**CHƯƠNG 1: TỔNG QUAN KIỂM THỬ**

* Quy trình phát triển phần mềm là các giai đoạn mà quá trình phát triển phần mềm phải trải qua.
  + Mỗi giai đoạn cần xác định rõ:
    - **Mục tiêu, kết quả** đạt được của giai đoạn trước.
    - **Kết quả** chuyển giao cho giai đoạn tiếp sau.
* Tiếp cận quy trình phát triển:
  + Quy trình có kế hoạch (plan-driven process): là quy trình mà tất cả các hoạt động đã được **lên kế hoạch trước** và tiến độ được xác định thông qua kế hoạch này.
  + Quy trình phát triển nhanh (agile process): các kế hoạch **tăng trưởng dần** và quy trình này dễ dàng phản ứng với những thay đổi yêu cầu của khách hàng.
* Quy trình thác nước:
  + Xđ yêu cầu ⬄ Phân tích ⬄ Thiết kế ⬄ Cài đặt ⬄ Kiểm thử ⬄ Triển khai
  + Các giai đoạn tuần tự nối tiếp nhau, giai đoạn này xong mới tiến hành giai đoạn tiếp theo, và không được phép quay lùi
  + Khó khăn trong việc xử lý các thay đổi ở một giai đoạn trước đó
* Quy trình lặp (Iterative Process) phát triển phần mềm theo nhiều vòng lặp, mỗi vòng lặp sẽ cho ra một phiên bản của phần mềm.
  + Kết thúc mỗi vòng lặp sẽ đánh giá, cải thiện để xđ y/c cho phiên bản của vòng lặp tt
  + Đặc trưng: có **đại diện người dùng** trong quá trình kiểm thử.
  + Nhược điểm: thiếu các tài liệu chính thức khiến quá trình kiểm thử trở nên khó khăn. Khi lập trình viên thực hiện các thay đổi theo y/c khách hàng không cần ghi nhận lại một cách chính thức khiến khả năng truy vết (traceability) trong các tiến độ dự án bị giảm.
* Quy trình tăng trưởng:
  + Chia hệ thống thành các thành phần nhỏ để phát triển, mỗi thành phần áp dụng quy trình thác nước hoặc một quy trình nào đó để tiến hành đến khi thành phần cuối cùng được hoàn tất thì quá trình phát triển toàn bộ hệ thống kết thúc.
  + Linh hoạt, giúp phát triển phần mềm nhanh, giảm chi phí khi thay đổi phạm vi và yêu cầu phần mềm, dễ dàng kiểm thử và phát hiện vấn đề trong từng vòng lặp nhỏ hơn.
  + **Nhược điểm**: có chi phí cao hơn quy trình thác nước, cần có kế hoạch và thiết kế đầy đủ trước khi chia nhỏ và phát triển tăng trưởng dần.
* Quy trình Scrum: quy trình phát triển phần mềm theo **mô hình linh hoạt** (agile).
  + Chia dự án thì **nhiều vòng lặp** phát triển gọi là sprint, một sprint thường kéo dài 2-4 tuần (30 ngày).
  + phù hợp các dự án có **nhiều thay đổi** và yêu cầu **tốc độ cao**.
* Chất lượng phần mềm: chỉ định mức độ để phần mềm, một thành phần của phần mềm hoặc một quy trình đáp ứng được các yêu cầu chỉ định (IEEE).
  + Đánh giá thông qua các thuộc tính chất lượng: Tính chính xác (Correctness) ▪ Tính tin cậy (Reliability) ▪ Tính khả dụng (Usabiliy) ▪ Tính toàn vẹn (Integrity) ▪ Tính khả chuyển (Portability) ▪ Tính bảo trì (Maintainability) ▪ Tính tương tác (Interoperability)
* Kiểm thử (testing) là quá trình đánh giá hệ thống hoặc các thành phần của hệ thống, nhằm kiểm tra nó **đúng với bản đặc tả yêu cầu** và **tìm ra lỗi** trước khi đưa vào sử dụng
  + Kiểm thử nghĩa là thực thi chương trình với **dữ liệu được tạo trước**.
  + Mục đích
    - Chứng minh nhóm phát triển phần mềm đã phát triển đúng với yêu cầu của khách hàng (***validation testing***).
    - Phát hiện ra những trường hợp phần mềm còn lỗi, thiếu sót hoặc hoạt động không đúng (***defect testing***) với đặc tả của nó.
* 7 nguyên tắc kiểm thử:
  + Kiểm thử chứng minh sự hiện diện của lỗi (Testing is context dependent)
  + Kiểm thử toàn bộ là không khả thi (Exhausive testing is impossible)
  + Kiểm thử càng sớm càng tốt (Early testing)
  + Lỗi thường được phân bố tập trung (Defect clustering)
  + Nghịch lý thuốc trừ sâu (Pesticide paradox)
  + Kiểm thử phụ thuộc vào ngữ cảnh (Testing shows presence of defects)
  + Quan niệm sai lầm về việc “hết lỗi” (Absent-oferrors fallacy)
* **Thuật ngữ**
* **Erro**r (lỗi): lỗi do con người gây ra.
* **Fault** (sai sót): những sai sót do dư hoặc thiếu hoặc không đúng các yêu cầu phần mềm cần thực hiện. Error và fault gọi chung là **bug**.
* **Failure** (hỏng): xảy ra khi các sai sót (fault) được thực thi.
* **Incident** (biến cố): là những trường hợp mà phần mềm có những thực thi đáng nghi ngờ, nguyên nhân của nó có thể là do cấu hình môi trường kiểm thử không đúng, dữ liệu dùng kiểm thử sai hoặc do lỗi của kiểm thử viên,... Incident được xem là lỗi (defect hoặc bug) khi nguyên nhân của vấn đề thuộc chính thành phần đang kiểm thử.
* Stub: một mẫu code để mô phỏng hoạt động của thành phần đang thiếu.
* Driver: một mẫu code để truyền test case (dữ liệu kiểm thử) đến mẫu code khác (module cần kiểm thử), nhận kết quả và xuất ra.
* QC (Quality Control): những hđ, kỹ thuật nhằm đảm bảo chất lượng sản phẩm, ngừa lỗi
* QA (Quality Assurance): những kế hoạch, hoạt động mang tính hệ thống nhằm đảm bảo quá trình sản xuất sẽ tạo những sản phẩm có chất lượng -> tìm lỗi
* **Các cấp độ kiểm thử**

1. **Unit test** (Kiểm thử đơn vị)

* Dùng kiểm tra trên từng đơn vị riêng rẽ
* Đc thực hiện trong suốt giai đc implementation
* Được thực hiện bởi chính developer phát triển đơn vị đó nhằm đảm bảo mã nguồn viết cho đơn vị nào đó đúng với đặc tả yêu cầu, xác nhận mã nguồn trong đơn vị đó có thể thực thi.
* Một đơn vị (hàm, thủ tục, lớp, phương thức) là thành phần nhỏ nhất có thể kiểm thử.
* Các lỗi tìm thấy và sửa trong giai đoạn kiểm thử đơn vị thường k cần ghi trong tài liệu.

1. **Integration test** (**Kiểm thử tích hợp**)

* kiểm tra **sự tương tác** các tp riêng rẽ.
* Các chiến lược tích hợp
* Chiến lược Big-bang: các đơn vị, thành phần được tích hợp cùng 1 lúc để có HT đầy đủ.
* Ưu điểm: quá trình kiểm thử bắt đầu chỉ khi mọi thứ của HT đã hoàn tất
* Khuyết điểm: khi có lỗi xảy ra rất khó cô lập lỗi để tìm ra nguyên nhân.
* Lựa chọn k tốt cho chiến lược tích hợp, có thể dẫn đến nhiều rủi ro sau đó của dự án, thời điểm tốn nhiều chi phí để giải quyết vấn đề tìm thấy.
* Chiến lược Incremental: kiểm thử từng phần riêng biệt.
* Ưu điểm: lỗi được phát hiện sớm và dễ dàng tìm ra nguyên nhân.
* Khuyết điểm: tốn nhiều thời gian vì phải phát triển các stub&driver để thực hiện kiểm thử. Hai chiến lược incremental thường được sd là topdown và bottom-up.
  + top-down: xd theo từng giai đoạn, bắt đầu từ thành phần cao nhất (top) gọi các tp khác ở cấp thấp hơn, y/c sự tương tác giữa các tp phải được kiểm thử trước khi xây dựng các tp. Các tp ở cấp thấp hơn chưa xây dựng sẽ được thay thế = stub.
  + bottom-up: ngược lại với chiến lược top-down, các thành phần được tích hợp theo thứ tự từ dưới lên.

1. **System test** (**Kiểm thử hệ thống**)

* Kiểm thử trên **hệ thống đầy đủ** sau khi các thành phần đã được tích hợp.
* Mục đích: đảm bảo phần mềm **tuân thủ các đặc tả yêu cầu** của người dùng.
* Nhóm kiểm thử chỉ dựa trên đặc tả yêu cầu (yêu cầu chức năng và phi chức năng) mà không được xem mã nguồn hệ thống.
* Yc chức năng: các yc chỉ định các chức năng của HT hoặc các thành phần cần được thực hiện, nó chỉ định chi tiết những gì HT cần làm. Trong suốt quá trình kiểm thử chức năng, cần ktra chức năng như cài đặt, chạy thử ứng dụng trên máy cục bộ, HĐH cục bộ, ktra các chức năng của ứng dụng, các xử lý chuỗi, vb như copy, paste cho các ký tự mở rộng
* Yc phi chức năng: yc không liên quan đến **dịch vụ** được chỉ định, mà liên quan đến những **thuộc tính ràng buộc** hệ thống như tính tin cậy, tính hiệu quả, tính tiện dụng:
  + - * + Kiểm thử tính tin cậy (Reliabilty Tesing).
        + Kiểm thử tính tiện dụng (Usablilty Testing).
        + Kiểm thử tính hiệu quả (Efficiency Testing).
        + Kiểm thử khả năng bảo trì (Maintainablity Testing).
        + Kiểm thử tính khả chuyển (Portable Testing).
        + Kiểm thử hiệu năng (Performance Testing)
* **Kiểm thử hiệu năng**:
* xđ một số vấn đề về thắt cổ chai (bottleneck) hoặc hiệu năng của hệ thống
* Kiểm thử tính bền vững (Endurance Testing) ktra hành vi của HT với lượng tải đáng kể được thực hiện trong một khoảng thời gian dài. Sự tiêu thụ bộ nhớ thường là nguyên nhân dẫn đến failure của hệ thống.
* Scalability Testing kiểm thử hệ thống để đo khả năng mở rộng cho các yêu cầu phi chức năng.
* Volume Testing kiểm thử hệ thống với một lượng dữ liệu nhất định.
* Để ktra **khả năng chịu tải** của phần mềm thực hiện bằng cách chạy hàng loạt test, tăng dần sức tải cho đến khi hệ thống k thể thực thi được nữa, gọi là load test.
* Với cách tiếp cận này, test case thường được thiết kế xung quanh giá trị biên.
* Với cách này, ta kiểm tra được 2 thứ
  + Cách xử lý của hệ thống khi nó quá tải.
  + Phát hiện những lỗi mà không thể phát hiện ở các trường hợp thông thường.
* Việc kt HT với những giá trị ngoài phạm vi (abnormal conditions) đã được thiết kế gọi là ***stress test*** => **xác định những điểm gãy** (breaking point) của phần mềm.

1. **Acceptance Test** (Kiểm thử chấp nhậ)

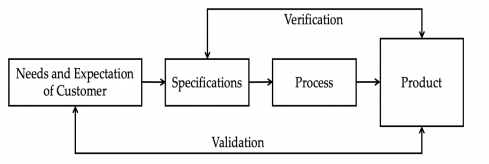
* Được thực hiện bởi ***khách hàng*** để đảm bảo sản phẩm phần mềm hoạt động đúng những gì khách hàng mong đợi và khách hàng chấp nhận sản phẩm – thanh toán hợp đồng.
* Với những sản phẩm có người dùng đa dạng trên thị trường thường có hai loại kiểm thử:
  + Alpha testing: người dùng kiểm tra phần mềm ngay môi trường phát triển **phần mềm**.
    - Thực hiện bởi một nhóm người trong tổ chức phát triển (in-house) phần mềm **độc lập** với nhóm phát triển (developer).
    - Developer qsát người dùng sd phần mềm, ghi nhận lỗi phát sinh để sửa chữa.
    - Giúp phát hiện ra vấn đề thực tế khi sd phần mềm mà nhóm phát triển chưa biết.
  + Beta testing: người dùng kiểm tra phần mềm trong môi trường thực của **người dùng**.
    - Tester: nhóm người từ khách hàng (e**xternal**).
    - Phần mềm cũng có thể công khai cho những người dùng quan tâm dùng thử.
    - Khách hàng ghi nhận lỗi và báo lại cho các developer.
    - Giúp phát hiện những vấn đề tương tác giữa phần mềm và những đặc trưng trong môi trường nó được sử dụng.
    - Beta testing được thực hiện sau alpha testing.
* Kiểm thử tĩnh
* Là các kỹ thuật kiểm thử không cần thực thi mã nguồn chương trình.
* Hai kỹ thuật được sử dụng thường xuyên trong kiểm thử tĩnh là đánh giá (Review) và phân tích tĩnh (Static Analysis). Review thường được thực hiện thủ công, static analysis thường được thực hiện tự động bằng các công cụ.
  + Kỹ thuật review để tìm và gỡ bỏ các lỗi, những điều chưa rõ trong các tài liệu trước khi chúng được sử dụng trong quy trình phát triển.
    - Kỹ thuật review kiểm thử bất cứ thứ gì được tài liệu hoá như đặc tả yêu cầu, thiết kế hệ thống, mã nguồn, kế hoạch kiểm thử, các test case. ▪
    - Có hai loại review
      * Formal review được thực hiện theo quy trình chính thức, có kế hoạch. (Planning, Kick-off, Preparation, Review Meeting, Rework, Follow-up)
      * Informal review khi thực hiện không xác định trước các bước, quy trình thực hiện (Walkthrought, Technical Review, Inspectation)
  + Kỹ thuật phân tích tĩnh (Static Analysis) cho phép mã nguồn được phân tích các vấn đề cấu trúc chương trình hoặc những điểm yếu của chương trình (Source Code), các mô hình phần mềm (Model) một cách hệ thống mà không cần thực thi chương trình.
    - Giúp phát hiện các vấn đề sớm, đưa ra những cảnh báo những điểm đáng nghi ngờ trong mã nguồn chương trình hoặc trong thiết kế bằng cách tính toán một số độ đo (Metrics) như độ phức tạp chương trình.
    - Giúp cải thiện khả năng bảo trì mã nguồn và các thiết kế của phần mềm.
    - Các vấn đề thường được phát hiện:
      * Tham chiếu đến biến nhưng gt chưa được chỉ định (Undefined).
      * Biến khai báo nhưng không được sử dụng.
      * Phát hiện các hàm và thủ tục không được gọi.
      * Vi phạm chuẩn lập trình.
      * Chương trình thiếu tính an toàn (Security).
      * Sử dụng các cú pháp không chính xác trong ngôn ngữ lập trình hay một ngôn ngữ thiết kế phần mềm đang sử dụng.
* Kiểm thử động
* Kiểm thử hộp đen (Black-box):
  + Kỹ thuật kiểm thử mà **không cần biết** những công việc bên trong của ứng dụng gọi là black-box testing.
  + Tester sẽ **tương tác với giao diện người dùng**, cung cấp dữ liệu vào và kiểm tra dữ liệu ra, không cần biết cách thức và nơi nào xử lý dữ liệu vào đó
* Kiểm thử hộp trắng (White-box):
  + Kỹ thuật kiểm thử white-box yc **biết logic và cấu trúc mã nguồn** bên trong CT.
  + Tester cần biết rõ những công việc bên trong mã nguồn, làm rõ từng đơn vị hoặc đoạn mã nguồn không phù hợp.
* Kiểm thử hộp xám (Gray-box)
  + Gray-box testing yc tester phải có chút ít hiểu biết công việc bên trong của ứng dụng.
  + Tester có thể truy cập vào tài liệu thiết kế và cơ sở dữ liệu. Với các tài liệu này, tester có thể chuẩn bị dữ liệu và kịch bản kiểm thử tốt hơn.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Black-box** | **Gray-box** | **White-box** |
| ***Cấu trúc bên trong CT*** | Không biết | Biết một ít | Biết đầy đủ |
| ***Người thực hiện*** | End-user | End-user Tester developer | Tester developer |
| ***Cơ sở thực hiện*** | Dựa trên đầu vào, đầu ra mong muốn | Dựa trên lược đồ cơ sở dữ liệu, sơ đồ luồng dữ liệu | Cấu trúc bên trong chương trình |
| ***Chi phí*** | Thấp | Trung bình | Cao |
| ***Kiểm tra thuật toán*** | Không phù hợp | Không phù hợp | Phù hợp |

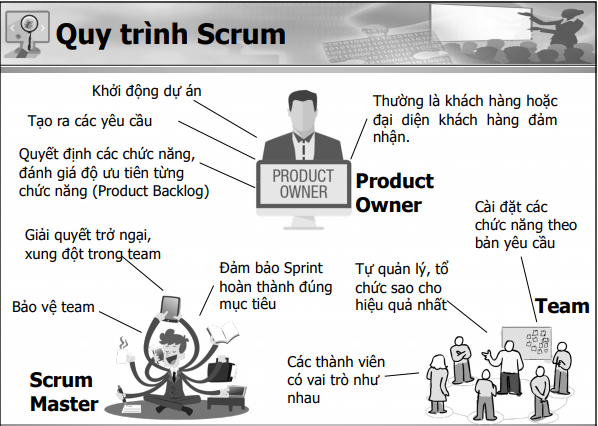
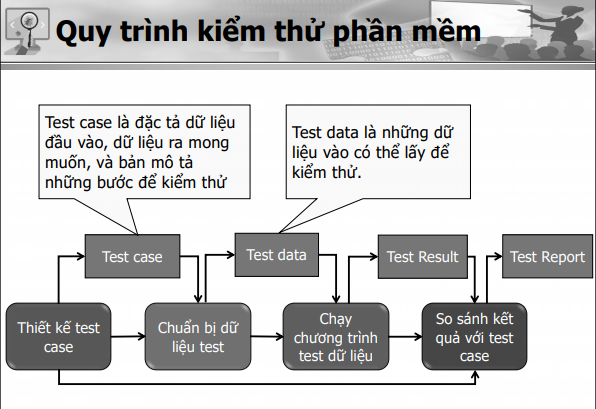
* Kiểm thử tĩnh và động:

|  |  |
| --- | --- |
| **Kiểm thử tĩnh** | **Kiểm thử động** |
| ▪ Xem mã nguồn, không thực thi.  ▪ Thực hiện trong quy trình Verfication.  ▪ Code inspection, walk through, code reviews, deskchecking | ▪ Thực thi mã nguồn.  ▪ Thực hiện trong quy trình Validation.  ▪ Black-box testing, white-box testing |

* Mô hình V&V
* Gồm **một dãy rộng các hđ trong đảm bảo chất lượng** phần mềm như technial reviews, documentation review, database review, algorithm analysis. Trong đó kiểm thử (testing) đóng vai trò cực kỳ quan trọng



* Verficiation đảm bảo sản phẩm phần mềm đúng với đặc tả yêu cầu của nó (right way) được thực hiện khi bắt đầu quy trình phát triển, bao gồm các buổi review để đánh giá các tài liệu, kế hoạch, mã nguồn, các tài liệu về yêu cầu và đặc tả yêu cầu.
  + Là quá trình đảm bảo sản phẩm phần mềm đúng với đặc tả của nó.
  + Thực hiện bởi developer
  + “Are we building the product right?
* Validation là quá trình đảm bảo sản phẩm phần mềm đáp ứng được yêu cầu của người dùng (user needs). Validation được thực hiện vào cuối của quy trình phát triển và sau khi verification được hoàn thành.
  + Là quá trình đảm bảo sản phẩm phần mềm đáp ứng được yêu cầu của người dùng.
  + Thực hiện bởi tester
  + “Are we building the right product?”
* V Model
* V-Model cũng là một mô hình quy trình phát triển phần mềm dựa trên sự kết hợp các giai đoạn kiểm thử cho mỗi giai đoạn tương ứng trong quy trình phát triển.
* Quá trình kiểm thử được thực hiện sau mỗi kết quả đạt được ở từng bước, điều này giúp phát hiện ra những vấn đề sớm nhất trong vòng đời phát triển của phần mềm.
* V-Model chỉ tập trung vào kiểm thử động, mà bỏ qua hiệu quả của kiểm thử tĩnh.
* W-Model khắc phục được điều có bằng cách cho phép thực hiện quá trình kiểm thử song song với quá trình phát triển phần mềm, các kỹ thuật kiểm thử tĩnh được áp dụng trong giai đoạn đầu của quy trình phát triển phần mềm, mỗi hoạt động trong quá trình phát triển đều có một hoạt động kiểm thử tương ứng.
* Test case là một tài liệu trong quá trình kiểm thử phần mềm được phát triển cho một kịch bản cụ thể để xác minh tính đúng đắn của một yêu cầu cụ thể. ▪
* Test case thể hiện cho một hành vi nào đó trong chương trình.
* Một test case gồm dữ liệu vào, các bước thực hiện và dữ liệu ra mong muốn.
* Số định danh test case (Test case ID)
* Kịch bản kiểm thử (Test scenario)
* Mô tả test case
* Các bước thực hiện kiểm thử
* Điều kiện tiên quyết (Prerequisite)
* Dữ liệu kiểm thử
* Kết quả mong muốn (expected result)
* Các tham số kiểm thử
* Kết quả thực sự (actual result)
* Thông tin môi trường kiểm thử
* Ghi chú (comment)



* Checklist là một danh sách các test case được thực thi trong một thủ tục xác định, nó giúp ta nhanh chóng nắm được các test case đã được thực thi hết hay chưa và có bao nhiêu test bị fail
* Test plan
* Tài liệu phác thảo chiến lược kiểm thử phần mềm như tài nguyên sử dụng, môi trường kiểm thử, phạm vi kiểm thử, lịch trình cho các hoạt động kiểm thử.
* Tài liệu này thường được phát triển trong giai đoạn đầu phát triển phần mềm

▪ Các giả thiết khi kiểm thử phần mềm.

▪ Giới thiệu tài liệu test plan

▪ DS các test case để k.thử phần mềm

▪ DS các đặc trưng (feature) sẽ đc k.thử.

▪ Danh sách sp (deliverabled) sẽ đc kthử.

▪ Xđ ưu tiên cách tiệp cận cho việc kthử.

▪ Tài nguyên sử dụng cho việc kiểm thử.

▪ Nêu các rủi ro có thể xảy ra trong quá trình kiểm thử

▪ Lịch trình task, milestone cần đạt đc

* Test report thể hiện kết quả kiểm thử chính thức.
* Tài liệu này lưu lại kết quả kiểm thử phần mềm một cách có tổ chức, mô tả điều kiện và môi trường kiểm thử, so sánh kết quả kiểm thử với mục tiêu kiểm thử đề ra.
* Test scenario đảm bảo các luồng xử lý từ đầu đến cuối. Một test scenario bao gồm nhiều bước được thực thi.
* Traceability Matrix
* Là một bảng để lưu vết (trace) các yêu cầu trong vòng đời phát triển phần mềm.
* Mỗi yêu cầu bảng này được liên kết đến một test case để quá trình kiểm thử được thực hiện. Ngoài ra, Bug ID cũng được liên kết đến yêu cầu này.
* Mục đích chính của bảng này
  + Đảm bảo phần mềm phát triển như yêu cầu đề cập.
  + Thuận tiện tìm nguyên nhân chính (root cause) gây ra bug.
  + Lưu vết các tài liệu được phát triển trong vòng đời phát triển phần mềm.

**CHƯƠNG 2: KIỂM THỬ HỘP ĐEN**

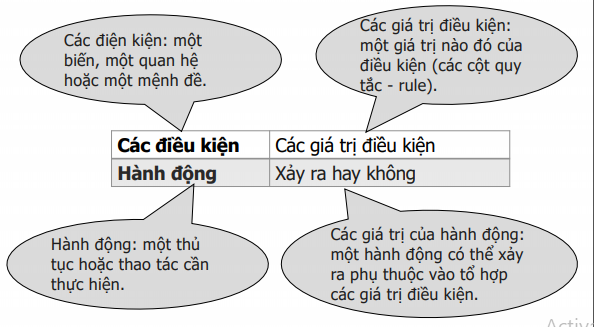
* Kiểm thử hộp đen (black-box testing) còn gọi kiểm thử **dựa trên đặc tả** (specification-based testing), thông tin **duy nhất** làm cơ sở để kiểm thử hộp đen là bảng đặc tả yêu cầu **chức năng** của từng thành phần phần mềm.
* Kiểm thử viên **tương tác với giao diện** người dùng, **cung cấp dữ liệu đầu vào, kiểm tra dữ liệu đầu ra**, không quan tâm mã nguồn, chi tiết các công việc bên trong, cách thức và nơi nào xử lý dữ liệu đầu vào đó

|  |  |
| --- | --- |
| **Ưu điểm** | **Khuyết điểm** |
| - Không cần truy cập mã nguồn.  - Tách biệt khung nhìn của user và developer.  - Nhiều người có thể tham gia test | - Kiểm thử không có hiệu quả cao  - Khó thiết kế test case, kiểm thử phủ được hết các trường hợp  - Không có định hướng kiểm thử rõ ràng |

* **Quy trình:**
  + Phân tích đặc tả chức năng của các thành phần phần mềm.
  + Thiết kế test case để kiểm thử.
  + Thực thi các test case để kiểm thử
  + So sánh kết quả đạt được với kết quả mong muốn trong từng test case.
  + Lập báo cáo kết quả kiểm thử.
* **Phân loại kiểm thử hộp đen:**
  + Kiểm thử chức năng (functional testing) kiểm thử chức năng chỉ định, tính an toàn của hệ thống (security testing), sự tương tác của hệ thống với các thành phần được chỉ định.
  + Kiểm thử phi chức năng (non-functional testing) kiểm thử các khía cạnh hành vi của hệ thống: tính tiện dụng, tính khả chuyển, hiệu năng hệ thống (load test hay stress test)
  + Kiểm thử có sự thay đổi:
    - Sau khi lỗi tìm thấy được giải quyết thì hệ thống cần được kiểm tra lại để xác nhận vấn đề đã thực sự được giải quyết: **retesting /confirmation testing**.
    - Khi mã nguồn hệ thống có thay đổi hoặc môi trường hệ thống thay đổi, ta cũng cần thực thi lại một tập các test case đã thiết kế trước đó, để đảm bảo không có lỗi mới phát sinh từ sự thay đổi đó: **kiểm thử hồi quy (regression testing).**
* **Các kỹ thuật kiểm thử hộp đen:**
  + Phân vùng tương đương (Equivalence Partitioning hoặc Equivalence Class), Phân tích giá trị biên (Boundary Value Analysis) **=>** áp dụng cho các tình huống các **đầu vào cụ thể và ít có quan hệ ràng buộc nhau.**
  + Bảng quyết định (Decision Tables hay còn gọi là Cause Effect), Dịch chuyển trạng thái (State Transition Testing) **=>** tập trung vào **logic** và **quy tắc (rule) nghiệp vụ**.
* **Phân vùng tương đương**
* Ý tưởng: nếu 1 gt đại diện trong nhóm đúng -> các gt còn lại trong nhóm cũng đúng và ngược lại. **PP đạt được độ phủ (coverage) hết đầu vào và đầu ra**
* Phương pháp phù hợp cho các bài toán có giá trị đầu vào là một *miền xác định*.
* Phân vùng tương đương (EP): phân chia một tập các điều kiện kiểm thử thành các tập con có các giá trị tương đương nhau và kiểm thử các tập con này.
* Mục đích: giảm thiếu số lượng test case không cần thiết khi kiểm thử.
* EP có thể áp dụng tất cả mức độ kiểm thử.
* Hai giá trị tương đương theo một trong các cách sau:
  + Chúng tương tự nhau (intuitive similarity). ▪
  + Tài liệu đặc tả mô tả ct xứ lý theo cùng 1 cách thức (specified as equivalent)
  + Chúng lái chương trình theo cùng đường (vd cùng nhánh if) (equivalent path).
  + Chúng cho cùng kết quả với những giả thiết đưa ra.
* **Phân tích giá trị biên** (Boundary Value Analysis)
* Phân tích giá trị biên (BVA): dựa các **giá trị tại biên** giữa các phân vùng tương đương, bao gồm trường hợp: ▪ Hợp lệ (valid). ▪ Không hợp lệ (invalid).
* **Quy tắc** xác định giá trị biên khi thiết kế test case ▪ Giá trị nhỏ nhất (biên dưới) ▪ Giá trị nhỏ thứ hai (cận biên dưới) ▪ Giá trị lớn nhất (biên trên) ▪ Giá trị lớn thứ hai (cận biên trên) ▪ Giá trị bình thường

|  |  |
| --- | --- |
| Ưu điểm | Khuyết điểm |
| - Đơn giản.  - Hiệu quả cho các hàm có biến độc lập.  - Có thể tự động sinh test case khi xác định được giá trị biên của các biến | - K quan tâm đặc trưng của hàm, ngữ nghĩa các biến, cũng như quan hệ giữa các biến.  - Khó áp dụng cho trường hợp các biến có quan hệ ràng buộc nhau. |

* **Bảng quyết định**:
* Bảng quyết định (DT): xác định **những kịch bản** (scenario) kiểm thử cho những trường hợp có logic nghiệp vụ phức tạp.
* DT giúp tester xác định hiệu quả sự **kết hợp các đầu vào khác nhau** với các tình trạng phần mềm thực thi đúng quy tắc nghiệp vụ.
* DT nên được sử dụng trong những chương trình có **nhiều lệnh rẽ nhánh** và **các biến đầu vào có mối quan hệ** với nhau.



* Xây dựng bảng quyết định:
  + Liệt kê tất cả **các điều kiện/đầu vào**.
  + Tính **số lượng kết hợp** các giá trị của các điều kiện/ đầu vào và đặt các kết hợp đó vào trong phần giá trị các điều kiện.
  + **Xác định các test case** tương ứng cho các điều kiện được thỏa mãn.
  + **Các hành động** chính là kết quả mong đợi của test case.
* Các quy tắc chuyển từ bảng quyết định thành các test case:
  + Nếu giá trị các điều kiện nhập là các giá trị **luận lý** (true/false) thì **mỗi cột** quy tắc được chuyển thành **một test case**.
  + Nếu các giá trị đk nhập có **nhiều giá trị** thì **mỗi cột** quy tắc được chuyển thành **nhiều test case** sd kỹ thuật phân vùng tương đương hoặc phân tích giá trị biên.
* **Chuyển đổi trạng thái**:
* Các khía cạnh của hệ thống được mô tả thông qua **máy trạng thái hữu hạn**.
  + Hệ thống xử lý cùng đầu vào, nhưng cho kết quả đầu ra khác nhau (máy trạng thái hữu hạn). Máy trạng thái hữu hạn đc biểu diễn giống **lược đồ trạng thái**.
* Hệ thống có nhiều trạng thái khác nhau, sự dịch chuyển từ một trạng thái này sang trạng thái khác được **quyết định** bởi **một sự kiện** nào đó.
* Một **mô hình chuyển đổi trạng thái** có 4 phần cơ bản:
  + Các trạng thái (**states**) phần mềm có thể xảy ra.
  + Sự dịch chuyển (**transitions**) từ trạng thái này sang trạng thái khác.
  + Các sự kiện (**events**) dẫn đến sự dịch chuyển trạng thái.
  + Các hành động (**actions**) là kết quả của việc dịch chuyển trạng thái.
* **Kiểm thử dựa trên use-case**
* Hữu dụng trong việc thực hiện các **quy tắc nghiệp vụ hoặc các luồng xử lý**, phát hiện được lỗ hổng, điểm yếu của HT khi kiểm thử từng thành phần riêng rẽ k phát hiện ra.
* Nền tảng phát triển các test case trong cấp độ **kiểm thử hệ thống** & **kiểm thử chấp nhận** do các use case đại diện cho khả năng **sử dụng thực sự** của người dùng.
* Lược đồ use case: mô tả các chức năng gì của HT được thực hiện bởi actor nào và vai trò của actor trong HT. Giữa hai actor có thể có mqh tổng quát hoá và chuyên biệt hoá.
* 1 đặc tả use case có các TT: -Use case id (định danh use case) -Tên use case -Mô tả vắn tắt mục đích use case -Các actor chính -Các actor phụ -Tiền điều kiện (Pre-conditions) -Hậu điều kiện (Post-conditions) -Luồng hoạt động chính (Main Flows) -Luồng hoạt động thay thế (Alternative flows) -Luồng hoạt động ngoại lệ (Exception flows)
* **Kiểm thử dựa trên kinh nghiệm**
* **Đoán lỗi** (error guessing) áp dụng sau khi tất cả các kỹ thuật chính thức được áp dụng.
  + Dựa trên kỹ năng, kinh nghiệm, trực giác của kiểm thử viên đã kiểm thử các ứng dụng tương tự nhằm xác định những test case đặc biệt không được phát hiện khi áp dụng các kỹ thuật chính thức trước đó.
* **Kiểm thử thăm dò** (exploration testing) làm rõ hệ thống có gì, không có gì, cái nào làm việc tốt, cái nào làm việc chưa tốt để kiểm thử viên có cơ sở đưa ra quyết định những gì cần kiểm thử tiếp theo.
  + Kỹ thuật tiếp cận có cấu trúc kết hợp kiến thức, sự quan sát, kinh nghiệm của các kiểm thử viên để kiểm thử những chổ đặc tả yêu cầu bị thiếu, không đầy đủ (poor specifications) hoặc có nhiều áp lực thời gian thực hiện.
  + Tập trung thực thi kiểm thử hơn là việc lên kế hoạch kiểm thử. Việc thiết kế test case và thực thi test case được thực hiện song song, và cũng không cần một tài liệu chính quy về test condition, test case hoặc test script.
  + Thực hiện trong các **giai đoạn đầu** của vòng đời phát triển phần mềm khi mã nguồn trải qua những thay đổi nhanh chóng
  + Lập trình viên sử dụng kiểm thử thăm dò để thực hiện các Unit Test, còn kiểm thử viên có thể làm quen hệ thống bằng kỹ thuật này.

**CHƯƠNG 3: KIỂM THỬ HỘP TRẮNG**

* Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing hoặc Structural Testing là một pp kiểm thử phần mềm trong đó tester biết về **cấu trúc** nội bộ / thiết kế

|  |  |
| --- | --- |
| **Ưu điểm** | **Khuyết điểm** |
| - **Dễ xác định loại dữ liệu** để kiểm tra, nên việc kiểm tra hiệu quả hơn.  - Nhờ biết mã nguồn, nên tester **có** **thể phủ tối đa** khi viết kịch bản kiểm thử. | - **Chi phí tăng** vì tester cần có kỹ năng đọc và hiểu mã nguồn.  - Khó xét hết các ngõ ngách trong mã nguồn nên **có thể sót đường dẫn** không được test |

* **Đồ thị luồng** (flow graph): đồ thị có hướng, dùng mô tả ***luồng điều khiển logic***.
  + Gồm 2 thành phần cơ bản: ***các đỉnh*** tương ứng với các lệnh/nhóm câu lệnh và ***các cạnh*** kết nối tương ứng với dòng điều khiển giữa các câu lệnh/nhóm câu lệnh.
  + **Đồ thị luồng** được xây dựng dựa trên sơ đồ **luồng điều khiển** (flow chart).
* **Đường dẫn độc lập**
  + Đường thông qua CT có ít nhất một tập các ***câu lệnh xử lý mới*** hoặc ***điều kiện mới***.
  + Trong đồ thị luồng, đường dẫn mới sẽ có cạnh mới.
* **Độ phức tạp Cyclomatic**
  + Độ đo phần mềm cung cấp thước đo **định lượng độ phức tạp về mặt logic** của CT.
  + Là **số lượng các đường độc lập** trong tập cơ sở của CT và cung cấp **chặn trên** (upper bound) số lượng các test case được thực thi **đảm bảo phủ đường dẫn cơ sở** của CT.
* **Kiểm thử đường dẫn cơ** **sở**
  + Đảm bảo các **đường dẫn độc lập** được kiểm thử qua ít nhất một lần.
  + Các bước thực hiện: ▪ Vẽ đồ thị luồng G. ▪ Tính độ phức tạp Cyclomatic V(G). ▪ Xác định tập cơ sở các đường dẫn độc lập. ▪ Viết test case thực thi mỗi đường dẫn trong tập cơ sở.
* **Kiểm thử cấu trúc điều khiển**
* Kiểm thử luồng điều khiển (Control Flow Testing) hoặc bao phủ (Coverage Testing)
* Kiểm thử luồng dữ liệu (Data Flow Testing)
* Kiểm thử vòng lặp (Loop Testing)
* **Kiểm thử bao phủ**
  + Dùng kiểm tra mức độ phủ (coverage) của các test case.
  + Các loại kiểm thử bao phủ:
    - Phủ câu lệnh (statement coverage): **mỗi** **câu lệnh** đc thực thi ít nhất một lần
    - Phủ nhánh (branch coverage): **mỗi** **nhánh** phải đc thực hiện ít nhất một lần.
      * Phủ nhánh đảm bảo **phủ câu lệnh**.
      * Trong đồ thị luồng
        + Phủ nhánh có nghĩa là **các cạnh được đi qua ít nhất một lần**.
        + Để phủ nhánh phải thiết kế dữ liệu kiểm thử sao cho mỗi **nút vị từ** (predicate) xảy ra tất cả các kết quả (true/false) có thể của nó, phủ nhánh còn gọi là **phủ quyết định** (decision coverage).
    - Phủ đường (path coverage): ***mỗi đường dẫn*** qua ít nhất một lần
      * Đường là một tập các nhánh, nên **phủ đường** chắc chắn **phủ nhánh**, nhưng ngược lại chưa chắc đúng.
    - Phủ điều kiện (condition coverage): **mỗi điều kiện** **trong các vị từ** đc thực hiện ít nhất một lần cho cả true và false (k bắt buộc các kết hợp giữa chúng).
    - Phủ đa điều kiện (multi-conditions coverage): mỗi đk trong biểu thức vị từ và **các kết hợp** giữa chúng đc thực hiện ít nhất 1 lần cho trường hợp true & false.
* **Kiểm thử luồng dữ liệu**
  + Kiểm thử **vòng đời của biến** trong từng luồng thực thi của chương trình
  + Vòng đời của một biến được thể hiện thông qua ba hành động: ▪ Định nghĩa biến (Define). ▪ Sử dụng biến (Use). ▪ Xóa biến (Delete).
  + Lựa chọn các đường dẫn để kiểm thử dựa trên **vị** **trí** định nghĩa (**define**) và sử dụng (**use**) các biến trong chương trình.
  + **DEF(v, n)** biến v được định nghĩa trong câu lệnh nào đó của nút n trong đồ thị luồng. Các lệnh nhập, lệnh gán, lệnh gọi thủ tục là các lệnh định nghĩa biến.
  + **USE(v, n)** giá trị của biến v được sử dụng trong một lệnh nào đó của nút n trong đồ thị luồng. Các lệnh xuất dữ liệu, tính toán, các biểu thức điều kiện, các lệnh trả về trong hàm, phương thức là các lệnh sử dụng biến.
    - Nếu câu lệnh sd biến v là biểu thức ***vị từ*** thì ký hiệu là **p-use** (predicate use).
    - Nếu câu lệnh sd biến v là biểu thức ***tính toán*** thì KH là **c-use** (computation use).
  + Kiểm thử luồng dữ liệu giúp phát hiện các vấn đề sau:
    - Một biến được **khai báo, nhưng không sử dụng**.
    - Một biến **sử dụng nhưng không khai báo**.
    - Một biến được **định nghĩa nhiều lần** trước khi được sử dụng.
    - **Xóa biến trước khi sử dụng**.
  + Cho biến x € DEF(S) ⊂ USE(S’), trong đó S và S’ là các câu lệnh. ▪ **Đường DC** (definition-clear path) của biến x là đường nối từ S đến S’ trên đồ thị luồng sao cho không tồn tại một định nghĩa nào khác của x trên đường này. ▪ **Cặp DU** (definition-use pairs) của biến là cặp S và S’, sao cho tồn tại ít nhất một đường DC nối S và S’.
* **Kiểm thử vòng lặp:** kiểm tra **tính hợp lệ** của cấu trúc vòng lặp
  + Thời điểm kiểm thử: ▪ Lúc vừa vào vòng lặp. ▪ Lúc đang xử lý trong vòng lặp. ▪ Lúc rời khỏi vòng lặp
  + Có 4 loại cấu trúc vòng lặp
    - Lặp đơn giản (simple loop)
    - Lặp lồng nhau (nested loop)
      * Bắt đầu test vòng lặp ***trong cùng*** (innermost) và các vòng lặp khác thiết lập 1 giá trị tối thiểu.
      * Dùng chiến lược kiểm thử vòng lặp **đơn giản** cho vòng lặp trong cùng và giữ các vòng lặp ngoài ở giá trị tối thiểu.
      * Thực hiện tương tự cho các vòng lặp ngoài, cho đến khi vòng lặp ngoài cùng (outermost) được test.
    - Lặp nối tiếp (concatenated loop)
      * Vòng lặp là ***độc lập*** thì AD cách tiếp cận cho các vòng lặp ***đơn giản***.
      * Vòng lặp ***phụ thuộc*** thì AD cách tiếp nhận cho các vòng lặp ***lồng nhau***.
    - Lặp không cấu trúc (unstructured loop): y/c thiết kế lại CTđể đảm bảo tính cấu trúc của CT.

**CHƯƠNG 4: KIỂM THỬ TỰ ĐỘNG**

* **Automation Testing** là kỹ thuật ***sử dụng công cụ*** nào đó **để thực thi các test case một cách tự động** và đưa ra kết quả kiểm thử một cách nhanh chóng.
  + Đặc biệt có hiệu quả trong **kiểm** **thử hồi quy** (regression testing).
* **Mục đích**: giảm thời gian, chi phí, tăng độ tin cậy, nâng cao hiệu suất công việc, giảm sự nhàm chán cho kiểm thử viên.
* Một số **công cụ** kiểm thử: Selenium, LoadRunner, jMeter, jUnit/NUnit, Quick Test Professional (HP), Katalon Studio
* **Selenium**: hđ trên nhiều trình duyệt và hệ điều hành khác nhau
  + Là công cụ **mã nguồn mở** dùng cho kiểm thử tự động các **ứng dụng web** được phát triển bởi ThoughWorks năm 2004.
  + Các công cụ của Selenium
    - **Selenium IDE**: là 1 plugin trình duyệt Firefox và Chrome. Nó có thể ghi nhận những hành động cần kiểm thử.
      * Mở Mozilla FireFox > Tools > Add-ons (Ctrl+Shift+A). ▪ Trong cửa sổ Add-ons Manager, tìm từ khóa Selenium IDE.
    - **Selenium RC**: là 1 framework cho phép thực hiện nhiều hành động trên trình duyệt và thực thi tuyến tính, có thể dùng trong nhiều ngôn ngữ lập trình như Java, C#, Python, PHP.
    - **Selenium WebDriver**: cho phép gửi lệnh trực tiếp lên trình duyệt và tìm kiếm kq
      * Các thuộc tính của webdriver
        + url: thiết lập/hoặc lấy url của trang web.
        + title: lấy chủ đề trang web.
        + page\_source: lấy mã nguồn HTML của trang web
      * PT của webdriver
        + close(): đóng cửa sổ webdrive đang điều khiển.
        + quite(): đóng tất cả các cửa sổ mở bởi IWebDriver.
      * Cung cấp hai loại chờ: implicit và explicit.
        + Explicit Wait: chờ 1 đk thoả mãn trc khi thực hiện 1 việc nào đó.

Mặc định kt đkmong muốn (ExpectedCondition) mỗi 500 mili giây để tìm thành phần mong muốn.

Nếu sau khoảng thời gian chờ thành phần cần tìm chưa có thì ngoại lệ **TimeoutException** sẽ được ném ra.

* + - * + Implicit Wait: chờ 1 thành phần DOM trong khoảng tg nào đó để thử lấy nó.
    - **Selenium Grid**: công cụ dùng chạy các test song song trên nhiều máy và nhiều trình duyệt khác nhau một cách đồng thời để tối ưu thời gian thực thi.
* Tương tác HTML DOM
* **find\_element** trả về thành phần đầu tiên tìm thấy trong cây HTML DOM. Nếu không tìm thấy, nó sẽ ném ngoại lệ **NoSuchElementException**.
* **find\_elements** trả về một danh sách các thành phần tìm thấy trong cây HTML DOM, và sẽ trả về **danh sách rỗng** nếu không tìm thấy
* Điền văn bản vào ô nhập liệu: element.send\_keys(text)
* Xoá giá trị ô nhập liệu: element.clear()
* Giả lập bấm phím:
  + from selenium.webdriver.common.keys import Keys
  + element.send\_keys(Keys.RETURN)
* Submit form: element.submit()

