, look through the versions in order and stop at the first that contains the line.

A solution

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

versions = [
    ("first drive", "from bugbot import *\nconnect()\nforward(50, distance=20)"),
    ("stop at the wall", "from bugbot import *\nconnect()\nwhile distance() > 20:\n
forward(50, distance=5)"),
    ("faster approach", "from bugbot import *\nconnect()\nwhile distance() > 20:\n
forward(100, distance=50)\nled(\"green\")"),
    ("beep when parked", "from bugbot import *\nconnect()\nwhile distance() > 20:\n
forward(100, distance=50)\nled(\"green\")\ntone(880, 0.3)"),
]
BUG = "forward(100, distance=50)"

def changes(old, new):
    before, after = {}, {}
    for line in old.splitlines():
        before[line] = before.get(line, 0) + 1
    for line in new.splitlines():
```

```
        after[line] = after.get(line, 0) + 1
    added = sum(max(0, n - before.get(line, 0)) for line, n in after.items())
    removed = sum(max(0, n - after.get(line, 0)) for line, n in before.items())
    return added, removed

previous = ""
for n, (message, text) in enumerate(versions, start=1):
    added, removed = changes(previous, text)
    print(f"commit {n}: {message} (+{added} -{removed})")
    previous = text

for n, (message, text) in enumerate(versions, start=1):
    if BUG in text:
        print(f"bug introduced in commit {n}: {message}")
        break
```

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