The Basics of Unit Testing

The Art of Unit Testing, 3e

Summer

2024/03/28

Agenda

- 什麼是「單元」
- 什麼是好的單元測試
- 單元測試 vs 整合測試
- TDD

<Timer>

Remaining seconds: 3  Remaining seconds: 2  Remaining seconds: 1  Time's Up

Demo: https://playcode.io/1812654

SUT

```
const Timer = () => {
  const [seconds, setSeconds] = useState(3);
  const intervalIDRef = useRef(null);

  const startTimer = useCallback(() => {
    intervalIDRef.current = setInterval(
      () => setSeconds((prev) => prev - 1),
      timerTickInterval
    );
  }, []);

  const stopTimer = useCallback(() => {
    clearInterval(intervalIDRef.current);
    intervalIDRef.current = null;
  }, []);

  useEffect(() => {
    startTimer();
    return () => clearInterval(intervalIDRef.current);
  }, []);

  useEffect(() => {
    if (seconds === 0) {
      stopTimer();
    }
  }, [seconds]);

  return (
    <div>{seconds === 0 ? `Time\'s Up` : `Remaining seconds:
${seconds}`}</div>
  );
};
```

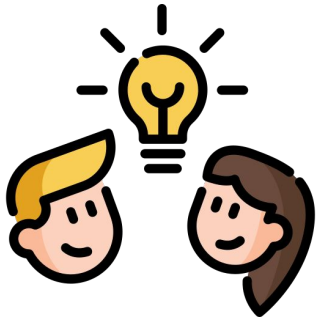
Unit of Work

```
const Timer = () => {
  // entry point 函式被呼叫
  const [seconds, setSeconds] = useState(3);
  const intervalIDRef = useRef(null);

  const startTimer = useCallback(() => {
    // exit point 呼叫外部函式
    intervalIDRef.current = setInterval(
      () => setSeconds((prev) => prev - 1),
// exit point 改變狀態
      timerTickInterval // 1000
    );
  }, []);

  const stopTimer = useCallback(() => {
    // exit point 呼叫外部函式
    clearInterval(intervalIDRef.current);
    intervalIDRef.current = null;
  }, []);

  useEffect(() => {
    startTimer();
    return () =>
clearInterval(intervalIDRef.current); // exit
point 呼叫外部函式
  }, []);

  useEffect(() => {
    if (seconds === 0) {
      stopTimer();
    }
  }, [seconds]);

  // exit point 函式回傳結果
  return (
    <div>
      {seconds === 0 ? `Time\'s Up` :
`Remaining seconds: ${seconds}`}
    </div>
  );
};
```

```
describe('Timer component', () => {
  beforeAll(() => jest.useFakeTimers());

  afterAll(() => jest.useRealTimers();
  });

  it('should show remaining 2 seconds when after 1 second', () => {
    const { getByText } = render(<Timer />);

    // Mock-object-based test
    act(() => jest.advanceTimersByTime(1000));

    // Return-value-based exit point
    expect(getByText('Remaining seconds: 2')).toBeInTheDocument();
  });
```

```
  it("should show Time's Up when after 3 seconds", () => {
      const { getByText } = render(<Timer />);

      // Mock-object-based test
      act(() => jest.advanceTimersByTime(3000));

      // Return-value-based exit point
      expect(getByText("Time's Up")).toBeInTheDocument();
  });

  it('cleans up the timer on unmount', () => {
    const { unmount } = render(<Timer />);
    const spyOnClearInterval = jest.spyOn(global, 'clearInterval');

    unmount();

    // State-based test
    expect(spyOnClearInterval).toHaveBeenCalledTimes(1);
  });
});
```

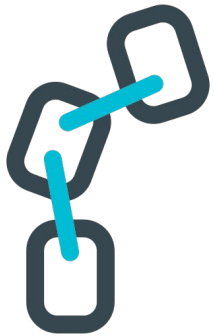
單元測試的特質



簡單、清楚、易執行

- 目標明確、提示清楚
  - (O) should get 3 when 1 + 2
  - (X) get correct answer when execute sum
- 好懂好維護
  - (O) 對程式的外部行為進行測試
  - (X) 關注內在實作細節
- 配置簡單
  - (O) 一鍵執行
  - (X) 過多資源、設定、前置工作
- 自動執行
  - (O) 完全由程式自動測試
  - (X) 需搭配手動測試

單元測試的特質



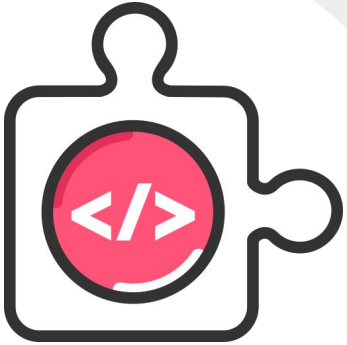
穩定的、可預期的

- 有完全的控制權
  - (O) 模擬依賴
  - (X) 真實依賴
- 隔離於獨立的狀態與環境、與其他測試沒有相依
  - (O) 執行前處於相同的起始狀態
  - (X) 共用狀態
- 無資源相依
  - (O) mock 或固定測試資料
  - (X) 寫檔、寫資料庫、網路
- 同步且線性執行
  - (O) 完全由程式自動測試
  - (X) 需搭配手動測試

單元測試的特質

執行速度快

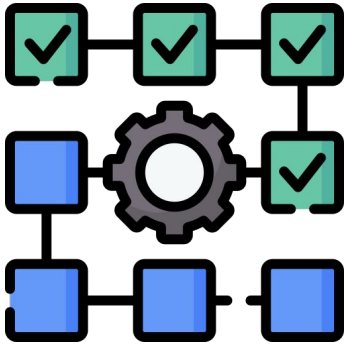
- 一次只檢測一個目的
- 比對特定元素，減少快照的使用



Unit Testing

單元測試

VS



Integration Testing

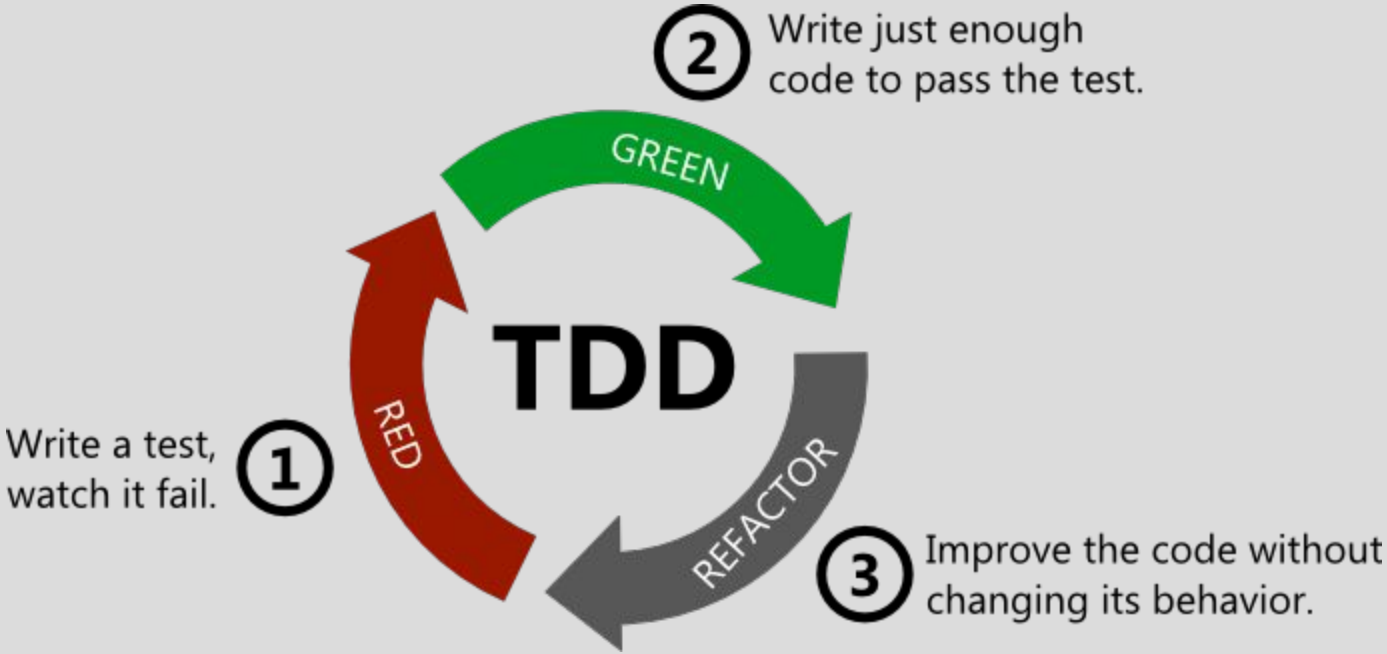
整合測試

良好的單元測試

Readability
可讀性

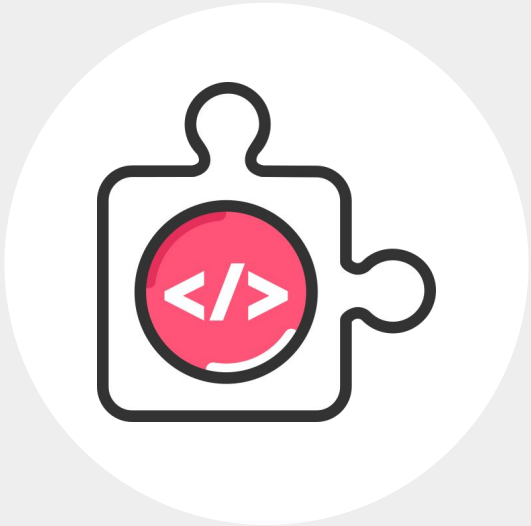
Maintainability
可維護性

Trust
可信性



https://www.linkedin.com/pulse/how-make-test-driven-development-unit-testing-easier-greg-goldshteyn

推行 TDD 成功的三要素



高品質的測試

TDD

先寫測試

SOLID

如何寫一份前端的開發設計文件？
https://bit.ly/48Rp5U5

https://stevenma.medium.com/the-solid-design-principles-why-are-they-important-2c73c1f0b983

總結

- 什麼是好的單元測試：速度快、對要測試的程式碼有完全的主控權、沒有與其他測試或資源相依、同步和線性執行。
- public function 是 entry point，用來進入 unit of work 以觸發測試；exit point 是測試會檢查的地方，包含回傳結果、改變狀態、呼叫 third-party dependency，分別需要不同的測試技術來做檢測。
- unit of work 是從 entry point 到 exit point 的過程，是用來觀察程式碼的流程，之後就能利用這個流程來決定 exit point。
- integration testing 即是有相依狀況的 unit testing。
- 好的測試應具備可讀性、可維護性、可信任性。
- TDD 是指在開發前先寫測試的技術，TDD 的好處是驗證測試的正確性，看到測試失敗後再寫產品的程式碼，就可以確保這些測試在功能失效時會失敗。

Thank you!