**Chapter 1.Writing a Program - Viết một chương trình**

1. Cho biết yêu cầu phần mềm, yêu cầu chức năng, yêu cầu phi chức năng là gì? Cho ví dụ một yêu cầu chức năng và một yêu cầu phi chức năng.

2. Trình bày ba loại yêu cầu phi chức năng tiêu biểu.

Bài làm:

1. Yêu cầu phần mềm là: những gì cấu thành nên chương trình, bao gồm những gì chương trình cần phải làm và chất lượng các yêu cầu chức năng cần đạt được.

Các chức năng tối thiểu mà hệ thống cần có nhằm đáp ứng các yêu cầu của các đối tượng tham gia vào việc quản lý, vận hành, duy trì, thực hiện nghiệp vụ bên trong, sử dụng dịch vụ cung cấp ra bên ngoài của hệ thống

Yêu cầu chức năng là: những gì chương trình cần phải làm

VD: Chương trình nhập điểm sinh viên có yêu cầu nhập và lưu trữ điểm đúng với đối tượng được nhập.

Yêu cầu phi chức năng là: chất lượng chương trình, cách thức mà các yêu cầu chức năng cần phải đạt được để đảm bảo hiệu xuất, khả năng thực hiện, khả năng bảo trì.

Những ràng buộc và điều kiện đối với các yêu cầu chức năng của hệ thống.

VD: Chỉ cho phép nhập điểm trong phạm vi 0.00-10.00, nhập sai có thể nhập lại, có thể sửa điểm

2. Ba loại yêu cầu phi chức năng tiêu biểu:

Hiệu xuất (performance) : Yêu cầu về thời gian; Tài nguyên sử dụng; Công suất tối đa;

Khả năng sử dụng (usability) : là mức độ sử dụng được và làm hài lòng người sử dụng như: Phù hợp với nhu cầu; Dễ dàng học cách sử dụng; Giao diện người sử dụng; Khả năng truy cập, khai thác;

Khả năng bảo trì (maintainability) : là Phân tích được; Hiệu chỉnh được và Khả chuyển là Mức độ hiệu suất và hiệu quả của việc dịch chuyển một hệ thống từ một nền tảng phần cứng, phần mềm, hệ điều hành từ môi trường sử dụng này sang môi trường sử dụng khác; hỗ trợ khả năng dịch chuyển từ nền tảng chạy trên Unix sang nền tảng chạy trên Windows Server một cách dễ dàng, không phát sinh thêm chi phí hoặc kinh phí không đáng kể; Thích ứng: là hỗ trợ và sử dụng các trình duyệt thông dụng hiện nay như Micrsoft Internet Explorer, Google Crome, Mozila Firefox…; Cài đặt được; Vận hành; khai thác; Khả năng thay thế được là mức độ một sản phẩm phần mềm của hệ thống có thể được thay thế bởi một sản phẩm phần mềm khác có cùng mục đích sử dụng trong cùng một môi trường.

(1) Hiệu năng hoạt động: Yêu cầu về thời gian; Tài nguyên sử dụng; Công suất tối đa;

(2) Tương thích: Cùng tồn tại; Tương tác liên thông;

(3) Tính khả dụng: là mức độ sử dụng được và làm hài lòng người sử dụng như: Phù hợp với nhu cầu; Dễ dàng học cách sử dụng; Giao diện người sử dụng; Khả năng truy cập, khai thác;

(4) Tính tin cậy: Trưởng thành; Sẵn sàng; Khả năng chịu lỗi; Khả năng phục hồi; Thời gian giữa các lần xảy ra sự cố gián đoạn hoạt động của hệ thống;

(5) An toàn thông tin: Bảo mật; Toàn vẹn; Xác thực.

(6) Duy trì được là Phân tích được; Hiệu chỉnh được và Khả chuyển là Mức độ hiệu suất và hiệu quả của việc dịch chuyển một hệ thống từ một nền tảng phần cứng, phần mềm, hệ điều hành từ môi trường sử dụng này sang môi trường sử dụng khác; hỗ trợ khả năng dịch chuyển từ nền tảng chạy trên Unix sang nền tảng chạy trên Windows Server một cách dễ dàng, không phát sinh thêm chi phí hoặc kinh phí không đáng kể; Thích ứng: là hỗ trợ và sử dụng các trình duyệt thông dụng hiện nay như Micrsoft Internet Explorer, Google Crome, Mozila Firefox…; Cài đặt được; Vận hành; khai thác; Khả năng thay thế được là mức độ một sản phẩm phần mềm của hệ thống có thể được thay thế bởi một sản phẩm phần mềm khác có cùng mục đích sử dụng trong cùng một môi trường.

(7) Các yêu cầu phi chức năng khác: Yêu cầu phi chức năng cho công cụ tiềm kiếm, thống kê, xây dựng báo cáo; Lưu trữ: là tính đáp ứng của hệ thống đối với danh mục tiêu chuẩn, quy chuẩn kỹ thuật về ứng dụng công nghệ thông tin trong cơ quan nhà nước; Nền tảng công nghệ nền, phát triển ứng dụng là yêu cầu, định hướng lựa chọn công nghệ nền tảng dùng để xây dựng, phát triển hệ thống trong các dịch vụ công trực tuyến cấp huyện được triển khai độc lập, ưu tiên lựa chọn hệ quản trị cơ sở dữ liệu miễn phí đối với các phần mềm ứng dụng nội bộ đang có trên thị trường, ưu tiên lựa chọn các phần mềm miễn phí mã nguồn mở.

**Chapter 2:**

1. Độ phức tạp của phần mềm theo chiều rộng và chiều sâu đc xác định thông qua yếu tố nào

Các yếu tố xác định độ phức tạp của phần mềm:

+ Theo chiều rộng:

- Các hàm chính (Major Functions)

- Đặc tính bên trong của từng vùng chức năng (Features within each functional area)

- Giao diện của các hệ thống bên ngoài khác (Interfaces of the external system)

- Đa nhiệm (Simualtaneous users \_ Nhiều người dùng đồng thời)

- Kiểu dữ liệu và cấu trúc dữ liệu

+ Theo chiều dài:

- Liên kết và mối quan hệ giữa các phần (Linkage and relationship among items)

-- Liên kết: chia sẻ dữ liệu; điều khiển truyền tin

-- Mối quan hệ: thứ bậc (hierarchical), tuần tự(sequential), vòng lặp(loop), đệ quy(recursive) hoặc các dạng khác.

2. Cho biết các vấn đề kỹ thuật và phi kỹ thuật trong phát triển các hệ thống lớn

**“Technical” Issues:**

Problem and Solution Simplification chia thành các component nhỏ có liên quan tới nhau

Decomposition: phân tích

Modularization: hoàn thành các yêu cầu 1 cách tuần tự

Separation:

Incremental iterations:

Technology and tools choices

Development platform

Development language

Database

Network

Configuration management

Process and Methodology

Choice of process

Choice of methodologies

Choice tools to support the process

**Non-Technical issues:**

*Project Effort Estimation and Scheduling*

Needs to consider and estimate more items

Needs to coordinate more items in terms of pre-requisites and co-requisites

Needs to consider more potentials of risks and variations

*Assignments and Communications: giao việc và giao tiếp*

More people with an increased variety of skills

More communications among the people

More errors and modifications

*Difficult in Assignment:*

*không thể hiểu hết hệ thống, vấn đề khó dễ của dự án*

*không nắm rõ sở trường từng người*

Communication: giao tiếp cần thiết trong hệ thống lớn, cần trao đổi dữ liệu, cách sử dụng…

3. Cho biết các yếu tố cần xem xét để xác định số lượng người cần cho các công việc sau khi chuyển giao phần mềm cho khách hàng

Supporting a System

**Pre-release education and preparation**

Number of expected users

Number of known problems and expected quality

Amount of user and support personnel for training

number of fix and maintenance cycles

**Post-release user and customer support**

Call center and problem resolutions

Major problem fixes and code changes

Functional modifications and enhancements

**Chương 3:**

**Câu 7** Cho biết tám nguyên tắc về đạo đức nghề nghiệp trong công nghệ phần mềm của IEEE-CS và ACM.

8 principles for ethics and professional practices in software engineering

1. Software engineers shall act consistently with the public interest: do not do anything that only bring the benefits to the owner of some groups but it harms to most of people, to the community. It shall make better life for community.
2. Software engineers shall act in a manner that is in the best interest of their client and employer, consistent with the public interest: as the main purpose, it must provide what it’s ordered based on the public interest with best experience for the user and should be easy for using and graduating.
3. Software engineers shall ensure that their products and related modifications meet the highest professional standards possible: every industries have it own standards, they are basis for a good, safety,
4. Software engineers shall maintain integrity and independence in their professional judgment
5. Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance
6. Software engineers shall advance the integrity and reputation of the profession consistent with the public interest
7. Software engineers shall be fair to and supportive of their colleagues
8. Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

**Câu 8** Cho biết 15 nguyên tắc xây dựng phần mềm của Davis.

1. Make quality number 1
2. High quality software is possible
3. Give products to customers early
4. Determine the problem before writing the requirements
5. Evaluate design alternatives
6. Use an appropriate process model
7. Use different languages for different phases
8. Minimize intellectual distances
9. Put techniques before tools
10. Get it right before you make it faster
11. Inspect code
12. Good management is more important than good technology
13. People are the key to success
14. Follow with care
15. Take responsibility

1. Make quality Number 1: It is shortsightedness to deliver a product that is of poor quality. Quality has many definitions and means different things to different constituents. For example, to the developer, it may mean “elegant design,” and to the customers it may mean ”good response time." In any case, quality must be viewed as the first requirement.

2. High-quality software is possible: Large software systems can be built with good quality, although it may carry a high price.

3. Give products to customers early: It is very difficult to completely understand and capture the users' needs during the requirements phase; thus it is more effective to give the users a prototype of the product and let them play with it. Gather the feedback and then go into full-scale development of the product.

4. Determine the problem before writing the requirements: Before the software engineers rush to offer the solution, ensure that the problem is well understood. Then explore the potential solution and various alternatives.

5. Evaluate design alternatives: After the requirements are understood and agreed upon, explore a variety of design architecture and related algorithms. Ensure that the selected design and algorithms are the best match to satisfy the goals of the requirement.

6. Use an appropriate process model: Since there is no universal process model that applies to all projects, each project must select a process that best fits the project based on parameters such as corporate culture, project circumstances, user expectations and requirements volatility, and resource experiences.

7. Use differentlanguages for differentphases: No one single language is optimal for all phases of software development. Therefore select the best methodology and language for the different phases of software development. The d ifflculty in transition from one phase to another is not necessarily solved by using a single language across all phases.

8. Minimize intellectual distance: It is easier to create the solution if the distance between the real-world problem and the computerized solution to that problem is minimized. That is, the software solution structure needs to be as close as possible to the real-world problem structure.

9. Put technique before tools: Before using the tool, the technique needs to be well understood. Otherwise, the tool just rushes us into performing the wrong thing faster.

10. Get it right before you make it faster: It is essential to make the software execute correctly first and then work on improving it. Do not worry about optimization for either execution speed or code during initial coding.

11. Inspectcode: Inspection as first proposed by IBM’s Mike Fagan is a much better way to find errors than testing. Some early data in inspections showed a reduction of 50% to 90% of the time-to-test.

12. Good management is more important than good technology: A good manager can produce extraordinary results even with limited resources. Even the best technology cannot compensate for terrible management because it cannot motivate the people as good mangers can.

13. People are the key to success: Software is a labor-intensive profession. and people with experience, talent, and appropriate drive are the key. The right people can overcome many of the shortcomings in process, methodology, or tools. There is no substitute for quality people.

14. Follow with care: Be careful in adopting tools, process, methodology, and so on. Do not follow just because someone else is doing it or using it. Run some experiments before making a major commitment.

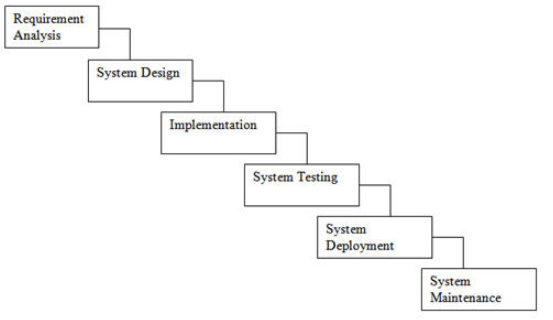
15. Take responsibility. If you developed the system, then you should take responsibility to do it right. Blaming the failure or the problem on others, on the schedule, or on the process is irresponsible.

**Câu 9** Vẽ biểu đồ minh họa và giải thích các hoạt động cho mô hình tiến trình phát triển phần mềm thác nước (waterfall model).

Chapter 4:

**Câu 9** Vẽ biểu đồ minh họa và giải thích các hoạt động cho mô hình tiến trình phát triển phần mềm thác nước (waterfall model).

Biểu đồ:



Giải thích:

Mô hình thác nước là một ví dụ của mô hình Sequential (Tuần tự). Trong mô hình này, hoạt động phát triển phần mềm được chia thành các giai đoạn khác nhau và từng giai đoạn bao gồm hàng loạt các nhiệm vụ và có các mục tiêu khác nhau.

Mô hình Thác nước là giai đoạn đầu trong quá trình SDLC. Trên thực tế, nó là mô hình đầu tiên được sử dụng rộng rãi trong ngành công nghiệp phần mềm. Nó được chia thành các pha, đầu ra của một pha trở thành đầu vào của pha tiếp theo. Nó là giai đoạn bắt buộc được hoàn thành trước khi bắt đầu giai đoạn tiếp theo. Nói tóm lại, không có sự chồng chéo nào trong mô hình thác nước.

Trong thác nước, sự phát triển của một pha chỉ bắt đầu khi giai đoạn trước hoàn thành. Do tính chất này, mỗi giai đoạn của mô hình thác nước phải được xác định khá chính xác. Các giai đoạn chuyển từ mức cao xuống mức thấp hơn, giống như một thác nước nên mô hình này được đặt tên là mô hình thác nước.

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Giai đoạn | Các hoạt động đã thực hiện | Các tài liệu có thể chuyển |
| 1 | Phân tích yêu cầu | 1. Nắm được tất cả các yêu cầu.  2. Thảo luận và hướng dẫn để hiểu các yêu cầu.  3. Thực hiện thử nghiệm các yêu cầu để đảm bảo rằng các yêu cầu có thể kiểm chứng được hay không. | RUD (Yêu cầu hiểu tài liệu) |
| 2 | Thiết kế hệ thống | 1. Theo yêu cầu, tạo ra thiết kế  2. Thảo luận về yêu cầu phần cứng /phần mềm.  3. Tài liệu thiết kế | HLD (Tài liệu thiết kế mức độ cao), LLD (Tài liệu thiết kế mức độ thấp) |
| 3 | Thực hiện | 1. Theo thiết kế tạo ra các chương trình / code  2. Tích hợp code cho giai đoạn tiếp theo.  3. Unit testing | Các chương trình, Unit TCs và kết quả |
| 4 | Thử nghiệm hệ thống | 1. Tích hợp unit tested code và kiểm tra nó để đảm bảo nó hoạt động như mong đợi.  2. Thực hiện tất cả các hoạt động thử nghiệm (Functional and non functional) để đảm bảo hệ thống đáp ứng các yêu cầu.  3. Trong trường hợp bất thường, báo cáo.  4. Theo dõi tiến độ về kiểm tra thông qua các công cụ như số liệu truy xuất nguồn gốc, ALM  5. Báo cáo hoạt động thử nghiệm | Test case, Test report, Report lỗi, Updates |
| 5 | Triển khai Hệ thống | 1. Hãy chắc chắn rằng môi trường đang hoạt động 2. Hãy chắc chắn rằng không có lỗi mở server. 3. Đảm bảo rằng các tiêu chí test được đáp ứng. 4. Triển khai ứng dụng trong môi trường tương ứng. 5. Thực hiện kiểm tra về môi trường sau khi ứng dụng được triển khai để đảm bảo ứng dụng không gặp vấn đề | Hướng dẫn sử dụng Định nghĩa / đặc tả môi trường |
| 6 | Bảo trì hệ thống | 1. Hãy chắc chắn rằng ứng dụng đang chạy và chạy trong môi trường tương ứng. 2. Trong trường hợp người dùng gặp lỗi, chắc chắn giải quyết và khắc phục các vấn đề. 3. Trong trường hợp bất kỳ vấn đề fixed; code cập nhật được triển khai trong môi trường. 4. Ứng dụng luôn được tăng cường để kết hợp nhiều tính năng, cập nhật môi trường với các tính năng mới nhất | Hướng dẫn sử dụng, Danh sách ticket, Danh sách các tính năng mới được triển khai. |

2. Mô hình Thác nước SDLC được sử dụng khi:

Yêu cầu ổn định và không thay đổi thường xuyên.

Một ứng dụng nhỏ.

Không có yêu cầu mà không hiểu hoặc không rõ ràng.

Môi trường ổn định

Các công cụ và công nghệ được sử dụng là ổn định

Nguồn lực được đào tạo và sẵn sàng.

3.1. Ưu điểm của việc sử dụng mô hình thác nước như sau:

Đơn giản, dễ hiểu và sử dụng.

Đối với các dự án nhỏ hơn, mô hình thác nước hoạt động tốt và mang lại kết quả phù hợp.

Vì các giai đoạn của mô hình thác nước cứng nhắc và chính xác, một pha được thực hiện một lần, nó rất dễ dàng để maintain.

Các tiêu chí đầu vào và đầu ra được xác định rõ ràng, do đó nó dễ dàng và có hệ thống để tiến hành chất lượng.

Kết quả được ghi chép tốt.

3.2. Nhược điểm của việc sử dụng mô hình thác nước:

Không thể chấp nhận thay đổi yêu cầu

Nó trở nên rất khó khăn để di chuyển trở lại giai đoạn. Ví dụ, nếu ứng dụng đã chuyển sang giai đoạn thử nghiệm và có thay đổi về yêu cầu, gặp khó khăn để quay lại và thay đổi nó.

Việc giao hàng của sản phẩm cuối cùng là muộn vì không có mẫu thử nghiệm được chứng minh trung gian.

Đối với các dự án lớn và phức tạp, mô hình này không tốt vì yếu tố rủi ro cao hơn.

Không thích hợp cho các dự án mà yêu cầu được thay đổi thường xuyên.

Không làm việc cho các dự án dài và đang diễn ra.

Kể từ khi thử nghiệm được thực hiện ở giai đoạn sau, nó không cho phép xác định những thách thức và rủi ro trong giai đoạn trước đó nên chiến lược giảm thiểu rủi ro rất khó để chuẩn bị.

**Câu 10** Vẽ biểu đồ minh họa và giải thích mô hình tiến trình phát triển phần mềm xoắn ốc (spiral model)

**Câu 11** Cho biết các vấn đề tồn tại của các tiến trình phát triển phần mềm truyền thống.

**Câu 12** Cho biết các đặc trưng của các phương pháp/quy trình phát triển phần mềm linh hoạt/nhanh (Agile processes).

1. Khái niệm:

Phương thức phát triển phần mềm Agile là một tập hợp các phương thức phát triển lặp và tăng dần trong đó các yêu cầu và giải pháp được phát triển thông qua sự liên kết cộng tác giữa các nhóm tự quản và liên chức năng. Agile là cách thức làm phần mềm linh hoạt để làm sao đưa sản phẩm đến tay người dùng càng nhanh càng tốt càng sớm càng tốt và được xem như là sự cải tiến so với những mô hình cũ như mô hình “Thác nước (waterfall)” hay “CMMI”.

2. Nguyên tắc áp dụng:

1.Kiểm thử giúp dự án nhanh chóng được bàn giao

Ở dự án truyền thống, kiểm thử thường được xem là bước cuối kiểm tra chất lượng sản phẩm. Và việc ngăn chặn lỗi của phầm mềm bị coi như trách nhiệm của QA/tester. Bug được tìm thấy dù quan trọng hay không thì cũng sẽ làm chậm quá trình bàn giao sản phẩm.

Trong dự án Agile, chúng ta xây dựng một sản phẩm tốt ngay từ ban đầu, sử dụng kiểm thử để phản hồi trên ngay khi phát triển để làm thế nào cho sản phẩm tương đồng với yêu cầu.

Nghe có vẻ là thay đổi nhỏ, nhưng thực chất thì việc này có ý nghĩa lớn. Mối liên hệ giữa tester và dev cần là sự cộng tác, tương hỗ lẫn nhau

2. Kiểm thử không chỉ là một giai đoạn của dự án

Kiểm thử không phải là một giai đoạn trong quá trình phát triển Agile mà cần được tham gia sâu vào quy trình phát triển từ sớm.

Cách tiếp cận Agile tập trung vào việc xác nhận những điều đúng đắn ngay từđầu, giảm sự cần thiết phải có nhiều kiểm thử viên (QA Tester) ở cuối quy trình để đạt được kết quả. Đảm bảo tiến độ dự án được liên tục.

3. Cá nhân và sự tương hỗ quan trọng hơn quy trình

Với dự án truyền thống, tester làm việc độc lập và chịu trách nhiệm với toàn bộ hoạt động test.

Đối với Agile, các hoạt động test được thực hiện bởi toàn bộ dự án. Để thực hiện được hết test thì cần thực hiện lặp lại qua các sprint.

Tuy nhiên tới khi dự án lớn hoặc phức tạp lên thì sẽ có lúc không thể test hết các testcase đề ra và không thể thực hiện được mục tiêu ban đầu đề ra. Có nghĩa team không thể thực hiện nhanh như họ nghĩ. Vì nếu test chưa xong thì feature cũng không thể xong được, vì vậy để đẩy nhanh tốc độ thì cả team phải làm cùng nhau và đẩy nhanh phần chậm nhất, test cùng nhau.

4. Rút ngắn vòng lặp phản hồi

Thời gian từ khi viết code và thực hiện code tới khi biết được code vận hành như thế nào được gọi là feedback loop (vòng phản hồi).

Nếu một phần mềm không được thực hiện test cho tới khi kết thúc và được bàn giao thì vòng phản hồi này bị kéo dài tới cả tháng, như thế là quá dài.

Agile tạo nên một vòng phản hồi ngắn hơn bởi với dự án Agile, phần mềm được sẵn sàng để test ngay từ khi bắt đầu. Đặc thù của Agile là đội dự án có rất nhiều cấp độ kiểm thử để có thể tấn công được nhiều loại dữ liệu khác nhau.

Agile sử dụng nhiều test tự động vì nó trả lại phản hồi nhanh. Test hồi quy thủ công mất nhiều thời gian thực hiện hơn, cần có nhân lực và có thể không thực hiện ngay lập tức được. Kiểm tra thủ công vẫn còn quan trọng.

Tuy nhiên, đội Agile thường thấy rằng những thông tin phản hồi nhanh chóng tạo nên bởi hồi quy tự động là chìa khóa để phát hiện các vấn đề một cách nhanh chóng, do đó làm giảm rủi ro và giảm việc phải làm lại.

5. Thỏa mãn mong muốn của khách hàng

Cho dù là test tự động hay test thủ công thì kịch bản test cần phải khớp với yêu cầu và mong đợi của khách hàng. Trước khi tốn thời gian tìm bug nên đặt câu hỏi để làm sáng tỏ mong muốn của khách hàng đối với chức năng sản phẩm

6.Giữ code rõ ràng

Nguyên tắc này là một ví dụ về một nguyên tắc mà đội Agile phải có. Sẽ mất nhiều công sức và thời gian để sửa lỗi khi chúng được tìm thấy.

Nếu đó là một lỗi chính đáng nó sẽ được sửa trong vòng lặp và đôi khi kết quả sau khi sửa sẽ không được tốt bằng làm từ đầu vì nó ảnh hưởng tới những phần khác.

7.Giản lược tài liệu kiểm thử

Thay vì viết dài dòng thì Agile test

Tái sử dụng checklist

Tập trung vào bản chất của các thử nghiệm chứ không phải là các chi tiết ngẫu nhiên

Sử dụng các tài liệu hướng dẫn đơn giản

Nắm bắt những ý tưởng thử nghiệm trong điều lệ kiểm nghiệm thăm dò

8. Chưa thể hoàn thành khi chưa qua giai đoạn kiểm thử

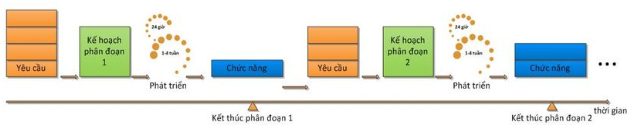
Trong dự án truyền thống có sự phân tách rõ ràng giữa dev và test, đó là đặc trưng cho việc dev nói “xong” với phần họ phát triển nhưng nó vẫn chưa được test. Do đó thực tế là phần phát triển ấy vẫn chưa xong cho tới khi test xong và bug được fix.

Đó là lý do vì sao mà phần mềm chỉ được để “90% done”. Agile không tính là “done” mà nó cần được sẵn sàng cho sự chấp nhận của Product Owner và khách hàng cho tới khi nó được thực thi và test

9. Test-Last và Test-Driven

Trong môi trường phát triển truyền thống, test được lấy từ tài liệu yêu cầu. Yêu cầu và design đầu tiên, sau đó đến kiểm thử. Quá trình kiểm thử diễn ra ở cuối dự án. Tuy nhiên kiểm thử cung cấp một ví dụ về ý nghĩa của việc phát triển thỏa mãn yêu cầu. Test được định hướng từ các thành phần của project, trong đó có tài liệu dự án. Việc thực hiện test được tiến hành vào thời điểm cuối cùng của project. Đây gọi là cách tiếp cận "testlast" - Test sau cùng

II. Các giai đoạn kiểm thử phần mềm tương ứng với các giai đoạn phát triển phần mềm trong mô hình Agile



1. Tiền-Phân-đoạn (Pre-iteration):####

Yêu cầu được phân tích chi tiết bởi BA (Business Analyst – chuyên viên phân tích nghiệp vụ) và các tiêu chí chấp nhận (acceptance criteria). Và QA sử dụng các yêu cầu này ngay từ đầu, ta cần phải xác minh (verify) những yêu cầu đó từ sớm và thường xuyên.

2. Xác minh Yêu cầu:

+ Kiểm thử Agile thiên về việc đưa ra phản hồi sớm, và phải bắt đầu bằng việc kiểm tra các yêu cầu từ sớm bởi QA hoặc tester để làm sáng rõ ý nghĩa và tính khả-kiểm (testability). Việc này sẽ đảm bảo các yêu cầu luôn rõ ràng và có thể kiểm thử được.

+ Yêu cầu cần đủ nhỏ để có ý nghĩa trong bối cảnh xác định

+ Tiêu chí chấp nhận (acceptance criteria những story thường được sử dụng cho các tiêu chí chấp nhận) không nên bị trùng lặp, chồng chéo từ những story khác nhau

+ Các giai đoạn: Agile-Testing-Flow2.png

- user story: là một tóm tắt đơn giản, ngắn gọn về chức năng mà khách hàng mong muốn.

- Tiêu chí chấp nhận (Acceptance Criteria): là những tiêu chí dùng để đánh giá sản phẩm, chức năng đã thực hiện đúng yêu cầu hay chưa? Có thể coi đó là các tiêu chí xác nhận hoàn thành story.

+ Các tiêu chí đặt ra phải đáp ứng các đặc tính sau:

- Tính khả dụng (usability): là tiêu chí trả lời cho câu hỏi: Có dễ sử dụng hay không?

- Tính chức năng (Functionality)

- Xử lý lỗi (error handing) : Liệt kê ra những lỗi có thể gặp phải trong quá trình sử dụng chương trình và phương thức để xử lý. Ví dụ người dùng có thể thực hiện sai thứ tự các bước và khi đó chương trình sẽ xử lý như thế nào?

- Hiệu suất( Performance)

- Stress tests: Là tiêu chí trả lời cho câu hỏi: Hệ thống sẽ hoạt động như thế nào dưới những áp lực như có nhiều người truy cập tại cùng 1 thời điểm, có quá nhiều request được gửi đến hệ thống…

Để đạt được mục tiêu của giai đoạn này cần có sự giao tiếp chặt chẽ giữa các bên Đội Phát triển / Nhà phân tích nghiệp vụ/ Đảm bảo chất lượng.

- Khả kiểm (Testable): Các khía cạnh có thể kiểm thử được phải được xem xét chi tiết. Những yếu tố này thường là:

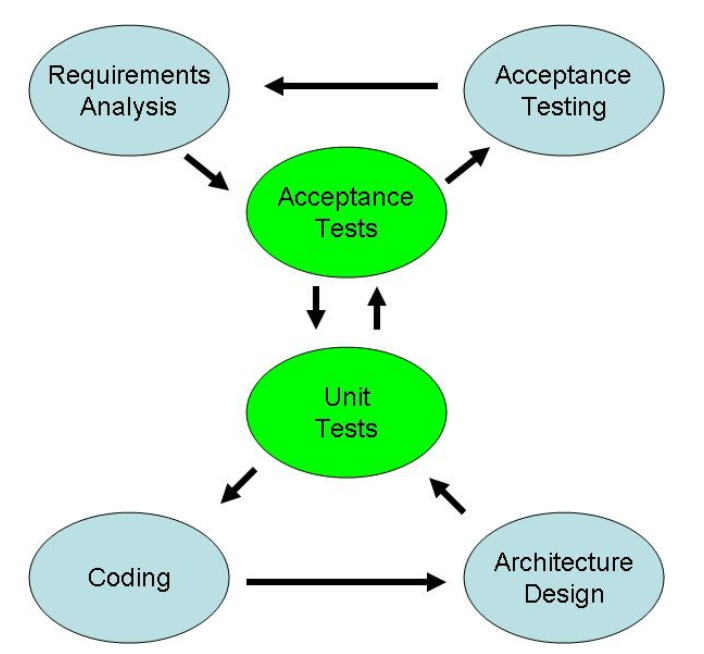
-- Tìm kiếm các yêu cầu ẩn

-- Môi trường

-- Dữ liệu kiểm thử (test data)

-- Sự phụ thuộc vào các yêu cầu khác

3. Các hoạt động Đảm bảo chất lượng:####



Kiểm thử chấp nhận là các yêu cầu về phương diện kiểm thử cần được thực hiện để hiểu các yêu cầu phần mềm. Các kiểm thử chấp nhận này được sinh ra tự động và dùng để hướng dẫn quá trình phát triển.

Các kiểm thử chấp nhận không nên bao gồm toàn bộ các tình huống (case scenarios) do điều này có thể tạo ra những sự ngưng trệ không cần thiết và có thể tạo ra quá nhiều bộ kiểm thử tự động (automated test) tương tự nhau.

Kiểm thử chấp nhận trong các dự án Agile rất khác biệt so với các dự án truyền thống. Không giống như các dự án truyền thống, nơi kiểm thử chấp nhận xảy ra ở phần cuối của vòng đời phần mềm, trong dự án Agile kiểm thử chấp nhận được thực hiện trước khi phần mềm được chuyển giao. Kiểm thử chấp nhận cũng có xu hướng được tự động hóa để họ có thể chạy như là kiểm thử hồi quy (regression test).

Kiểm thử tự động rất quan trọng đối với mọi dự án Agile. Các bản build thường xuyên yêu cầu các chu kỳ phản hồi ngắn, do đó kiểm thử hồi quy phải nhanh chóng và chính xác.

Trong các dự án Agile, kiểm thử tự động được thực hiện bởi tất cả các cấp độ – lập trình viên, kiểm thử viên bảo đảm chất lượng(QA tester) và các nhà phân tích nghiệp vụ (BA). Sự tham gia của tất cả mọi người làm gia tăng tính xác đáng của các phần kiểm thử và thường giúp xác định đúng các phần kiểm thử. Tuy nhiên, điều này không có nghĩa là tất cả mọi người phải đều phải viết mã kiểm thử.

**Câu 13** Cho biết các thực hành trong lập trình cực độ.

**Câu 14** Liệt kê và mô tả các bước cơ bản trong quá trình xây dựng yêu cầu phần mềm.

**Câu 15** Cho biết ba công việc chính cần thực hiện trước khi xây dựng yêu cầu phần

mềm.

**Câu 16** Mô tả các mục cần xem xét khi xác định mức độ ưu tiên của các yêu cầu.

**Câu 17** Cho biết cấu trúc và các đặc điểm của kiểu kiến trúc đường ống và bộ lọc (Pipe and filter)

**Câu 18** Cho biết cấu trúc và các đặc điểm của kiểu kiến trúc Khách-Chủ (Client-Server).

**Câu 19** Cho biết cấu trúc và các đặc điểm của kiểu kiến trúc dữ liệu dùng chung (Shared-Data).

**Câu 20** Liêt kê và mô tả các bước trong thiết kế cơ sở dữ liệu.

**Câu 21** Mô tả ba trong số các loại biểu đồ UML trong phân tích và thiết kế hướng đối tượng.

**Câu 22** Trình bày cách tính độ đo dung lượng (Volume metric) của Halstread để phân tích độ phức tạp của chương trình nguồn.

**Câu 23** Trình bày các cấp độ ghép nối (coupling) giữa các module.

**Câu 24** Trình bày các cấp độ kết dính (cohension) giữa các phần tử trong một module.

**Câu 25** Trình bày 8 quy tắc thiết kế giao diện người dùng của Shneiderman và Plaisant.

**Câu 26** Cho biết các lỗi lập trình thường gặp.

**Câu 27** Liệt kê 10 thông lệ lập trình.

**Câu 28** Liệt kê các chuẩn lập trình Java.

**Câu 29** Vẽ và mô tả biểu đồ biểu diễn tiến trình lập trình hướng kiểm thử (test driven development).

**Câu 30** Liệt kê 10 mục cần kiểm tra khi thanh tra chương trình

**Câu 31** Cho biết các cách phân lớp của các kỹ thuật kiểm thử phần mềm.

**Câu 32** Cho biết 4 cấp độ kiểm thử phần mềm.

**Câu 33** Trình bày kỹ thuật kiểm thử phần mềm phân lớp tương đương (Equivalence Class Partitioning).

**Câu 34** Trình bày kỹ thuật kiểm thử phần mềm phân tích giá trị biên (Boundary Value Analysis).

**Câu 35** Trình bày kỹ thuật kiểm thử phần mềm phân tích đường dẫn

**Câu 36** Liệt kê 10 sản phẩm được quản lý cấu hình.

**Câu 37** Cho biết vai trò của quản lý cấu hình và các chức năng cơ bản của quản lý cấu hình.

**Câu 38** Cho biết mô hình lưu trữ và truy cập các thành phần (Parts Storage and Access Model) trong quản lý cấu hình và các tính năng chính của mô hình này.

**Câu 39** Cho biết các công việc cần thực hiện trong hỗ trợ khắc phục khiếm khuyết phần mềm và cho biết xu hướng thay đổi của số lượng khiếm khuyết theo thời gian sau khi chuyển giao phần mềm.

**Câu 40** Cho biết các công việc thực hiện trong giai đoạn trước khi kết thúc việc hỗ trợ phần mềm, hay còn gọi là giai đoạn mặt trời lặn (product sunset period).

**Câu 41** Trình bày các mục cần có trong một mẫu yêu cầu thay đổi trong giai đoạn bảo trì phần mềm.

**Câu 42** Cho biết các giai đoạn và các mục tiêu của quy trình quản lý dự án phần mềm.

**Câu 43** Cho biết các bước của giai đoạn lập kế hoạch trong quy trình quản lý dự án phần mềm.

**Câu 44** Cho biết các công việc được thực hiện trong giai đoạn tổ chức trong quy trình quản lý dự án phần mềm

**Câu 45** Trình bày chương trình khung để xây dựng cấu trúc phân rã công việc (Work breakdown Structure) trong việc ước lượng công sức hoàn thành dự án.