***ĐỀ CƯƠNG ÔN TẬP: NHẬP MÔN CÔNG NGHỆ PHẦN MỀM***

1. **QUY TRÌNH PHÁT TRIỂN PHẦN MỀM:**

**Câu 1: Quy trình phát triển phần mềm. Các hoạt động chính trong một quy trình phát triển phần mềm (Fundamental software engineering activitives).**

**Trả lời:**

**Quy trình phát triển phần mềm:**

* Quy trình phát triển phần mềm (QT PTPM) là một tập các hoạt động có liên quan với nhau, nhằm sản xuất ra sản phẩm phần mềm chất lượng cao [1].
* Quy trình phát triển phần mềm là một tập các hoạt động được thực hiện theo một trật tự nhất định nhằm tạo ra sản phẩm phần mềm chất lượng cao [2].
* Thuật ngữ: Software development process.
* Các hoạt động này có thể nhằm phát triển phần mềm từ “con số 0” (scratch) hoặc chỉnh sửa từ một hệ thống phần mềm đã có.

**Các quy trình phát triển phần mềm thường bao gồm 4 hoạt động quan trọng sau:**

1. Đặc tả yêu cầu phần mềm (Software specification, requirements engineering): Định nghĩa các chức năng chính của phần mềm và các ràng buộc xung quanh nó.
2. Phát triển phần mềm (Software development, còn gọi là design and implementation): Phần mềm được thiết kế và xây dựng.
3. Xác minh và thẩm định phần mềm (verification and validation, aka V&V): Phần mềm được kiểm tra xem có đúng theo đặc tả và có đáp ứng theo nhu cầu của người dùng không?
4. Tiến hóa phần mềm (Software evolution, hoặc software maintenance): Phần mềm được cập nhật/chỉnh sửa đáp ứng theo những thay đổi yêu cầu của khách hàng.

**Câu 2: Mô hình phát triển phần mềm thác nước:**

**Trả lời:**

* Mô hình Thác nước theo hướng tiếp cận tuần tự (sequence approach), mỗi hoạt động của quy trình được tách rời và liên tiếp nhau một cách tuyến tính.
  + Trên thực tế, các giai đoạn có sự chồng lấn lên nhau.
* Trong mô hình Thác nước, chúng ta phải lên kế hoạch rất chi tiết (đặc tả, thiết kế kiến trúc, thiết kế chi tiết...) trước khi bắt tay vào thực hiện.

**Các giai đoạn trong mô hình Thác nước:**

1. Định nghĩa yêu cầu
2. Thiết kế hệ thống
3. Hiện thực và kiểm thử đơn vị
4. Tích hợp và kiểm thử hệ thống
5. Vận hành và bảo trì

**Các giai đoạn trong mô hình Thác nước:**

* Về nguyên tắc: giai đoạn sau sẽ không được thực hiện cho đến khi giai đoạn trước đó hoàn thành và được chấp thuận các kết quả (tài liệu, bản thiết kế).
* Trên thực tế: các giai đoạn thường chồng lấn lên nhau và có thông tin phản hồi ngược về giai đoạn trước.
* Ví dụ:
  + Trong quá trình thiết kế, một số vấn đề về yêu cầu được tìm thấy.
  + Trong quá trình code, một số vấn đề thuộc giai đoạn thiết kế được phát triển.

**Mô hình Thác nước có hạn chế gì? Điểm mạnh gì?**

**Hạn chế của mô hình Thác nước:**

* Sự tương tác với khách hàng ít
  + Chủ yếu tập trung ở giai đoạn đầu, khi thu thập phân tích yêu cầu.
  + Ít có cơ hội nhận phản hồi của khách hàng.
* Khách hàng phải đợi rất lâu mới thấy được sản phẩm
  + Đến lúc thấy sản phẩm, có khi lại đổi ý, hoặc nhận ra là không đúng như cái mình mong muốn.
* Có thể phải làm lại, quay lại giai đoạn trước (rework)
  + Khi phát hiện điểm không hợp lý, phải rework dẫn đến tốn nhiều chi phí sửa đổi, thậm chí ảnh hưởng kiến trúc toàn hệ thống, đặc biệt là những phát hiện ở giai đoạn càng trễ.

**Điểm mạnh của mô hình Thác nước:**

* Đơn giản, dễ hiểu, dễ tổ chức.
* Quản lý dự án dễ dàng.
* Cấu trúc, kiến trúc toàn hệ thống chặt chẽ, rõ ràng.
* Tài liệu được ghi lại đầy đủ và chi tiết
  + Là cơ sở tốt để bảo trì, đào tạo huấn luyện người mới, chuyển giao dự án (cho nhóm khác)…

**Nên áp dụng mô hình Thác nước khi:**

* Các yêu cầu đã được hiểu rõ.
* Khả năng thay đổi yêu cầu thấp.

**Câu 3: Mô hình phát triển tiến hóa(Evolutionary development, còn gọi là mô hình phát triển tăng dần, Incremental development).**

**Trả lời:**

**Các mô hình phát triển phần mềm Tiến hoá (Tăng dần):**

* Các mô hình phát triển phần mềm Tiến hóa dựa trên ý tưởng chung:
  + Phát triển một phiên bản đầu tiên (initial version)
  + Lấy phản hồi từ người dung
  + Nâng cấp (tiến hóa) phần mềm thông qua các phiên bản cho đến khi hệ thống được chấp thuận.
* Các hoạt động trong quy trình (đặc tả yêu cầu, phát triển, thẩm định) được đan xen nhau để nhận thông tin phản hồi một cách nhanh chóng (thay vì phân tách thành các giai đoạn riêng biệt)

**Các mô hình phát triển phần mềm Tiến hóa có ưu, nhược điểm gì?**

**Điểm mạnh của các mô hình PTPM Tiến hoá (Tăng dần):**

* Khách hàng có thể kiểm tra các tính năng quan trọng của hệ thống một cách sớm nhất và nhiều lần nhất
  + Vì các phiên bản ban đầu thường bao gồm các tính năng quan trọng nhất, cần thiết có sớm nhất (most important, most urgent).
* Giúp trả lời sớm câu hỏi: liệu sản phẩm có đáp ứng đúng yêu cầu của khách hàng hay không?
  + Nếu có sự thay đổi, chỉ những phần mà nhà phát triển đang thực hiện bị ảnh hưởng (current increment) (ít ảnh hưởng đến những phần trước đó), hoặc các phần sẽ thực hiện sau này.

**Hạn chế của PTPM Tiến hoá (Tăng dần):**

* Quy trình không rõ ràng (the process is not visible)
  + Người quản lý dự án phải dựa vào các đợt bàn giao sản phẩm để đánh giá tiến độ.
  + Nếu hệ thống được phát triển một cách nhanh chóng, việc viết tài liệu cho mỗi phiên bản sẽ tốn nhiều thời gian và không hiệu quả (phải cập nhật liên tục)
* Cấu trúc hệ thống có khuynh hướng không ổn định (System structure tends to degrade)
  + Khi có nhiều phần được tích hợp, thêm vào hệ thống trong quá trình phát triển, cấu trúc hệ thống bị ảnh hưởng.
* Hai điểm hạn chế của PTPM Tiến hóa trên còn được thể hiện rõ đối với các hệ thống lớn, phức tạp, có nhiều đội cùng tham gia phát triển.
  + Những hệ thống lớn cần một kiến trúc và cấu trúc ổn định. Sự phân chia công việc vào các phần phải rõ ràng.
  + Điều đó có nghĩa cần phải lên kế hoạch một cách kỹ lưỡng trước (hơn là phát triển tăng dần)

**Lợi ích của PTPM Tiến hoá (Tăng dần) so với mô hình Thác nước:**

* Chi phí để đáp ứng những thay đổi yêu cầu giảm:
  + Chi phí cho sự phân tích lại yêu cầu, viết lại tài liệu giảm nhiều
* Dễ dàng lấy phản hồi của khách hàng:
  + Với các phần được bàn giao, nhà phát triển dễ dàng lấy phản hồi từ khách hàng (điều này rất khó nếu khách hàng chỉ dựa vào tài liệu đặc tả yêu cầu và các bản thiết kế)
* Bàn giao sản phẩm cho khách hàng sớm:
  + Thậm chí khi sp chưa hoàn thiện, khách hàng vẫn có thể sử dụng (ở một mức độ giới hạn)

**Nên áp dụng mô hình nào? Mô hình Thác nước hay Tiến hoá?**

* Với những hệ thống trung bình và nhỏ (ít hơn 500.000 dòng code), hướng tiếp cận Tiến hoá sẽ tốt hơn.
* Với những hệ thống lớn, nên kết hợp cả 2 hướng tiếp cận: Thác nước và Tiến hoá
  + Phát triển Tiến hoá để xây dựng các bản thảo (Prototype), lấy phản hồi và làm rõ các điểm yêu cầu còn mơ hồi, chưa rõ ràng.
  + Những phần (module, chức năng) đã hiểu rõ -> áp dụng Thác nước.
  + Những phần chưa hiểu rõ -> áp dụng Tiến hoá.

**Câu 4: Mô hình công nghệ phần mềm dựa trên thành phần (Component-based software engineering, còn gọi là mô hình công nghệ phần mềm hướng tái sử dụng, Reuse-oriented software engineering ).**

**Trả lời:**

* Ý tưởng: trên cơ sở sử dụng lại các thiết kế, mã nguồn đã có sẵn (tương tự với cái cần để xây dựng hệ thống), chỉnh sửa, tích hợp vào hệ thống mới.
* Giai đoạn **Đặc tả yêu cầu** và **Thẩm định hệ thống** tương tự như các quy trình khác.
* Các giai đoạn trung gian trong mô hình này thì đặc biệt hơn so với các quy trình khác.
* **Phân tích thành phần (component analysis)**
  + Với bản đặc tả yêu cầu, nhà phát triển sẽ tìm kiếm các thành phần có sẵn và phân tích xem chúng có thể đáp ứng được thế nào trong việc xây dựng hệ thống mới.
  + Thông thường, các thành phần tìm thấy sẽ không đáp ứng 100% yêu cầu.
* **Chỉnh sửa yêu cầu (Requirement modification)**
  + Ở giai đoạn này, các yêu cầu sẽ được phân tích dựa trên thông tin về các thành phần (component) đã tìm thấy.
  + Các yêu cầu có thể được chỉnh sửa lại cho phù hợp với các thành phần tìm thấy.
  + Nếu việc chỉnh sửa yêu cầu không thể thực hiện được, trở lại bước tìm kiếm các thành phần -> tìm giải pháp khác.
* **Thiết kế hệ thống với các thành phần tái sử dụng (System design with reuse)**
  + Thiết kế kiến trúc, nền tảng (framework) cho hệ thống với các thành phần tái sử dụng.
* **Phát triển và tích hợp (Development and Integration)**
  + Các thành phần tái sử dụng sẽ được tích hợp vào hệ thống mới.
  + Phát triển các thành phần mới.
* Thông thường, có 3 kiểu thành phần có thể tái sử dụng (theo tổng hợp của Ian Sommerville [1])
  + Web services: cho phép triệu gọi từ xa thông qua web
    - Dịch vụ vé máy bay, dự báo thời tiết, giá vàng, kiểm lỗi chính tả…
  + Các thư viện/plugins:
    - Thư viện xuất ra file Excel, PDF…
    - Plugin hiển thị sliders (wordpress), comment facebook trên website…
  + Hệ thống phần mềm hoàn chỉnh:
    - Hệ thống CT-scan xuất ra file ảnh, dùng import vào hệ thống quản lý y khoa.

**PTPM theo hướng tái sử dụng có ưu, nhược điểm gì?**

**Điểm mạnh:**

* Giảm khối lượng công việc phát triển phần mềm.
* Giảm khối lượng công việc kiểm thử.
* Giảm chi phí và rủi ro.
* Bàn giao sản phẩm cho khách hàng nhanh hơn.

**Hạn chế:**

* Có thể không đáp ứng 100% yêu cầu của khách hàng.
* Không chủ động (trong việc nâng cấp/ bảo trì) vì các thành phần tái sử dụng thuộc quyền kiểm soát của tổ chức khác.
* Phụ thuộc về giấy phép sử dụng (license).

**Câu 5:Khung quy trình phát triển Scrum?**

**Trả lời:**

**Giai đoạn 1:** Lên kế hoạch sơ bộ, xác định các mục tiêu của dự án và thiết kế kiến trúc hệ thống.

**Giai đoạn 2:** Các vòng Sprint, mỗi Sprint sẽ tạo ra một phần tăng trưởng.

**Giai đoạn 3:** Đóng dự án, tổng hợp, hoàn tất các tài liệu dự án, rút bài học kinh nghiệm...

**Các vai trò trong Scrum:**

**Scrum Team =** Product Owner + Scrum Master + **Development Team**

**Product Owners:**

* Quản lý các yêu cần (Product backlog).
* Định nghĩa và quyết định thứ tự ưu tiên các yêu cầu.

**Scrum Master:**

* Làm cho quy trình hoạt động trôi chảy.
* Loại bỏ các rào cản ảnh hưởng đến hiệu suất làm việc.
* Tổ chức và tạo thuận lợi cho các cuộc họp.

**Development Team:**

* Tự tổ chức self-organizing
  + Không chịu sự chỉ dẫn, quản lý bởi ai (kể cả Scrum Master)
  + Tự quyết định phân chia task
* Liên chức năng cross-functional
  + Team có đầy đủ các kỹ năng cần thiết để xây dựng sản phẩm mà không phụ thuộc bên ngoài.

**Các sự kiện trong Scrum:**

* **Sprint (**một giai đoạn, 2-4 tuần)
* **Sprint planning** (cuộc họp lên kế hoạch cho Sprint)
  + Scrum master lần lượt nêu các User story
  + Các thành viên thảo luận, cho điểm (Story point)
  + Nếu điểm số chênh lệch, Scrum master cho người đánh giá điểm cao nhất và thấp nhất trình bày.
  + Cho đánh điểm lại
* **Daily Scrum** (họp hằng ngày, Stand-up meeting)
* **Sprint Review** (họp demo sản phẩm)
* **Sprint Retrospective** (họp rút kinh nghiệm)

**SO SÁNH MÔ HÌNH SCRUM VÀ MÔ HÌNH WATERFALL (THÁC NƯỚC) , SPRIAL (XOẮN ỐC)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Đặc điểm** | **Waterfall** | **Spriral** | **Scrum** |
| Xác định các giai đoạn phát triển | Bắt buộc | Bắt buộc | Chỉ có giai đoạn lập kế hoạch và kết thúc |
| Sản phẩm cuối cùng | Được xác định trong quá trình lập kế hoạch | Được xác định trong quá trình lập kế hoạch | Xác định trong suốt dự án |
| Chi phí sản phẩm | Được xác định trong quá trình lập kế hoạch | Thay đổi cục bộ | Xác định trong quá trình xây dựng dự án |
| Ngày hoàn thành sản phẩm | Được xác định trong quá trình lập kế hoạch | Thay đổi cục bộ | Xác định trong quá trình xây dựng dự án |
| Đáp ứng với môi trường sử dụng | Trong kế hoạch ban đầu | Trong kế hoạch ban đầu | Xuyên suốt từ kế hoạch đến xây dựng và kết thúc |
| Kinh nghiệm trao đổi | Đào tạo trước cho đến khi bắt tay làm dự án | Đào tạo trước cho đến khi bắt tay làm dự án | Thực hiện trong quá trình làm dự án |
| Khả năng thành công | Thấp | Trung bình thấp | Cao |

1. **YÊU CẦU PHẦN MỀM:**

**Câu 6. Yêu cầu phần mềm (Software requirements). Yêu cầu chức năng, yêu cầu phi chức năng.**

**Trả lời:**

* Yêu cầu phần mềm (Sofware requirements) là những mô tả về những cái mà hệ thống cần phải làm:
  + Những chức năng (dịch vụ) mà hệ thống phải cung cấp.
  + Những ràng buộc đối với hệ thống.
* Yêu cầu phần mềm phản ánh nhu cầu (mong muốn) của khách hàng với hệ thống.
* Việc mô tả các yêu cầu có nhiều cấp độ, có thể chia thành 2 mức [1]:
  + Yêu cầu người dùng (user requirements): những mô tả yêu cầu mức cao, trừu tượng (high-level abstract requirements).
    - User requirements:
      * Là những phát biểu tổng quan, bằng ngôn ngữ tự nhiên (hoặc được bổ sung bởi một số biểu đồ) về những dịch vụ (chức năng) mà hệ thống cần cung cấp cho người dung và các ràng buộc đi kèm.
      * Ví dụ: Hệ thống cho phép thêm một sản phẩm mới.
  + Yêu cầu hệ thống (system requirements): những mô tả yêu cầu mức chi tiết (detailed description of what the system should do).
    - System requirements:
      * Là những mô tả chi tiết hơn về các chức năng và ràng buộc mà hệ thống cần cung cấp.
      * Nó mô tả chính xác những hạng mục nào cần phải làm.
      * Nó có thể là một phần trong bản hợp đồng giữa khách hàng và nhà phát triển.
      * Ví dụ: Hệ thống cho phép thêm sản phẩm mới gồm các trường thông tin: tên sản phẩm, giá, kích thước..., Hệ thống tự động tạo mã sản phẩm theo mã danh mục sản phẩm.

**Yêu cầu chức năng và yêu cầu phi chức năng :**

**Yêu cầu chức năng:**

* Yêu cầu chức năng (Functional requirements) là những phát biểu về những chức năng (dịch vụ) mà hệ thống cần phải cung cấp.
* Trong một số trường hợp, nó cũng phát biểu cụ thể những cái hệ thống sẽ không cung cấp
* Ví dụ:
  + Người dùng có thể tìm kiếm sách theo mã sách, tiêu đề hoặc tên tác giả.
  + Hệ thống có thể xuất báo cáo thống kê số lượt mượn sách trong một tháng.

**Yêu cầu phi chức năng:**

* Yêu cầu phi chức năng (non-functional requirements) là những ràng buộc (constraints) đối với các chức năng (dịch vụ) mà hệ thống cung cấp.
* Các ràng buộc như:
  + về thời gian
  + về hiệu năng
  + về quy trình phát triển phần mềm…
* Yêu cầu phi chức năng thường áp dụng lên toàn hệ thống hơn là lên từng chức năng riêng lẻ.
* Ví dụ:
  + Hệ thống phải dễ sử dụng
  + Hệ thống phải hoạt động tốt trên Firefox x.x, Chrome x.x
  + Hệ thống phải phản hồi trong vòng < 1s đối với tất cả các yêu cầu tìm kiếm sách.

**Câu 7: Vai trò của việc làm bản thảo (Prototyping) trong phát triển phần mềm.**

**Trả lời:**

Trong quá trình phân tích và đặc tả yêu cầu (requirement engineering), bản thảo cho phép:

* Người dùng trải nghiệm với hệ thống, từ đó tinh chỉnh lại các yêu cầu.
* Phát hiện những ý tưởng mới.
* Nhận ra những điểm mạnh, điểm yếu của phần mềm.
* Phát hiện những lỗi phát sinh, những bất hợp lý trong yêu cầu.
* Từ đó, bản đặc tả yêu cầu có thể sẽ được thay đổi.

Trong quá trình thiết kế hệ thống (system design), bản thảo có thể giúp:

* Trải nghiệm để kiểm tra tính khả thi của bản thiết kế được đề xuất.

Chẳng hạn: bản thảo cơ sở dữ liệu có thể được xây dựng để kiểm tra việc truy cập/truy vấn vào cơ sở dữ liệu có hiệu quả không?

Bản thảo về kiến trúc hệ thống được xây dựng để kiểm tra luồng dữ liệu đi có phù hợp không?

* Hỗ trợ thiết kế giao diện người dùng (UI).

**Câu 8: Thuộc tính đánh giá chất lượng của đặc tả yêu cầu phần mềm.**

**Trả lời:**

**Các thuộc tính của bản đặc tả yêu cầu chất lượng:**

* Tính chính xác (Correct):
  + Nếu mỗi yêu cầu được phát biểu trong SRS là khớp với những mong muốn, yêu cầu cần có của khách hàng trong sản phẩm cuối.
* Tính đầy đủ (Complete):
  + Nếu tất cả các chức năng, ràng buộc ở sản phẩm cuối đều được phát biểu trong SRS.
* Tính không nhập nhằng, mơ hồ (Unambigous):
  + Nếu mọi yêu cầu được phát biểu một cách rõ ràng, không hiểu theo nhiều nghĩa.
  + Thường phát biểu bằng ngôn ngữ tự nhiên dễ gây nhập nhằng ◊ có thể dùng các biểu đồ hỗ trợ
* Tính có thể kiểm tra được (Verifiable):
  + Nếu sau khi làm xong sản phẩm, có thể kiểm tra xem sản phẩm đó có được xây dựng đúng với yêu cầu không?
  + Nên dùng có đơn vị đo lường (cho các yêu cầu phi chức năng)
* Tính nhất quán (Consistent):
  + Nếu các yêu cầu không xung đột với nhau: về thời gian, về logic...
  + Ví dụ:
    - Yêu cầu x: sự kiện a trước sự kiện b.
    - Yêu cầu y: sự kiện b trước sự kiện a.
* Sắp xếp độ quan trọng và độ ổn định của mỗi yêu cầu (Ranked for importance and/of stability):
  + Các yêu cầu thường có độ quan trọng không giống nhau.
  + Có những yêu cầu ít có khả năng thay đổi, nhưng cũng có những yêu cầu có thể thay đổi theo thời gian.
  + Tài liệu đặc tả yêu cầu thể hiện được độ quan trọng, độ ổn định của mỗi yêu cầu ->giúp nhà phát triển có kế hoạch thực hiện tốt hơn.

**Câu 9. Các hoạt động chính trong một quy trình kỹ nghệ yêu cầu (Requirements engineering process).**

**Trả lời:**

Quy trình kỹ nghệ yêu cầu gồm 3 nhiệm vụ cơ bản:

1. Thu thập và Phân tích yêu cầu:

* Kết thúc giai đoạn này, người phân tích (analyst) sẽ cơ bản hiểu được hệ thống cần phải như thế nào:
  + Có những chức năng gì, các ràng buộc, các dữ liệu vào và kết quả đầu ra của hệ thống.
* Quá trình này thường liên quan tới rất nhiều người, gọi là các bên liên quan (stackholder).
  + Khách hàng, người dùng cuối, người phân tích, người quản lý...
* Trong quá trình phân tích yêu cầu, nhà phát triển (analyst) và phía khách hàng, người dùng cuối (clients, users) sẽ có một chuỗi các cuộc họp:
  + Các cuộc họp ban đầu, khách hàng và người dùng cuối sẽ mô tả về công việc của họ, môi trường của họ, các quy trình...
  + Người phân tích sẽ lắng nghe, ghi chép, thu thập các thông tin, biểu mẫu...
  + Một khi đã hiểu được ở mức độ cơ bản, người phân tích sẽ viết các tài liệu yêu cầu người dùng, một số mô hình, một số phác thảo ->để làm rõ những điểm chưa hiểu, hoặc để xác nhận lại với khách hàng xem có đúng ý như vậy không?

1. Đặc tả yêu cầu:

* Sau khi phân tích yêu cầu, người phân tích đã hiểu được về hệ thống cần phải làm gì.
* Đặc tả yêu cầu là việc ghi vào tài liệu những mô tả về hệ thống (tài liệu đặc tả yêu cầu – SRS – Software Requirement Specification)
* Tại hoạt động này, những vấn đề như: giao diện, ngôn ngữ sử dụng, công cụ... cũng được bàn tới.

1. Thẩm định yêu cầu:

* Thẩm định yêu cầu tập trung vào việc đảm bảo bản đặc tả yêu cầu trên là đúng với những yêu cầu/mong muốn/mục tiêu của khách hàng về hệ thống cần xây dựng.
* Kết quả của hoạt động này là một tài liệu đặc tả yêu cầu đã được thẩm định (validated SRS), một bản đặc tả yêu cầu chất lượng tốt.

1. **THIẾT KẾ PHẦN MỀM:**

**Câu 10. Vai trò của thiết kế kiến trúc hệ thống.**

**Trả lời:**

* Là phương tiện giao tiếp để truyền đạt ý tưởng, đề xuất (Understanding and communication)
* Giúp xác định các thành phần tái sử dụng
* Giúp dễ dàng phân chia công việc xây dựng hệ thống (Construction and Evolution)
* Giúp phân tích một số thuộc tính quan trọng

**Câu 11. Các góc nhìn kiến trúc hệ thống.**

**Trả lời:**

* Trong lĩnh vực phần mềm, những góc nhìn kiến trúc khác nhau sẽ thể hiện những khía cạnh khác nhau về hệ thống phần mềm cần xây dựng.
* Một góc nhìn (a view) thể hiện cấu trúc của hệ thống:
  + Có những thành phần nào trong hệ thống (elements)
  + Mối quan hệ giữa các thành phần đó như thế nào (relationships)

**Câu 12. Độ đo coupling và cohesion trong thiết kế phần mềm.**

**Trả lời:**

**Coupling:**

* Coupling thể hiện sự phụ thuộc, sự kết nối giữa các module với nhau.
* Trong thiết kế, người ta hướng tới mục tiêu Low coupling.
* Low coupling (Loose coupling): – 2 modules được gọi là low coupling nếu chúng ít phụ thuộc vào nhau, sự thay đổi trong module này ít làm ảnh hưởng hoặc không làm ảnh hưởng đến module kia.
* High coupling (Tight coupling): – 2 modules được gọi là high coupling nếu chúng phụ thuộc chặt chẽ vào nhau, sự thay đổi trong module này dẫn đến phải thay đổi ở module kia.

**Conhesion:**

* Cohesion thể hiện sức mạnh liên kết giữa các thành phần trong cùng một module.
* Khi thiết kế module, cần hướng tới High cohesion.
* High cohesion:
  + Một module được gọi là high cohesion nếu các thành phần bên trong nó có liên kết chặt chẽ với nhau.
  + Nếu xét một module là class, thì các thành phần bên trong nó sẽ là các trường dữ liệu, các phương thức..
* Low cohesion:
  + Nếu các thành phần bên trong nó rời rạc, không liên kết với nhau.
  + Một module mà thực hiện nhiều chức năng khác nhau, sẽ dễ dẫn tới low cohesion.
    - Ví dụ lớp Utility: định dạng ngày giờ, định dạng đường dẫn.
* Có nhiều cách để chia hệ thống thành các module nhỏ khác nhau.
* Việc phân chia tốt sẽ làm cho hệ thống đạt được tính chất high cohesion trong mỗi module và tính chất low coupling giữa các module.
* Xét trường hợp module là class, khi thiết kế phải trả lời các câu hỏi:
  + Nên có bao nhiêu lớp? Là những lớp nào?
  + Trong lớp đó, sẽ định nghĩa những phương thức gì, lưu dữ liệu gì?

**Các loại Conhesion:**

Một số kiểu kết hợp, tạo nên module có các loại cohesion khác nhau. Dưới đây là các loại được tổng hợp, từ tệ nhất đến tốt nhất:

* Coincidental cohesion (worst)
* Logical cohesion
* Temporal cohesion
* Procedural cohesion
* Communicational/informational cohesion
* Sequential cohesion – Functional cohesion (best)

1. **LẬP TRÌNH PHẦN MỀM**

**Câu 13: Hai hướng tiếp cận cho việc viết mã:Lập trình tăng dần(Incremental coding) và lập trình hướng kiểm thử(Test-driven development)**

**Trả lời:**

**Quy trình viết mã tăng dần:**

* Viết mã cho một số chức năng ban đầu.
* Viết test script để kiểm thử.
* Chạy test script, sửa lỗi nếu có.
* Tiếp tục bổ sung viết mã cho các chức năng khác.

**Điểm mạnh:**

* Dễ debug (dễ phát hiện lỗi): vì lập trình viên viết và chạy test script sau mỗi lần viết mã cho chức năng mới, nên nếu chạy test script phát hiện lỗi, thì lỗi này thuộc về phần mã mới bổ sung vào gần nhất.
* Việc viết test script có ý nghĩa rất lớn khi mã lệnh càng ngày càng nhiều: test script giúp kiểm tra nhanh, chính xác và thường xuyên.
* Test script là cơ sở để thực hiện kiểm thử đơn vị (unit testing) trước khi đưa module lên server (repository)

**Hạn chế:**

* Giữa mã nguồn và test scripts thường không đồng bộ: Viết mã rất nhiều, nhưng test scripts lại ít, không đủ độ phủ kiểm tra hết các đoạn mã đã viết ra.

**Quy trình viết mã hướng kiểm thử:**

* Viết test script cho một số chức năng ban đầu dựa trên các đặc tả.
* Viết mã để thoả các test cases.
* Chạy test script.
  + Nếu có lỗi thì sửa, rồi chạy lại test scripts, lặp lại cho đến khi thoả.
  + Nếu không có lỗi, kiểm tra xem có cần cải tiến mã không, nếu có thì cải tiến, rồi chạy lại test scripts lần nữa.
* Tiếp tục viết test scripts cho các chức năng khác (nếu còn) dựa trên các đặc tả.

**Điểm mạnh:**

* Mã nguồn luôn đồng bộ với test scripts.
* Vì phải viết test cases dựa trên các đặc tả, và viết code sau đó để thỏa các test cases nên hệ thống xây dựng ra sẽ đáp ứng tốt các yêu cầu đặt ra.
* Những chức năng quan trọng thường được viết test script trước, nên chúng sẽ được lập trình trước và được kiểm thử nhiều nhất

**Hạn chế:**

* Tính đầy đủ của mã nguồn phụ thuộc vào tính toàn diện, đầy đủ của bộ test cases được thiết kế. Thông thường, khó có thể viết các test cases một cách đầy đủ (thường sót đối với các trường hợp đặc biệt).
* Đôi khi việc viết mã để thỏa test case sau lại phải dẫn đến việc thay đổi thuật toán, các đoạn mã đã có.

**Câu 14: Kỹ thuật tái cấu trúc mã nguồn(Refactoring)**

**Trả lời:**

**Tái cấu trúc mã nguồn:**

* Refactoring là một kỹ thuật cho việc tái cấu trúc mã nguồn đã có, thay đổi cấu trúc bên trong mà không làm thay đổi hành vi bên ngoài.
* Theo định nghĩa của Martin Fowler (https://refactoring.com/): Refactoring là việc làm thay đổi cấu trúc bên trong của phần mềm, làm cho nó dễ hiểu, chi phí sửa đổi thấp mà không làm thay đổi hành vi bên ngoài.
* Mục tiêu cơ bản của Refactoring là cải tiến thiết kế (ở giai đoạn viết mã, không phải ở giai đoạn thiết kế).
  + Kết quả: làm tăng cohesion và giảm coupling, tuân theo nguyên lý đóng-mở tốt hơn.
* Đối tượng của Refactoring là: mã nguồn.
* Rủi ro khi làm refactoring là: làm hỏng hệ thống. Để giảm thiểu rủi ro, cần tuân theo 2 nguyên tắc:
  + Tái cấu trúc từng bước nhỏ một.
  + Có test scripts để kiểm tra lại sau khi thay đổi.
* Khi nào thì cần thực hiện Refactoring? Đó là khi trong mã nguồn có một số dấu hiệu không tốt (gọi là “bad smells”):
  + Trùng mã (Duplicate code)
  + Phương thức quá dài (Long method)
  + Tham số phương thức quá nhiều (Long parameter list)
  + Lớp quá dài (Long class)

1. **KIỂM THỬ PHẦN MỀM**

**Câu 15: Kiểm thử phần mềm là gì?Xác minh và thẩm định(Software Verification và Software Validation)**

**Trả lời:**

**Kiểm thử phần mềm:**

Theo Glen Myers, kiểm thử phần mềm là quá trình vận hành để tìm ra lỗi.

Một ca kiểm thử (test case) tốt: là ca kiểm thử có xác suất cao tìm ra lỗi chưa phát hiện.

Mục tiêu: thiết kế các ca kiểm thử để có thể phát hiện ra một cách có hệ thống các loại lỗi khác nhau với chi phí thời gian và công sức ít nhất có thể.

**Xác minh và thẩm định:**

Xác minh và thẩm định là tên gọi chung của các hoạt động SQA, nhằm đảm bảo phần mềm được:

* Phát triển đúng theo đặc tả
* Đáp ứng nhu cầu của người dùng

Tuy nhiên V&V là hai hoạt động khác nhau. Theo Boehm:

* Xác minh (verification) là sự kiểm tra xem sản phầm có đúng với đặc tả không, tức là chú trọng vào việc phát hiện lỗi phần mềm.
* Thẩm định (validation) là sự kiểm tra xem sản phẩm có đáp ứng được nhu cầu người dùng không, tức là chú trọng vào sự khác biệt giữa phần mềm làm ra và cái người dùng mong đợi.

Tuỳ theo thời điểm và đối tượng của hoạt động kiểm tra, người ta phân biệt V&V tĩnh và động:

* V&V tĩnh: là sự kiểm tra phần mềm mà không thực hiện chương trình. Ví dụ: xét duyệt yêu cầu, xét duyệt thiết kế, thanh tra mã nguồn (code review)
* V&V động: là sự kiểm tra phần mềm thông qua việc thực thi phần mềm (sau khi phần mềm hoặc một bộ phận phần mềm được xây dựng)

**V&V động thường được gọi là kiểm thử phần mềm (testing).**

**Câu 16: Các phương pháp kiểm thử (Black box testing, White box testing, Grey box testing).**

**Trả lời:**

**Các phương pháp kiểm thử:**

* Kiểm thử hộp đen (Blackbox testing)
* Kiểm thử hộp trắng (Whitebox testing)
* Kiểm thử hộp xám (Greybox testing)

**Đặc điểm:**

*Đặc điểm kiểm thử hộp đen:*

* Xem phần mềm như một hộp đen, người kiểm thử không cần quan tâm đến cấu trúc và hoạt động bên trong.
* Cách thực hiện: nhập dữ liệu đầu vào qua giao diện, thực thi chương trình, xem kết quả đầu ra có đúng với mong đợi không?
* Còn được gọi là kiểm thử chức năng (functional testing) hay kiểm thử hành vi (behavior testing)
* Cơ sở của kiểm thử hộp đen là đặc tả chức năng và dữ liệu đầu vào mà nó sử dụng.

Điểm mạnh:

* Không cần truy cập mã nguồn
* Không cần kiến thức về các ngôn ngữ lập trình
* Kiểm thử được thực hiện với góc nhìn của người dùng ->Mang tính khách quan (tách biệt với góc nhìn của lập trình viên)

Hạn chế:

* Khó thiết kế test cases
* Kiểm thử tất cả các khả năng là không khả thi

*Đặc điểm kiểm thử hộp trắng:*

* Là việc kiểm tra các đoạn mã chương trình, xem xét cấu trúc bên trong, xem nó có vận hành đúng theo thiết kế hay không.
* Người kiểm thử yêu cầu phải hiểu biết về lập trình, về sự vận hành của chương trình.
* Các tên gọi khác: glass testing, structural testing, open box testing, clear box testing.
* Cơ sở của kiểm thử hộp trắng: dựa trên mã nguồn: các rẽ nhánh, vòng lặp (if, while, for...)

Điểm mạnh:

* Rất hiệu quả trong việc tìm lỗi, đặc biệt là các lỗi ẩn (mà blackbox testing không phát hiện ra)

Hạn chế:

* Phải truy cập vào mã nguồn
* Có thể không nhìn ra những chức năng bị thiếu (chưa viết)

*Đặc điểm của kiểm thử hộp xám*:

* Là sự kết hợp giữa Whitebox + Blackbox testing
* Người kiểm thử không truy cập vào mã nguồn.
* Nhưng biết được về thông tin cấu trúc bên trong (có giới hạn) thông qua : Các tài liệu đặc tả, tài liệu thiết kế chi tiết, thiết kế kiến trúc, thuật toán, cấu trúc dữ liệu sử dụng...

Điểm mạnh:

* Sử dụng được các điểm mạnh của hai phương pháp kiểm thử Hộp đen và Hộp trắng.
* Có thể thiết kế được một số kịch bản test phức tạp và thông minh (dựa vào việc biết được một phần cấu trúc bên trong, các bước xử lý, con đường đi dữ liệu...)
* Vẫn đảm bảo sự tách biệt (về góc nhìn) giữa người kiểm thử và lập trình viên.

Hạn chế:

* Không thể test hết tất cả trường hợp (vì vẫn không truy cập vào mã nguồn)

**Câu 17:Các loại hình kiểm thử (Type of testing).**

**Trả lời:**

* System Test (ST)
* User Acceptance Test (UAT)
* Functional Test (FT)
* User Interface (UI) Test
* Server side Test
* Security Tests
* Database Testing
  + Transaction errors
  + Concurrency errors
* Performance Test
* Documentation Test
* Regression Test