TRƯỜNG ĐẠI HỌC PHENIKAA

KHOA CÔNG NGHỆ THÔNG TIN

HỌC PHẦN:

KỸ THUẬT PHẦN MỀM

Đề tài:

……..

Sinh viên	**1. Mè Trung Đức**	**MS: 2017601671**
thực hiện:	**2. Dương Thị Ngọc Hảo**	**MS: 2017600889**
	3. Lê Minh Hiếu	**MS: 2017601003**
	4. Nguyễn Thị Thanh Huyền	**MS: 2017603305**
	5. Bạc Bảo Long	**MS: 2017602398**
Lớp:	**KTPM- 01**	

Hà Nội, tháng 04 năm 2023

MỤC LỤC

LỜI MỞ ĐẦU

Với sự phát triển không ngừng của khoa học và công nghệ, đặc biệt là các thiết bị được hỗ trợ công nghệ xử lý ảnh ngày càng hiện đại và được sử dụng phổ biến trong đời sống con người đã làm cho lượng thông tin thu được bằng hình ảnh ngày càng tăng và phổ biến. Theo đó, lĩnh vực xử lý ảnh cũng được chú trọng phát triển, ứng dụng rộng rãi trong đời sống xã hội hiện đại. Không chỉ dừng lại ở việc chỉnh sửa, tăng chất lượng hình ảnh mà với công nghệ xử lý ảnh hiện nay chúng ta có thể giải quyết các bài toán nhận dạng chữ viết, nhận dạng dấu vân tay, đặc biệt là nhận dạng khuôn mặt…

Công nghệ nhân diện khuôn mặt (Facial Recognition Technology) hiện là một công nghệ đang được sử dụng khá phổ biến tại các quốc gia phát triển. Công nghệ này có khả năng xác định hoặc xác nhận một người từ hình ảnh kỹ thuật số được lấy mẫu trước đó hoặc từ một khung hình trong một nguồn video khác. Đây là một phương pháp xác minh độc đáo khi thiết bị sẽ dựa vào những điểm khác nhau tiêu biểu nhất trên khuôn mặt của một người để tiến hành phân biệt giữa người này với người khác. Do vậy đối với các trường hợp như song sinh thì người dùng có thể yên tâm rằng máy vẫn sẽ phát hiện ra. Chính vì đặc điểm này thì ngoài được ứng dụng trong việc quản lý nhân sự ra thì nó còn là sự lựa chọn của rất nhiều đơn vị hoạt động trong lĩnh vực an ninh, bảo mật, giao dịch.

Xây dựng một ứng dụng nhận dạng khuôn mặt nhằm giúp việc quản lý, điểm danh, giao dịch hay thống kê theo từng yêu cầu phục cụ cho các mục đích khác nhau là cấp thiết.

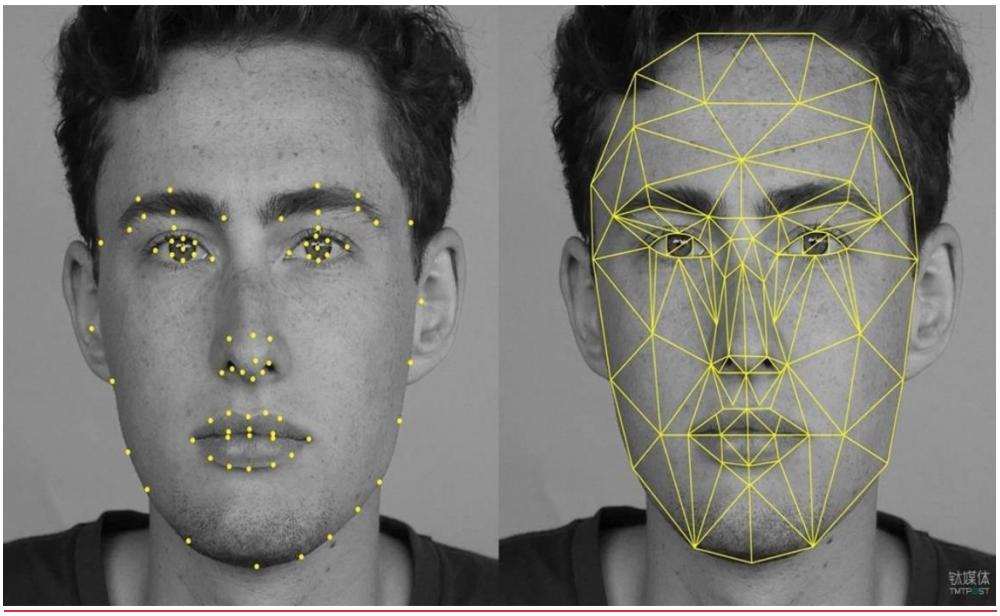
DANH MỤC HÌNH

MỞ ĐẦU

(Viết từ 1-2 trang: giới thiệu, phân tích các đóng góp chính bài toán mà nhóm đã giải quyết)

a. Giới thiệu

…….



Hình 1- 1: Nhận diện khuôn mặt.

b. ……

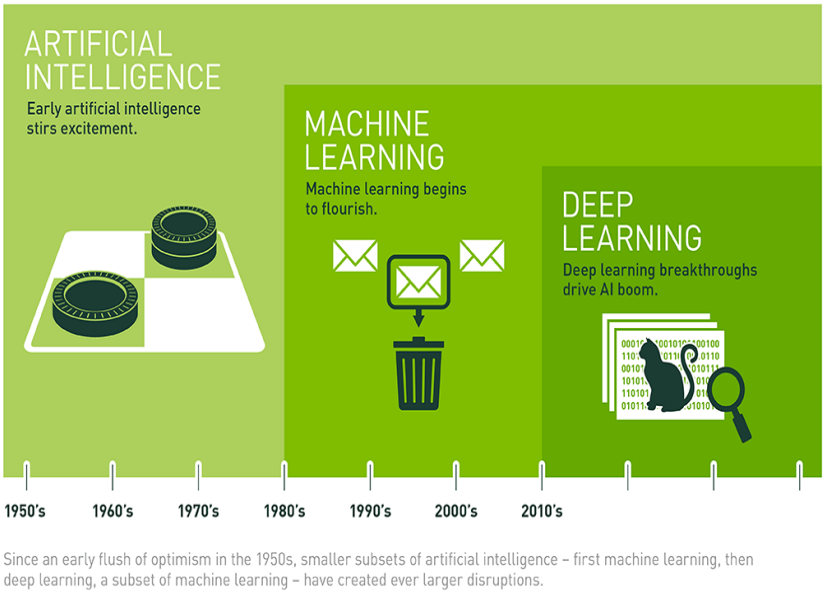
c. ……..

2. PHÁT BIỂU YÊU CẦU CỦA KHÁCH HÀNG

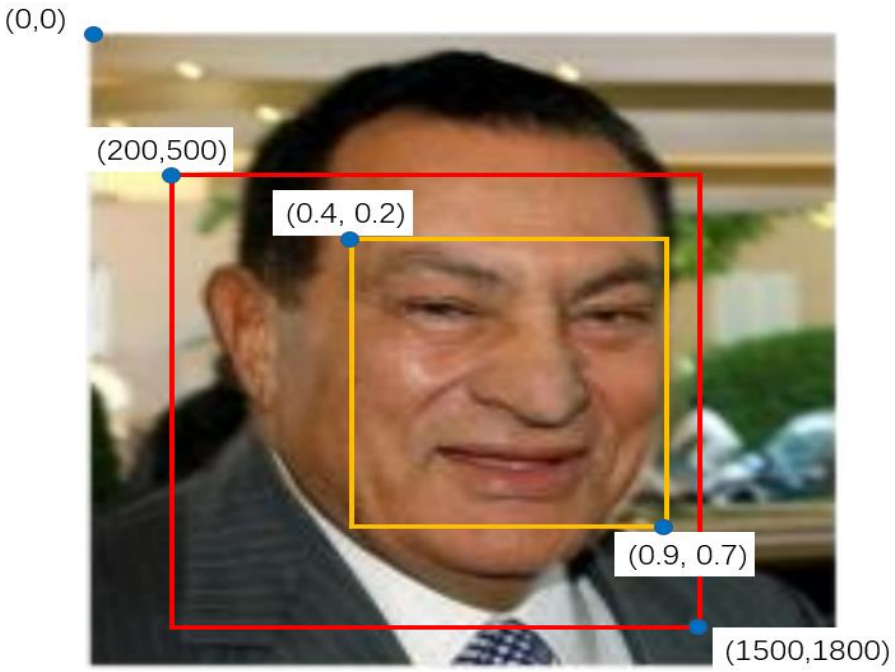
2.1. Chữ đậm

2.1.1. Chữ đậm và nghiêng

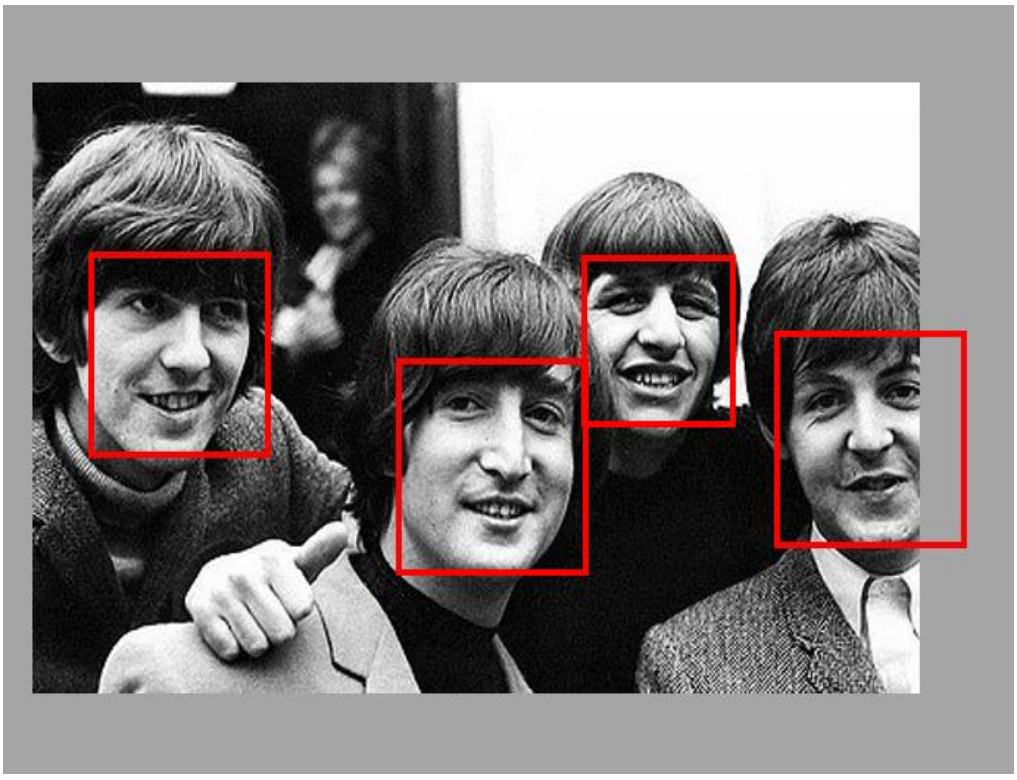
Đây là một tài liệu quan trọng trong quá trình phát triển phần mềm, được sử dụng để xác định các yêu cầu và mong muốn của khách hàng đối với sản phẩm phần mềm. Bản tường trình yêu cầu của khách hàng thường bao gồm mô tả chi tiết về các chức năng, tính năng, yêu cầu kỹ thuật và các yêu cầu phi chức năng khác của sản phẩm



Hình 2-1: Tổng quan AI



Hình 2-2: Ở đây, hộp màu đỏ là nhân 12 x 12, trong khi hộp màu vàng là hộp giới hạn bên trong nó.



Hình 2-3: The Beatles và các hộp giới hạn của họ. Hộp của Paul McCartney nằm ngoài giới hạn và cần có đệm.

3. TỪ ĐIỂN THUẬT NGỮ

3.1. Chữ đậm

Đây là một danh sách các thuật ngữ chuyên ngành được sắp xếp theo thứ tự chữ cái và cung cấp định nghĩa và giải thích cho từng thuật ngữ. Từ điển thuật ngữ thường được sử dụng để giúp người đọc hiểu và sử dụng các thuật ngữ chuyên ngành một cách chính xác và nhất quán.

4. CÁC YÊU CẦU HỆ THỐNG

4.1 Các yêu cầu chức năng

Đây là một phần quan trọng trong tài liệu yêu cầu của phần mềm, trong đó các yêu cầu chức năng của phần mềm được liệt kê một cách chi tiết và rõ ràng. Các yêu cầu chức năng liệt kê trong phần này thường bao gồm các chức năng cần thiết, điều kiện hoạt động, tương tác với người dùng và hệ thống khác, đầu vào và đầu ra mong đợi. Liệt kê các yêu cầu chức năng giúp đảm bảo rằng phần mềm được phát triển đầy đủ và đáp ứng được nhu cầu của khách hàng.

4.2 Các yêu cầu phi chức năng

Đây là một phần quan trọng trong tài liệu yêu cầu của phần mềm, trong đó các yêu cầu phi chức năng của phần mềm được liệt kê một cách chi tiết và rõ ràng. Các yêu cầu phi chức năng trong phần này thường bao gồm các yêu cầu về hiệu suất, độ tin cậy, bảo mật, khả dụng và độ bảo mật của phần mềm. Liệt kê các yêu cầu phi chức năng giúp đảm bảo rằng phần mềm được phát triển đầy đủ và đáp ứng được các yêu cầu chất lượng của khách hàng.

4.3 Yêu cầu giao diện người dùng

Đây là một phần quan trọng trong tài liệu yêu cầu của phần mềm, trong đó các yêu cầu về giao diện người dùng được liệt kê một cách chi tiết và rõ ràng. Các yêu cầu giao diện người dùng thường bao gồm các yêu cầu về cách mà giao diện người dùng phải được thiết kế, vị trí, kích thước và màu sắc của các phần tử trong giao diện, cách thức tương tác với người dùng, cách phản hồi và thông báo cho người dùng. Liệt kê các yêu cầu giao diện người dùng giúp đảm bảo rằng phần mềm được phát triển đầy đủ và đáp ứng được nhu cầu sử dụng của người dùng.

5. ĐẶC TẢ CÁC YÊU CẦU CHỨC NĂNG

Đây là một tài liệu quan trọng trong quá trình phát triển phần mềm, trong đó các yêu cầu chức năng của phần mềm được mô tả một cách chi tiết và rõ ràng. Thông số kỹ thuật yêu cầu chức năng thường bao gồm các yêu cầu về chức năng, tính năng, điều kiện hoạt động, tương tác với người dùng và hệ thống khác, đầu vào và đầu ra mong đợi. Thông số kỹ thuật yêu cầu chức năng là tài liệu quan trọng giúp đảm bảo rằng phần mềm được phát triển đầy đủ và đáp ứng được các yêu cầu chức năng của khách hàng.

5.1 Các đối tượng liên quan

Trong phát triển phần mềm, các bên liên quan là các cá nhân, tổ chức hoặc nhóm mà phần mềm ảnh hưởng đến hoặc ảnh hưởng đến phần mềm đó. Các bên liên quan bao gồm khách hàng, nhà phát triển, quản lý dự án, người dùng cuối và các bên thứ ba có liên quan đến phần mềm như đối tác hoặc nhà cung cấp. Các bên liên quan có quyền quyết định hoặc ảnh hưởng đến các quyết định trong quá trình phát triển phần mềm, vì vậy quan hệ giữa các bên liên quan rất quan trọng để đảm bảo rằng phần mềm đáp ứng được các yêu cầu và mục tiêu của tất cả các bên liên quan.

5.2 Các tác nhân và mục đích

Trong phát triển phần mềm, "Actors" là những người hoặc hệ thống có tương tác với phần mềm, bao gồm cả người dùng cuối và các hệ thống khác. "Goals" là các mục tiêu hoặc kết quả mà phần mềm phải đáp ứng được để thỏa mãn nhu cầu của các Actor.

Thông qua việc xác định các Actors và Goals, nhóm phát triển phần mềm có thể hiểu rõ hơn về các yêu cầu chức năng và phi chức năng của phần mềm. Các Actors và Goals cũng giúp nhóm phát triển phần mềm phân tích và thiết kế hệ thống phần mềm sao cho phù hợp với nhu cầu của các Actor và đáp ứng được các mục tiêu đề ra. Các Actors và Goals thường được đặc tả trong tài liệu yêu cầu phần mềm và được sử dụng trong quá trình phát triển và kiểm thử phần mềm.

5.3 Các ca sử dụng

"Use Cases" có thể được dịch sang tiếng Việt là "Các ca sử dụng". Trong phát triển phần mềm, Use Cases là một phương pháp để mô tả các tình huống hoặc kịch bản mà người dùng sẽ sử dụng phần mềm trong thực tế. Các Use Cases được sử dụng để mô tả cách thức hoạt động của phần mềm từ góc độ của người dùng cuối, cung

cấp cho nhóm phát triển phần mềm một cái nhìn tổng quan về các tác nhân, các sự kiện và các hành động được thực hiện trong các tình huống khác nhau.

Mỗi Use Case sẽ bao gồm một số thông tin, bao gồm: tên của Use Case, mô tả ngắn gọn, tên của các tác nhân được liên quan đến Use Case, các bước thực hiện cần thiết để hoàn thành Use Case và các điều kiện tiên quyết và hậu quả. Use Cases giúp định hình và hiểu rõ hơn các tính năng và yêu cầu của phần mềm, cũng như giúp đảm bảo rằng phần mềm được thiết kế để đáp ứng nhu cầu và mục tiêu của người dùng cuối.

5.3.1 Miêu tả đơn giản

"Casual Description" có thể được dịch sang tiếng Việt là "Mô tả bình thường". Trong phát triển phần mềm, Casual Description là một phương pháp để mô tả các tính năng hoặc yêu cầu của phần mềm một cách dễ hiểu và trực quan. Nó được sử dụng để truyền tải thông tin một cách thông thường, tránh sử dụng các thuật ngữ kỹ thuật phức tạp và đảm bảo rằng tất cả các bên đều có thể hiểu được ý nghĩa của yêu cầu hoặc tính năng.

Mô tả bình thường có thể bao gồm các ví dụ, đồ họa hoặc mô phỏng, và nó thường được sử dụng trong các tài liệu yêu cầu phần mềm. Mục đích của nó là giúp cho các bên liên quan có thể đồng thuận về các yêu cầu và tính năng của phần mềm một cách dễ dàng và đơn giản. Nó cũng giúp đảm bảo rằng các tính năng và yêu cầu của phần mềm được hiểu đúng cách và đáp ứng được nhu cầu của người dùng.

5.3.2 Lược đồ ca sử dụng

Use Case Diagram: Lược đồ Use Case là một công cụ trực quan để mô tả các Use Case và các tương tác giữa các tác nhân và hệ thống trong quá trình sử dụng phần mềm. Sơ đồ Use Case là một phần quan trọng của phương pháp phát triển phần mềm hướng Use Case.

Sơ đồ Use Case bao gồm các thành phần sau:

Use Case: Một Use Case là một kịch bản hoặc tình huống mà người dùng cuối sẽ sử dụng phần mềm để đáp ứng nhu cầu của họ. Mỗi Use Case được biểu diễn bằng một hình chữ nhật trong sơ đồ Use Case.

Tác nhân: Tác nhân là một thực thể bên ngoài hệ thống, như là người dùng cuối hoặc một hệ thống khác, tương tác với hệ thống để thực hiện các Use Case. Mỗi tác nhân được biểu diễn bằng một hình vuông nhỏ trong sơ đồ Use Case.

Kết nối: Kết nối là các mũi tên trên sơ đồ Use Case, biểu thị các tương tác giữa các Use Case và tác nhân. Các mũi tên này chỉ ra rằng tác nhân được sử dụng để thực hiện Use Case hoặc Use Case sử dụng tác nhân để thực hiện một tác vụ cụ thể.

Lư đồ Use Case giúp định hình và hiểu rõ hơn các tính năng và yêu cầu của phần mềm, cũng như giúp đảm bảo rằng phần mềm được thiết kế để đáp ứng nhu cầu và mục tiêu của người dùng cuối.

5.3.3 Ma trận truy vết

"Traceability Matrix" là "Ma trận truy vết".

Ma trận truy vết là một công cụ quản lý yêu cầu trong quá trình phát triển phần mềm, giúp theo dõi quá trình phát triển từ các yêu cầu đầu vào cho đến các yêu cầu đầu ra và các thành phần liên quan. Ma trận truy vết được sử dụng để đảm bảo rằng tất cả các yêu cầu được phát triển và triển khai đúng cách và đáp ứng đầy đủ các yêu cầu của khách hàng.

Ma trận truy vết bao gồm hai loại yêu cầu chính:

Yêu cầu đầu vào: là các yêu cầu được đưa ra bởi khách hàng hoặc những người quan tâm đến sản phẩm phần mềm, chẳng hạn như yêu cầu chức năng, yêu cầu phi chức năng, yêu cầu kỹ thuật, v.v.

Yêu cầu đầu ra: là các thành phần phần mềm, bao gồm các tính năng, các ca sử dụng, mã nguồn, tài liệu hướng dẫn, v.v. được phát triển và triển khai để đáp ứng các yêu cầu đầu vào.

Ma trận truy vết giúp các nhà phát triển phần mềm theo dõi các yêu cầu đầu vào và đầu ra, đảm bảo rằng tất cả các yêu cầu đầu vào được triển khai và đáp ứng đầy đủ. Ma trận truy vết cũng giúp quản lý rủi ro trong quá trình phát triển phần mềm và cải thiện chất lượng của sản phẩm phần mềm cuối cùng.

5.3.4 Miêu tả đầy đủ

Fully-Dressed Description (Mô tả đầy đủ) là một cách tiếp cận phát triển Use Case trong kỹ thuật phần mềm. Nó bao gồm tất cả các yếu tố cần thiết để mô tả một Use Case một cách đầy đủ và chi tiết, bao gồm các yếu tố như tên Use Case, các bước trong quá trình sử dụng, các điều kiện tiên quyết, các điều kiện sau khi thực hiện, và các yếu tố khác như actor và mục tiêu.

Cụ thể, một Fully-Dressed Description bao gồm:

Tên Use Case: đây là tên đại diện cho Use Case và nó phải rõ ràng và dễ hiểu.

Actor: đây là những người hoặc hệ thống mà sẽ tương tác với hệ thống của bạn trong Use Case.

Mục tiêu: đây là mục tiêu chính của Use Case, mô tả mục đích của Use Case và điều gì được mong đợi từ hệ thống.

Các bước trong quá trình sử dụng: bao gồm các hành động cụ thể mà các actor phải thực hiện để đạt được mục tiêu của Use Case.

Điều kiện tiên quyết: đây là các điều kiện mà phải được đáp ứng trước khi Use Case được thực hiện.

Điều kiện sau khi thực hiện: đây là các điều kiện mà phải được đáp ứng sau khi Use Case được thực hiện.

Triggers: đây là các sự kiện hoặc hành động mà kích hoạt Use Case.

Kết quả dự kiến: đây là kết quả dự kiến khi Use Case được thực hiện.

Các yếu tố bổ sung: có thể bao gồm các yêu cầu phi chức năng, các giả định hoặc các ràng buộc khác của Use Case.

Một Fully-Dressed Description cung cấp một cách tiếp cận rõ ràng và cụ thể cho việc mô tả các Use Case, giúp các nhà phát triển phần mềm hiểu rõ hơn về các yêu cầu và các chức năng cần thiết cho hệ thống của họ.

5.4 Sơ đồ luồng

System Sequence Diagrams (SSD) là một kỹ thuật phát triển phần mềm để mô tả luồng tương tác giữa người dùng và hệ thống, tập trung vào các tác nhân bên ngoài hệ thống và các thông điệp được trao đổi giữa chúng. SSD thường được sử dụng trong kỹ thuật phát triển phần mềm hướng đối tượng để mô hình hoá các chức năng của hệ thống.

SSD bao gồm các thành phần sau:

Các thông điệp (Message): các thông điệp được trao đổi giữa các tác nhân và hệ thống. Chúng có thể là yêu cầu, phản hồi hoặc thông tin liên lạc giữa các tác nhân và hệ thống.

Hệ thống (System): là tập hợp các đối tượng và thành phần của hệ thống để thực hiện các chức năng.

SSD thường được sử dụng để mô hình hoá các Use Case trong kỹ thuật phát triển phần mềm. Nó cung cấp một cách trực quan để hiểu các tương tác giữa các tác nhân và hệ thống. SSD giúp nhà phát triển phần mềm hiểu rõ hơn các yêu cầu chức

năng của hệ thống và xác định các tương tác giữa các tác nhân và hệ thống. Nó cũng giúp các nhà phát triển phần mềm phát hiện ra các lỗ hổng hoặc các vấn đề trong thiết kế hệ thống.

6. ĐÁNH GIÁ ĐỘ PHỨC TẠP CÁC CA SỬ DỤNG

Effort Estimation using Use Case Points (UCP) là một phương pháp ước tính công sức trong kỹ thuật phần mềm, dựa trên các Use Case (các tác vụ chức năng của hệ thống) để ước tính khối lượng công việc cần thực hiện. Phương pháp này dựa trên việc đánh giá khối lượng công việc và độ phức tạp của một hệ thống phần mềm dựa trên các Use Case đã xác định.

Để ước tính khối lượng công việc bằng UCP, các thông số sau được sử dụng:

1. Điểm Use Case (UCP): Được tính bằng tổng điểm của tất cả các Use Case trong hệ thống, được tính bằng cách thực hiện các bước sau:

- Đếm số lượng Use Case trong hệ thống.
- Xác định loại Use Case (đơn giản, trung bình, phức tạp).
- Gán điểm cho mỗi loại Use Case theo bảng điểm đã được xác định.

2. Độ phức tạp kỹ thuật (TCC): Được tính bằng tổng các yếu tố kỹ thuật của hệ thống, bao gồm mức độ sử dụng các công nghệ mới, độ phức tạp của cấu trúc hệ thống và độ tin cậy của hệ thống.
3. Độ phức tạp môi trường (ECF): Được tính bằng tổng các yếu tố môi trường của hệ thống, bao gồm mức độ sử dụng các thiết bị phần cứng và phần mềm mới, yêu cầu bảo mật và độ tin cậy của hệ thống.

- Sau khi tính toán các thông số trên, công sức ước tính có thể được tính bằng cách sử dụng công thức sau:
- Effort = UCP x TCC x ECF

Phương pháp ước tính công sức bằng UCP có thể giúp nhà phát triển phần mềm ước tính chi phí và thời gian thực hiện các dự án phần mềm. Nó cũng cung cấp một cách tiếp cận chính xác để đánh giá khối lượng công việc cần thực hiện trong quá trình phát triển phần mềm.

7. PHÂN TÍCH MIỀN

Domain Analysis là quá trình phân tích và hiểu rõ về lĩnh vực hoạt động của một hệ thống phần mềm cụ thể. Nó bao gồm việc phân tích các yêu cầu chức năng và phi chức năng của hệ thống, các đối tượng và mối quan hệ giữa chúng, cũng như các quy trình kinh doanh và yêu cầu của khách hàng. Mục đích của Domain Analysis là giúp nhà phát triển phần mềm có cái nhìn toàn diện về lĩnh vực hoạt động của hệ thống và giúp họ xác định các yêu cầu chức năng và phi chức năng của hệ thống một cách chính xác.

Quá trình Domain Analysis bao gồm các bước sau:

1. Xác định các đối tượng chính: Đây là các thực thể quan trọng nhất trong lĩnh vực hoạt động của hệ thống, bao gồm cả các đối tượng bên trong và bên ngoài hệ thống.
2. Xác định các mối quan hệ giữa các đối tượng: Đây là các mối quan hệ giữa các đối tượng trong hệ thống, bao gồm cả các mối quan hệ chức năng và phi chức năng.
3. Phân tích quy trình kinh doanh: Đây là quá trình phân tích các quy trình kinh doanh trong lĩnh vực hoạt động của hệ thống, bao gồm cả quy trình hiện tại và quy trình mong muốn.
4. Xác định yêu cầu chức năng: Dựa trên các đối tượng và mối quan hệ giữa chúng, ta có thể xác định các yêu cầu chức năng của hệ thống.
5. Xác định yêu cầu phi chức năng: Ngoài các yêu cầu chức năng, ta cũng phải xác định các yêu cầu phi chức năng của hệ thống, bao gồm cả các yêu cầu về hiệu suất, độ tin cậy và bảo mật.

Quá trình Domain Analysis giúp nhà phát triển phần mềm có cái nhìn toàn diện về lĩnh vực hoạt động của hệ thống và giúp họ xác định các yêu cầu chức năng và phi chức năng của hệ thống một cách chính xác. Nó cũng giúp đảm bảo rằng hệ thống được thiết kế và triển khai đáp ứng được các yêu cầu.

7.1 Mô hình miền (Domain Model)

Domain Model là một biểu đồ được sử dụng để mô tả các đối tượng quan trọng trong một lĩnh vực hoạt động cụ thể và mối quan hệ giữa chúng. Nó là một phần quan trọng của quá trình phân tích hướng đối tượng trong phát triển phần mềm và giúp xác định các yêu cầu chức năng và phi chức năng của hệ thống.

Domain Model bao gồm hai phần chính là các đối tượng và mối quan hệ giữa chúng. Các đối tượng thường được biểu diễn bằng các hình hộp và các mối quan hệ

giữa chúng thường được biểu diễn bằng các mũi tên. Domain Model thường được xây dựng dựa trên các thông tin thu thập được từ Domain Analysis và sử dụng cho việc xây dựng các yêu cầu và thiết kế hệ thống.

Việc tạo ra một Domain Model hữu ích để trình bày các đối tượng quan trọng trong hệ thống, định nghĩa các thuộc tính và phương thức của chúng, và mô tả mối quan hệ giữa các đối tượng. Nó cung cấp một cái nhìn tổng quan về hệ thống và giúp cho các nhà phát triển và khách hàng có thể thảo luận về cách thức triển khai các tính năng của hệ thống.

Ví dụ, nếu đang phát triển một ứng dụng đặt chỗ vé máy bay, Domain Model sẽ gồm các đối tượng như Khách hàng, Vé máy bay, Chuyến bay, Sân bay, v.v. Mỗi đối tượng sẽ có các thuộc tính như tên, địa chỉ, giá vé, thời gian khởi hành, v.v. Các mối quan hệ giữa chúng sẽ được biểu diễn bằng các mũi tên, ví dụ như mỗi khách hàng có thể đặt nhiều vé máy bay và mỗi vé máy bay được đặt bởi một khách hàng.

7.1.1 Xác định các khái niệm (Concept Definitions)

Trong Domain Model, Concept Definitions là mô tả chi tiết về các khái niệm, thuật ngữ, đối tượng, mối quan hệ giữa các đối tượng và các quy trình hoạt động trong lĩnh vực hoạt động của hệ thống. Concept Definitions được sử dụng để mô tả các khái niệm quan trọng trong Domain Model và giúp cho các thành viên trong dự án hiểu rõ hơn về cách thức hoạt động của hệ thống.

Các Concept Definitions thường được sử dụng để giải thích các thuật ngữ chuyên ngành trong lĩnh vực hoạt động của hệ thống, cũng như mô tả các đối tượng và các mối quan hệ giữa chúng. Các định nghĩa này có thể được sử dụng để xác định các yêu cầu của hệ thống và hướng đến mục tiêu của dự án.

Ví dụ, trong một hệ thống quản lý thư viện, các Concept Definitions có thể bao gồm:

- Thư viện: một tổ chức chứa các tài liệu và sách cho mượn.
- Thành viên: người sử dụng thư viện và có thể mượn sách.
- Sách: một loại tài liệu được chứa trong thư viện và có thể được mượn.
- Phiếu mượn: một bản ghi trong hệ thống thể hiện mượn sách của một thành viên.
- Thủ thư: người quản lý thư viện và có thể tạo phiếu mượn sách cho thành viên.

- Trạng thái sách: Trạng thái của một cuốn sách, ví dụ như đang mượn, trả lại hoặc bị mất.

Các Concept Definitions giúp các nhà phát triển và các chuyên gia trong lĩnh vực hoạt động của hệ thống hiểu rõ hơn về các yêu cầu của hệ thống và cách thức hoạt động của nó.

7.1.2 Xác định mối quan hệ

Trong Domain Model, Association Definitions là mô tả các mối quan hệ giữa các đối tượng trong hệ thống. Các quan hệ này có thể là đơn giản, hoặc có thể là phức tạp với nhiều mức độ và cấp độ khác nhau. Các Association Definitions được sử dụng để hiểu rõ hơn về cách các đối tượng trong hệ thống tương tác với nhau và hỗ trợ phát triển và hiện thực hóa hệ thống.

Ví dụ, trong một hệ thống quản lý bán hàng, Association Definitions có thể bao gồm:

1. Khách hàng - đơn hàng: Mối quan hệ giữa khách hàng và đơn hàng, trong đó mỗi đơn hàng được liên kết với một khách hàng cụ thể.
2. Sản phẩm - đơn hàng: Mối quan hệ giữa sản phẩm và đơn hàng, trong đó mỗi sản phẩm được liên kết với một đơn hàng cụ thể.
3. Đơn hàng - thanh toán: Mối quan hệ giữa đơn hàng và thanh toán, trong đó một đơn hàng có thể được thanh toán bởi nhiều phương thức khác nhau.
4. Nhân viên - đơn hàng: Mối quan hệ giữa nhân viên và đơn hàng, trong đó mỗi đơn hàng được xử lý bởi một nhân viên cụ thể.

Các Association Definitions giúp cho các nhà phát triển hiểu rõ hơn về các mối quan hệ giữa các đối tượng trong hệ thống và cung cấp một bức tranh toàn cảnh về cách thức hoạt động của hệ thống. Nó cũng giúp định nghĩa các yêu cầu cho các mối quan hệ và xác định các tương tác cần thiết giữa các đối tượng để hỗ trợ các chức năng của hệ thống.

7.1.3 Xác định các thuộc tính

Trong Domain Model, Attribute Definitions là mô tả các thuộc tính của các đối tượng trong hệ thống. Các thuộc tính này được sử dụng để mô tả các tính chất của các đối tượng, như kích thước, trạng thái, màu sắc, v.v. Các Attribute Definitions được sử dụng để hiểu rõ hơn về các thuộc tính của các đối tượng trong hệ thống và hỗ trợ phát triển và hiện thực hóa hệ thống.

Ví dụ, trong một hệ thống quản lý bán hàng, Attribute Definitions có thể bao gồm:

- Sản phẩm: Kích thước, trọng lượng, giá bán, số lượng tồn kho, mô tả sản phẩm, v.v.
- Đơn hàng: Ngày đặt hàng, trạng thái đơn hàng, phương thức thanh toán, v.v.
- Khách hàng: Tên, địa chỉ, email, số điện thoại, loại khách hàng, v.v.
- Nhân viên: Tên, địa chỉ, email, số điện thoại, chức vụ, v.v.

Các Attribute Definitions giúp cho các nhà phát triển hiểu rõ hơn về các thuộc tính của các đối tượng trong hệ thống và cung cấp một bức tranh toàn cảnh về các tính chất của hệ thống. Nó cũng giúp định nghĩa các yêu cầu cho các thuộc tính và xác định các giá trị của chúng để hỗ trợ các chức năng của hệ thống.

7.1.4 Ma trận truy xuất

Traceability Matrix trong Domain Model là một bảng liệt kê các thành phần của Domain Model (bao gồm các khái niệm, các quan hệ giữa các khái niệm và các thuộc tính) và các tài liệu liên quan đến chúng. Nó cung cấp cho nhà phát triển và người quản lý dự án một cách tiếp cận hệ thống để hiểu rõ các yêu cầu, thiết kế và triển khai của hệ thống.

Traceability Matrix trong Domain Model cung cấp một phương tiện để theo dõi các liên kết giữa các thành phần khác nhau của hệ thống. Nó giúp đảm bảo rằng các yêu cầu của khách hàng được đáp ứng, các thiết kế được xây dựng đúng cách và các thành phần được phát triển và kiểm tra một cách hợp lý. Điều này có thể giúp đảm bảo rằng hệ thống được phát triển theo đúng hướng và đáp ứng được yêu cầu của khách hàng.

Một số thông tin mà Traceability Matrix trong Domain Model có thể bao gồm:

- Các khái niệm của Domain Model
- Các quan hệ giữa các khái niệm
- Các thuộc tính của các khái niệm
- Các yêu cầu của khách hàng liên quan đến các khái niệm trong Domain Model
- Các ca sử dụng liên quan đến các khái niệm trong Domain Model
- Các thiết kế liên quan đến các khái niệm trong Domain Model

- Các đặc tả kiểm thử liên quan đến các khái niệm trong Domain Model

Traceability Matrix trong Domain Model giúp đảm bảo rằng các thành phần của hệ thống được đáp ứng đầy đủ các yêu cầu và giúp cho quản lý dự án đánh giá được tiến độ của dự án. Nó cũng giúp tăng tính hiệu quả và độ chính xác trong quản lý và triển khai dự án.

7.2 Các hợp đồng hoạt động hệ thống (System Operation Contracts)

System Operation Contracts là một công cụ trong Domain Analysis giúp mô tả chi tiết các thao tác hoặc chức năng của hệ thống và quy định trách nhiệm và hành vi của các đối tượng khác nhau trong hệ thống.

Mỗi System Operation Contract mô tả một chức năng hoặc thao tác cụ thể của hệ thống, bao gồm các trạng thái khác nhau của hệ thống và các điều kiện tiên quyết và điều kiện sau khi thực hiện chức năng hoặc thao tác đó. System Operation Contract cũng mô tả các đối tượng trong hệ thống có liên quan đến chức năng hoặc thao tác đó và trách nhiệm của các đối tượng đó.

System Operation Contract thường được sử dụng để mô tả các chức năng hoặc thao tác của hệ thống trong phân tích yêu cầu, thiết kế và triển khai hệ thống. Chúng được sử dụng để phân tích chi tiết các yêu cầu chức năng của hệ thống và đảm bảo rằng hệ thống được phát triển và triển khai đúng cách.

Ví dụ, một System Operation Contract có thể mô tả chức năng của hệ thống để đăng ký người dùng mới. Các điều kiện tiên quyết cho chức năng này có thể bao gồm việc người dùng cung cấp thông tin cá nhân chính xác và đầy đủ, trong khi các điều kiện sau khi thực hiện chức năng có thể bao gồm việc hệ thống cập nhật cơ sở dữ liệu người dùng mới. System Operation Contract cũng sẽ xác định các đối tượng trong hệ thống liên quan đến chức năng này, chẳng hạn như người dùng, cơ sở dữ liệu người dùng và giao diện người dùng.

7.3 Mô hình toán học

Trong Domain Analysis của Kỹ thuật phần mềm, Mathematical Model là một công cụ được sử dụng để biểu diễn một cách toán học các khái niệm và quan hệ giữa các đối tượng trong hệ thống.

Một Mathematical Model được xây dựng dựa trên các phương trình, hệ thức toán học hoặc các thuật toán được sử dụng để mô tả hoạt động của hệ thống. Mathematical Model giúp định nghĩa các ràng buộc và điều kiện về tính chất, đặc

tính của hệ thống, cũng như các giới hạn và hạn chế trong việc thiết kế và triển khai hệ thống.

Mathematical Model được sử dụng để giải quyết các vấn đề phức tạp trong lĩnh vực Kỹ thuật phần mềm, chẳng hạn như đánh giá hiệu suất của hệ thống, dự đoán khả năng mở rộng và đo lường sự tương tác giữa các thành phần khác nhau của hệ thống.

Ví dụ, một Mathematical Model có thể được sử dụng để mô tả một hệ thống phần mềm chuyên dụng trong lĩnh vực y tế để dự đoán thời gian phản hồi của hệ thống trên các yêu cầu khác nhau. Mathematical Model sẽ định nghĩa các ràng buộc về tính chất của hệ thống, bao gồm tốc độ xử lý, dung lượng và tài nguyên của hệ thống. Nó cũng sẽ xác định các giới hạn và hạn chế trong việc thiết kế và triển khai hệ thống, bao gồm các tài nguyên cần thiết và mức độ mở rộng của hệ thống.

8. BIỂU ĐỒ TƯƠNG TÁC (Interaction Diagram)

Biểu đồ tương tác được sử dụng để mô tả các tương tác giữa các đối tượng trong một hệ thống. Interaction Diagram bao gồm hai loại chính là Sequence Diagram và Communication Diagram.

- Sequence Diagram được sử dụng để mô tả các tương tác giữa các đối tượng trong một hệ thống theo thứ tự thời gian. Biểu đồ này biểu thị các thông điệp được gửi và nhận bởi các đối tượng, cũng như thời gian mà chúng được gửi và nhận. Sequence Diagram thường được sử dụng để mô tả các ca sử dụng cụ thể trong hệ thống.
- Communication Diagram, còn được gọi là Collaboration Diagram, được sử dụng để mô tả các tương tác giữa các đối tượng trong một hệ thống dưới dạng một mạng lưới các đối tượng. Biểu đồ này biểu thị các đối tượng và các thông điệp được gửi và nhận giữa chúng, cũng như các quan hệ giữa các đối tượng.

Cả Sequence Diagram và Communication Diagram đều hỗ trợ việc xác định các lỗi, rủi ro và các vấn đề khác trong thiết kế và triển khai hệ thống. Chúng cũng giúp định nghĩa các tương tác giữa các đối tượng, tạo ra một cách để các nhà phát triển và các nhà quản lý dự án có thể dễ dàng hiểu và đánh giá các yêu cầu của hệ thống, và giúp cải thiện hiệu suất và độ tin cậy của hệ thống.

9. LƯỢC ĐỒ LỚP VÀ MIÊU TẢ GIAO DIỆN

Class Diagram là một biểu đồ được sử dụng để mô tả các lớp trong một hệ thống và các quan hệ giữa chúng. Biểu đồ này cho phép các nhà phát triển hình dung được cấu trúc của hệ thống, bao gồm các đối tượng, thuộc tính, phương thức và các quan hệ giữa chúng.

Interface Specification là một tài liệu mô tả cách mà các lớp và đối tượng trong hệ thống tương tác với nhau thông qua các giao diện. Tài liệu này cũng mô tả các yêu cầu và các giới hạn của giao diện, bao gồm các đối tượng, phương thức, thuộc tính và các quan hệ giữa chúng.

Class Diagrams và Interface Specification là hai thành phần quan trọng trong Kỹ thuật phần mềm, và chúng thường được sử dụng trong quá trình thiết kế hệ thống.

Cụ thể, Class Diagrams cung cấp cho các nhà phát triển một cách nhìn tổng quan về cấu trúc của hệ thống, giúp họ tìm ra các lớp và quan hệ giữa chúng một cách rõ ràng và dễ hiểu. Trong khi đó, Interface Specification mô tả cách mà các lớp và đối tượng trong hệ thống tương tác với nhau thông qua các giao diện, giúp đảm bảo tính đúng đắn và hoạt động hiệu quả của hệ thống.

Khi sử dụng Class Diagrams và Interface Specification, các nhà phát triển có thể dễ dàng hiểu và đánh giá các yêu cầu của hệ thống, giúp cải thiện tính tin cậy và hiệu suất của hệ thống.

10. THIẾT KẾ VÀ KIẾN TRÚC HỆ THỐNG

System Architecture và Design là phần quan trọng của quá trình phát triển phần mềm, trong đó kiến trúc của hệ thống được thiết kế và triển khai. Kiến trúc của hệ thống bao gồm cấu trúc các thành phần của hệ thống và cách chúng tương tác với nhau.

Cụ thể, System Architecture bao gồm các yếu tố sau:

- Phân tích yêu cầu hệ thống: đây là bước đầu tiên trong thiết kế kiến trúc hệ thống, trong đó các yêu cầu của khách hàng và các bên liên quan được phân tích và xác định.
- Xác định cấu trúc kiến trúc hệ thống: sau khi yêu cầu được phân tích và xác định, kiến trúc hệ thống được thiết kế và triển khai, bao gồm các thành phần cơ bản, quan hệ giữa chúng và cách chúng tương tác với nhau.
- Lựa chọn công nghệ và nền tảng: để triển khai kiến trúc hệ thống, các công nghệ và nền tảng phù hợp cần được lựa chọn và triển khai.
- Đảm bảo tính khả chuyển và mở rộng: kiến trúc hệ thống cần đảm bảo tính khả chuyển và mở rộng để có thể đáp ứng được các yêu cầu và nhu cầu của khách hàng và các bên liên quan.

System Design là quá trình thiết kế chi tiết các thành phần của hệ thống dựa trên kiến trúc đã được thiết kế trước đó. Các thành phần của hệ thống, bao gồm các module, chức năng, giao diện và cơ chế xử lý được thiết kế và triển khai. Trong quá trình thiết kế, các yêu cầu chức năng và phi chức năng của hệ thống được xác định và đảm bảo tính khả chuyển và mở rộng của hệ thống.

System Architecture và Design đóng vai trò quan trọng trong việc đảm bảo tính tin cậy, khả năng mở rộng và hiệu suất của hệ thống. Nó cũng giúp các nhà phát triển đưa ra các quyết định thiết kế chính xác và tối ưu hóa hệ thống, tăng tính hiệu quả và giảm thiểu chi phí phát triển.

10.1 Kiểu kiến trúc

Một số kiểu kiến trúc phổ biến trong kỹ thuật phần mềm bao gồm:

- Kiến trúc lớp: Kiểu kiến trúc này chia hệ thống thành một loạt các lớp, mỗi lớp thực hiện một tập hợp nhiệm vụ cụ thể. Các lớp giao tiếp với các lớp ngay trên và dưới của chúng, và các lớp cấp cao phụ thuộc vào

các lớp cấp thấp. Kiểu kiến trúc này cung cấp tính tổ chức và cho phép bảo trì và cập nhật dễ dàng.

- Kiến trúc client-server: Trong kiểu kiến trúc này, hệ thống được chia thành hai thành phần: một thành phần client tương tác với người dùng và một thành phần server cung cấp dịch vụ hoặc tài nguyên cho client. Kiểu kiến trúc này thường được sử dụng trong các hệ thống dựa trên web, trong đó client là trình duyệt web và server là máy chủ web.
- Kiến trúc microservices: Kiểu kiến trúc này phân tách hệ thống thành một tập hợp các dịch vụ nhỏ, độc lập mà giao tiếp với nhau sử dụng các giao thức nhẹ. Mỗi dịch vụ thực hiện một chức năng cụ thể, và hệ thống chủ yếu phục vụ các mục đích kinh doanh. Kiểu kiến trúc này cung cấp tính linh hoạt và độ phân tán cao.

10.2 Lược đồ gói các hệ thống con (Subsystem Package Diagram)

Subsystem Package Diagram là một loại biểu đồ trong kỹ thuật phần mềm dùng để mô tả cấu trúc phân cấp của các thành phần trong hệ thống. Nó cho phép các thành phần phụ thuộc vào nhau được nhóm lại thành các gói (packages), giúp cho việc quản lý và phát triển hệ thống dễ dàng hơn.

Các gói trong Subsystem Package Diagram có thể chứa các thành phần khác như các lớp (class), interface, use case, component, activity, và các gói con khác. Các gói cũng có thể chứa các phụ thuộc về mã, tài liệu và các tài nguyên khác.

Subsystem Package Diagram thường được sử dụng trong giai đoạn thiết kế hệ thống để mô tả cấu trúc phân cấp của các thành phần và quan hệ giữa chúng, từ đó giúp cho việc triển khai hệ thống được dễ dàng và có tổ chức hơn.

10.3 Ánh xác các hệ thống con sang phần cứng

Trong kỹ thuật phần mềm, việc ánh xạ (mapping) các subsystems (các hệ thống con) vào phần cứng (hardware) là một bước quan trọng trong thiết kế và triển khai hệ thống.

Để ánh xạ các subsystems vào phần cứng, trước tiên cần xác định các yêu cầu phần cứng cho mỗi subsystem. Các yêu cầu này có thể bao gồm các thông số kỹ thuật về bộ vi xử lý, bộ nhớ, băng thông mạng, bộ điều khiển, cổng giao tiếp và các yếu tố khác liên quan đến phần cứng.

Sau đó, các subsystems được ánh xạ vào các thiết bị phần cứng tương ứng. Việc ánh xạ này có thể được thực hiện bằng cách sử dụng các công cụ mô phỏng hoặc

các phần mềm thiết kế mạch điện tử để đảm bảo rằng các yêu cầu phần cứng được đáp ứng và các thành phần phần cứng được kết nối đúng cách.

Cuối cùng, kiểm thử phần cứng cần được thực hiện để đảm bảo rằng việc ánh xạ các subsystems vào phần cứng được thực hiện đúng và hệ thống hoạt động như mong đợi.

10.4 Lưu trữ (Persistent Data Storage_

Persistent Data Storage là khái niệm trong kỹ thuật phần mềm, đề cập đến cách lưu trữ dữ liệu một cách vĩnh viễn để có thể lưu giữ thông tin giữa các phiên làm việc và bảo đảm tính toàn vẹn của dữ liệu.

Trong một hệ thống phần mềm, dữ liệu thường được lưu trữ trong các tệp hoặc cơ sở dữ liệu. Tùy vào tính chất của ứng dụng, dữ liệu có thể được lưu trữ trên các loại đĩa cứng, ổ đĩa SSD, bộ nhớ flash, hay các loại lưu trữ đám mây (cloud storage).

Trong các hệ thống phân tán, dữ liệu có thể được lưu trữ trên nhiều nút khác nhau, được đồng bộ hóa với nhau để đảm bảo tính toàn vẹn của dữ liệu.

Việc lựa chọn và thiết kế phương pháp lưu trữ dữ liệu phù hợp là một phần quan trọng của thiết kế hệ thống phần mềm. Một cách lưu trữ không tốt có thể dẫn đến việc mất dữ liệu, hoặc làm giảm hiệu suất của hệ thống.

Ngoài ra, việc đảm bảo tính bảo mật của dữ liệu cũng là một yếu tố quan trọng cần được xem xét trong thiết kế và triển khai hệ thống lưu trữ dữ liệu.

10.5 Giao thức mạng

Trong kỹ thuật phần mềm, Network Protocol (giao thức mạng) là một tập các quy tắc và tiêu chuẩn được sử dụng để điều khiển và hỗ trợ giao tiếp giữa các thiết bị trong mạng máy tính. Giao thức này đảm bảo rằng các thiết bị trong mạng có thể truyền thông với nhau một cách hiệu quả và đáng tin cậy.

Một số giao thức mạng phổ biến bao gồm TCP/IP (Transmission Control Protocol/Internet Protocol), HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), DNS (Domain Name System) và SSH (Secure Shell). Mỗi giao thức có chức năng và mục đích sử dụng khác nhau, tùy thuộc vào loại mạng và các yêu cầu của ứng dụng cụ thể.

10.6 Luồng điều khiển toàn cục (Global Control Flow)

Global Control Flow (luồng điều khiển toàn cục) là một khái niệm trong thiết kế kiến trúc và thiết kế hệ thống trong kỹ thuật phần mềm, đề cập đến quá trình điều khiển và tương tác giữa các thành phần hệ thống.

Global Control Flow mô tả cách mà các thành phần của hệ thống giao tiếp với nhau, trao đổi dữ liệu và thực hiện các hoạt động. Điều này bao gồm các trạng thái của hệ thống, các sự kiện và tương tác giữa các thành phần của hệ thống.

Trong thiết kế kiến trúc, Global Control Flow là một phần quan trọng của thiết kế, giúp đảm bảo tính đúng đắn và hiệu quả của hệ thống. Các kiến trúc sử dụng Global Control Flow để xác định các thành phần của hệ thống, các trạng thái của chúng, và các tương tác giữa các thành phần.

Trong thiết kế hệ thống, Global Control Flow được sử dụng để mô tả cách thức mà hệ thống hoạt động, các trạng thái của hệ thống, và cách thức mà các tương tác được quản lý và điều khiển. Các thiết kế hệ thống sử dụng Global Control Flow để đảm bảo tính đúng đắn, an toàn và hiệu quả của hệ thống.

10.7 Các yêu cầu phần cứng (Hardware Requirements)

Trong thiết kế kiến trúc và thiết kế hệ thống, các yêu cầu phần cứng thường bao gồm:

1. Bộ vi xử lý (CPU): CPU là trung tâm xử lý của hệ thống và cần phải có tốc độ xử lý đủ để đáp ứng các yêu cầu tính toán và xử lý dữ liệu của hệ thống.
2. Bộ nhớ: Hệ thống cần có đủ bộ nhớ để lưu trữ các dữ liệu và chương trình. Bộ nhớ càng lớn, hệ thống có thể xử lý được các ứng dụng phức tạp hơn.
3. Bộ lưu trữ: Hệ thống cần có bộ lưu trữ đủ lớn để lưu trữ các dữ liệu của người dùng và dữ liệu của ứng dụng. Các lựa chọn bộ lưu trữ bao gồm ổ cứng (HDD) hoặc ổ đĩa rắn (SSD).
4. Màn hình: Màn hình là giao diện trực tiếp giữa người dùng và hệ thống. Hệ thống cần có màn hình đủ lớn và độ phân giải cao để hiển thị thông tin một cách rõ ràng và chi tiết.
5. Bàn phím và chuột: Bàn phím và chuột là các thiết bị nhập liệu cho hệ thống. Chúng cần phải có độ nhạy cao để đáp ứng nhu cầu nhập liệu của người dùng.

6. Card mạng: Card mạng là thiết bị cho phép hệ thống kết nối với các mạng khác nhau. Hệ thống cần có card mạng đủ mạnh để đáp ứng nhu cầu kết nối của người dùng.
7. Nguồn cung cấp điện: Nguồn cung cấp điện cần đáp ứng nhu cầu sử dụng điện của hệ thống. Cần đảm bảo rằng hệ thống được cấp đủ điện để hoạt động ổn định và tránh các tình huống mất điện đột ngột.

11. CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

11.1 Thuật toán

Trong kỹ thuật phần mềm, thuật toán là một bước xử lý được thực hiện bởi chương trình máy tính để giải quyết một vấn đề. Thuật toán bao gồm một chuỗi các hành động cụ thể được thực hiện theo một trình tự nhất định để giải quyết vấn đề được đưa ra. Các thuật toán phổ biến trong kỹ thuật phần mềm bao gồm các thuật toán sắp xếp, tìm kiếm, mã hóa/giải mã, tính toán đường đi ngắn nhất, phân loại và ràng buộc. Thuật toán được thiết kế để có hiệu suất tối ưu, tức là tốn ít thời gian và tài nguyên tính toán nhất có thể. Việc lựa chọn thuật toán phù hợp đóng vai trò quan trọng trong việc xây dựng phần mềm hiệu quả.

11.2 Cấu trúc dữ liệu

Cấu trúc dữ liệu là một phần quan trọng trong kỹ thuật phần mềm, được sử dụng để tổ chức và lưu trữ dữ liệu một cách hiệu quả. Cấu trúc dữ liệu đóng vai trò quan trọng trong việc tối ưu hóa hiệu suất của chương trình, giảm thiểu thời gian truy cập dữ liệu và tối đa hóa khả năng mở rộng của hệ thống.

Các cấu trúc dữ liệu phổ biến trong kỹ thuật phần mềm bao gồm:

Mảng (Array): Một cấu trúc dữ liệu đơn giản nhưng mạnh mẽ, được sử dụng để lưu trữ một danh sách các giá trị cùng loại.

Danh sách liên kết (Linked List): Một cấu trúc dữ liệu động, cho phép thêm hoặc xóa phần tử một cách linh hoạt.

Cây (Tree): Một cấu trúc dữ liệu phân cấp, được sử dụng để lưu trữ dữ liệu theo thứ tự.

Đồ thị (Graph): Một cấu trúc dữ liệu được sử dụng để biểu diễn các mối quan hệ giữa các đối tượng.

Bảng băm (Hash Table): Một cấu trúc dữ liệu được sử dụng để lưu trữ các cặp khóa/giá trị, giúp truy cập dữ liệu nhanh chóng và hiệu quả.

Các cấu trúc dữ liệu này được sử dụng rộng rãi trong kỹ thuật phần mềm để giải quyết các vấn đề phức tạp liên quan đến lưu trữ và truy xuất dữ liệu. Tuy nhiên, việc lựa chọn cấu trúc dữ liệu phù hợp với vấn đề cần giải quyết là rất quan trọng để đảm bảo tính hiệu quả và hiệu suất của chương trình.

12. CÀI ĐẶT VÀ THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG

12.1 Thiết kế

Design trong cài đặt và thiết kế giao diện người dùng của Kỹ thuật phần mềm là quá trình xác định cách mà giao diện người dùng sẽ được triển khai, bao gồm các phương thức, hàm và thuộc tính được sử dụng để hiển thị các thông tin và chức năng của hệ thống cho người dùng.

Trong quá trình này, các yêu cầu của người dùng sẽ được sử dụng để tạo ra các thiết kế khác nhau để đáp ứng các nhu cầu của họ. Các thiết kế này sẽ bao gồm các yếu tố như cấu trúc, màu sắc, kiểu chữ và các phương thức tương tác để đảm bảo tính thẩm mỹ và khả năng sử dụng của giao diện.

Sau khi thiết kế giao diện được hoàn thành, các kỹ sư phần mềm sẽ sử dụng các công cụ lập trình để triển khai giao diện này trong hệ thống. Quá trình này bao gồm việc tạo các hàm và phương thức để hiển thị các thông tin và chức năng, tương tác với các thành phần khác trong hệ thống và đảm bảo tính tương thích với các trình duyệt và thiết bị khác nhau.

12.2 Ước tính hiệu quả người dùng (User Effort Estimation)

Trong cài đặt và thiết kế giao diện người dùng của Kỹ thuật phần mềm, ước lượng hiệu quả người dùng (User Experience - UX) là một yếu tố quan trọng trong việc đảm bảo sản phẩm phần mềm đáp ứng được nhu cầu của người dùng. Các yếu tố cơ bản để đánh giá hiệu quả người dùng bao gồm:

Tốc độ: Tốc độ là thời gian mà người dùng phải chờ đợi để tải hoặc truy cập thông tin hoặc tính năng trên giao diện người dùng. Tốc độ nhanh sẽ giúp cải thiện trải nghiệm của người dùng và tạo sự thoải mái khi sử dụng sản phẩm.

Dễ sử dụng: Giao diện người dùng phải được thiết kế sao cho dễ sử dụng, đơn giản và trực quan. Người dùng không nên mất quá nhiều thời gian để tìm hiểu hoặc sử dụng tính năng của sản phẩm.

Tính tương thích: Giao diện người dùng phải được thiết kế để tương thích với nhiều loại thiết bị và trình duyệt khác nhau. Điều này đảm bảo rằng người dùng có thể truy cập sản phẩm từ bất kỳ thiết bị nào và sử dụng các tính năng một cách dễ dàng.

Thiết kế đồ họa: Thiết kế đồ họa của giao diện người dùng cần phải được tối ưu hóa để tạo ra sự thẩm mỹ và hấp dẫn. Điều này giúp người dùng có trải nghiệm tốt hơn khi sử dụng sản phẩm.

Tính tiện lợi: Giao diện người dùng cần phải đảm bảo tính tiện lợi và tiết kiệm thời gian cho người dùng. Ví dụ, nếu người dùng có thể tìm kiếm thông tin nhanh chóng hoặc truy cập tính năng một cách dễ dàng, họ sẽ có trải nghiệm tốt hơn.

Tương tác: Giao diện người dùng cần cung cấp các tương tác phù hợp để giúp người dùng tương tác với sản phẩm một cách dễ dàng. Các tính năng như kéo và thả, click và cuộn, cũng như các phương pháp tương tác khác có thể giúp cải thiện hệ thống.

13. THIẾT KẾ KIỂM THỬ

13.1 Các ca kiểm thử (Test Cases)

Trong thiết kế kiểm thử, test cases là một tập hợp các bước cụ thể mà một người kiểm thử phải thực hiện để kiểm tra tính năng hoặc chức năng của một ứng dụng hoặc hệ thống. Mỗi test case đại diện cho một trường hợp kiểm thử cụ thể để đảm bảo rằng ứng dụng hoặc hệ thống đang hoạt động đúng như mong đợi.

Một test case bao gồm các phần sau:

- Mô tả: mô tả rõ ràng mục đích của test case và các bước kiểm thử cần thiết để kiểm tra tính năng hoặc chức năng của ứng dụng hoặc hệ thống.
- Dữ liệu đầu vào: cung cấp các giá trị đầu vào mà người kiểm thử cần phải nhập để thực hiện test case.
- Bước thực hiện: đưa ra một loạt các bước cụ thể mà người kiểm thử cần thực hiện để kiểm tra tính năng hoặc chức năng của ứng dụng hoặc hệ thống.
- Dữ liệu đầu ra: đưa ra kết quả mong đợi mà người kiểm thử cần xác nhận để kiểm tra tính chính xác của ứng dụng hoặc hệ thống.
- Trạng thái kiểm thử: xác định kết quả của test case sau khi được thực hiện, bao gồm trạng thái pass (đạt) hoặc fail (không đạt).
- Tiêu chí đánh giá: xác định các tiêu chí để đánh giá kết quả kiểm thử của test case.

Các test case đóng vai trò quan trọng trong quá trình kiểm thử của một ứng dụng hoặc hệ thống, giúp đảm bảo rằng tính năng và chức năng của nó hoạt động đúng như mong đợi.

13.2 Tích hợp kiểm thử

Integration testing là một phương pháp kiểm thử phần mềm, trong đó các module con hoặc các thành phần phần mềm được kết hợp với nhau để kiểm tra tính tương tác và tính toàn vẹn của hệ thống. Trong quá trình này, các module được kiểm tra xem chúng hoạt động đúng cách khi được kết hợp với các module khác. Mục đích của integration testing là đảm bảo rằng các module con hoạt động tốt với nhau và hệ thống là hoàn chỉnh và đáng tin cậy.

Các kỹ thuật tích hợp thường được sử dụng trong integration testing bao gồm top-down integration, bottom-up integration và mixed integration. Trong top-down

integration, các module cao nhất trong hệ thống được kiểm tra trước, sau đó các module cấp thấp hơn được kết hợp với chúng. Trong bottom-up integration, các module cấp thấp nhất được kiểm tra trước, sau đó các module cao hơn được kết hợp với chúng. Trong mixed integration, một số module được kết hợp với nhau theo cách top-down và một số khác được kết hợp với nhau theo cách bottom-up.

Việc thực hiện integration testing đòi hỏi một chiến lược kiểm thử tốt và các kỹ thuật kiểm thử hiệu quả để đảm bảo rằng hệ thống được kiểm tra đầy đủ và kịp thời. Integration testing thường được thực hiện sau khi đã hoàn thành các bước kiểm thử đơn vị và trước khi kiểm thử hệ thống.

14. KẾT LUẬN VÀ CÁC CÔNG VIỆC TRONG TƯƠNG LAI

TÀI LIỆU THAM KHẢO

[1]. ….

[2]. …..

[3]. ….