***Bài 2:***  
1/ Một số khái niệm liên quan đến KTPM – Software Testing

1.1/ The classic statements ( câu phát biểu kinh điển)  
- Phân biệt các thuật ngữ: **error, fault, failure, defect, mistake, bug**. Phải phân biệt trong ngữ cảnh, đoạn văn, câu chuyện  
=> Cùng có 1 ý nghĩa chung: có 1 thứ không như mong đợi xảy ra, sự sai lệch nào đó theo dự kiến nó phải là thế này, mà nó lại ra thế khác!!!

Khái niệm bug: Bug là 1 lỗi hay xuất hiện ở trong app, sai sót trong phần mềm và được tester phát hiện thấy được những người phát triển phần mềm sửa nó.

=> Bug là thứ gây nên sự bất ổn trong app, developer phải fix

***Bài 4:***

1/ Định nghĩa kiểm thử

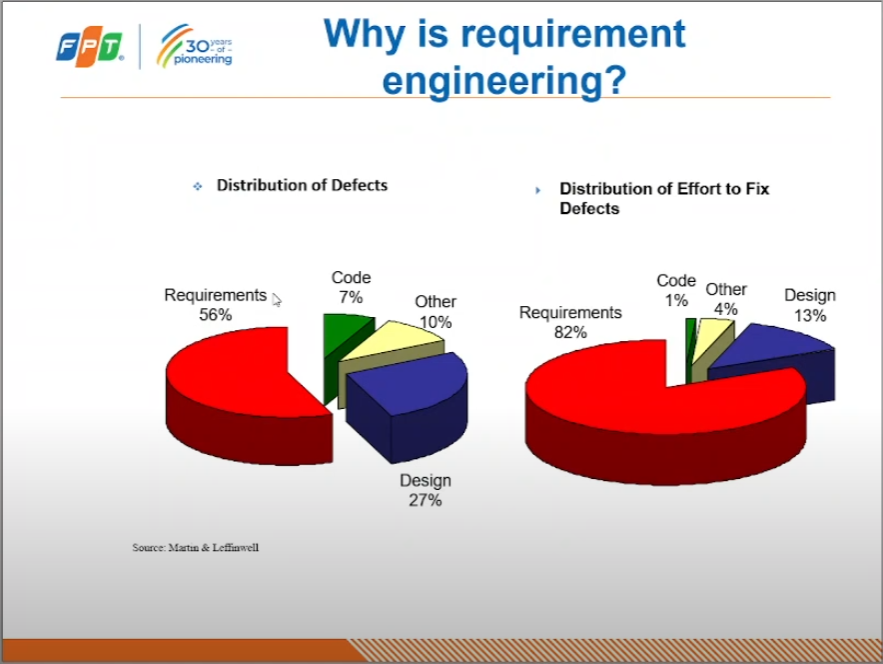
So sánh:

- Giữa cái mình kì vọng app phải như thế này ( Expected ) và cái thực tế App đang run và trả ra kết quả không như mong đợi

- Chạy app, chạy 1 chức năng, màn hình và nhìn kết quả và đánh giá ( evaluate/compare ) kết quả tính toán/hiển thụ của app so với cái ta kì vọng

( dynamic testing – chạy app kiểm tra )

- Kiểm tra xem app đã viết/ implement/ hiện thực có đúng như thiết kế/ mô tả / specification ( static testing – không chạy app chỉ kiểm tra các bản thiết kế đối chiếu với việc app được code/ design ra sao )

- Kiểm tra requirement có ổn không, khả thi không, phù hợp vs nhu cầu người dùng hay không  


***Bài 5:***

1/ Ai tham gia vào phán quyết App ổn, App có bug?

- Developer: - Người chịu trách nhiệm chất lượng code của mình

- Đọc code mình viết

- Chạy code mình viết, kiểm tra xem Expected có == Actual hay không

- Kỹ thuật nào để kiểm tra code của mình hiệu quả  
 - TDD, Unit Test

Test First Development

Test Driven Development ( thay đổi 1 chút hành vi viết code )

🡪 dẫn đến CI/CD/DevOps Continuous Integration

- Tester ( QC – Quality Control ): thánh soi, dân kiểm thử phần mềm ( thường sẽ không mở code hiểu thuật toán, cách viết code…) lỗi được ghi nhận bởi Tester/Dev team, sau đó fix

- Tester căn cứ trên cái gì để tìm sai sót/bug/lỗi

- Document (Specification, SRS, FRS, BRS,…) 🡺 static testing, là thánh soi, soi ngay khi viết Reqs, thiết kế UI, DB, kiến trúc

\* Mở rộng: Job title trong nghề làm P/M

- BA ( Business Analyst ) = reqs

- Designer ( UI )

- DB Designer

- Developer ( code ): FE, BE, Full-stack

- Tester ( QC )

- Supporter ( cài đặt, hướng dẫn sử dụng, ghi nhận sự cố… )

- Kiến trúc sư ( Solution Architect/ Software Architect 4-5k$/month )

- PM ( Project Manager )

- PM khác ( Product Manager )

- CIO ( cao cấp ) Chief Information officer

- CTO

***Bài 6:***

- Sếp bên tester: QC Manager, QC Lead/Leader | QC: Quality Control

- Quản lý các công việc liên quan đến quá trình kiểm soát chất lượng sản phẩm, quản lý các công việc liên quan quá trình tìm bug

- Lên kế hoạch về việc kiểm thử, lúc nào thì test app

- Phân bổ nguồn lực vào việc kiểm thử, chia người, chia task, bố trí bao nhiêu người, máy móc thiết bị dành cho việc test

Giả sử cần Test app bán hàng của siêu thị GS25

Manager: -Biết lộ trình phát triển App của Dev: tháng nào, sprint nào xong module nào

- Biết được độ phức tạp của app cần test: thuật toán, xử lí, thiết bị, AI…

- Cần bao nhiêu người để kiểm thử APP

- Lên kế hoạch tìm hiểu/ nghiên cứu bản thiết kế App, Specification – mô tả reqs hiểu được app xử lí làm gì -> Phân chia người thiết kế kịch bản test/ test cases tưởng tượng các luồng đi của user khi xài app, test thử demo, **luồng đi => test case, phân công người design luồng này**

KHI NÀO BẮT ĐẦU TIẾN HÀNH KIỂM THỬ CHẤT LƯỢNG PHẦN MỀM

- Càng sớm càng tốt, ngay cả khi chưa viết code

- Test ngay từ lúc lấy reqs, test ngay cái design: DB, UI, Architecture,…

OOP: 4 + 5 ( 4 NGUYÊN LÝ CĂN BẢN CỦA OOP + 5 SOLID )

TỔNG QUAN CHUNG VỀ CÁC GÓC NHÌN LÀM APP: 4 + 1 MODEL

SCRUM/AGILE: 4 ( 4 ĐIỀU TRONG AGILE MANIFESTO )

TESTING: 7 (7 ĐỊNH LUẬT/ĐỊNH LÝ//NGUYÊN LÝ) CẦN NHỚ KHI LÀM NGHẾ KIỂM THỬ

***Bài 7:***

1/ 7 định luật/nguyên lí/định lí cần nhớ khi làm nghề kiểm thử?

Những điều sẽ nói dưới đây coi như là định lí, tiên đề để định hướng việc kiểm thử của chúng ta

1.1/ Kiểm thử phần mềm không phải là việc chứng minh, hay khẳng định, hay phải tuyên bố phần mềm tui không có lỗi – free of errors/defects

Hiểu ngược lại: phần mềm luôn có lỗi, còn lỗi, việc của kiểm thử/ testing là tìm lỗi, nhưng tiêu chí sản xuất luôn là: HƯỚNG ĐẾN CHẤT LƯỢNG, GIẢM THIỂU BUG NẾU CÓ THỂ

- các công ty luôn đưa ra những quy trình để đảm bảo sp làm ra có chất lượng

1.2/ Không thể test hết các tổ hợp sử dụng/ tình huống sử dụng phần mềm của USER

- Khi ta bán phần mềm ra thị trường (product-based)

- Khi ta bàn giao phần mềm theo đặt hàng (service-based, customized app, bespoke app)

Người dùng/user sẽ có n cách thức xài app

Dân kiểm thử/tester trước cái app theo cách mà người dùng xài app, xài theo cách đó, coi có văng lỗi hay không

Số tổ hợp cách xài app là vô chừng, do đó có tiền đề 2: ko thể test hết các cách xài của user vs app   
1.3/ EARLY TESTING: KIỂM THỬ CÀNG SỚM CÀNG TỐT, VÌ VÀO GIAO ĐOẠN CODE, VIỆC SỬA APP CÀNG TỐN CHI PHÍ NGAY TỪ KHI LẤY REQS ĐÃ CẦN TÉT DOCUMENT RỒI, TEST THIẾT KẾ RỒI

1.4/ Defect clustering – Sự phân bố hay tập trung của bug

Lỗi thường xảy ra nhiều ở 1 số chỗ trong app => Dân kiểm thử biết điều này để tập trung thời gian vào trong việc test chỗ này

1.5/ Pesticide paradox: nghịch lí thuốc trừ sâu, hiện tượng “lờn” thuốc “kháng” thuốc

- Chủ quan, quen việc test của dân kiểm thử, test mãi 1 cái app

-> dẫn đến bỏ sót bug

Mời bạn vào đảm bảo chất lượng p/m, bạn lại để bug to

- Nên thay đổi project cần test, module cần test, môi trường app nay test mobile, mai test web

1.6/ Kiểm thử phụ thuộc ngữ cảnh

- các loại app khác nhau, môi trường chạy app khác nhau, thì phải có phương pháp kiểm thử khác nhau, cách thức kiểm thử khác nhau

- app chạy đa nền tảng khác app hỗ trợ 1 platform

- responsive là 1 dạng test khi app chạy trên màn hình to nhỏ

- tính tương thích của thiết bị: in bill loại giấy, loại máy in

- IOT cần có thể môi trường giả lập hoặc ra thực địa để xem app chạy thế nào

1.7 Ảo tưởng về việc app hết lỗi

- Vỗ ngực app ngon, hết lỗi, ít lỗi, app tao chất lượng

Viết app chất lượng là default, bán hàng phải bán có chất lượng

Chuyện quan trọng hơn phải là: APP tao bắt trend, app tui có lượng user/customer đông đảo => làm hài lòng khách hàng

REQS phải chuẩn trong hết mức khả năng

***Bài 8:***

Testing level: là cách phân chia công việc test, tiến hành công việc kiểm tra chất lượng phần mềm dựa trên tiến trình hoàn thiện code, hoàn thiện app chia 4 mức độ/ 4 giai đoạn kiểm thử app dựa trên việc viết code

-> Thực sự việc kiểm thử phần mềm phải đc tiến hành sớm hơn, ko chờ đến giai đoạn viết code, test ngay ở giai đoạn reqs, nguyên lý early testing

Chia 4 mức độ/ 4 giai đoạn kiểm thử app dựa trên việc viết code, việc hoàn thiện code

1/ Unit Test/Testing

- Unit: đơn vị code, có thể xem 1 hàm, 1 class là 1 đơn vị code cơ bản

Developer sẽ phải là người có trách nhiệm đảm bảo rằng hàm/class của mình ok

Ok cho chính mình sử dụng, ok cho người khác sử dụng

Devloper phải test code của mình, test hàm, method, class

Làm sao test code của mình:

- Mở code ra đọc

- Chạy thử code

- Chạy = cách gọi hàm ở trong main()

- In ra trang web, hoặc windows form, hoặc ghi ra LOG file (.txt)

2 cách này đều ổn cả, nhưng bị 1 vấn đề:

- nếu ta có nhiều hàm cần test thì sao? Code chạy thử hàm cũng dài dài

- mắt ta phải nhìn các output trả về của hàm ( ACTUAL – thực tế )

Ta tự so sánh với cái ta nghĩ rằng app phải trả ra thế này ( EXPECTED – mong đợi )

Thư viện dùng để test các hàm của ta theo mô tả ở trên được gọi là Unit Test Framework, vs mỗi NNLT khác nhau có Unit test framework khác nhau

Framework khác library thế nào? -> Framework phải viết đúng theo nguyên tắc mà nó đã đặt ra, đúng cú pháp, dàn khung -> ràng buộc kỹ hơn so với library viết tự do

Dùng Junit để test code của mình, các hàm của mình, các class của mình chưa quan tâm UI, mức UNIT TEST, DEVELOPER phải làm việc này, còn không thì dùng console, log file

2/ Integration test

3/ System test

4/ UAT ( User Acceptance test )