**A. Nhiệm vụ:**

1. Tìm hiểu vòng đời và quy trình phát triển phẩn mềm

2. Phân biệt quy trình phát triển phần mềm Agile (Scrum) và Waterfall

3. Phân biệt kiến trúc Microservices và kiến trúc Monolithic

**B. Sinh viên thực hiện:**

1. Trần Hửu Quân

2. Phan Văn Hiếu

3 . Hồ Văn Tiến Dũng

4. Đặng Ngọc Quốc Bảo

5. Lê Văn Mẫn

6. Nguyễn Hữu Phương

7. Huỳnh Sơn Đông

I)**Vòng đời phần mềm.**

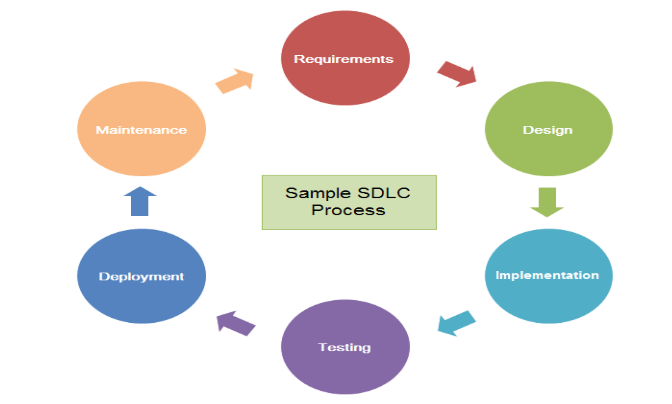
**1) Giới thiệu**

-Vòng đời phát triển phần mềm có tên tiếng Anh là Software Development Life Cycle. Quy trình phát triển phần mềm ( SDLC ), cũng được gọi là phát triển ứng dụng vòng đời , là một thuật ngữ được sử dụng trong hệ thống kỹ thuật , hệ thống thông tin và công nghệ phần mềm để mô tả một quá trình lập kế hoạch, tạo, thử nghiệm và triển khai một hệ thống thông tin. Một số mô hình SDLC hoặc các phương pháp đã được tạo ra, chẳng hạn như thác nước , xoắn ốc , phát triển phần mềm Agile , tạo mẫu nhanh , gia tăng và đồng bộ hóa và ổn định. SDLC được sử dụng trong quá trình phát triển một dự án CNTT, nó mô tả các giai đoạn khác nhau liên quan đến dự án từ bản vẽ, thông qua việc hoàn thành dự án.

-SDLC bao gồm các quy trình được lên kế hoạch một cách chi tiết nhất. Từ đó đảm bảo cho ra mắt những phần mềm chất lượng. Hiện nay có rất nhiều mô hình phát triển phần mềm với ứng dụng khác nhau. Nổi bật phải nhắc đến là: mô hình thác nước, mô hình chữ V, mô hình Agile.

-Hoạt động kiểm thử phần mềm có liên quan mật thiết với vòng đời phát triển của phần mềm. Để tạo ra thành phẩm là những phần mềm chạy mượt, chất lượng đòi hỏi bạn phải có kiến thức về SDLC. Nhiều chuyên gia phần mềm đã nhận định rằng SDLC là một cấu trúc đặc biệt. Cấu trúc này quyết định sự phát triển của phần mềm.

**2) Những giai đoạn của vòng đời phần mềm**



Vòng đời phát triển phần mềm SDLC chia thành nhiều giai đoạn khác nhau. Cụ thể là: Requirement Analysis, Design, Implementation, Testing, Deployment và Maintenance.

– Requirement Analysis: Đây là giai đoạn đầu tiên của một SDLC. Các kỹ sư sẽ tiến hành thu thập các thông tin sơ khai về phần mềm muốn phát triển.

– Design: Thiết kế tổng thể, chi tiết về phần mềm muốn phát triển. Đây là giai đoạn sơ khai để phát triển một phần mềm bất kỳ.

– Implementation: Hai bên có thể trao đổi để chốt được phương án phát triển tốt nhất. Ở giai đoạn Implementation các lập trình viên tiến hành coding theo những tài liệu và mẫu thiết kế được khách phê duyệt.

– Testing: Tiến hành kiểm thử và cập nhật những lỗi lên các tool quản lý theo yêu cầu. Kiểm thử viên và lập trình viên sẽ sửa lỗi và cập nhật lại tình hình của phần mềm. Các kiểm thử viên tiến hành test lỗi dựa vào các mã hóa đã được tạo. Tester có thể áp dụng các phương pháp kiểm thử khác nhau để phát hiện ra lỗi nhanh nhất và tiến hành fix lại lỗi.

– Deployment: Khi đã vượt qua vòng testing và sửa lỗi thành công, nếu không còn vấn đề gì thì có thể đến bước Deployment. Kiểm thử viên tiến hành triển khai phát triển phần mềm ở môi trường thực tế. Đồng thời quan sát các biến động.

– Maintenance: Bảo trì phần mềm và xử lý các lỗi phát sinh nếu có. Đây là giai đoạn kết thúc của một vòng phát triển phần mềm. Lúc này, phần mềm có thể chạy tốt, công việc của người kiểm thử viên và lập trình viên là bảo trì và tiến hành nâng cấp phần mềm.

**II) Phân biệt hai quy trình phát triển phần mềm Agile(scrum) và Waterfall**

**1)Khái niệm**

**1.1) Phương pháp Agile**

-Phương pháp Agile là một thực hành hỗ trợ sự lặp lại liên tục của quá trình phát triển và thử nghiệm trong quá trình phát triển phần mềm. Trong mô hình này, chúng tôi thực hiện đồng thời các hoạt động phát triển và thử nghiệm, ngược lại với mô hình Waterfall. Quá trình này tự động dẫn đến giao tiếp nhiều hơn giữa khách hàng, nhà phát triển, người quản lý và người thử nghiệm.

-Scrum LÀ một trong các quy trình phát triển Agile vì nó có chứa tất cả các đặc điểm cốt lõi của Agile. SCRUM là một nhóm hoạt động, được ví như là một bao gói, trong đó mọi người trong nhóm cùng nhau thực hiện các công việc. Nó thực hiện dự án với thời gian và chi phí tối thiểu. là một khuôn khổ để phát triển các sản phẩm và hệ thống phức tạp. Nó là căn cứ trong quá trình thực nghiệm và lý thuyết về kiểm soát. Scrum sử dụng một cách tiếp cận lặp đi lặp lại và tăng lên để tối ưu hóa khả năng dự đoán và kiểm soát rủi ro.

**1.2) Phương pháp Waterfall**

-Mô hình thác nước (waterfall model) là 1 trong quy mô của quy trình cải cách và phát triển phần mềm, trong những số ấy quá trình trở nên tân tiến trông y như một cái rã, cùng với những trộn được thực hiện theo bơ vơ tự chặt chẽ cùng không tồn tại sự con quay lui giỏi dancing vượt pha là: đối chiếu trải nghiệm, xây dựng, thực thi triển khai, kiểm thử, link và duy trì. là mô hình vòng đời tuần tự tuyến tính. Nó rất đơn giản để hiểu và sử dụng. Trong mô hình Waterfall, mỗi giai đoạn phải được hoàn thành trước khi giai đoạn tiếp theo có thể bắt đầu và không có sự chồng chéo trong các giai đoạn.

-Mô hình Waterfall là cách tiếp cận SDLC (chu trình phát triền phần mêm) sớm nhất được sử dụng để phát triển phần mềm.

**2) So sánh và phân biệt hai quy trình phát triển phần mềm Agile(scrum) và Waterfall**

|  |  |  |
| --- | --- | --- |
| Các tiêu chí | Agile | Waterfall |
| Phương pháp đối ứng hướng đến khách hàng | cho phép khách hàng có thể request thay đổi yêu cầu hoặc bổ sung specs bất kỳ lúc nào trong quá trình phát triển | khách hàng phải thực hiện thay đổi specs và các yêu cầu đặc tả khác trước khi bắt đầu phát triển |
| **Phương pháp đối ứng hướng đến thay đổi requirement** | do sẽ phát triển cho mỗi lần lặp (Iteration), do vậy nó có thể thay đổi linh hoạt trong suốt thời gian phát triển | các yêu cầu phải được chốt trước khi phát triển |
| Cung cấp dịch vụ | có thể cung cấp dịch vụ ở mỗi lần lặp (Iteration) nên sẽ nhanh chóng | chỉ cho phép cung cấp dịch vụ sau khi đã hoàn thành toàn bộ các quy trình, do vậy sẽ tốn thời gian hơn |
| Tính bắt buộc phải có tài liệu | không bắt buộc phải có tài liệu, tuy nhiên nếu đó là một phương thức truyền đạt thông tin hiệu quả thì vẫn nên tạo tài liệu | tài liệu cần được tạo cho mỗi quy trình và được kết nối với quy trình tiếp theo |
| Tần suất test | yêu cầu việc test thường xuyên ứng với từng implement | việc kiểm tra chỉ được thực hiện trong giai đoạn test |
| Ai là người phát triển | mỗi kỹ sư sẽ thực hiện công việc phát triển của mình mà không cần phải có một lĩnh vực chuyên môn cụ thể | bắt buộc phải có một người phụ trách từng quy trình |
| Phát triển nguồn lực | bạn sẽ học hỏi thực tế thông qua các phát triển lặp đi lặp lại | thiên về việc sử dụng người có chuyên môn, kinh nghiệm nên việc đào tạo/tuyển dụng nhân lực tương đối dễ dàng |
| Phương pháp cải thiện quy trình | có thể cải thiện quy trình cho mỗi lần lặp; có thể cải thiện ở những lần tiếp theo . | chỉ có thể đúc rút, áp dụng kinh nghiệm cho các dự án tiếp theo |

**III) Phân biệt kiến trúc phần mềm Microservices và Monolithic**

1. **Kiến trúc phần mềm Monolithic**

**1.1)Khái quát**

Monolithic – nguyên khối – đây là ứng dụng mà mọi dịch vụ đều nằm trên một hệ thống và dùng chung 1 database. Người dùng cứ thẳng đuột mà thao tác với giao diện, giao diện gọi đến chức năng, chức năng lại tìm đến dữ liệu trong database. Đó chính là kiến trúc Monolithic – Nguyên khối.

khi ứng dụng được lập trình theo kiến trúc một khối, kết quả thường đạt được như sau:

-Nó có thể hỗ trợ nhiều loại client như browser hay các app native trên cả desktop và mobile.

-Nó có thể expose API cho bên thứ ba.

-Nó có thể tích hợp với các ứng dụng khác thông qua REST/SOAP web service hoặc các message queue.

-Nó có thể xử lý các request dạng HTTP, thực hiện logic nghiệp vụ, truy cập cơ sở dữ liệu và có thể trao đổi dữ liệu với các hệ thống khác.

-Nó có thể chạy trên các container như Tomcat, JBoss,..

-Nó có thể scale theo chiều ngang (vertical scalability) bằng cách tăng sức mạnh của phần cứng hoặc scale theo chiều dọc (horizontal scalability) bằng cách load balancers.

**1.2)Ưu điểm**

-Dễ phát triển vì các stack công nghệ thống nhất ở tất cả các layer.

-Dễ test do toàn bộ project được đóng gói trong một package nên dễ dàng chạy test integrantion và test end-to-end.

-Deploy đơn giản và nhanh chóng nếu bạn chỉ có một package để bận tâm.

-Dễ scale vì chúng ta có thể có nhiều instance cho load banlancer.

-Yêu cầu team size nhỏ cho việc maintain app.

-Team member có thể chia sẻ ít nhiều về skill.

-Tech stack đơn giản và đa số là dễ học.

-Phát triển ban đầu nhanh hơn do đó có thể đem sale hoặc marketing nhanh hơn.Yêu cầu cơ sở hạ tầng đơn giản. Thậm chí một container đơn giản cũng đủ để chạy ứng dụng.

**1.3)Nhược điểm**

-Các component được liên kết chặt chẽ với nhau dẫn đến side effect không mong muốn như khi thay đổi một component ảnh hưởng đến một component khác.

-Theo thời gian thì project trở nên phức tạp và lớn dần. Các tính năng mới sẽ mất nhiều thời gian hơn để phát triển và tái cấu trúc các tính năng hiện có sẽ nhiều khó khăn hơn.

-Toàn bộ ứng dụng cần được triển khai lại cho bất kỳ thay đổi nào.

-Không hề dễ để hiểu project do các module liên quan chặt chẽ lẫn nhau. Một issue nhỏ cũng có thể làm chết toàn bộ ứng dụng.

-Áp dụng công nghệ mới khó khăn vì toàn bộ ứng dụng phải thay đổi. Do đó nhiều ứng dụng một khối thường phụ thuộc một công nghệ cũ và lỗi thời.

-Các service quan trọng không thể scale riêng dẫn đến lãng phí tài nguyên vì toàn bộ ứng dụng phải scale theo.

-Các ứng dụng một khối lớn sẽ có thời gian khởi động lâu và tốn tài nguyên CPU cũng như bộ nhớ.

-Các team tham gia vào dự án phải phụ thuộc lẫn nhau và tất khó để mở rộng quy mô team.

**2)Kiến trúc phần mềm Microservices**

**2.1) Khái quát**

Đây là loại kiến trúc đang dần trở nên phổ biến trong những năm gần đây nhờ khả năng mudule hóa và khả năng mở rộng. Kiến trúc microservice có thể cung cấp hầu hết các tính năng của một ứng dụng một khối. Ngoài ra, nó cung cấp nhiều tính năng và linh hoạt hơn, do đó nó là sự lựa chọn ưu việt cho ứng dụng phức tạp. Không giống như kiến trúc một khối, khá khó để khái quát hóa kiến trúc microservice vì nó có thể thay đổi nhiều tùy thuộc vào trường hợp sử dụng và triển khai. Nhưng nhìn chung thì chúng cũng có một số đặc điểm như sau:

-Các component trong kiến trúc microservice liên kết khá lỏng lẻo (loose coupling). Các component có thể được phát triển, test, deploy và scale độc lập mà không ảnh hưởng đến các component khác.

-Các component không cần phải được phát triển cùng một stack công nghệ. Điều này có nghĩa là một component có thể chọn stack công nghệ và ngôn ngữ lập trình của riêng nó.

-Chúng thường sử dụng các tính năng nâng cao như truy tìm dịch vụ (service discovery), circuit breaking, cân bằng tải (load balance),…

-Các component chủ yếu là nhẹ và chúng làm theo chức năng riêng biệt. Ví dụ dịch vụ xác thực sẽ chỉ quan tâm xác thực người dùng vào hệ thống.

-Thường có một thiết lập giám sát (monitor) và xử lý sự cố.

**2.2) Ưu điểm**

-Các component có kết nối lỏng lẻo dẫn đến dễ cách ly, dễ test và khởi động nhanh.

-Vòng đời phát triển nhanh hơn. Tính năng mới được phát triển nhanh hơn và tính năng cũ được cấu trúc lại dễ hơn.

-Các service có thể deploy độc lập nên ứng dụng dễ đọc, dễ tạo các bản vá hơn.

-Những issue, ví dụ liên quan đến memry leak một trong các service, bị cô lập và có thể không làm sập ứng dụng.

-Việc áp dụng các công nghệ mới dễ hơn. Các component có thể được nâng cấp độc lập với nhau.

-Các mô hình scale phức tạp và hiệu quả hơn có thể được thiết lập. Các service quan trọng có thể scale hiệu quả hơn. Các component riêng sẽ khởi động nhanh hơn và cải thiện thời gian khởi động của cả hệ thống.

-Các team tham gia sẽ ít phụ thuộc lẫn nhau. Kiến trúc này rất thích hợp cho các đội Agile.

**2.3)Nhược điểm**

-Phức tạp hơn về mặt tổng thể vì các component khác nhau có các stack công nghệ khác nhau nên buộc team phải tập trung đầu tư thời gian để theo kịp công nghệ.

-Khó thực hiện test end-to-end và integration test vì có nhiều stack công nghệ khác nhau.

-Deploy toàn bộ ứng dụng phức tạp hơn vì có nhiều container và nền tảng ảo hóa liên quan.

-Ứng dụng được scale hiệu quả hơn nhưng thiết lập nâng cấp sẽ phức tạp hơn vì nó sẽ yêu cầu nâng cao nhiều tính năng như truy tìm dịch vụ (service discovery), định tuyến DNS,…

-Yêu cầu một team-size lớn để maintain ứng dụng vì có nhiều component và công nghệ khác nhau.

-Các thành viên trong team chia sẻ các skill khác nhau dựa trên component họ làm nên sẽ tạo ra sự khó khăn khi thay thế và chia sẻ kiến thức.

-Stack công nghệ phức tạp và khó để học hơn.

-Thời gian phát triển ban đầu là chậm nên thời gian để có thể làm marketing lâu hơn.

-Yêu cầu cơ sở hạ tầng phức tạp. Thông thường sẽ yêu cầu nhiều container (Docker) và nhiều máy JVM để chạy.

**3) So sánh hai kiến trúc phần mềm Microservices và Monolithic**

**3.1) Độ phức tạp**

Về độ phức tạp ta sẽ phải xem xét về 2 vấn đề:

* Độ phức tạp của quá trình phát triển
* Độ phức tạp của việc triển khai phần mềm.

Đối với sự phức tạp của việc phát triển, kích thước của codebase có thể nhanh chóng phát triển khi xây dựng phần mềm dựa trên microservice. Nhiều mã nguồn có liên quan, sử dụng các framework, ngôn ngữ khác nhau và các phiên bản thư viện ở từng dịch vụ có thể sẽ 0khác nhau. Đối với khía cạnh điều hành và giám sát, số lượng dịch vụ bị ảnh hưởng có liên quan nhiều. Kiến trúc nguyên khối chỉ giao tiếp trong phạm vi nhất định, đó có nghĩa là mỗi chức năng cụ thể sẽ luôn có một quy trình xử lý của mình. Bên cạnh đó, một chức năng trong kiến trúc microservice có thể ảnh hưởng đến các dịch vụ khác nhau.

Với việc triển khai hệ thống microservices có thể dùng nhiều công cụ khác nhau, lấy ví dụ như Kubernetes là một công cụ phổ biến thì việc triển khai cũng khá phức tạp, mặc dù Kubernetes cho phép các khả năng như tự động mở rộng quy mô, nhưng nó không phải là một hệ thống dễ quản lý. Bên cạnh đó, để triển khai một hệ thống sử dụng kiến trúc nguyên khối, chỉ cần một thao tác sao chép đơn giản là đủ. Để bắt đầu hoặc dừng hệ thống nguyên khối thường chỉ cần một lệnh đơn giản là đủ. Các dữ liệu giữa các dịch vụ cũng làm tăng thêm sự phức tạp khi chạy ở kiến trúc microservice, so với một khối nguyên khối. Ngoài ra việc đồng bộ hóa cũng là 1 thử thách ở kiến trúc microservice. Ví dụ: việc xử lý các requests không tốt có thể gây ra lỗi thanh toán hai lần. Các kiến trúc microservice có thể quản lý điều này bằng cách sử dụng các kỹ thuật như bộ điều phối trung tâm. Tuy nhiên, trong một kiến trúc nguyên khối, các giao dịch rất dễ xử lý hoặc thậm chí minh bạch đối với nhà phát triển.

**3.2) Độ tin cậy**

Khi thiết kế hệ thống ta phải giả định rẳng hệ thống sẽ gặp sự cố tại một thời điểm nào đó. Với hệ thống sử dụng kiến trúc Microservcies, một dịch vụ trong hệ thống bị lỗi, người quản lý chỉ cần khoanh vùng và đưa ra giải pháp để xử lý dịch vụ đó khiến cho hệ thống sẽ vẫn hoạt động bình thường với các chức năng đang hoạt động. Kiến trúc Microservices đã cung cấp một số các giải pháp để giải quyết vấn đề này như Chaos Monkey (Netflix).

Bên cạnh đó với kiến trúc nguyên khối việc xảy ra sự cố sẽ gây tê liệt toàn bộ hệ thống dẫn đén thiệt hại khá lớn

**3.3) Tài nguyên sử dụng**

Với một hệ thống nhỏ việc gọi đến một request thì kiến trúc Microservice sẽ sử dụng nhiều tài nguyên hơn (trong trường hợp thuật toán giống nhau). Một số chi phí phát sinh có thể do docker, máy ảo, việc ghi nhật ký, giám sát tiến trình …

Tuy nhiên, microservices cho phép chúng ta sử dụng tài nguyên thông minh hơn rất nhiều. Vì người quản lý có thể phân bổ tài nguyên khi cần thiết nên mức sử dụng tài nguyên thực tế có thể thấp hơn nhiều.

Do việc so sánh 2 kiến trúc thường sẽ thực hiện trên các hệ thống có quy mô lớn nên đây có thể coi là điểm tốt hơn của microservices so với monolithic.

**3.4) Khả năng mở rộng**

Nhờ tính linh hoạt khi kết hợp các dịch vụ trong hệ thống nên việc mở rộng đối với kiến trúc Microservices sẽ dễ dàng hơn. Với kiến trúc Monolithic việc thêm các tính năng sẽ càng làm cho codebase của dự án lớn lên, các công nghệ có thể xung đột khi thêm một chức năng mới. Ngoài ra chi phí cũng là một điểm cần lưu ý do tính linh hoạt khi tính phí theo từng khoảng thời của các dịch vụ riêng lẻ trong hệ thống sử dụng kiến trúc

Microservices, điều này sẽ giúp tiết kiệm hơn so với việc chạy liên tục của hệ thống sử dụng kiến trúc nguyên khối.

Để mở rộng quy mô một cách chính xác và sử dụng tài nguyên tốt hơn thì kiến trúc Microservices sẽ mang lại nhiều lợi ích hơn

**3.5) Hiệu suất truyền tải**

Trong các khối lượng công việc không thể chạy đồng thời trên toàn mạng, kiến trúc nguyên khối có thể mang lại hiệu suất tốt hơn. Dữ liệu cần được gửi giữa các dịch vụ và cũng như tất cả cơ sở hạ tầng gây ra một chi phí nhất định. Nếu khối lượng công việc không thể được chia tỷ lệ cho nhiều trường hợp, một hệ thống sử dụng kiến trúc nguyên khối có thể mang lại hiệu suất truyền tải cao hơn.

Kết luận:

|  |  |
| --- | --- |
| Kiến trúc Microservices | Kiến trúc Monolithic |
| Dễ dàng phát triển mở rộng | Ít phức tạp hơn |
| Tài nguyên sử dụng sẽ ít hơn | Hiệu suất truyền tải dữ liệu tốt hơn |
| Đảm bảo an toàn khi gặp sự cố |  |