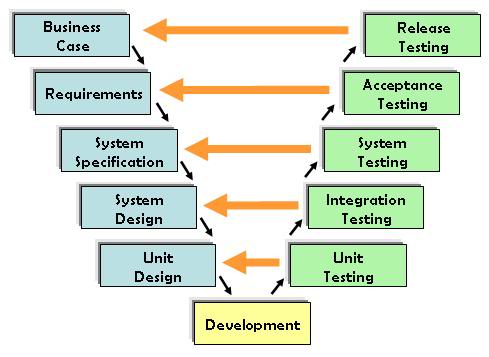
Giới thiệu quy trình phát triển dự án phần mềm

Đây là mô hình phát triển dự án



Mũi tên màu đen là hành động triển khai tiếp theo. Màu cam thể hiện triển khai có vấn đề thì quay lại triển khai lại quá trình trước đó

Các ô màu xanh thẫm thể hiện đặc tính tài liệu

Các ô màu xanh lá cây thể hiện đặc tính test.

Nếu có lỗi thành phần nào thì phải quay lại đặc tính tài liệu đó!

Nào chúng ta đi sâu tập trung chi tiết nhé, để hiểu rõ bạn nên dịch tiếng anh sang tiếng việt dễ hiểu hơn nhé:

Bussiness Case : Bài toán kinh doanh cụ thể

Requirement : Yêu cầu dự án

Viết Specfication : Miêu tả, mô tả hình dung về dự án

Viết Design : Thiết kế cơ bản

Viết Detail Design: Thiết kế chi tiết

Development : phát triển

Unit test : Kiểm thử đơn vị

IT Testing : kiểm thử tích hợp

System Testing : kiểm thử hệ thống

Acceptance Test : kiểm thử để chấp nhận

Release Testing : Test để xuất bản , đề xuất ra thị trường

Nào chúng ta đi vào chi tiết làm sao viết cho chuẩn nhé

Viết Bussiness Case như thế nào

Một đề án kinh doanh (Businness Case) là bản điều tra, ghi nhận hoặc tài liệu nhưng bắt buộc thật đơn giản dễ hiểu để người đọc có thể nắm được nội dung cụ thể. Tránh biệt ngữ,ngôn ngữ mang tính chuyên ngành cao và cung cấp một lời giải thích, chú giải rõ ràng.

Bạn nên xem xét Businness Case có đạt yêu cầu này không:

Đối tượng áp dụng có rõ ràng không?

Các nội trong đề án có cần được làm rõ không?

Đề xuất và các mục tiêu yêu cầu có khả thi không?

Những rủi ro và kế hoạch giảm thiểu có được làm rõ?

Những người đại diện chính có hỗ trợ gì không?

Đội nhóm có khả năng thực hiện nó không?

SỰ CHẤP THUẬN

Nếu nó có giá cả phù hợp với ngân sách thì Businness Case đạt yêu cầu, nếu không thì phải thương thảo lại các yêu cầu cho phù hợp ngân sách và năng lực.

Requirement

Requirement là gì, nhiều người sẽ nói “Requirement là balalala “. Đơn giản dịch tiếng anh là yêu cầu cụ thể là thôi. Không lằng nhằng

Đơn chức năng : là chức năng cơ bản, không liên quan tới chức năng khác

* Ví dụ : chiếc điện thoại đi động cơ bản (không có camera) có chức năng đèn pin => chức năng này không ảnh hưởng tới chức năng gọi điện, hay gửi tin nhắn

Đa chức năng : là chức năng có nhiệm vụ liên quan tới nhau

* Ví dụ : chiếc điện thoại đi động thông minh có chức năng đèn pin => chức năng này liên quan đến chức năng chụp hình(đèn LED chụp hình)

chức năng tiềm tàng rủi ro: chức năng có nguy cơ xung đột nếu phát quyền ưu tiên hoặc chức năng không có trong tài liệu nhưng bắt buộc phải có .

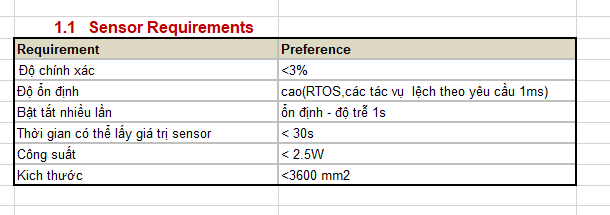
* Ví dụ - chức năng

Thế nào là 1 requiment chuẩn: Cái này thực ra trả lời rất khó vì phải phụ thuộc từng công ty, từng văn hóa và chính người sếp của bạn nữa!

Chúng tôi xin đề xuất định nghĩa 1 requment chuẩn như sau:

* Đơn giản
* Dễ hiểu
* Cụ thể,ngăn gọn
* Có chỉ tiêu cụ thể rõ ràng

Đây là ví dụ về 1 Requment đơn giản:



|  |  |
| --- | --- |
| Viết Specfication | Quy chuẩn |
| cơ bản |
| Phức tạp |
| Case abnomal |
| case khách hàng không nhìn thấy đượ |

Định nghĩa Specification

Specification là đặc tả,mô tả ( dịch tiếng anh), nó sẽ mô tả hoạt động của bài toán

Quy chẩn đặc tả cũng như Requirement

thu thập yêu cầu (requirements gathering, requirements capture), hoặc đặc tả yêu cầu (requirements specification) có tính đo được, kiểm thử được, có liên quan đến các nhu cầu hoặc cơ hội doanh nghiệp đã được xác định, và các yêu cầu phải được định nghĩa ở một mức độ chi tiết đủ cho việc thiết kế hệ thống.

Về khái niệm, việc phân tích yêu cầu bao gồm ba loại hoạt động sau:

Làm rõ yêu cầu (Eliciting requirements): giao tiếp với khách hàng và người sử dụng để xác định các yêu cầu của họ.

Xem xét yêu cầu (Analyzing requirements): xác định xem các yêu cầu được đặt ra có ở tình trạng không rõ ràng, không hoàn chỉnh, đa nghĩa, hoặc mâu thuẫn hay không, và giải quyết các vấn đề đó.

Làm tài liệu yêu cầu (Recording requirements): các yêu cầu có thể được ghi lại theo nhiều hình thức, chẳng hạn các tài liệu ngôn ngữ tự nhiên, các tình huống sử dụng (use case), câu chuyện sử dụng (user story), hoặc các đặc tả tiến trình.

Phân tích yêu cầu có thể là một quá trình dài và khó khăn, cần đến nhiều kĩ năng tâm lý khéo léo. Các hệ thống mới làm thay đổi môi trường và các mối quan hệ giữa con người, do đó điều quan trọng là phải xác định được tất cả những người có vai trò quan trọng, xem xét tất cả các nhu cầu của họ và đảm bảo rằng họ hiểu được các hàm ý của hệ thống mới. Các nhà phân tích có thể sử dụng một số kĩ thuật để làm rõ các yêu cầu của khách hàng. Trong lịch sử, các kỹ thuật này bao gồm các cuộc phỏng vấn, thành lập các nhóm trọng tâm (focus group) với các cuộc họp bàn về yêu cầu (requirements workshops), và tạo ra các danh sách yêu cầu. Các kỹ thuật hiện đại hơn gồm có tạo nguyên mẫu (prototyping), và tình huống sử dụng. Khi cần thiết, nhà phân tích sẽ kết hợp các phương pháp này để thiết lập các yêu cầu chính xác của những người có vai trò quan trọng, nhằm mục đích xây dựng một hệ thống thỏa mãn các yêu cầu doanh nghiệp.

Trong cuốn Rapid Development, Steve McConnell đã liệt kê một loạt các khả năng người dùng có thể cản trở quá trình thu thập yêu cầu:

Người dùng không hiểu họ muốn gì

Người dùng không tuân theo một bộ yêu cầu đã được tài liệu hóa

Người dùng nhất định đòi hỏi các yêu cầu mới sau khi chi phí và kế hoạch phát triển đã được hoạch định xong.

Mức độ giao tiếp với người dùng là thấp

người dùng thường không tham gia các đợt thẩm định hoặc không thể tham gia.

Người dùng không hiểu kỹ thuật

Người dùng không hiểu quy trình phát triển.

Những điều này có thể dẫn tới tình huống khi yêu cầu người dùng liên tục thay đổi ngay cả khi việc phát triển hệ thống hay sản phẩm đã được bắt đầu.

Vấn đề về kỹ sư/nhà phát triển

Trong quá trình phân tích yêu cầu, các vấn đề sau có thể nảy sinh từ phía các kỹ sư và nhà phát triển:

Nhân viên kỹ thuật và người dùng cuối có thể có ngôn từ khác nhau. Kết quả là họ có thể tin rằng họ hoàn toàn đồng thuận cho đến khi sản phẩm hoàn thiện được đưa ra.

Các kỹ sư và nhà phát triển có thể cố lái cho các yêu cầu khớp với một hệ thống hay mô hình sẵn có, thay vì phát triển một hệ thống theo sát nhu cầu của khách hàng

Việc phân tích có thể do các kỹ sư hoặc lập trình viên thực hiện, thay vì các nhân viên có kỹ năng và kiến thức miền ứng dụng để có thể hiểu các nhu cầu của khách hàng một cách đúng đắn

Giải pháp đã được thực hiện

Một giải pháp đối với các vấn đề về giao tiếp là thuê các chuyên gia về doanh nghiệp hoặc chuyên gia phân tích hệ thống.

Các kỹ thuật được đưa ra trong thập kỷ 1990 như tạo nguyên mẫu, UML, tình huống sử dụng. và phát triển phần mềm linh hoạt (Agile software development) cũng đã được dùng làm giải pháp cho các vấn đề trên.

cơ bản : là case rõ ràng, nội dung nó không ảnh hưởng yêu cầu khác

Phức tạp : là case không rõ ràng, nội dung nó không ảnh hưởng yêu cầu khác

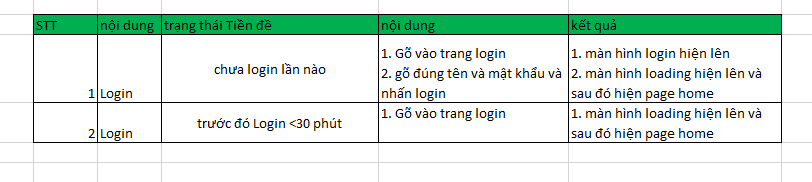
Case abnomal là case có vấn đề đặc biệt nghiêm trọng ảnh hưởng tới chu kỳ hoạt động bình thường của yêu cầu đặt ra

case khách hàng không nhìn thấy được . Nó là case cơ bản , hoặc phức tạp, hoặc abnormal nhưng trong quá trình thực hiện sẽ phát sinh ra vấn đề

Định nghĩa như vậy giúp các bạn đơn giản hóa vấn đề và tính giá tốt hơn

Quy chuẩn viết tài liệu:

Cũng giống như Requement tài liệu mô tả phải thể hiện ba đặc tính , tiền đề < trước đó đã là gì> hiện tại < làm cái gì > và kết quả mong muốn < tương lai>



ở đây là ví dụ về Login của 1 website thông thường

nếu trước đó chưa login lần nào thì phải gõ thông tin đăng nhập thì mới vào màn hình home, còn nếu đã login trước đó vào trang login thì màn hình chuyển sang màn hình home với ý nghĩa tư động login nếu thời gian hoạt động login trước đó <30 phút

|  |  |
| --- | --- |
| Viết Design | Design là gì |
| Các loại design và ứng dụng |
| Thiết kế design va kinh nghiệm |

Design là gì

“ Design chính là xây dựng một ý tưởng trở thành một sản phẩm tốt “

Ơ định nghĩa này có ổn không nhỉ => Design chỉ đơn giản là thiết kế mà thôi, quan trọng thiết kế chuẩn hay không phụ thuộc kinh nghiệm , trình độ người thiết kế. Vì vậy ông nào Dev càng lâu năm thiết kế càng chuẩn ( hình như có 1 chút mùi vị hơi sai sai)

Nói ngắn gọn lại design là vẽ ra mô hình để hiểu đơn giản nhất

Vậy trong lập trình và tạo ra sản IOT thiết kế như nào mới chuẩn!



<https://backlog.com/git-tutorial/vn/stepup/stepup2_5.html>

Các loại design và ứng dụng

Design phần mềm

Thiết kế giao diện

Thiết kế Giao tiếp, hoạt động cơ bản

Deign phần cứng

Thiết kế mạch

Thiết Kế Sản Phẩm Công Nghiệp (Industrial Product Design)

Thiết Kế Nội Thất (Interior Design)

Thiết Kế Thời Trang (Fashion Design)

Thiết Kế Kiến Trúc (Architectural Design)

Design quy trình : đây là design tầm cấp cao

Thiết kế design va kinh nghiệm:

B1: định nghĩa vấn đề của mình là gì

B2: kiểm tra yêu cầu

B3: vẽ desin ra

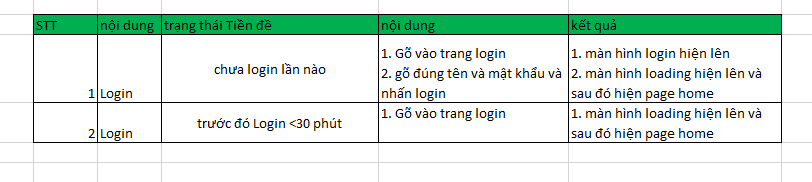
B4: kiểm tra lại yêu cầu

B5: quay lại B3 nếu có lỗi

Ví dụ cụ thể

Hệ thống quản lý sinh viên

Yêu cầu; quản lý , truy xuất, thêm, xóa sinh viên



B1: định nghĩa vấn đề là 1 máy tính kết nối với 1 máy chủ lưu trữ qua internet

B2: ờ thành phần chỉ 1 máy tính thôi nhỉ, với server lưu trữ

B3: vẽ desin ra

Server

Máy tính

B4: kiểm tra lại yêu cầu => hình như phải qua internet mà

B5: quay lại B3 nếu có lỗi => ờ lỗi cmnr rồi quay lại bước 3 thui

B3: vẽ desin ra

Server

Internet

Máy tính

B4: kiểm tra lại yêu cầu => đúng nhỉ

B5: ok rồi, xong

|  |  |
| --- | --- |
| Viết Detail Design | detail Design là gì |
| ứng dung |
| Thiết kế detail design va kinh nghiệm |

detail Design là gì

Đơn giản chỉ là thiết kế chi tiết đến mức sát với hoạt động ứng dụng nhất!

Thiết kế chi tiết như mức nào nhỉ!Câu hỏi thật khó phải không. À mình chỉ ra cho các bạn nhé

Đơn giản là thiết kế chi tiết đến mức dễ hiểu, người phát triển có thể đoán được và triển khai. Cụ thể như nào nhỉ!

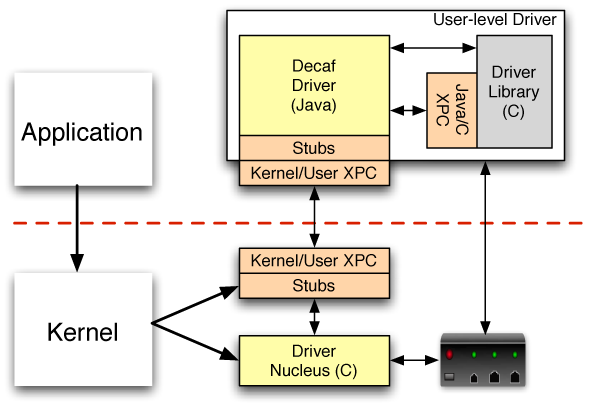
Ơ Cái này mình không biết đâu! Tùy trình độ lập trình!

Nhưng kinh nghiệm hệ thống IOT chia 4 loại design

* Event : lập trình web, mobile , hoặc chương trình có thể dễ dàng mô tả các event thì design không cần quá chi tiết
* Process: Quá trình lập trình bug sẽ chồng bug không như event , bug dễ dàng phát hiên, nên design thật chi tiết cụ thể, từng hành vi một, trạng thái, định nghĩa
* Mạch : thiết kế chi tiết theo từng khối và từng nội dung một
* Vỏ, kiểu dáng công nghiệp: mình không dám nói nhiều thêm, cái này dành cho các bạn cơ khí tự định nghĩa thêm nhé

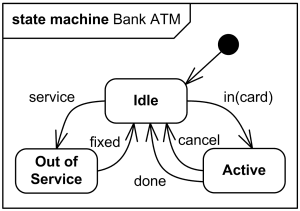
Các loại detail Design

* Dạng mô hình quan hệ



Dạng mô hình quan hệ là mối quan hệ thành phần chính với các thành phần khác mặc dù chúng ta không thể biết chi tiết quan hệ chúng với nhau:

Áp dụng cho design cơ bản, design ứng dụng ,design hệ thống



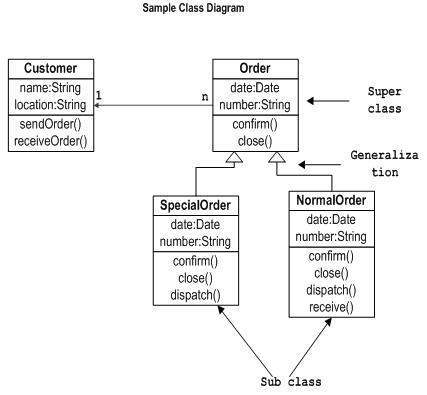
State machine là

State machine là một mô hình quản lý quá trình chuyển trạng thái. Nó được ứng dụng rộng rãi trong nhiều ứng dụng. Một trong đó là về cài đặt các workflow.

**Máy trạng thái hữu hạn** (finite-state machine **FSM**) hoặc **Máy tự động trạng thái hữu hạn** (finite-state automaton **FSA**), hoặc là**máy tự động hữu hạn**, hoặc gọi đơn giản là **máy trạng thái**, là một [mô hình tính toán toán học](https://vi.wikipedia.org/wiki/M%C3%B4_h%C3%ACnh_t%C3%ADnh_to%C3%A1n). Nó là một [máy trừu tượng](https://vi.wikipedia.org/w/index.php?title=Abstract_machine&action=edit&redlink=1) luôn có trạng thái nằm trong tổng hữu hạn các [trạng thái](https://vi.wikipedia.org/wiki/Tr%E1%BA%A1ng_th%C3%A1i_(khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh)) tại bất kỳ thời điểm nào. Máy trạng thái hữu hạn có thể chuyển từ trạng thái này sang trạng thái khác để phù hợp với [đầu vào](https://vi.wikipedia.org/wiki/%C4%90%E1%BA%A7u_v%C3%A0o_(m%C3%A1y_t%C3%ADnh)); sự thay đổi này được gọi là *quá trình chuyển đổi*. Máy trạng thái hữu hạn được xác định bởi danh sách các trạng thái của nó, trạng thái khởi đầu, và các điều kiện cho từng sự chuyển đổi trạng thái.

Hành vi của máy trạng thái có thể được quan sát qua nhiều thiết bị hiện đại, đó là việc thực hiện một chuỗi các hành động định trước tùy vào chuỗi sự kiện mà chúng được lập trình.

**Class diagram**



**Class Diagram là một trong những bản vẽ quan trọng nhất của thiết kế phần mềm, nó cho thấy cấu trúc và quan hệ giữa các thành phần tạo nên phần mềm. Trong quá trình xây dựng Class Diagram chúng ta sẽ phải quyết định rất nhiều yếu tố về thiết kế nên nó là bản vẽ khó xây dựng nhất. Bản vẽ này sẽ cho thấy cấu trúc tĩnh của phần mềm, tương tự như bản vẽ mặt bằng trong thiết kế của ngành xây dựng.**

**User case diagram**

Đầu tiên Use Case là một technique của [công việc Business Analyst](https://thinhnotes.com/chuyen-nghe-ba/business-analyst-la-gi-va-lam-nhung-gi/).

Use Case là kỹ thuật dùng để ***mô tả sự tương tác*** giữa người dùng và hệ thống với nhau, trong một ***môi trường cụ thể*** và vì một ***mục đích cụ thể.***

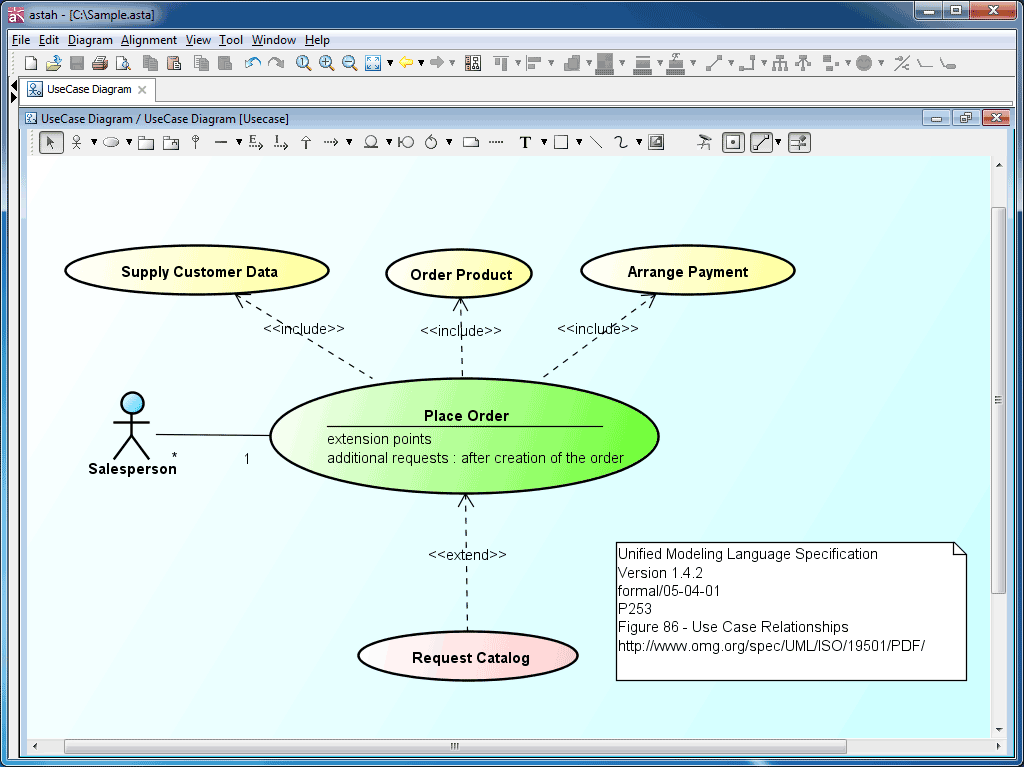
Sự tương tác ở đây có thể là:

* Người dùng tương tác với hệ thống như thế nào?
* Hoặc, hệ thống tương tác với các hệ thống khác như thế nào?

Và dĩ nhiên, sự tương tác này phải nằm trong một môi trường cụ thể, tức là nằm trong một bối cảnh, phạm vi chức năng cụ thể, hoặc rộng hơn là trong một hệ thống/ phần mềm cụ thể.

Sau cùng, việc mô tả sự tương tác này phải nhằm diễn đạt một mục đích cụ thể nào đó. Use Case phải diễn rả được Requirement **theo góc nhìn**cụ thể từ phía**người dùng.**

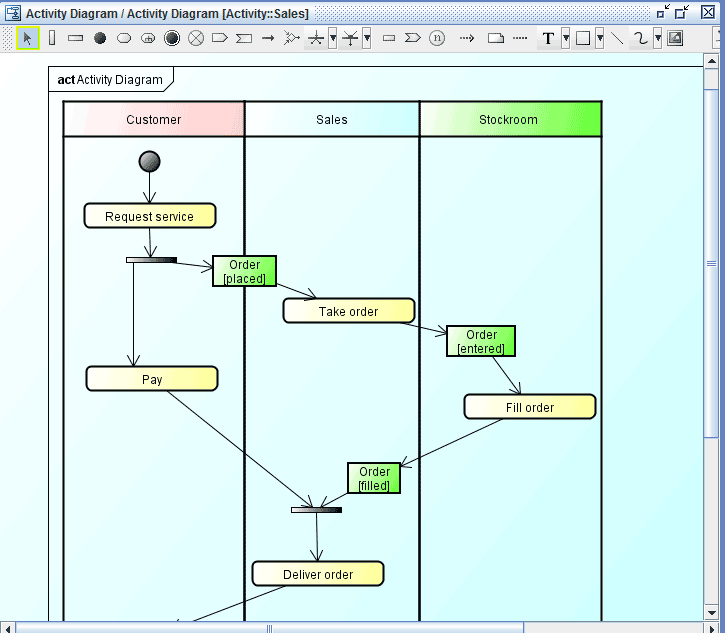
<https://thinhnotes.com/chuyen-nghe-ba/use-case-diagram-va-5-sai-lam-thuong-gap/>



Activity Diagram

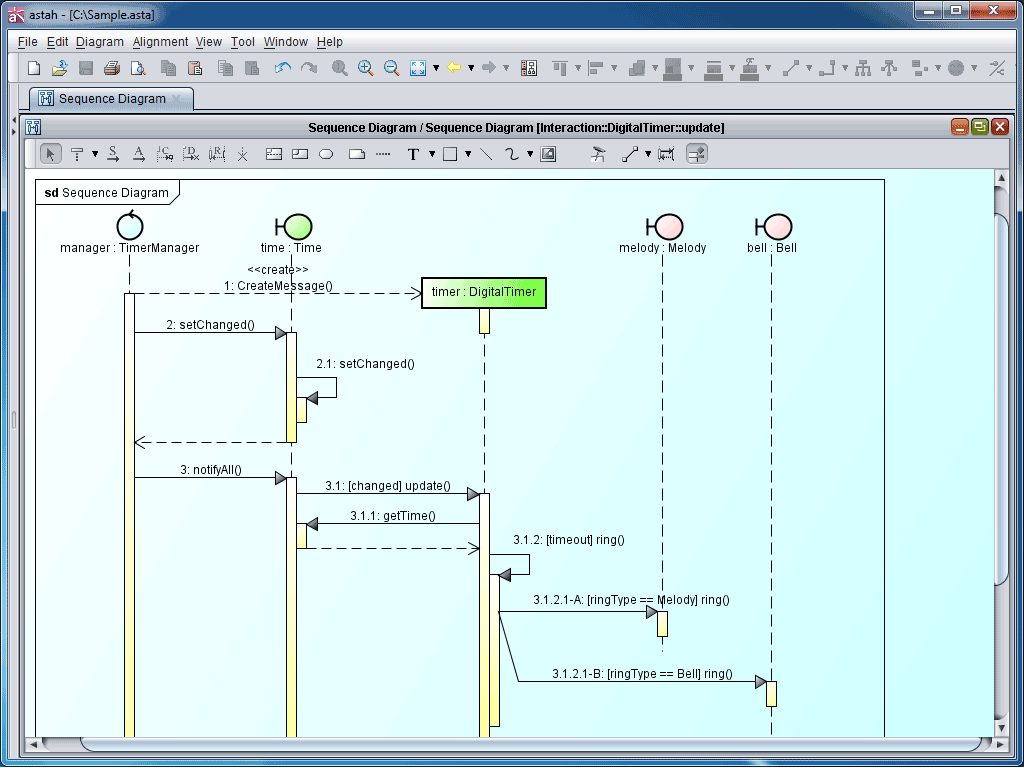
Activity Diagram là bản vẽ tập trung vào mô tả các hoạt động, luồng xử lý bên trong hệ thống. Nó có thể được sử dụng để mô tả các qui trình nghiệp vụ trong hệ thống, các luồng của một chức năng hoặc các hoạt động của một đối tượng.

Activity diagram (biểu đồ hoạt động ) là một mô hình logic được dùng để mô hình hoá cho các hoạt động trong một quy trình nghiệp vụ. Nó chỉ ra luồng đi từ hoạt động này sang hoạt động khác trong một hệ thống. Nó đặc biệt quan trọng trong việc xây dựng mô hình chức năng của hệ thống và nhấn mạnh tới việc chuyển đổi quyền kiểm soát giữa các đối tượng.

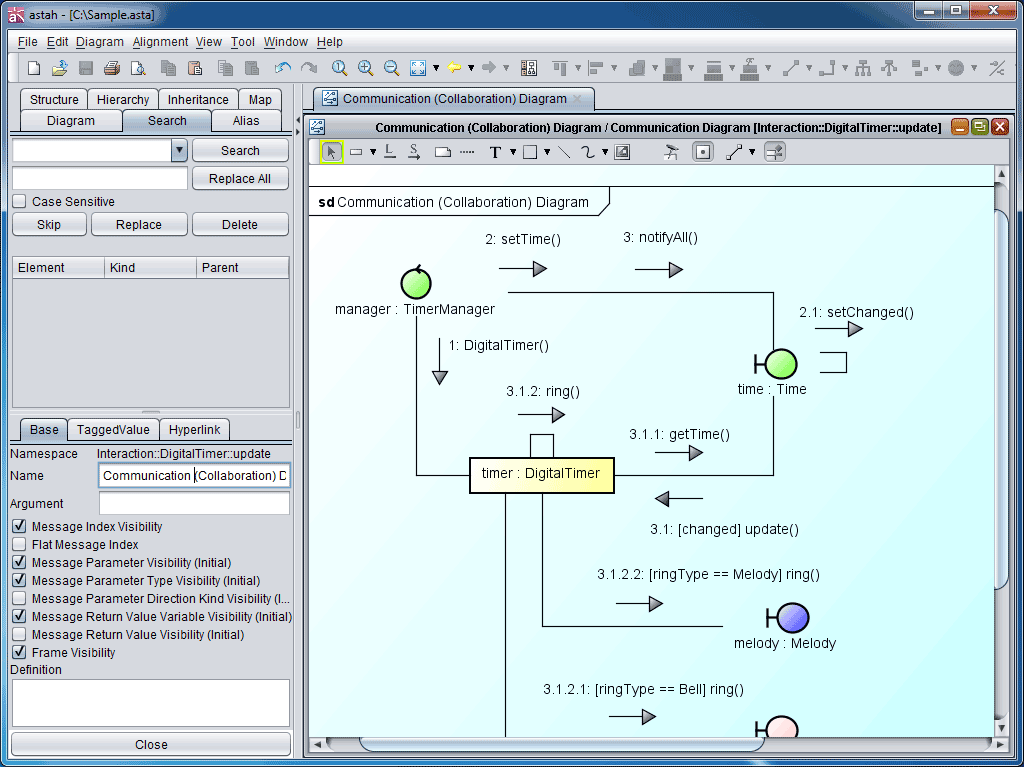


<https://iviettech.vn/blog/1066-activity-diagram-ban-ve-hoat-dong.html>

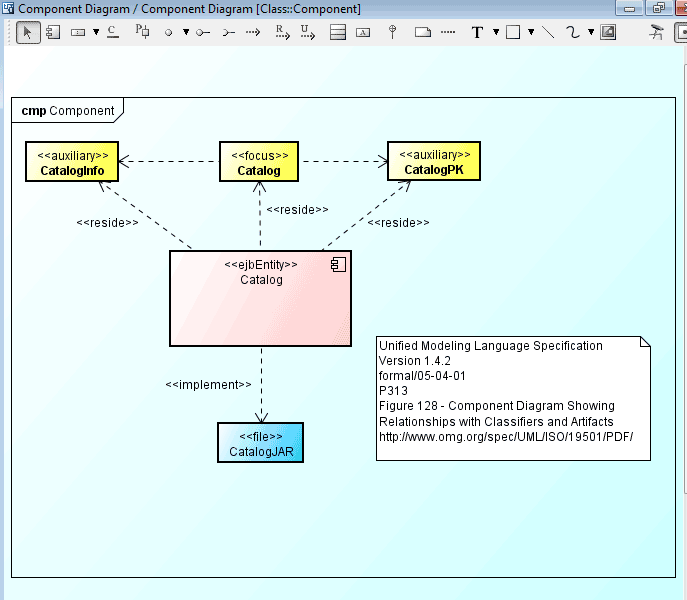
Sequence Diagram



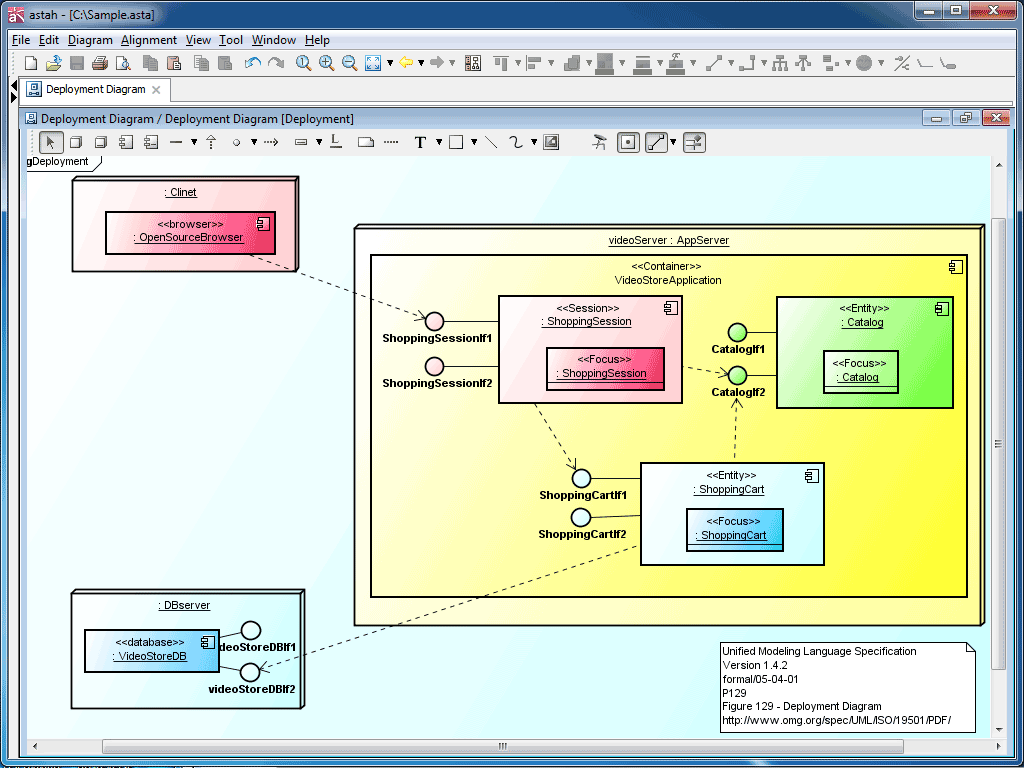
Communication Diagram



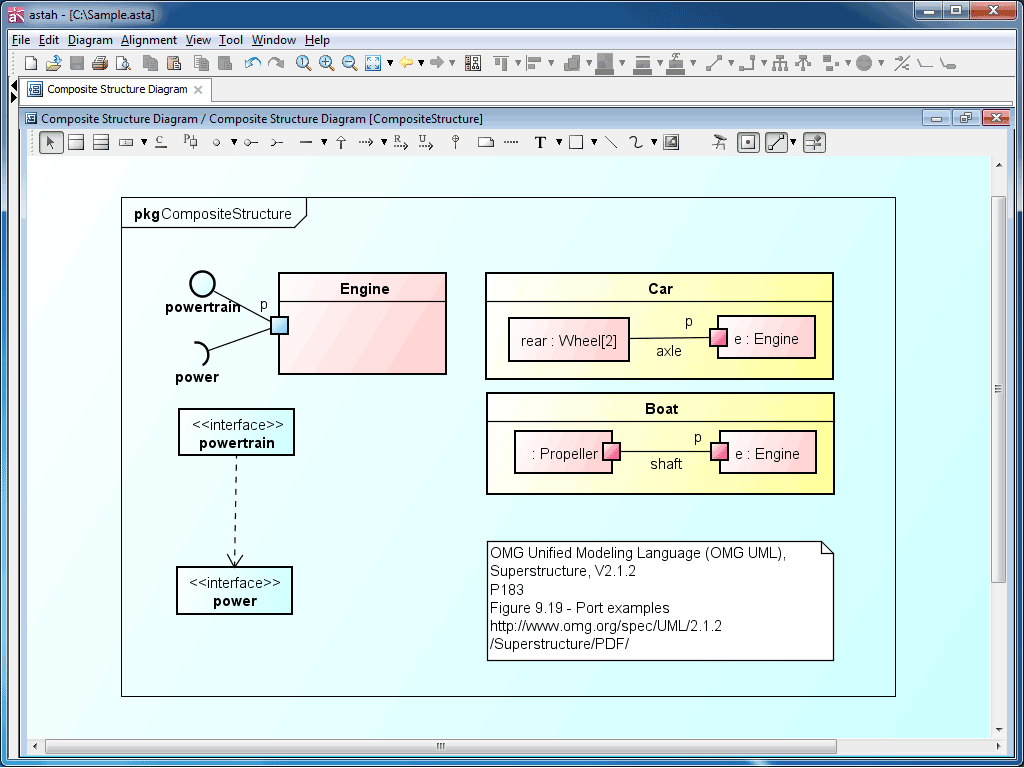
Component Diagram



Deployment Diagram



Composite Structure Diagram



Develop là gì

|  |  |
| --- | --- |
| Develop | Delevelop là gì |
| Develop những gì |
| Cách develop |
| Cách giao tiếp |
| Cách review |

Deleverlop là gì:

Deverlop đơn giản là phát triển, từ những yêu cầu cơ bản thành sản phẩm thực sự( chạy được hay ổn định không hay biết)

Develop tập trung các vấn đề sau:

* Tập trung design
* Quay lại requmemient
* Comfirm

Cách develop:

* Bài toán rõ ràng thì cứ thế mà code thôi
* Bài toán không rõ ràng thì phải qua các quy trình sau:

+ Đọc lại tài lieu

+ cofirm

* Tập trung làm khung trước-> chi tiết

|  |  |
| --- | --- |
| Unit test | Định nghĩa Unitest |
| Test case |
| Test Value |

unitTest là gi

Một Unit là một thành phần PM nhỏ nhất mà ta có thể kiểm tra được như các hàm (Function), thủ tục (Procedure), lớp (Class), hoặc các phương thức (Method).

Vì Unit được chọn để kiểm tra thường có kích thước nhỏ và chức năng hoạt động đơn giản, chúng ta không khó khăn gì trong việc tổ chức, kiểm tra, ghi nhận và phân tích kết quả kiểm tra nên việc phát hiện lỗi sẽ dễ dàng xác định nguyên nhân và khắc phục cũng tương đối dễ dàng vì chỉ khoanh vùng trong một Unit đang kiểm tra.

Mỗi UT sẽ gửi đi một thông điệp và kiểm tra câu trả lời nhận được đúng hay không, bao gồm:

* Các kết quả trả về mong muốn
* Các lỗi ngoại lệ mong muốn

Các đoạn mã UT hoạt động liên tục hoặc định kỳ để thăm dò và phát hiện các lỗi kỹ thuật trong suốt quá trình phát triển, do đó UT còn được gọi là kỹ thuật kiểm nghiệm tự động. UT có các đặc điểm sau:

* Đóng vai trò như những người sử dụng đầu tiên của hệ thống.
* Chỉ có giá trị khi chúng có thể phát hiện các vấn đề tiềm ẩn hoặc lỗi kỹ thuật.

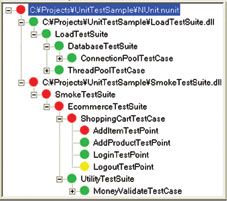
**Khi làm Unit test chúng ta thường thấy các khái niệm sau:**

1. Assertion: Là một phát biểu mô tả các công việc kiểm tra cần tiến hành, thí dụ: AreEqual(), IsTrue(), IsNotNull()… Mỗi một UT gồm nhiều assertion kiểm tra dữ liệu đầu ra, tính chính xác của các lỗi ngoại lệ ra và các vấn đề phức tạp khác như: – Sự tồn tại của một đối tượng – Điều kiện biên: Các giá trị có vượt ra ngoài giới hạn hay không – Thứ tự thực hiện của các luồng dữ liệu …
2. Test Point: Là một đơn vị kiểm tra nhỏ nhất, chỉ chứa đơn giản một assertion nhằm khẳng định tính đúng đắn của một chi tiết mã nào đó. Mọi thành viên dự án đều có thể viết một test point. Test Case: Là một tập hợp các test point nhằm kiểm tra một đặc điểm chức năng cụ thể, thí dụ toàn bộ giai đoạn người dùng nhập dữ liệu cho đến khi thông tin được nhập vào cơ sở dữ liệu. Trong nhiều trường hợp kiểm tra đặc biệt và khẩn cấp có thể không cần đến test case.
3. Test Suite: Là một tập hợp các test case định nghĩa cho từng module hoặc hệ thống con.
4. Regression Testing (hoặc Automated Testing): Là phương pháp kiểm nghiệm tự động sử dụng một phần mềm đặc biệt. Cùng một loại dữ liệu kiểm tra giống nhau nhưng được tiến hành nhiều lần lặp lại tự động nhằm ngăn chặn các lỗi cũ phát sinh trở lại. Kết hợp Regression Testing với Unit Testing sẽ đảm bảo các đoạn mã mới vẫn đáp ứng yêu cầu thay đổi và các đoạn mã cũ sẽ không bị ảnh hưởng bởi các hoạt động bảo trì.
5. Production Code: Phần mã chính của ứng dụng được chuyển giao cho khách hàng.
6. Unit Testing Code: Phần mã phụ để kiểm tra mã ứng dụng chính, không được chuyển giao cho khách hàng.

**2.Vòng đời Unit Test**

UT có 3 trạng thái cơ bản:

* Fail (trạng thái lỗi)
* Ignore (tạm ngừng thực hiện)
* Pass (trạng thái làm việc)
* Toàn bộ UT được vận hành trong một hệ thống tách biệt. Có rất nhiều PM hỗ trợ thực thi UT với giao diện trực quan. Thông thường, trạng thái của UT được biểu hiện bằng các màu khác nhau: màu xanh (pass), màu vàng (ignore) và màu đỏ (fail)



UT chỉ thực sự đem lại hiệu quả khi:

* Được vận hành lặp lại nhiều lần
* Tự động hoàn toàn
* Độc lập với các UT khác.

**3. Thiết kế Unit test**

Mỗi UT đều được tiết kế theo trình tự sau:

* Thiết lập các điều kiện cần thiết: khởi tạo các đối tượng, xác định tài nguyên cần thiết, xây dựng các dữ liệu giả…
* Triệu gọi các phương thức cần kiểm tra.
* Kiểm tra sự hoạt động đúng đắn của các phương thức.
* Dọn dẹp tài nguyên sau khi kết thúc kiểm tra.

**4. Ứng dụng Unit test**

* Kiểm tra mọi đơn vị nhỏ nhất là các thuộc tính, sự kiện, thủ tục và hàm.
* Kiểm tra các trạng thái và ràng buộc của đối tượng ở các mức sâu hơn mà thông thường chúng ta không thể truy cập được.
* Kiểm tra các quy trình (process) và mở rộng hơn là các khung làm việc(workflow – tập hợp của nhiều quy trình)

**5. Lợi ích của việc áp dụng Unit test**

Thời gian đầu, người ta thường do dự khi phải viết UT thay vì tập trung vào code cho các chức năng nghiệp vụ. Công việc viết Unit Test có thể mất nhiều thời gian hơn code rất nhiều nhưng lại có lợi ích sau:

* Tạo ra môi trường lý tưởng để kiểm tra bất kỳ đoạn code nào, có khả năng thăm dò và phát hiện lỗi chính xác, duy trì sự ổn định của toàn bộ PM và giúp tiết kiệm thời gian so với công việc gỡ rối truyền thống.
* Phát hiện các thuật toán thực thi không hiệu quả, các thủ tục chạy vượt quá giới hạn thời gian.
* Phát hiện các vấn đề về thiết kế, xử lý hệ thống, thậm chí các mô hình thiết kế.
* Phát hiện các lỗi nghiêm trọng có thể xảy ra trong những tình huống rất hẹp.
* Tạo hàng rào an toàn cho các khối mã: Bất kỳ sự thay đổi nào cũng có thể tác động đến hàng rào này và thông báo những nguy hiểm tiềm tàng.

Trong môi trường làm việc Unit Test còn có tác dụng rất lớn đến năng suất làm việc:

* Giải phóng chuyên viên QA khỏi các công việc kiểm tra phức tạp.
* Tăng sự tự tin khi hoàn thành một công việc. Chúng ta thường có cảm giác không chắc chắn về các đoạn mã của mình như liệu các lỗi có quay lại không, hoạt động của module hiện hành có bị tác động không, hoặc liệu công việc hiệu chỉnh mã có gây hư hỏng đâu đó…
* Là công cụ đánh giá năng lực của bạn. Số lượng các tình huống kiểm tra (test case) chuyển trạng thái “pass” sẽ thể hiện tốc độ làm việc, năng suất của bạn.

**6. Cách code hiệu quả với Unit Test**

Phân tích các tình huống có thể xảy ra đối với mã. Đừng bỏ qua các tình huống tồi tệ nhất có thể xảy ra, thí dụ dữ liệu nhập làm một kết nối cơ sở dữ liệu thất bại, ứng dụng bị treo vì một phép toán chia cho không, các thủ tục đưa ra lỗi ngoại lệ sai có thể phá hỏng ứng dụng một cách bí ẩn…

Mọi UT phải bắt đầu với trạng thái “fail” và chuyển trạng thái “pass” sau một số thay đổi hợp lý đối với mã chính.

Mỗi khi viết một đoạn mã quan trọng, hãy viết các UT tương ứng cho đến khi bạn không thể nghĩ thêm tình huống nào nữa.

Nhập một số lượng đủ lớn các giá trị đầu vào để phát hiện điểm yếu của mã theo nguyên tắc:

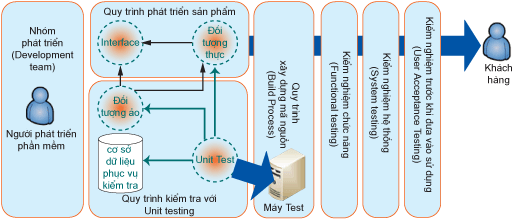
* Nếu nhập giá trị đầu vào hợp lệ thì kết quả trả về cũng phải hợp lệ
* Nếu nhập giá trị đầu vào không hợp lệ thì kết quả trả về phải không hợp lệ
* Sớm nhận biết các đoạn mã không ổn định và có nguy cơ gây lỗi cao, viết UT tương ứng để khống chế.

Ứng với mỗi đối tượng nghiệp vụ (business object) hoặc đối tượng truy cập dữ liệu (data access object), nên tạo ra một lớp kiểm tra riêng vì những lỗi nghiêm trọng có thể phát sinh từ các đối tượng này.

Để ngăn chặn các lỗi có thể phát sinh trở lại thực thi tự động tất cả UT mỗi khi có một sự thay đổi quan trọng, hãy làm công việc này mỗi ngày. Các UT lỗi cho chúng ta biết thay đổi nào là nguyên nhân gây lỗi.

Để tăng hiệu quả và giảm rủi ro khi viết các UT, cần sử dụng nhiều phương thức kiểm tra khác nhau. Hãy viết càng đơn giản càng tốt.

Cuối cùng, viết UT cũng đòi hỏi sự nỗ lực, kinh nghiệm và sự sáng tạo như viết PM.



Trước khi kết thúc phần này, tôi có một lời khuyên là viết UT cũng tương tự như viết mã một chương trình, điều bạn cần làm là không ngừng thực hành. Hãy nhớ UT chỉ thực sự mang lại lợi ích nếu chúng ta đặt vấn đề chất lượng phần mềm lên hàng đầu hơn là chỉ nhằm kết thúc công việc đúng thời hạn. Khi đã thành thạo với công việc viết UT, bạn có thể đọc thêm về các kỹ thuật xây dựng UT phức tạp hơn, trong số đó có mô hình đối tượng ảo sẽ được trình bày trong phần tiếp theo.

Hướng dẫn Viết test Case

|  |
| --- |
| Test case |
| Test Value |

|  |  |
| --- | --- |
| IT Testting | IT test là gì |
| Tư duy mipmap |
| Cách viết và test |

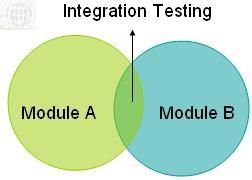
**. Định nghĩa:**

* **Kiểm thử tích hợp (Integration testing) hay còn gọi là tích hợp và kiểm thử (integration and testing, viết tắt: I&T)** là một giai đoạn trong kiểm thử phần mềm. Mỗi môđun phần mềm riêng biệt được kết hợp lại và kiểm thử theo nhóm.
* Kiểm thử tích hợp xảy ra sau kiểm thử đơn vị (Unit Test) và trước kiểm thử xác nhận. Kiểm thử tích hợp nhận các môđun đầu vào đã được kiểm thử đơn vị, nhóm chúng vào các tập hợp lớn hơn, áp dụng các ca kiểm thử đã được định nghĩa trong kế hoạch kiểm thử tích hợp vào tập hợp đó, và cung cấp đầu ra cho hệ thống tích hợp.

**2. Tại sạo lại phải thực hiện kiểm thử tích hợp**

Mặc dù mỗi module đều được kiểm thử đơn vị (Unit test) nhưng các lỗi vẫn còn tồn tại với các nguyên nhân sau:

* Một Module nói chung được thiết kế bởi một lập trình viên có hiểu biết và logic lập trình có thể khác với các lập trình viên khác. Kiểm thử tích hợp là cần thiết để đảm bảo tính hợp nhất của phần mềm.
* Tại thời điểm phát triển module vẫn có thể có thay đổi trong spec của khách hàng, những thay đổi này có thể không được kiểm tra ở giai đoạn unit test trước đó.
* Giao diện và cơ sở dữ liệu của các module có thể chưa hoàn chỉnh khi được ghép lại.
* Khi tích hợp hệ thống các module có thể không tương thích với cấu hình chung của hệ thống.
* Thiếu các xử lý ngoại lệ có thể xảy ra.

Hai mô-đun khác nhau ‘Mô-đun A và’ Mô-đun B, được tích hợp sau đó thử nghiệm tích hợp được thực hiện.

**3. Ví dụ về Kiểm thử Tích Hợp:**

Giả sử bạn làm việc cho một tổ chức CNTT đã được yêu cầu phát triển trang web mua sắm trực tuyến cho Camp World, một công ty bán dụng cụ cắm trại. Sau khi thu thập yêu cầu, phân tích và thiết kế hoàn tất, một nhà phát triển đã được chỉ định để phát triển từng mô-đun bên dưới.

* Đăng ký và xác thực người dùng / Đăng nhập
* Danh mục sản phẩm
* Giỏ hàng
* Thanh toán
* Tích hợp cổng thanh toán
* Theo dõi vận chuyển và gói hàng

Sau khi mỗi mô-đun được gán cho nhà phát triển, nhà phát triển bắt đầu mã hóa chức năng trên các máy riêng lẻ của họ. Họ đã triển khai các mô-đun tương ứng trên các máy của mình để xem những gì đã hoạt động và những gì đã làm, khi họ bắt đầu phát triển mô-đun. Sau khi họ hoàn thành việc phát triển, các nhà phát triển đã kiểm tra các chức năng cá nhân của họ như là một phần của kiểm thử đơn vị của họ và tìm thấy một số khiếm khuyết. Họ đã sửa những khuyết điểm này. Tại thời điểm này, họ cảm thấy các mô-đun của họ đã hoàn thành. Kiểm tra tích hợp nên được thực hiện để xác nhận rằng tất cả các mô-đun hoạt động cùng nhau. Khi họ triển khai tất cả mã của họ trong một máy chung, họ thấy rằng ứng dụng không hoạt động như mong đợi vì các mô-đun riêng lẻ không hoạt động tốt với nhau. Có một số lỗi như - sau khi đăng nhập, giỏ hàng của người dùng không hiển thị các mục họ đã thêm trước đó, số tiền hóa đơn không bao gồm chi phí vận chuyển, v.v.

Theo cách này, Kiểm thử tích hợp giúp chúng ta xác định, khắc phục các sự cố và đảm bảo rằng toàn bộ ứng dụng hoạt động như mong đợi.

**4. Cách tiếp cận, phương pháp, chiến lược của kiểm thử tích hợp:**

Có nhiều loại hoặc cách tiếp cận khác nhau để kiểm thử tích hợp. Các phương pháp phổ biến và được sử dụng thường xuyên nhất là Kiểm thử tích hợp Big Bang, Kiểm thử tích hợp Top-down, Kiểm thử tích hợp từ dưới lên và Kiểm thử tích hợp Bottom up. Sự lựa chọn của phương pháp phụ thuộc vào các yếu tố khác nhau như chi phí, độ phức tạp, mức độ quan trọng của ứng dụng, v.v. Ngoài ra, có nhiều loại thử nghiệm tích hợp ít được biết đến như tích hợp dịch vụ phân tán, thử nghiệm tích hợp sandwich, tích hợp đường trục, tích hợp tần số cao, tích hợp lớp, v.v.

|  |  |
| --- | --- |
| SysTemTessting | -Chạy Hàng tháng = Điều kiện đặc biệt |

**1. System testing là gì ?**

* Kiểm thử một hệ thống đã được tích hợp hoàn chỉnh để xác minh rằng nó đáp ứng được yêu cầu.
* Kiểm thử hệ thống bao gồm kiểm thử hộp đen vì vậy các kiến thức về thiết kế nội bộ hoặc cấu trúc hoặc code không cần thiết cho loại kiểm thử phần mềm này.
* Kiểm thử hệ thống bao gồm kiểm thử chức năng và phi chức năng.
* Kiểm thử hệ thống tập trung nhiều hơn vào các chức năng của toàn bộ hệ thống.
* Các trường hợp kiểm thử hệ thống bao gồm các chức năng của sản phẩm hoàn chỉnh và được thực hiện các trường hợp kiểm thử mức độ cao.
* Các hành vi của ứng dụng hoàn chỉnh được kiểm tra để đáp ứng các yêu cầu quy định.
* Các trường hợp kiểm thử và dữ liệu kiểm thử được thực hiện và các dữ liệu thực tế không được sử dụng trong loại kiểm thử này.
* Trong kiểm thử tích hợp hệ thống sẽ tích hợp các mô-đun khác nhau và kiểm tra giao diện giữa chúng để kiểm tra tính toàn vẹn dữ liệu.
* Trong khi thực hiện quá trình kiểm thử hệ thống, kiểm tra tính đúng đắn được thực hiện bởi nhân viên kiểm thử.

Có các kỹ thuật kiểm thử khác nhau được sử dụng trong kiểm thử hệ thống:

* Kiểm thử chức năng
* Kiểm thử giao diện , kiểm thử tính sử dụng
* Kiểm thử bảo mật
* Kiểm thử hiệu năng

**2. Acceptance Testing là gì ?**

* Đây là một kiểm thử liên quan đến nhu cầu của người sử dụng, yêu cầu và quy trình kinh doanh được tiến hành để xác định có hay không một hệ thống đáp ứng các tiêu chí chấp nhận và kiểm tra hệ thống đáp ứng yêu cầu của khách hàng.
* Kiểm thử chấp nhận kiểm thử các chức năng để kiểm tra hành vi của hệ thống bằng cách sử dụng dữ liệu thực tế. Nó cũng được gọi là thử nghiệm người dùng doanh nghiệp.
* Kiểm thử chấp nhận được thực hiện bởi người dùng cuối để kiểm tra hệ thống được xây dựng để phù hợp với yêu cầu kinh doanh của tổ chức.
* Trong kiểm thử này, tất cả các giao diện đã được kết hợp và hệ thống đã hoàn thành và đã được kiểm tra. Người dùng cuối cũng thực hiện các kiểm thử để kiểm tra khả năng sử dụng của hệ thống.
* Nhiều kỹ thuật kiểm thử chức năng sử dụng cho loại này là phân tích giá trị biên giới, phân vùng tương đương, bảng quyết định. Đây là loại kiểm tra tập trung chủ yếu vào các kiểm thử hợp lệ của hệ thống.

Trong kiểm thử chấp nhận có hai mức độ kiểm thử là Alpha testing và Beta testing

* Alpha Testing: Kiểm thử Alpha cũng được gọi là kiểm thử trang web off. Trong kiểm thử này quá trình kiểm thử sẽ kiểm tra các ứng dụng với sự hiện diện của người dùng cuối trong môi trường tổ chức.
* Beta Testing: Kiểm thử Beta nên được thực hiện bởi người dùng cuối trong môi trường riêng của họ với sự hiện diện của đội phát hành.

**3. Sự khác nhau giữa System Testing và Acceptance Testing**

| **System Testing** | **Acceptance Testing** |
| --- | --- |
| 1. Kiểm thử hệ thống được thực hiện để kiểm tra xem phần mềm đáp ứng các yêu cầu đã quy định. | 1. Kiểm thử chấp nhận là kiểm thử chức năng, được thực hiện để kiểm tra xem phần mềm đáp ứng các yêu cầu của khách hàng. |
|  |  |
| 2. Kiểm thử hệ thống được thực hiện bởi những người phát triển và các nhân viên kiểm thử. | 2. Kiểm thử chấp nhận được thực hiện bởi khách hàng , người dùng và các bên liên quan. |
| 3. Kiểm thử hệ thống kiểm tra cả các yêu cầu chức năng và phi chức năng | 3. Kiểm thử chấp nhận kiểm tra các yêu cầu chức năng |
| 4. Trong kiểm thử hệ thống, sẽ kiểm tra cách toàn bộ hệ thống được thực hiện, thực hiện kiểm tra các chức năng. | 4. Trong kiểm thử chấp nhận sẽ kiểm tra hệ thống đáp ứng các nhu cầu kinh doanh của tổ chức, khả năng sử dụng của sản phẩm |
| 5. Được thực hiện với dữ liệu demo và không phải là dữ liệu thực tế. | 5. Được thực hiện với các dữ liệu thời gian thực tế. |
| 6. Kiểm thử phần mềm cho đặc tả yêu cầu đầy đủ bao gồm cả phần cứng và phần mềm, bộ nhớ và số lượng người dùng. | 6. Kiểm thử phần mềm cho các nhu cầu sử dụng và nhu cầu của người sử dụng được đáp ứng trong phát triển phần mềm. |
| 7. Kiểm thử hệ thống bao gồm kiểm thử hệ thống và kiểm thử tích hợp hệ thống | 7. Kiểm thử chấp nhận bao gồm kiểm thử alpha và kiểm thử beta. |
| 8. Kiểm thử hệ thống được tiến hành trước kiểm thử chấp nhận | 8. Kiểm thử chấp nhận được thực hiện sau kiểm thử hệ thống. |
| 9. Kiểm thử hệ thống liên quan đến kiểm thử phi chức năng là hiệu suất tải (performance load) và kiểm thử stress . | 9. Kiểm thử chấp nhận liên quan đến kiểm thử chức năng đó là phân tích giá trị biên, phân vùng tương đương và bảng quyết định. |
| 10. Kiểm thử hệ thống được thực hiện bởi nhóm các nhân viên kiểm thử, nó sẽ chứa nhiều trường hợp kiểm thử bất thường (abnormal test cases) . | 10. Kiểm thử chấp nhận chứa nhiều các trường hợp kiểm thử thông thường (normal test cases) . |
| 11. Các lỗi tìm thấy trong kiểm thử hệ thống có thể được sửa dựa trên độ ưu tiên. | 11. Các lỗi tìm thấy trong kiểm thử chấp nhận được xem như là sự thất bại của sản phẩm. |
| 12. Kiểm thử hệ thống cho tất cả các dữ liệu đầu vào giả có thể. | 12. Kiểm thử với các dữ liệu ngẫu nhiên |

**Làm thế nào kiểm thử hệ thống và kiểm thử chấp nhận được thực hiện**

| **Quan điểm** | **Kiểm thử hệ thống** | **Kiểm thử chấp nhận** |
| --- | --- | --- |
| Điều kiện tiên quyết | 1. Unit testing được hoàn thành | 1. Kiểm thử hệ thống được hoàn thành |
|  | 2. Phần mền đã hoàn thành và đã được develop | 2. Yêu cầu doanh nghiệp có sẵn |
|  | 3. Môi trường kiểm thử phần mềm đã sẵn sàng. | 3. Môi trường kiểm thử chấp nhận sẵn sàng. |
| Tiến trình | 1. Tạo kế hoạch kiểm thử hệ thống. | 1. Phân tích các yêu cầu doanh nghiệp |
|  | 2. Tạo các trường hợp kiểm thử hệ thống. | 2. Tạo một kế hoạch kiểm thử chấp nhận. |
|  | 3. Tạo dữ liệu kiểm thử | 3. Xác định các kịch bản kiểm thử chấp nhận. |
|  | 4. Thực hiện test các trường hợp kiểm thử (test case) | 4. Tạo các trường hợp kiểm thử chấp nhận. |
|  | 5. Báo cáo lỗi | 5. Thực hiện test các trường hợp kiểm thử chấp nhận. |
|  | 6. Lặp lại các bước | 6. Xác nhận các mục tiêu doanh nghiệp. |
| Tiêu chí hoàn thành | 1. Tất cả các lỗi có độ ưu tiên cao và các lỗi phải sửa chưa ngay lập tức được sửa . | 1. Không có các lỗi quan trọng còn chưa được sửa. |
|  | 2. Bất kỳ lỗi có độ ưu tiên vừa, được xác nhận bởi người quản lý phân tích kinh doanh / người quản lý dự án. | 2. Quá trình kinh doanh đang làm việc tốt. |
|  | 3. Tất cả các trường hợp kiểm thử đã được passed. | 3. Có một dấu hiệu hoàn thành trên kiểm thử chấp nhận từ các khách hàng doanh nghiệp hoặc các bên liên quan. |
|  | 4. Không có vấn đề liên quan đến bảo mật | 4. Không có vấn đề liên quan đến bảo mật |

Trên đây là sự khác nhau giữa System testing và Accept testing hy vọng qua bài việt này mọi ng có thể hiểu rõ hơn về hai giai đoạn kiểm thử này.