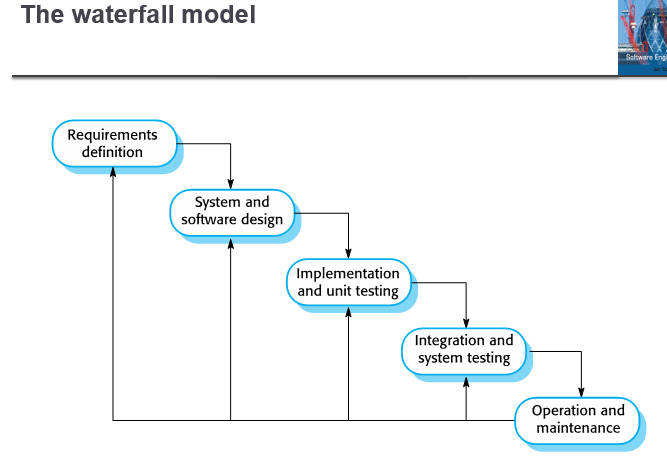
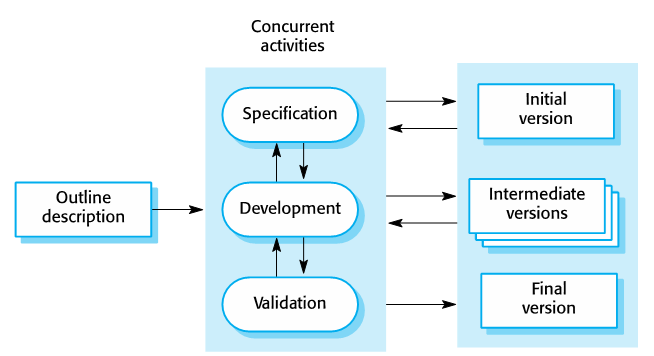
1. **Kỹ thuật phần mềm (Software Engineering)**

* Đầu tiên cần phải hiểu Software Engineering là gì?
  + Về cơ bản phần mềm đóng 1 vai trò rất quan trọng trong sự phát triển của một quốc gia
  + Rất nhiều hệ thống được controlled bởi các phần mềm
  + Kỹ thuật phần mềm, hay Software Engineering liên quan đến các lý thuyết, phương pháp và các công cụ để phát triển phần mềm chuyên nghiệp
* Software Cost
  + Chi phí cho cho phần mềm thường lớn hơn chi phí về phần cứng
  + Chi phí bảo trì thường lớn hơn chi phí phát triển, đối với những hệ thống được phát triển dài hạn, việc bảo trì hệ thống thường tốn chi phí gấp nhiều lần so với việc phát triển ứng dụng
  + SE sẽ chú trọng vào việc phát triển phần mềm hiệu quả với chi phí bỏ ra.
* Các câu hỏi thường gặp về SE:
  + Phần mềm là gì? Là các ứng dụng máy tính, di động và các tài liệu liên quan. Software Products có thể được phát triển cho những nhóm khách hàng nhất định, hoặc cho một thị trường lớn.
  + Các đặc tính của một phần mềm tốt?
    - Sản phẩm cuối cùng đáp ứng được các chức năng yêu cầu đề ra
    - Hiệu năng đến người dùng cuối cùng đạt hiệu quả như yêu cầu.
    - Phần mềm phải dễ dàng bảo trì, đáng tin cậy và sử dụng được.
  + SE là gì?
    - Là một ngành kỹ thuật, hay có thể coi là một môn học tập trung vào các khía cạnh của việc phát triển phần mềm từ những giai đoạn ban đầu của việc tạo ra các đặc tả cho yêu cầu của hệ thống cho đến khi phần mềm đã được đưa ra sử dụng và được bảo trì sau đó.
  + Các hoạt động cơ bản của SE?
    - Software Specification: đặc tả phần mềm
    - Software development: phát triển phần mềm
    - Software validation: kiểm thử phần mềm
    - Software Evolution: sự phát triển phần mềm
  + Sự khác nhau cơ bản giữa Kỹ thuật phần mềm và Khoa học máy tính?
    - Khoa học máy tính tập trung nhiều vào các lý thuyết và các nguyên tắc cơ bản, còn SE sẽ tập trung vào các vấn đề thực tế trong việc phát triển và đưa đến một phần mềm hữu dụng
  + Sự khác nhau giữa SE và system engineering?
    - system engineering rộng hơn SE, nó quan tâm đến tất cả các khía cạnh của hệ thống bao gồm cả phần cứng, phần mềm và các process của hệ thống. SE chỉ là một phần của các quy trình nói chung
  + Những khó khăn, thách thức của SE?
    - Sự đa dạng của các thiết bị
    - Thời gian yêu cầu để có được sản phẩm giảm
    - Phát triển được những phần mềm đáng tin cậy
  + Các nguyên tắc cơ bản của SE:
    - Hệ thống được phát triển sử dụng một process dễ quản lý và dễ hiểu. Tất nhiên những loại phần mềm khác nhau sẽ có những process khác nhau
    - Sự tin cậy và hiệu năng vô cùng quan trọng với tất cả mọi hệ thống.
    - Việc hiểu và nắm rõ các đặc tả của ứng dụng và các yêu cầu của ứng dụng là rất quan trọng
    - Trong những trường hợp, chúng ta nên sử dụng lại những phần mềm đã có hơn là tạo mới 1 phần mềm.
    - Web-bases là một nền tảng có thể chạy được các ứng dụng và nhiều tổ chức đang phát triển các hệ thống chạy trên web-base nhiều hơn là trên các hệ thống cục bộ.
    - Web-service cho phép các chức năng có thể được truy cập qua web
    - Cloud computing là một các để cung cấp các dịch vụ máy tính, trong đó các ứng dụng được chạy từ xa thông qua Cloud

1. **Software Process (quy trình phần mềm)**
   1. Software Process:
      1. Là một tập hợp các activities (hoạt động, hành động) cần thiết để tạo ra một phần mềm
      2. Có rất nhiều các SP, nhưng chung lại thì một hệ thống sẽ có:
         1. Specification (đặc tả): Định nghĩa những thứ mà hệ thống sẽ làm
         2. Design & implementation (thiết kế và thực hiện): xác định cách tổ chức của hệ thống và triển khai hệ thống
         3. Validation: kiểm thử xem hệ thống có thực hiện đúng những chức năng mà khách hàng mong muốn
         4. Evolution: thay đổi, bảo trì hệ thống theo yêu cầu của khách hàng
      3. Software process model là một cách nói trừu tượng của một process, nó sẽ mô tả một process ở một số góc nhìn cụ thể
      4. Khi chúng ta miêu tả và discuss về một process, chúng ta thường nói về các hoạt động trong process này như là xác định data model, designing giao diện người dùng … và thứ tự của các hoạt động này
      5. Process description còn có thể bao gồm:
         1. Products, là sản phẩm của các process activities
         2. Roles, phản ảnh trách nhiệm của những người tham gia vào process này
         3. Pre- và post- conditions: là các statements đúng lúc trước và sau khi process activities được cấu thành hoặc một product được tạo ra.
   2. Plan - driven and agile processes
      1. Plan-driven processes là một processes mà tất cả các process activities sẽ được planned và tiến độ sẽ được theo dõi theo kế hoạch này
      2. Trong agile processes, planning được thực hiện một cách gia tăng, việc thay đổi quy trình dễ dàng hơn để có thể đáp ứng được yêu cầu thay đổi của khách hàng
      3. Trên thực tế hầu hết các quy trình bao gồm các yếu tố của cả Plan-driven và agile process
      4. Không có quy trình phần mềm nào là đúng và sai.
2. **Software process models**
   1. Waterfall model:
      1. Plan-driven model, tách biệt giữa giai đoạn đặc tả và development

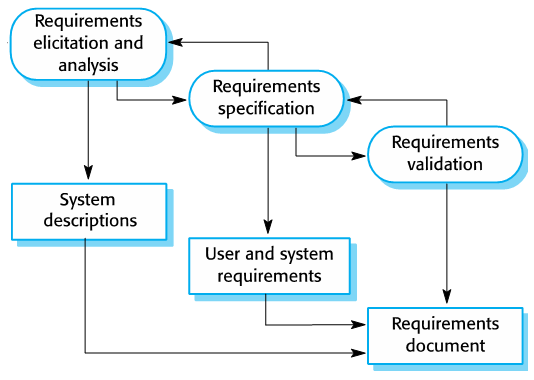


* + 1. Các giai đoạn được chia ra như sau:
       1. Requirements analysis & definition
       2. Thiết kế hệ thống và phần mềm
       3. Phát triển phần mềm và unit testing
       4. Test tích hợp và test hệ thống
       5. Vận hành và bảo trì
    2. Hạn chế của mô hình này đó là khó để đáp ứng lại sự thay đổi sau khi đã vận hành. Một giai đoạn phải được hoàn thành trước khi chuyển sang giai đoạn tiếp theo
       1. Việc không linh hoạt trong phân chia dự án gây khó khăn trong việc đáp ứng các yêu cầu thay đổi của khách hàng
       2. Chỉ thích hợp khi các yêu cầu được hiểu rõ ràng và các thay đổi sẽ ít trong quá trình thiết kế
       3. Rất ít hệ thống kinh doanh không có sự thay đổi
       4. Mô hình này chủ yếu được sử dụng cho các dự án kỹ thuật hệ thống lớn trong đó 1 hệ thống được phát triển tại một số sites
  1. Incremental development

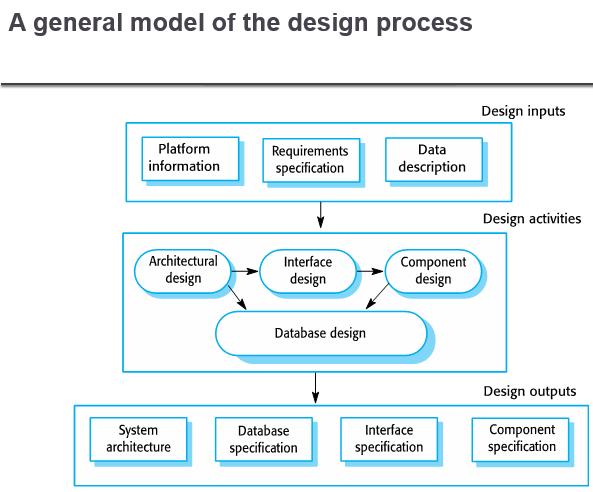


* + 1. Đặc tả, phát triển và kiểm thử được thực hiện xen kẽ. Có thể áp dụng plan-driven hoặc agile
    2. Chi phí cho việc đáp ứng cho sự thay đổi của khách hàng sẽ được giảm bớt so với waterfall model
    3. Có thể nhận được các feedback của khách hàng trong quá trình development trong những phần việc hay chức năng đã được hoàn thành
    4. Khách hàng có thể dễ dàng xem được tiến độ của công việc đã được hoàn thành
    5. Nhanh chóng ra được sản phẩm hơn so với Waterfall process
    6. Các hạn chế:
       1. Process sẽ không visible:
          1. Managers sẽ cần có các deliverables để đánh giá tiến độ. Nếu như hệ thống được phát triển quá nhanh thì việc tạo ra các documents để phản ảnh mọi phiên bản của hệ thống là không hiệu quả về mặt chi phí.
          2. Cấu trúc hệ thống có thể bị ảnh hưởng mỗi khi có các thay đổi.
  1. Integration và configuration (tích hợp và cấu hình)
     1. Hệ thống được phát triển dựa vào các components có sẵn, có thể sử dụng plan-driven hoặc agile model
     2. Các component được sử dụng lại có thể được config lại, điều chỉnh lại để phù hợp với yêu cầu mới
     3. Tái sử dụng là cách tiếp cận tiêu chuẩn để xây dựng nhiều loại hệ thống kinh doanh
     4. Ưu điểm:
        1. Giảm chi phí và rủi ro do tái sử dụng được các hệ thống cũ hoặc các components cũ
        2. Ra được sản phẩm nhanh hơn
     5. Các hạn chế:
        1. Các hệ thống ứng dụng độc lập được cấu hình để sử dụng trong một Env cụ thể
        2. Các dịch vụ web cần phát được phát triển theo dạng service standards để có thể remote được từ xa.
        3. Mất quyền kiểm soát về quá trình phát triển các hệ thống được sử dụng lại
  2. Trên thực tế các hệ thống lớn đều được phát triển sử dụng process kết hợp giữa các yếu tố của các models

1. **Process activities**
   1. Đặc tả phần mềm (software specification)
      1. Requirements engineering process



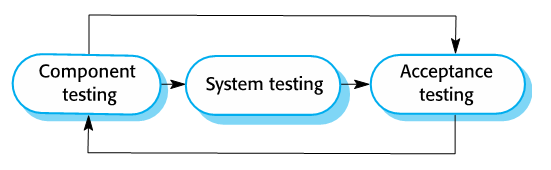
* + - 1. Các yêu cầu được tổng quát và đánh giá: trả lời cho câu hỏi “các stakeholder yêu cầu và kỳ vọng thế nào cho hệ thống này?”
      2. Đặc tả các yêu cầu
      3. Kiểm định lại các yêu cầu
    1. Là process để xác định các chức năng nào là cần phải thực hiện và các ràng buộc của hệ thống và thực hiện
  1. Software design & implementation

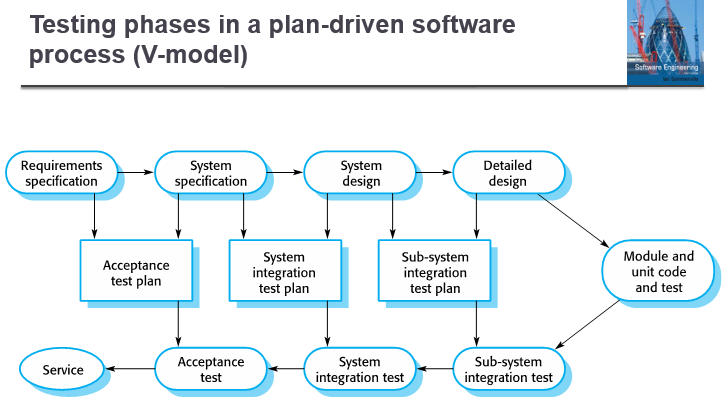


* + 1. Là process mà chúng ta sẽ converting system specification thành một hệ thống chạy được.
    2. Software design:
       1. Thiết kế software structure dựa vào các đặc tả
       2. Architectural: nơi chúng ta sẽ xác định các structure tổng quan của system, các thành phần chính, các subsystems hoặc các modules, mối quan hệ và cách mà các thành phần này làm việc với nhau.

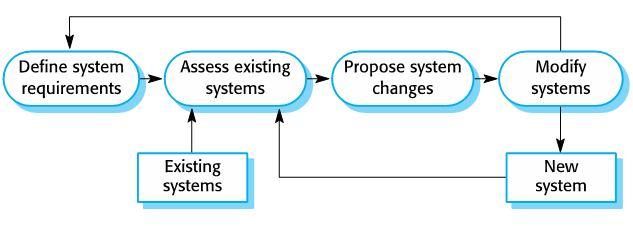
Database Design: design database của system, data structure của CSDL này

* + - 1. Interface design: nơi chúng ta interface giữa các system components
      2. Component selection và design: nơi chúng ta xác định các components có thể tái sử dụng được, nếu không có chúng ta sẽ design cách mà các component này vận hành.
    1. Implementation:
       1. Biến các structure thành một executable program
    2. Các activities của design và implementation có quan hệ chặt chẽ với nhau và có thể được thực hiện xen kẽ
  1. Software validation:
     1. Verification và validation (V&V) là bước xác định xem một hệ thống hoạt động đúng với đặc tả hay không, và có đáp ứng các yêu cầu của khách hàng hay không
     2. Sẽ bao gồm việc kiểm thử và review process và testing cho toàn hệ thống
     3. System testing sẽ bao gồm việc thực hiện chạy thử hệ thống với các test cases được tạo thành từ giai đoạn đặc tả với dữ liệu thật hoặc các dữ liệu phục vụ kiểm thử.
     4. Stages of Testing:



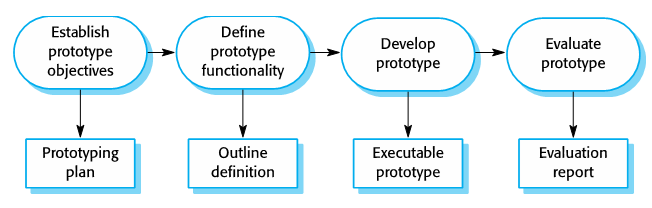


* 1. Software evolution:

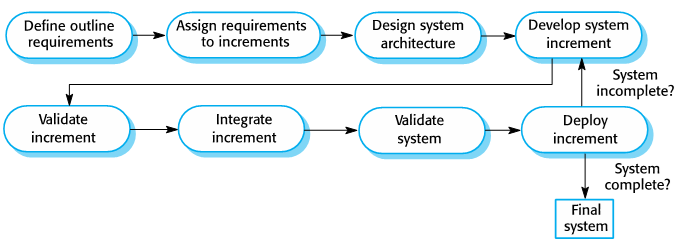


* + 1. Một ứng dụng có phải linh hoạt và có thể thay đổi
    2. Một ứng dụng có thể thay đổi do sự thay đổi về business (hoàn cảnh kinh doanh của khách hàng hoặc thị trường)

1. **Coping with change (đối mặt với sự thay đổi)**
   1. Thay đổi là điều chắc chắn xảy ra ở tất cả các phần mềm hay hệ thống lớn
      1. Thay đổi về business dẫn đến sự thay đổi về hệ thống
      2. Các công nghệ mới xuất hiện dẫn đến sự thay đổi cần thiết trong hệ thống
      3. Sự thay đổi về platform
   2. Mỗi sự thay đổi sẽ dẫn đến việc chúng ta phải rework lại công việc, do đó chi phí của một sự thay đổi cũng bao gồm cả những chi phí cho việc đánh giá lại requirement, và đương nhiên là chi phí cho việc implement lại chức năng hay sự thay đổi mới đó.
   3. Giảm bớt chi phí cho những sự thay đổi:
      1. Dự đoán những thay đổi, là một trong số những activities của SP giúp cho việc lường trước được những thay đổi có thể xảy ra, làm giảm đáng kể chi phí nếu cần phải rework
         1. Ví dụ: prototype system có thể được xây dựng để show ra các key feature của hệ thống đối với customers
      2. Process được thiết kế để làm sao có thể các thay đổi có thể được thực hiện với chi phí thấp nhất có thể, thường liên quan đến việc sử dụng các hình thức của phát triển gia tăng. Tức là những thay đổi được đề xuất sẽ được thực hiện dần dần chứ không thực hiện cùng 1 lúc.
   4. Đối mặt với sự thay đổi requirements
      1. System prototyping sẽ giúp tham khảo ý kiến của customers trước, khi mà các chức năng chính được tạo ra, xem xét tính khả thi của các chức năng, đồng thời cũng hỗ trợ dự đoán các thay đổi có thể có
   5. Incremental delivery: hệ thống được đưa đến tay customers để nhận xét và thử nghiệm, đồng thời hỗ trợ việc tránh các thay đổi, hoặc giảm bớt chi phí nếu có những thay đổi.
   6. Software prototyping:
      1. Là một phiên bản của hệ thống được dùng để thể hiện ra các chức năng chính hoặc để thử nghiệm các design option
      2. Thường được dùng trong:
         1. Trong Requirements engineering process để thực hiện việc thẩm định và gợi ý các requirements
         2. Trong design process, chúng ta dùng prototyping system để xem các options và thiết kế UI.
         3. Trong testing process, chạy back-to-back tests
      3. Ưu điểm:
         1. Tăng tính sử dụng của hệ thống
         2. Một phiên bản dùng để đánh giá nhu cầu của người dùng
         3. Tăng chất lượng design
         4. GIảm các efforts không cần thiết
         5. Tăng tính bảo trì.
      4. Process của prototype development:

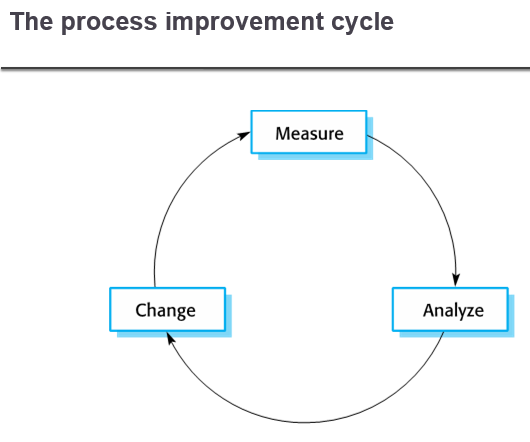


* + - 1. Có thể dựa vào những ngôn ngữ giúp phát triển nhanh prototype hoặc tools
      2. Có thể được giới hạn các chức năng hoặc giai đoạn, khi đó
         1. Prototype sẽ tập trung vào những chức năng của ứng dụng mà requirements chưa được rõ ràng
         2. Việc kiểm thử có thể được bỏ qua
         3. Tập trung vào các functional hơn là những non-functional requirements như là độ tin cậy hay bảo mật
    1. Prototype sẽ được loại bỏ vì nó không phải là một bộ khung tốt cho việc phát triển ứng dụng chạy thật
  1. Incremental delivery (phân phối gia tăng)



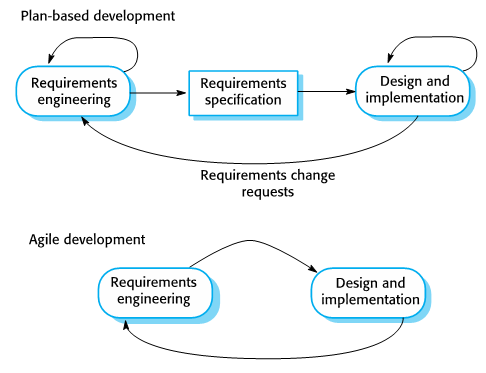
* + 1. Thay vì chỉ có 1 delivery system, chúng ta sẽ chuyển giao đến tay customers nhiều phiên bản của ứng dụng. Mỗi một phiên bản sẽ cung cấp 1 phần của mỗi chức năng trong ứng dụng
    2. Các requirement có độ ưu tiên cao hay các chức năng có độ ưu tiên cao sẽ được hoàn thành sớm hơn
    3. Các yêu cầu có thể được tạm dừng phát triển và có thể được quay lại phát triển tiếp về sau này.
    4. Ưu điểm:
       1. System cùng với các chức năng quan trọng sẽ đến tay người dùng hoặc customers sớm
       2. Những version ban đầu có thể đóng vai trò như là prototype để giúp việc phát triển các requirements có những lần gia tăng sau này
       3. Hạn chế được khả năng thất bại của project
       4. Các chức năng hay dịch vụ có độ ưu tiên cao thường nhận được nhiều sự kiểm thử nhất
    5. Nhược điểm:
       1. Hầu hết các hệ thống đều yêu cầu một tập hợp các phương tiện cơ bản, được những phần khác nhau trong hệ thống sử dụng, do đó các detail của requirements thường không được xác định chi tiết cho đến chức năng của phần đó được thực hiện ⇒ Khó xác định các tập hợp cơ bản đó
       2. Bản chất của iterative process là các đặc tả sẽ được phát triển cùng với phần mềm, điều này có thể gây khó khăn mô hình của nhiều tổ chức, khi mà đặc tả là 1 phần của hợp đồng phát triển hệ thống

1. **Process improvement (Cải tiến quy trình)**
   1. Cải tiến quy trình phần mềm (SP) để nâng cao chất lượng phần mềm, giảm chi phí, tăng tốc độ phát triển.
   2. Process improvement nghĩa là hiểu được các processes sẵn có và thay đổi nó làm sao cho tăng được chất lượng phần mềm + giảm chi phí và thời gian phát triển
   3. Agile approach, tập trung vào việc phát triển một cách lặp đi lặp lại, giảm chi phí trong quy trình phần mềm



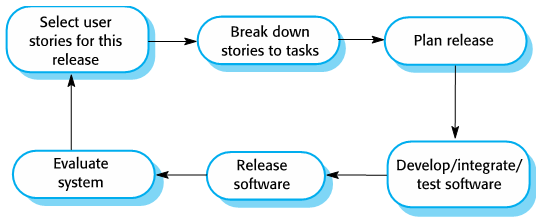
* + 1. Cung cấp một cách nhanh chính các chức năng
    2. Đáp ứng được nhu cầu thay đổi của khách hàng.
  1. Process improvement activities:
     1. Đánh giá process:
        1. Thực hiện việc đánh giá 1 hay nhiều các đặc tính của SP hoặc sản phẩm, từ đó đánh giá xem process improvement có thực sự hiệu quả hay không
     2. Process analyst:
        1. Xác định các điểm yếu của quy trình
        2. Các Process models được dùng để miêu tả các quy trình được thực hiện
     3. Process change:
        1. Được dùng để khắc phục các yếu điểm được xác định ở bước analyst

1. **Agile Software Development**
   1. Rapid software development:
      1. Yêu cầu phải nhanh chóng phát triển về có được delivery bây giờ là điều rất quan trọng đối với những Software System
      2. Plan-driven development là thiết yếu, tuy nhiên có một số loại hệ thống không thể áp dụng được phương pháp này
   2. Vậy Agile Software development là gì?



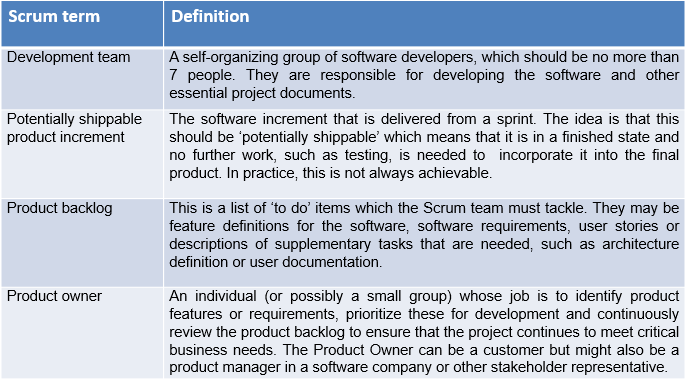
* + 1. Việc đặc tả hệ thống, thiết kế và phát triển được thực hiện xen kẽ nhau
    2. Hệ thống được phát triển thành nhiều versions, hoặc thành nhiều increments, với sự tham gia của các stakeholder trong cả quá trình đặc tả các chức năng cho từng phiên bản và phát triển
    3. Các phiên bản được delivery liên tục.
    4. Các công cụ, ví dụ như automation testing tools được sử dụng để support development
    5. Giảm thiểu số lượng documentation, tập trung vào coding
    6. Việc đặc tả phần mềm, thiết kế, triển khai và kiểm thử đều liên kết với nhau
  1. Agile Methods (phương pháp agile):
     1. Các phương pháp này bao gồm:
        1. Tập trung vào code hơn là design
        2. Dựa trên phát triển lặp (iterative)
        3. Nhanh chóng ra được sản phẩm
        4. Các sản phẩm phải có thể dễ dàng và đáp ứng các thay đổi nhanh chóng
     2. Mục đích của Agile methods là để giảm thiểu các chi phí trong SP, ví dụ như giảm số lượng các documentation, và đồng thời có thể nhanh chóng phản hồi lại các sự thay đổi về requirement và rework.
  2. Tuyên ngôn của Agile:
     1. Các developers nên tự vận động hơn là phụ thuộc vào quy trình quá nhiều, và đồng thời phải có sự tương tác với nhau để cùng phát triển
     2. Cộng tác với khách hàng: khách hàng sẽ tham gia vào quá trình phát triển như cung cấp các chức năng cần được ưu tiên, các yêu cầu hệ thống mới, đánh giá hệ thống sau mỗi giai đoạn
     3. Sản phẩm dùng được quan trọng hơn tài liệu sản phẩm: tập trung viết những tài liệu mà cần đọc.
     4. Phản hồi với sự thay đổi quan trọng hơn là bám sát kế hoạch.
  3. Khả năng áp dụng Agile:
     1. Các sản phẩm phần mềm dành cho công ty vừa và nhỏ dành cho sale, tuy nhiên hiện nay agile methods được áp dụng cho rất nhiều phần mềm với nhiều quy mô khác nhau
     2. Phát triển các hệ thống tùy chỉnh trong một tổ chức, nơi mà khách hàng sẽ tham gia nhiều vào quá trình phát triển, có ít quy tắc hay các quy định bên ngoài ảnh hưởng đến phần mềm.

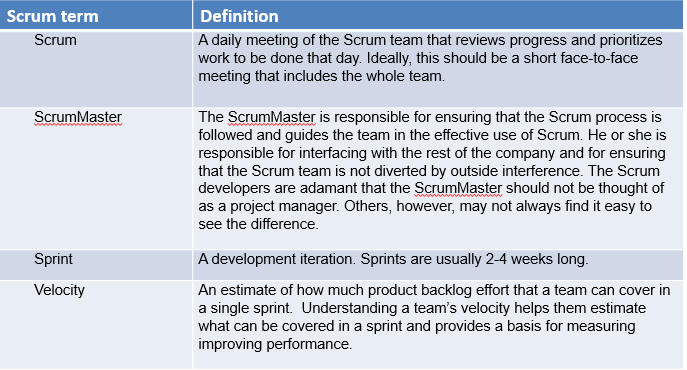
1. **Agile development techniques (các kỹ thuật để phát triển theo hình thức Agile)**
   1. Extreme programming
      1. Những versions mới có thể được built vài lần / ngày
      2. Các phiên bản increments sẽ được chuyển giao cho khách hàng mỗi 2 tuần
      3. Phải chạy tests trên tất cả các bản build, và một bản build chỉ được chấp nhận khi đã pass qua tất cả các test
      4. Release Cycle của XP:



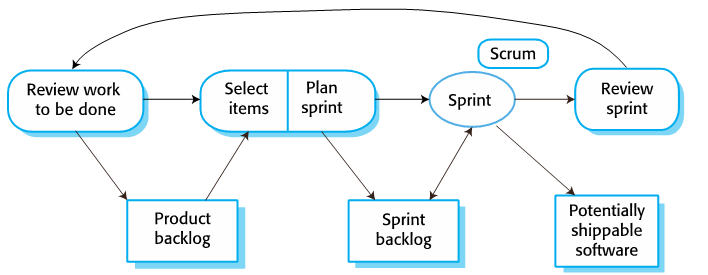
* + 1. Key practices:
       1. User stories for specification
       2. Refactoring
       3. Test-first development
       4. Pair programming

1. **Agile Project Management** 
   1. Trách nhiệm của một project manager trong một dự án Agile:
      1. Trách nhiệm của người quản lý dự án là phải quản lý để phần mềm được giao đúng thời hạn và trong ngân sách mà dự án cho phép
      2. Quản lý một dự án Agile yêu cầu 1 cách tiếp cận khác, điều chỉnh để phù hợp với sự phát triển gia tăng và các tuân theo các agile methods
   2. Scrum:
      1. Là một Agile method tập trung vào việc quản lý việc phát triển ứng dụng lặp đi lặp lại.
      2. Có 3 giai đoạn trọng Scrum:
         1. Bước khởi tạo là một giai đoạn lập kế hoạch phác thảo, thiết lập các mục tiêu chung cho dự án và thiết kế kiến trúc phần mềm
         2. Tiếp theo là một chu kỳ các sprint, nơi mà mỗi một chu kỳ sẽ là một increment của hệ thống
         3. Giai đoạn kết thúc dự án là giai đoạn mà hoàn thành các tài liệu cần thiết, các hướng dẫn sử dụng và đánh giá cho dự án
      3. Các thuật ngữ trong Scrum:



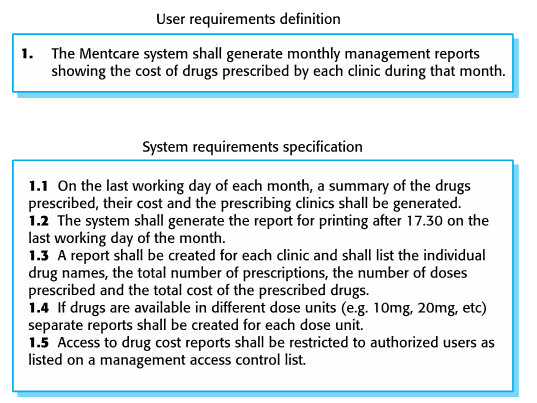


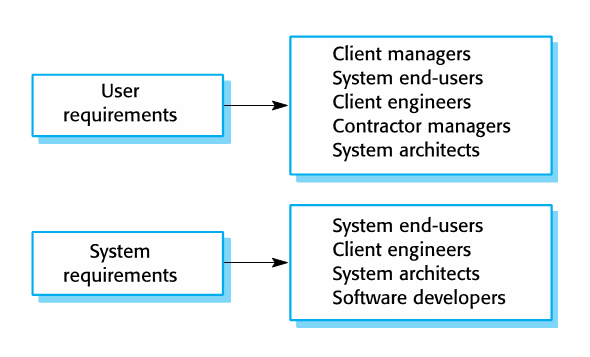
* + 1. Scrum sprint cycle:



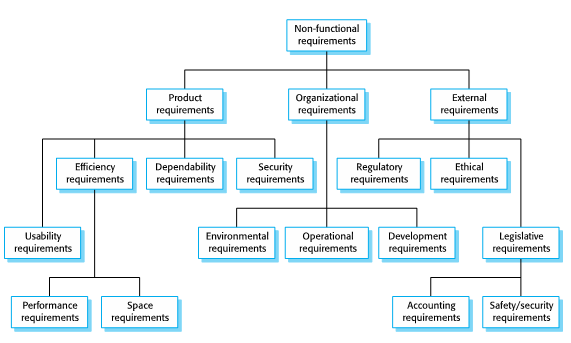
* + - 1. Sprint thường fixed length, 2 - 4 tuần
      2. Điểm bắt đầu của planning là product backlog, là list các công việc sẽ phải làm trong project
      3. Selection phase sẽ bao gồm tất cả project team làm việc với customer để lựa ra các features và chức năng từ những product backlog để thực hiện trong sprint.
    1. Lợi ích của Scrum:
       1. Sản phẩm được chia nhỏ thành các phần dễ hiểu và dễ quản lý
       2. Các chức năng không ổn định sẽ không làm process bị đóng băng
       3. Cả team sẽ có cái nhìn về cả ứng dụng, giao tiếp trong nhóm được cải thiện
       4. Customers sẽ luôn nắm bắt được tiến độ sản phẩm, và devs team sẽ luôn có được các feedback cần thiết từ phía khách hàng

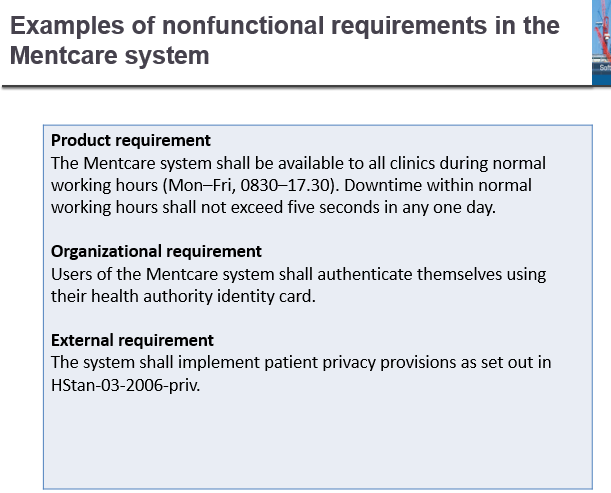
1. **Requirements Engineering:**
   1. Định nghĩa:
      1. Là một process để phân tích yêu cầu một cách có hệ thống
      2. System requirement (SR) là các miêu tả của system service và các ràng buộc được tạo ra trong quá trình thiết kế các yêu cầu.
   2. Requirement?
      1. Có thể từ các high-level abstract statements của một service hoặc các ràng buộc của hệ thống, hoặc là các yêu cầu chi tiết của chức năng
   3. Các loại requirements?



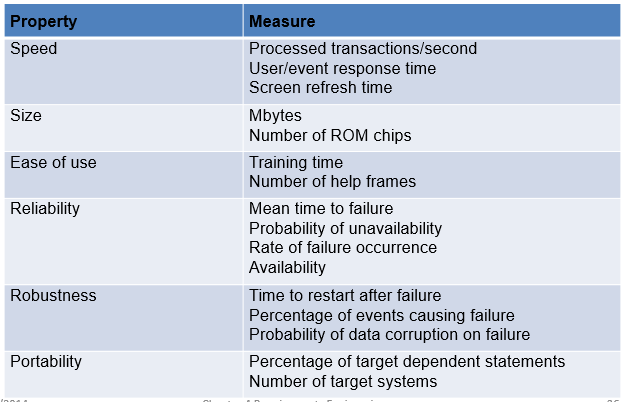


* + 1. User requirements:
       1. Statements được viết theo ngôn ngữ dễ hiểu, tự nhiên kết hợp với các diagrams của các services cung cấp các ràng buộc cho các service này có thể hoạt động, được viết cho khách hàng
    2. System Requirements:
       1. Một tài liệu có cấu trúc
       2. Mô tả chi tiết các chức năng các services và các ràng buộc vận hành
       3. Xác định rõ những gì nên được thực hiện để có thể là một phần của hợp đồng giữa khách hàng và nhà thầu
  1. System stakeholders:
     1. Bất cứ ai hoặc tổ chức nào bị ảnh bưởng với hệ thống theo 1 cách nào đó.
     2. Các loại stakeholder:
        1. End-users
        2. System managers
        3. System owners
        4. External Stakeholder.
  2. Agile methods & requirements:
     1. Rất nhiều các agile methods cho rằng việc tạo ra các System requirements details là tốn thời gian vì requirements thay đổi rất nhanh
     2. Requirements document luôn luôn ở tình trạng “hết hạn”
     3. Agile methods thường sử dụng incremental requirements engineering hay còn được gọi là User stories
  3. Functional and non-functional requirements
     1. Functional requirements:
        1. Định nghĩa:
           1. Là những statements của các services trong hệ thống cần phải cung cấp, cách mà hệ thống sẽ phản hồi lại khi có những đầu vào nhất định, cách mà hệ thống sẽ hoạt động trong các tình huống cụ thể
           2. Cũng có thể là những statements về những thứ mà system không được làm
           3. Tùy thuộc vào loại phần mềm, người dùng nhắm đến và loại của hệ thống nơi mà phần mềm được sử dụng
           4. Functional user requirements có thể ở dạng high-level statements, show ra những gì mà hệ thống nên làm
           5. Functional system requirements sẽ miêu tả các system services ở mức độ chi tiết.
        2. Yêu cầu không chính xác:
           1. Các vấn đề sẽ xảy ra nếu như yêu cầu không chính xác
           2. Nếu yêu cầu không đủ chi tiết, nó sẽ làm người dùng cũng như các lập trình viên hiểu theo những cách khác nhau
     2. Non-functional requirements:





* + - 1. Định nghĩa:
         1. Là những ràng buộc đối với các services hoặc chức năng được cung cấp bởi hệ thống như ràng buộc về thời gian phản hồi, quy trình phát triển, các tiêu chuẩn
         2. Thường được áp dụng cho toàn bộ hệ thống hơn là áp dụng cho các tính năng hay dịch vụ riêng lẻ
         3. Các yêu cầu về process cũng có thể được chỉ định, như IDE phải sử dụng là gì, ngôn ngữ lập trình là gì …
         4. Các Non-func có thể quan trọng hơn các FR vì nếu những yêu cầu này không được đáp ứng, hệ thống có thể trở nên không thể sử dụng được
      2. Các đơn vị hay các cách để đo lường non-func requirements



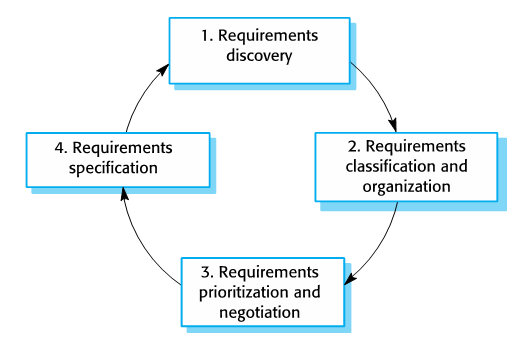
* + 1. Domain requirements
       1. Ràng buộc của hệ thống đến từ domain of operation
  1. Requirements engineering processes (REP)
     1. Process được dùng cho RE rất khác nhau, tùy thuộc vào application domain, những người kiên quan và tổ chức phát triển các requirements
     2. Tuy nhiên tựu chung lại chúng ta có 4 activities thường thấy ở tất cả các REP:
        1. Requirements elicitation (gợi ý các requirements), hay còn được gọi là requirements discovery (khám phá các requirements)
           1. Ở bước này, chúng ta sẽ cần các nhân viên kỹ thuật làm việc với khách hàng để tìm hiểu về application domain, các services mà hệ thống cần phải cung cấp, các hạn chế hay ràng buộc của hệ thống
           2. Có thể có sự tham gia của End-Users (EU), managers, engineers … Hay gọi chung là các stakeholder
           3. Bao gồm các giai đoạn:

Requirements discovery: bàn luận với các stakeholders để tìm hiểu các yêu cầu của họ

Requirements classification + organization: nhóm và phân loại các yêu cầu liên quan thành các nhóm rõ ràng

Requirements prioritization + negotiation (đàm phán): tìm các yêu cầu cần được ưu tiên và giải quyết các xung đột giữa các yêu cầu

Requirements specification: các yêu cầu được tài liệu hóa và được chuyển giao đến giai đoạn tiếp theo của mô hình spiral



* + - * 1. Các vấn đề của requirements elicitation:

Các stakeholder có thể không nắm rõ được cái mà họ thực sự cần

Các stakeholder thể hiện các yêu cầu bằng các cách của riêng họ hoặc bị ràng buộc hay chi phối bởi chính họ

Các requirements của các stakeholder có thể bị trùng lặp hoặc mâu thuẫn với nhau

Bị ảnh hưởng bởi các yếu tố liên quan đến tổ chức hay chính trị

Requirements có thể bị thay đổi trong quá trình phân tích

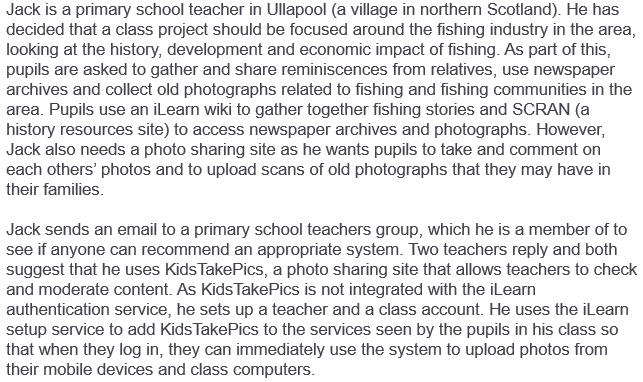
* + - * 1. Stories và scenarios:

Scenarios và user stories là những ví dụ thực tế cách mà chúng ta sử dụng hệ thống

Scenarios và user stories là những miêu tả về cách mà hệ thống được dùng trong những task khác nhau

Vì nó dựa trên những ví dụ thực tế nên các stakeholder có thể hiểu được và đóng góp xây dựng hoặc sửa đổi các story theo requirements mong muốn của họ

Ví dụ:



Scenarios:

Là một cấu trúc của một user story, bao gồm:

Miêu tả điểm bắt đầu của tình huống

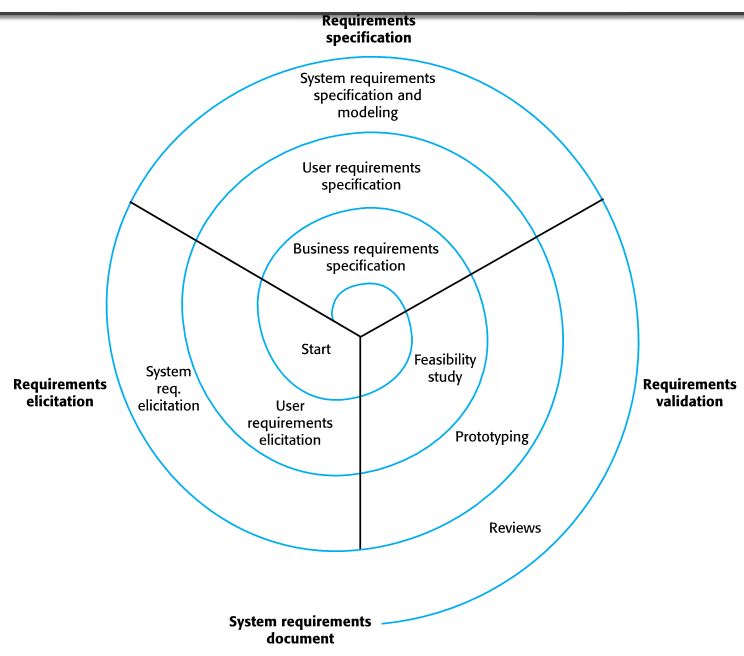
Miêu tả luồng chạy của các sự kiện trong story

Miêu tả những tình huống mà hệ thống có thể chạy sai

Thông tin về các hoạt động khác diễn ra đồng thời

Trạng thái mà khi đó scenarios sẽ kết thúc

* + - 1. Requirements analyst (Phân tích các yêu cầu)
      2. Requirements validation (Thẩm định các yêu cầu)
      3. Requirements management (Quản lý các yêu cầu)



* + 1. Trong thực tế, RE là một iterative activity, do đó các quá trình trên sẽ được thực hiện xen kẽ, theo dạng spiral view như trên
  1. Requirements specification
     1. Là process mà chúng ta sẽ tài liệu hóa các user và system requirements
     2. Các user requirements cần phải dễ hiểu bởi end-users và customers không có kiến thức về chuyên môn
     3. System requirements cần phải chi tiết và có thể bao gồm cả những technical information
        1. Các cách để viết một system requirements specification
           1. Use cases:

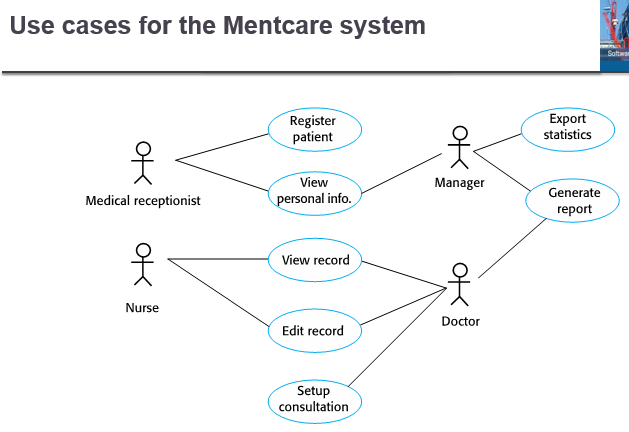
Use-cases có thể coi như là một dạng scenario có trong UML

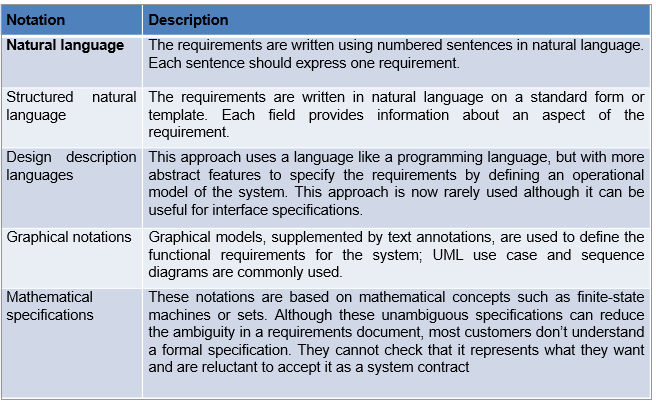
Use-cases sẽ xác định actors trong mỗi một tương tác của hệ thống, và miêu tả các tương tác đó

Một tập hợp các use cases có thể miêu tả tất cả các tương tác của một hệ thống

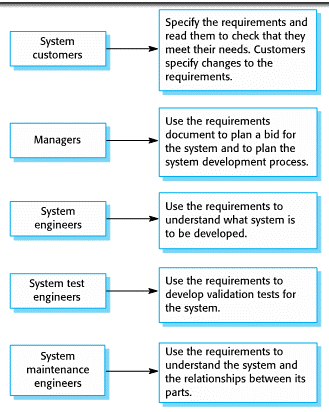
UML sequence diagrams có thể được dùng để bổ sung chi tiết cho các use cases bằng cách thể hiện trình tự xử lý các sự kiện trong hệ thống.

Ví dụ:

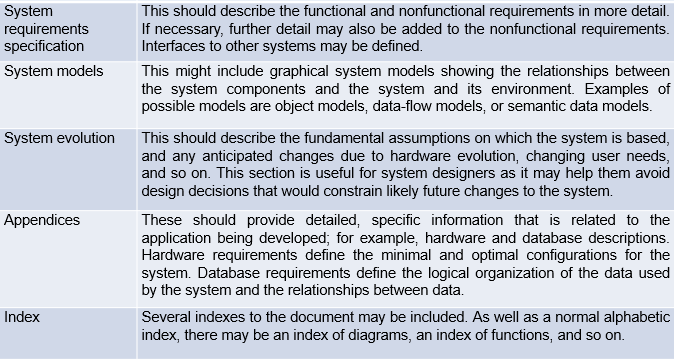




* + 1. Software requirements document:
       1. Là một tài liệu chứa những yêu cầu đối với các developer
       2. Sẽ bao gồm cả user requirements và các đặc tả của system requirements
       3. Đây KHÔNG PHẢI là design document, nó sẽ miêu tả những gì hệ thống phải làm hoặc nên làm, chứ không phải cách mà hệ thống làm việc
       4. Những người sẽ sử dụng Software requirements document:

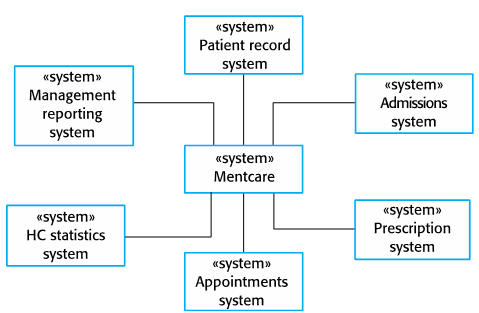


* + - 1. Cấu trúc của một requirements document
         1. Preface: define ra những người sẽ sử dụng tài liệu này, phiên bản của tài liệu, các thay đổi của tài liệu
         2. Introduce: giới thiệu về sự cần thiết của hệ thống sử dụng SRS này, miêu tả ngắn gọn về các chức năng, miêu tả cách mà hệ thống hoạt động trong bối cảnh chung của tổ chức
         3. Glossary (chú giải, chú thích): define các thuật ngữ sẽ được dùng trong SRS
         4. User requirements definition: miêu tả các services được cung cấp cho người dùng, các non-functional system requirements nên được miêu tả trong này. Nên được viết bằng ngôn ngữ tự nhiên, dễ hiểu cho cả người dùng và khách hàng.
         5. System architecture: tổng quan về về cấu trúc của hệ thống, cách mà các function hoặc services được phân phối trong ứng dụng. Các components được tái sử dụng nhiều nên được highlighted
         6. System requirements specification: Function + non-function requirements ở mức độ chi tiết hơn.
         7. System Model: chứa các biểu đồ hay diagrams models cho thấy các mối liên hệ giữa các components trong hệ thống cũng như môi trường
         8. System Evolution: miêu tả các tóm tắt cơ bản về based của hệ thống, các thay đổi có thể dự đoán trước trong quá trình phát triển của phần mềm về sau này, các thay đổi mà người dùng hay khách hàng có thể mong muốn …
         9. Phụ lục: cung cấp các thông tin chi tiết liên quan đến hệ thống được phát triển như phần cứng để chạy ứng dụng, database của ứng dụng, yêu cầu phần cứng để chạy ...

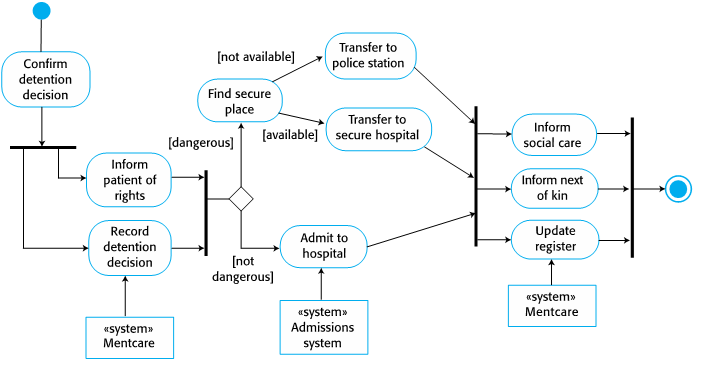


* + 1. Requirements and design
       1. Thông thường, requirements cần phải miêu tả rõ những gì mà hệ thống phải làm, còn thiết kế sẽ miêu tả cách mà hệ thống thực hiện những requirements này
       2. Trong thực tế, requirements và design là không thể tách rời
  1. Requirement validation
     1. Đây là bước để giải quyết những khúc mắc về việc các requirements có thực sự giúp xác định được những gì mà customers thực sự muốn hệ thống hay không
     2. Việc sai requirements sẽ tốn rất nhiều chi phí để khắc phục, do đó validation rất quan trọng
     3. Requirements checking:
        1. Validity (hiệu quả): Hệ thống có cung cấp những functions hỗ trợ tốt nhất cho các nhu cầu của khách hàng không?
        2. Consistency (nhất quán): các requirements có bị conflict với nhau không?
        3. Completeness (hoàn thiện): tất cả các chức năng mà khách hàng mong muốn đã được thể hiện qua các requirements đầy đủ chưa?
        4. Realism (tính thực tế): các requirements có thể thực hiện được với budget và công nghệ hiện tại không?
        5. Verifiability (khả năng xác minh): các requirements có thể kiểm tra được không?
  2. Requirements change
     1. Business hoặc technical environment của hệ thống luôn luôn thay đổi
     2. Người trả tiền cho hệ thống và người sử dụng hệ thống hiếm khi là 1
     3. Các hệ thống lớn thường có một cộng đồng người dùng lớn, do đó các người dùng có thể có các requirements khác nhau và có độ ưu tiên khác nhau, gây mâu thuẫn với những người dùng khác.
     4. Requirements management:
        1. Là một process quản lý sự thay đổi của các requirements trong quá trình REP và quá trình phát triển hệ thống

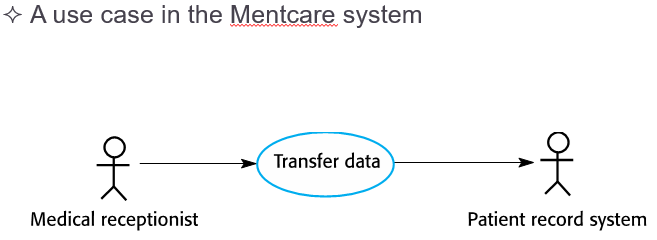
1. **System modeling (Mô hình hóa hệ thống)**
   1. Định nghĩa:
      1. Là một Process của việc trừu tượng hóa các models của một hệ thống, với mỗi models sẽ thể hiện một cách nhìn hoặc quan điểm khác nhau về hệ thống đó
      2. Hiện nay SM thường được thể hiện bằng cách sử dụng các loại đồ họa, hầu như là sử dụng UML (Unified Modeling Language)
      3. SM giúp đánh giá, phân tích và hiểu được các chức năng của hệ thống và đồng thời giúp việc giao tiếp với customers dễ dàng hơn
   2. Các góc nhìn đối với hệ thống:
      1. Góc nhìn từ bên ngoài: mô hình hóa bối cảnh hoặc môi trường của hệ thống
      2. Góc nhìn về sự tương tác: mô hình dựa trên tương tác giữa hệ thống và môi trường bên ngoài, hoặc giữa các thành phần của hệ thống
      3. Góc nhìn cấu trúc: mô hình hóa tổ chức của một hệ thống hoặc cấu trúc của dữ liệu được hệ thống xử lý
      4. Góc nhìn hành vi: mô hình hóa hành vi của hệ thống qua cách nó xử lý các sự kiện
   3. UML diagram types:
      1. Activity diagrams: thể hiện các hoạt động liên quan trong một process hoặc quá trình xử lý dữ liệu
      2. Use case diagrams: thể hiện các tương tác của hệ thống và môi trường
      3. Sequence diagrams: thể hiện tương tác giữa các actors và hệ thống và giữa các component trong hệ thống
      4. Class diagrams: thể hiện các classes có trong hệ thống, mối liên kết giữa các class này
      5. State diagrams: thể hiện cách mà hệ thống tương tác với các sự kiện bên trong và bên ngoài hệ thống
2. **Models:**
   1. Context Models:
      1. Minh họa các bối cảnh hoạt động của một hệ thống (Context), chỉ ra những gì nằm ngoài ranh giới của hệ thống
      2. Các vấn đề về xã hội hoặc tổ chức có thể ảnh hưởng đến ranh giới này
      3. Architectural models thể hiện hiện các mối quan hệ của hệ thống với các hệ thống khác.
      4. System Boundaries (ranh giới của hệ thống):
         1. Thể hiện những gì bên trong hoặc bên ngoài hệ thống, chỉ ra các hệ thống khác phụ thuộc hoặc liên kết với hệ thống được phát triển
         2. Vị trí của hệ thống có ảnh hưởng đến các yêu cầu của hệ thống
         3. Xác định ranh giới của hệ thống có thể liên quan đến vai trò, mức độ quan trọng, các chức năng … của hệ thống.

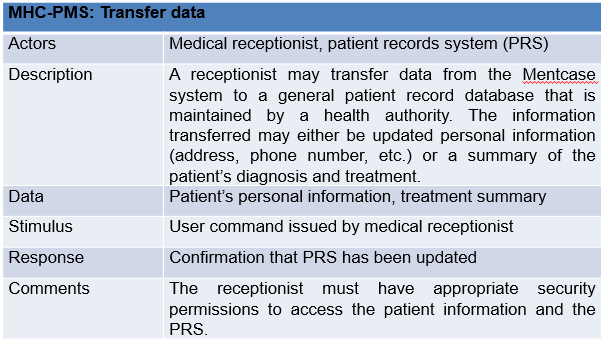


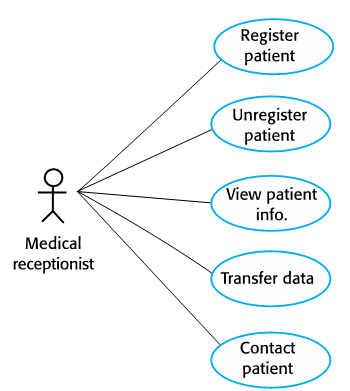
* + 1. Process perspective:
       1. Context models sẽ show ra các hệ thống khác trong môi trường, không show ra cách mà hệ thống đang được phát triển hoạt động thế nào
       2. Được dùng để cho thấy vai trò của hệ thống đang được phát triển trong tổng hệ các hệ thống đã có hoặc sắp được phát triển
       3. **UML activity diagrams** có thể được dùng để xác định business process models, ví dụ như dưới đây tương ứng với models ở trên, ta có thể dùng **activity diagrams** để diễn tả hệ thống:



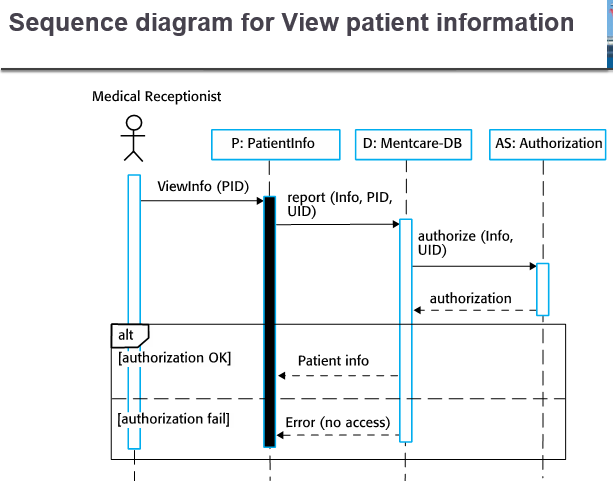
* 1. Interaction models
     1. Định nghĩa:
        1. Mô hình hóa các tương tác rất quan trọng trong việc giúp xác định user requirements
        2. Mô hình hóa sự tương tác hệ thống với hệ thống giúp làm nổi bật các sự liên kết có thể có
        3. Mô hình hóa sự tương tác giữa các components giúp chúng ta xác định xem cấu trúc hệ thống được đề xuất có khả năng mang lại hiệu suất và độ tin cậy như mong muốn hay không.
        4. **Use case diagrams** và **sequence diagrams** có thể được sử dụng để mô hình hóa các tương tác này
     2. Use case modeling:
        1. Được dùng để support cho requirements elicitation và bây giờ được hợp nhất với UML
        2. Mỗi một use case sẽ diễn tả một tương tác từ bên ngoài với hệ thống
        3. Actors có thể là người dùng của hệ thống, hoặc đại diện cho 1 hệ thống khác
        4. Cung cấp tổng quan và sẽ được biểu diễn chi tiết hơn dưới dạng bảng (tabular)

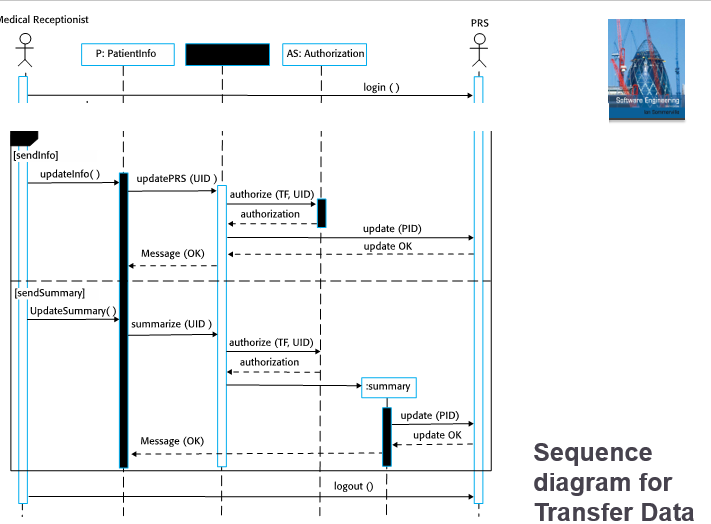




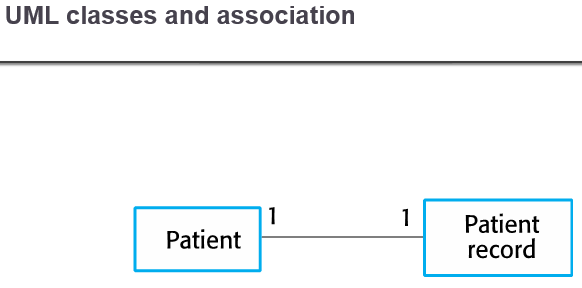


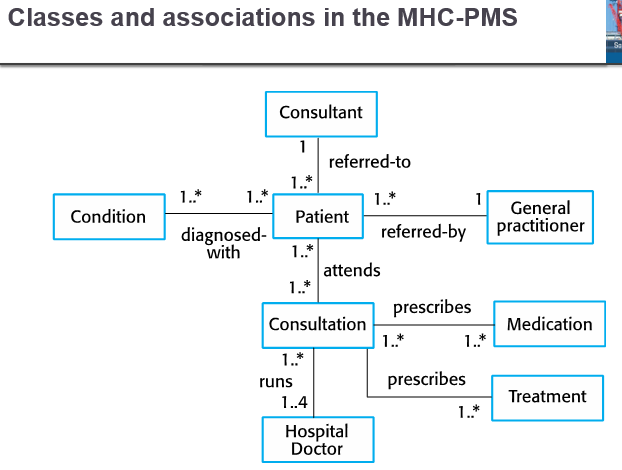
* + 1. Sequence diagrams:
       1. Là một phần của UML dùng để mô hình hóa tương tác giữa các actors và các object bên trong một hệ thống
       2. Thể hiện trình tự của các tương tác diễn ra trong use case
       3. Tương tác giữa các đối tượng được biểu thị bằng các mũi tên có chú thích

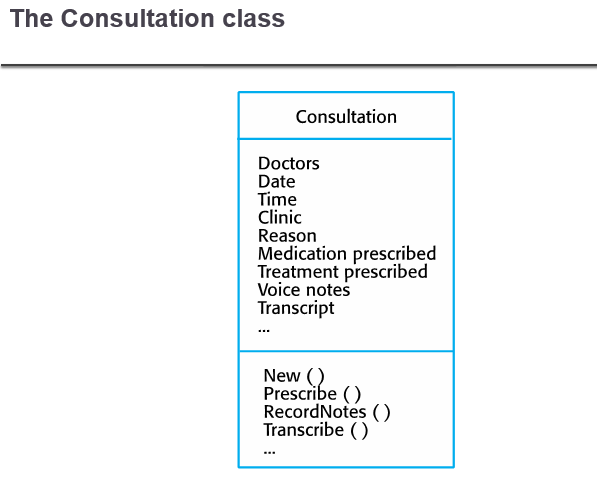




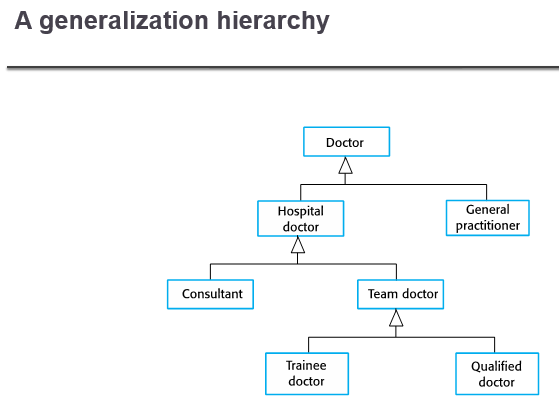
* 1. Structural Models:
     1. Định nghĩa:
        1. Hiển thị hệ thống thành một tổ chức các thành phần cấu thành nên nó và mối quan hệ giữa chúng
        2. Có thể biểu diễn cấu trúc của hệ thống (tĩnh) hoặc tổ chức của hệ thống khi nó đang hoạt động (động)
        3. Thường được tạo ra trong quá trúc thảo luận và thiết kế kiến trúc hệ thống
        4. Sử dụng class diagrams
     2. Class diagrams:
        1. Thường được dùng trong các hệ thống OOP để thể hiện các class trong hệ thống và các mối liên hệ giữa chúng
        2. Một object class có thể được coi như một đối tượng của hệ thống
        3. Các mối liên kết biểu thị quan hệ giữa các class này
        4. Trong giai đoạn ban đầu của việc thiết kế các models, các objects thường diễn tả những thứ “real world” như bệnh nhân, học sinh, quản lý …

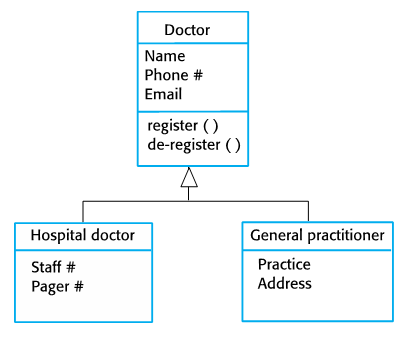




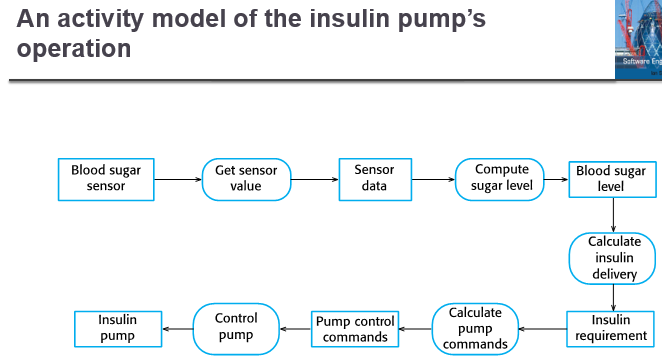


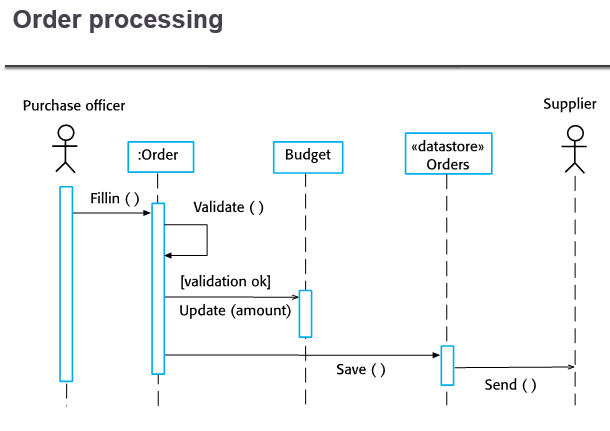
* + 1. Generalization (sự tổng quát hóa):
       1. Là một kỹ năng mà chúng ta dùng để quản lý những thứ phức tạp
       2. Thay vì tìm hiểu về chi tiết các đặc tính của những entities trong ứng dụng, chúng ta đặt các entities này vào các lớp tổng quan hơn, và tìm hiểu các đặc điểm của các lớp này.
       3. Cho phép chúng ta suy từ các class này sang class khác có cùng đặc điểm chung
       4. Sử dụng tính kế thừa để tổng quát hóa, giúp tái sử dụng trong việc diễn tả các class.



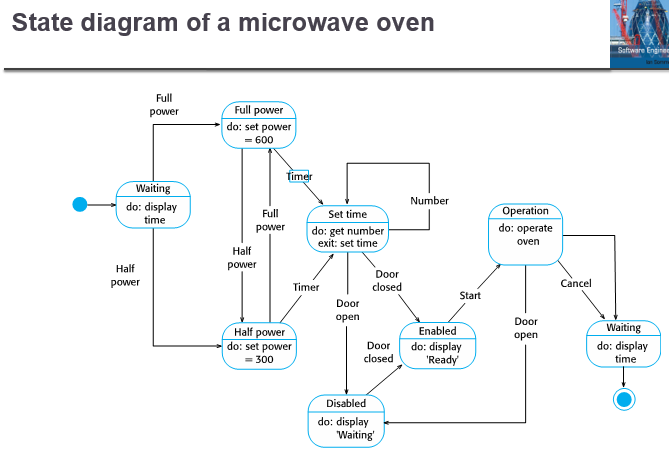


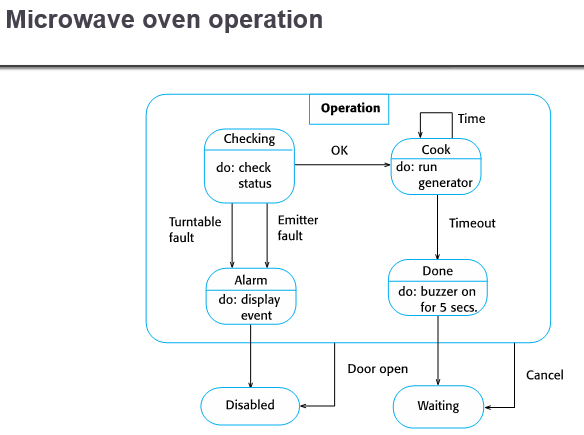
* 1. Behavioral models (Mô hình hóa hành vi)
     1. Định nghĩa:
        1. Là một models miêu tả hành vi của hệ thống khi nó đang thực thi, cho thấy những gì xảy ra khi một hệ thống phản hồi lại sự kiện từ môi trường
        2. Sự kiện có thể là:
           1. Data: dữ liệu được đưa đến và được xử lý bởi hệ thống
           2. Events: các sự kiện diễn ra và trigger hệ thống
     2. Data-driven modeling:
        1. Các hệ thống kinh doanh là các hệ thống xử lý dữ liệu, các hệ thống này kiểm soát dữ liệu đầu vào và xử lý và đưa ra dữ liệu đầu ra tương ứng
        2. Models dạng này sẽ cho thấy các luồng xử lý dữ liệu và các hành động bên trong hệ thống lúc làm việc với dữ liệu.
        3. Đặc biệt hữu ích trong việc phân tích các requirements



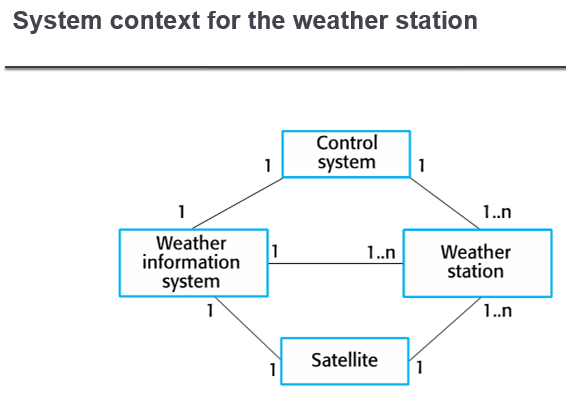


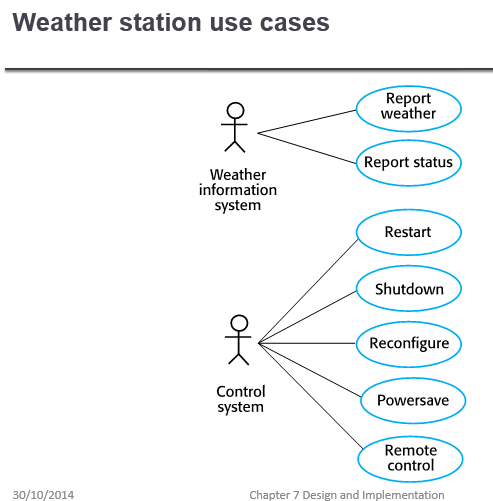
* + 1. Event-driven modeling (mô hình hướng sự kiện):
       1. Các hệ thống real-time thường là theo dạng hướng sự kiện, với rất ít luồng xử lý dữ liệu
       2. Event-driven modeling sẽ miêu tả các hệ thống response với các sự kiện bên trong và bên ngoài
       3. Giả định là một hệ thống có một số trạng thái hữu hạn các sự kiện có thể gây ra sự thay đổi trạng thái của hệ thống
    2. State machine models:
       1. Cũng là model behaviour của hệ thống response các sự kiện bên trong và bên ngoài hệ thống
       2. Thể hiện hệ thống dưới dạng các **nodes** và các events như là các **arc (vòng cung?)** giữa các nodes đó. Khi có các sự kiện, hệ thống sẽ di chuyển từ trạng thái này sang trạng thái khác
       3. **State chart** là một phần của UML và hay được dùng để mô hình hóa state machine

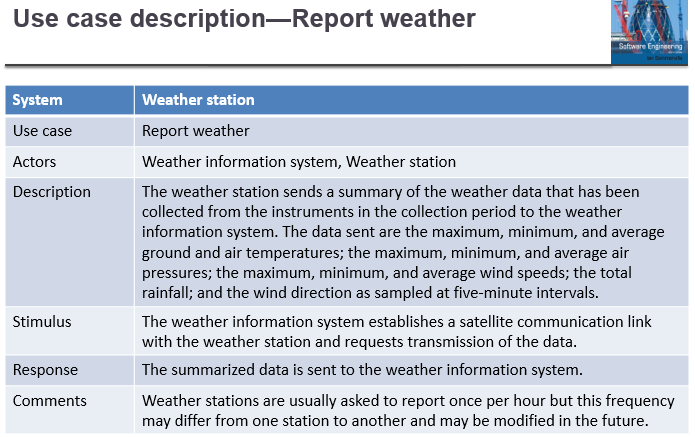




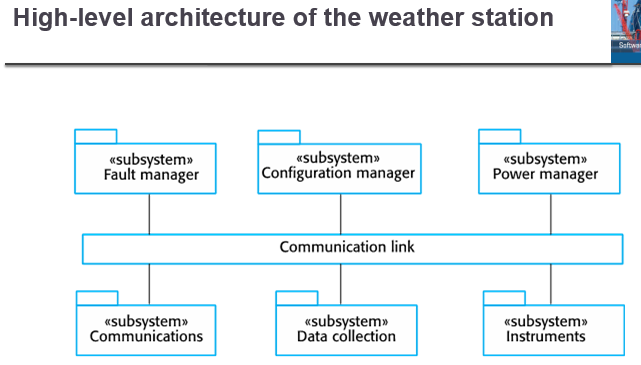
1. **Model-driven engineering (Kỹ thuật hướng mô hình - MDE):**
   1. Định nghĩa:
      1. Là một cách tiếp cận để phát triển phần mềm mà mô hình chứ không phải là kết quả là đầu ra của process.
      2. Ứng dụng sẽ được chạy trên một nền tảng phần cứng / phần mềm được tự động generate từ models
   2. Usage of MDE:
      1. Vẫn còn ở giai đoạn phát triển và chưa rõ mức độ ảnh hưởng đến SE
      2. Ưu điểm:
         1. Cho phép hệ thống có thể được trừu tượng ở mức độ cao
         2. Việc code được tự động generated ⇒ giảm chi phí khi chuyển đổi nền tảng
      3. Nhược điểm:
         1. Thiếu khả năng custom cần thiết
         2. Không phải lúc nào mô hình quá trừu tượng cũng là cần thiết
         3. Tiết kiệm từ việc code được generated 1 cách tự động có thể không bằng được chi phí để phát triển trình dịch cho các nền tảng mới
2. **Design & Implementation (Thiết kế và thực hiện)**
   1. Định nghĩa:
      1. Là một process để tạo ra phần mềm có thể chạy được.
      2. Việc design và implementation sẽ luôn được thực hiện liên tục cùng với nhau
         1. Design là một hoạt động mà chúng ta sẽ xác định các thành phần của phần mềm và mối liên hệ giữa chúng, dựa trên yêu cầu phần mềm
         2. Implementation là hoạt động hiện thực hóa các thiết kế thành một phần mềm
   2. Object-oriented design sử dụng UML
      1. Định nghĩa:
         1. Quy trình này thiết kế hướng đối tượng thường liên quan đến việc phát triển một số mô hình (models) hệ thống khác nhau
         2. Đối với các mô hệ thống nhỏ, điều này không hiệu quả về mặt chi phí do sẽ tốn rất nhiều effort để làm điều này
         3. Phù hợp với những hệ thống lớn.
      2. Process stages:
         1. Có rất nhiều Object Oriented design process, phụ thuộc vào tổ chức sử dụng process đó.
         2. Các activities thông thường:
            1. Xác định context và các models sử dụng hệ thống
            2. Thiết kế kiến trúc phần mềm
            3. Xác định các đối tượng chính của hệ thống
            4. Phát triển design models
            5. Xác định các interface của các objects
         3. Ví dụ cho một trạm thời tiết

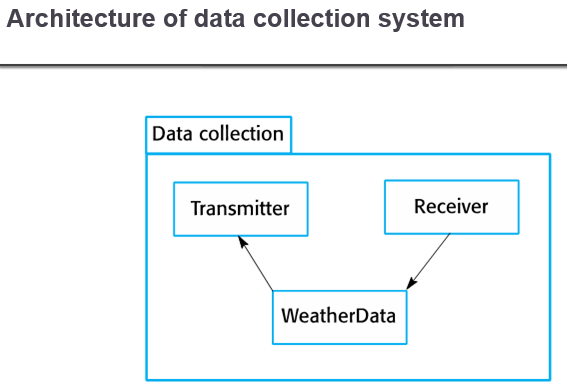




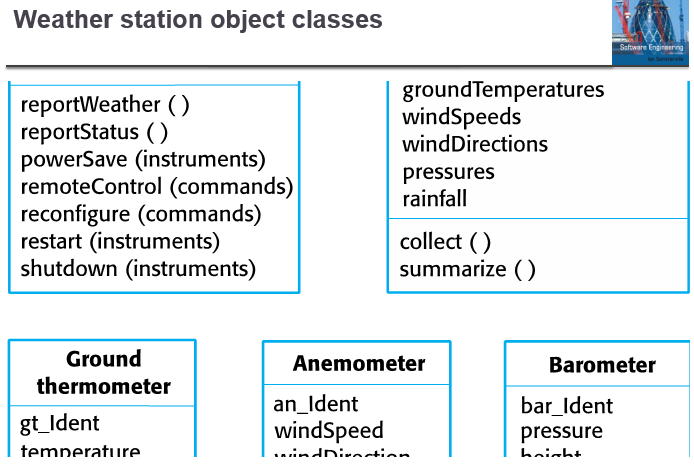


* + 1. System context và các tương tác:
       1. Hiểu được quan hệ giữa các software sắp được design và môi trường hoạt động của nó là điều cơ bản để quyết định các chức năng hệ thống cần thiết và cấu trúc hệ thống để ứng dụng có thể làm việc được với môi trường xung quanh
       2. Giúp xác định ranh giới của hệ thống và các chức năng cần có để nó có thể làm việc được với các hệ thống khác.
       3. System context model là một mô hình cấu trúc thể hiện các hệ thống khác có cùng môi trường với hệ thống đang được phát triển
       4. Interaction models là một mô hình động show ra các action của hệ thống đang được phát triển tương tác với các hệ thống khác khi nó đang chạy hoặc được sử dụng
    2. Architectural design (thiết kế kiến trúc)
       1. Khi mà sự tương tác giữa hệ thống và môi trường đã tường minh, chúng ta sẽ dùng những thông tin này để thiết kế kiến trúc hệ thống
       2. Xác định các components chính (major) tạo nên bộ khung cho hệ thống và các tương tác, sau đó tổ chức lại các components bằng các pattern như mô hình layered hoặc client - server model
       3. Ứng dụng trạm thời tiết ở trên bao gồm các hệ thống con độc lập, giao tiếp bằng các phát tin nhắn trên một cơ sở hạ tầng chung:





* + 1. Object class identification:
       1. Là một phần khó trong OO design
       2. Phụ thuộc nhiều vào kỹ năng, kinh nghiệm và hiểu biết của designer
       3. Cũng là 1 iterative process, thường sẽ không đúng ngay từ lần đầu.
       4. Phương pháp tiếp cận:
          1. Dựa vào description của hệ thống, sử dụng ngôn ngữ tự nhiên
          2. Dựa vào những thứ hữu hình trong miền ứng dụng
          3. Sử dụng bebaviour approach và xác định đối tượng, thuộc tính dựa trên những gì tham gia vào hành vi nào
          4. Sử dụng phân tích tình huống
       5. Ví dụ dựa vào weather station system dựa trên **phần cứng** và **dữ liệu hữu hình** trong hệ thống:
          1. Nhiệt kế mặt đất, máy đo gió, khí áp kế …
          2. Trạm thời tiết
          3. Dữ liệu thời tiết

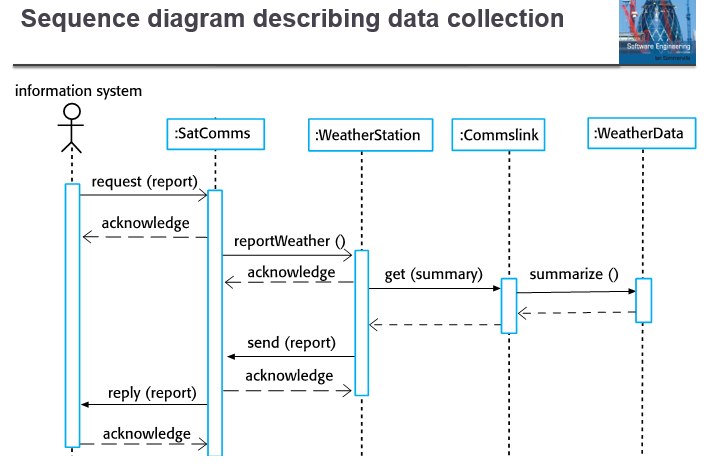


* + 1. Design model:
       1. Định nghĩa:
          1. Design models thể hiện các objects và các object classes và quan hệ giữa các entities này
          2. Có 2 dạng design models

Structural models: miêu tả tĩnh các cấu trúc của hệ thống dưới dạng các lớp đối tượng và các mối quan hệ

Dynamic models: miêu tả tương tác giữa các object

* + - 1. Ví dụ các design models:
         1. Subsystem models thể hiện các logical groupings của các objects thành các hệ thống con nhất quán
         2. Sequence models thể hiện trình tự của bác objects khi có tương tác

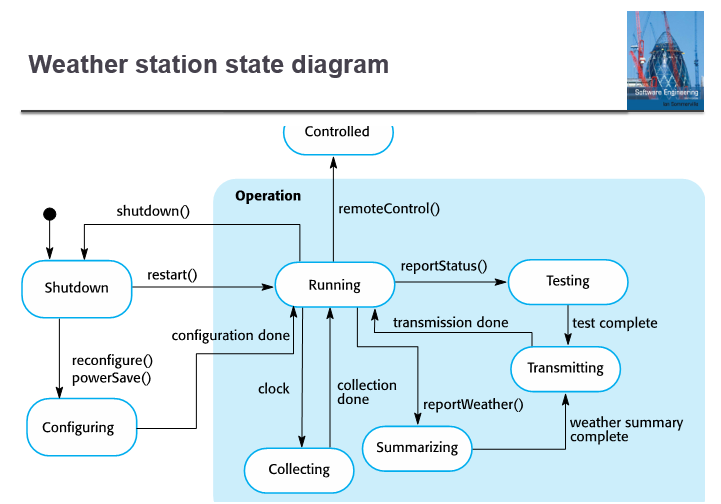


* + - * 1. State machine models thể hiện sự thay đổi của các objects lúc phản ứng lại với các events

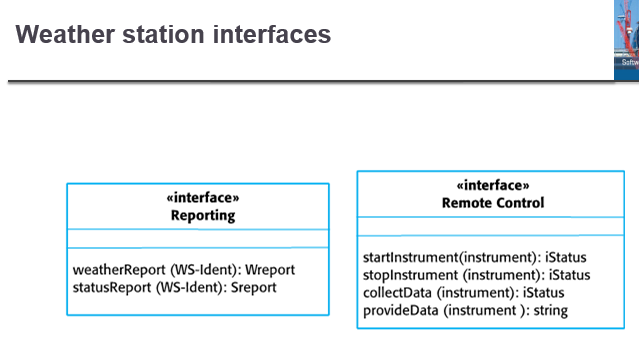
Các events có thể là các requests và sự chuyển đổi trạng thái được trigger bởi các requests này

State diagrams rất có ích ở high-level models của hệ thống hoặc trong miêu tả các behaviour trong quá trình run time

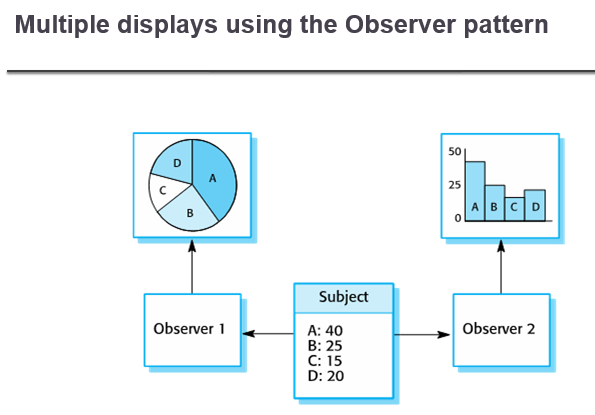
Không phải lúc nào state diagrams cũng cần thiết

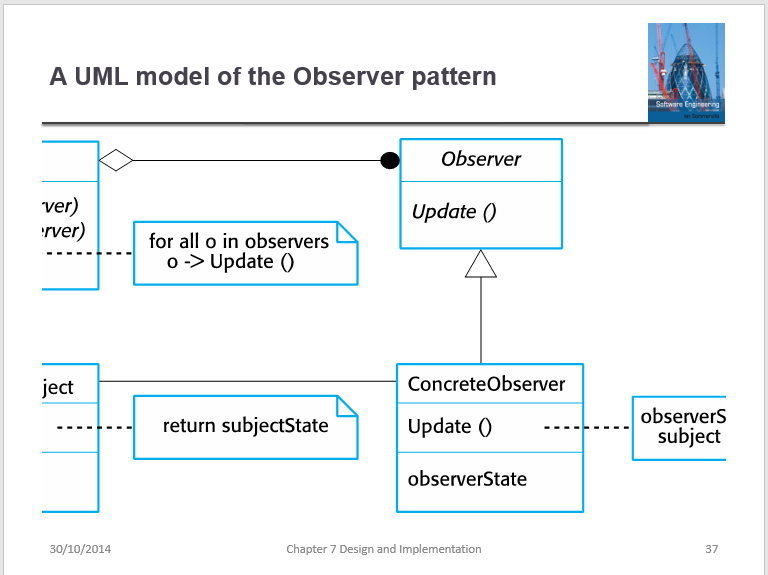


* + - * 1. Các models khác như use-case models, mô hình tổng hợp, mô hình tổng quát hóa …
      1. Interface specification:
         1. Object interfaces cần được xác định để các object và các components khác có thể được thiết kế tương tự
         2. Class diagrams có thể được sử dụng trong trường hợp này



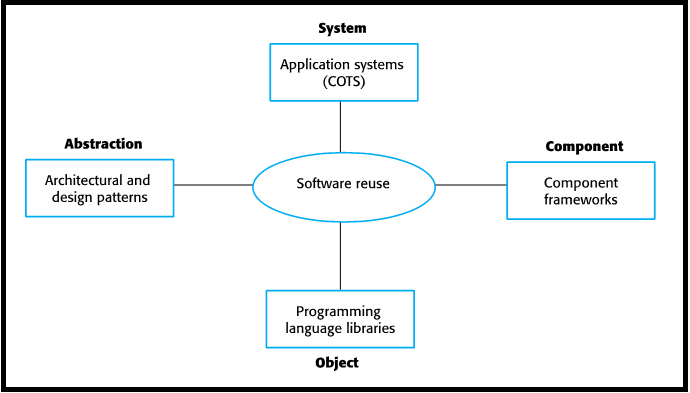
1. **Design pattern:**
   1. Định nghĩa:
      1. Là một cách chúng ta sẽ tái sử dụng các kiến thức về các vấn đề và các giải pháp đã được trừu tượng hóa
      2. Miêu tả về một vấn đề, và bản chất của vấn đề đó
      3. Cần được trừu tượng để tái sử dụng trong các trường hợp khác
      4. Thường sử dụng có đặc điểm như thừa kế và đa hình để có thể tái sử dụng
      5. Pattern hay Pattern language là cách để chúng ta miêu tả một best practice, một design tốt, hoặc một kinh nghiệm theo cách mà những người khác, hay những dự án khác có thể sử dụng lại pattern này.
      6. “*Pattern hay Pattern language là cách để mổ tả các phương pháp hay nhất, các thiết kế tốt nhất, có thể coi đây như là việc chúng ta tái sử dụng lại kinh nghiệm của một người khác đối với 1 vấn đề tương tự đã được giải quyết*”
   2. Các elements của một pattern:
      1. Tên: phải có ý nghĩa trong việc xác định pattern
      2. Miêu tả về vấn đề.
      3. Miêu tả về giải pháp: không mang tính cụ thể mà là một khuôn mẫu để design giải pháp có thể được tạo ra theo nhiều cách khác nhau để giải quyết vấn đề đang gặp phải.
      4. Kết quả: kết quả và các đánh đổi của việc sử dụng mô hình
   3. The Observer pattern (ví dụ về một pattern):
      1. Name: Observer
      2. Description: chia tách việc hiển thị trạng thái của Object khỏi chính Object đó. Khi state của đối tượng thay đổi, tất cả các hiển thị sẽ được tự động cập nhật để phản ánh sự thay đổi đó
      3. Problem description: được sử dụng khi mà có nhiều hình thái của state cần được theo dõi hoặc cần sử dụng. Mẫu này đặc biệt thích hợp khi đối tượng cần theo dõi cần được theo dõi ở nhiều hình thái hiển thị, và cần những thông tin này thay đổi khi mà đối tượng cần theo dõi cũng có sự thay đổi.
      4. Solution Description:



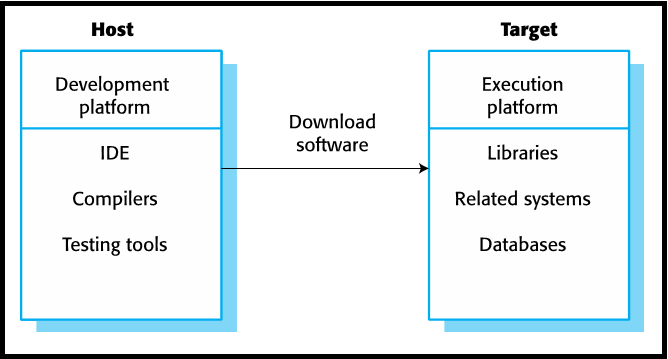


* + 1. Kết quả: tối ưu hóa để nâng cao hiệu suất hiển thị là không thực tế
  1. Design problems (của observer pattern):
     1. Để sử dụng pattern trong thiết kế, chúng ta phải biết rằng bất cứ design problem nào mà chúng ta phải đổi mặt đều có một pattern liên quan để có thể áp dụng
        1. Thông báo với các object rằng trạng thái của các object đã có sự thay đổi (Observer Pattern)
        2. Dọn dẹp các giao diện cho một số object liên quan phát triển tăng dần (Facade pattern)
        3. Cung cấp 1 cách chuẩn để truy cập các elements trong tập hợp (Iterative pattern)
        4. Cho phép khả năng mở rộng chức năng của một lớp tại run-time (Decorator pattern)

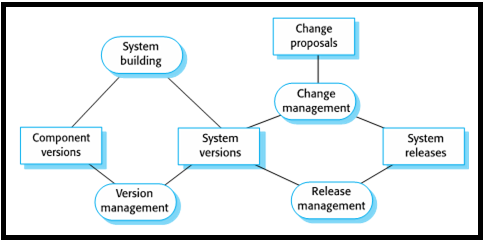
1. **Các vấn đề gặp phải của implementation:**
   1. Trọng tâm ở đây không phải là việc code (programming), mặc dù đây hiển nhiên là một bước quan trọng, nhưng những vấn đề trong việc implementation đôi khi lại không nằm trong việc lập trình:
      1. Reuse: đa số các Software đều được xây dựng bởi việc reuse lại components hoặc systems. Khi chúng ta developing software, chúng ta cần tái sử dụng càng nhiều càng tốt. Chúng ta có các cấp độ về việc reuse như sau:
         1. Abstraction level: ở cấp độ này, chúng ta không reuse software trực tiếp, mà sử dụng những kiến thức về các abstraction đã được kiểm chứng vào việc design software của mình
         2. Object level: sử dụng lại các objects từ các thư viện
         3. Component level: Components là tập hợp các objects và classes mà chúng ta sẽ tái sử dụng trong hệ thống
         4. System level: tái sử dụng lại toàn bộ hệ thống



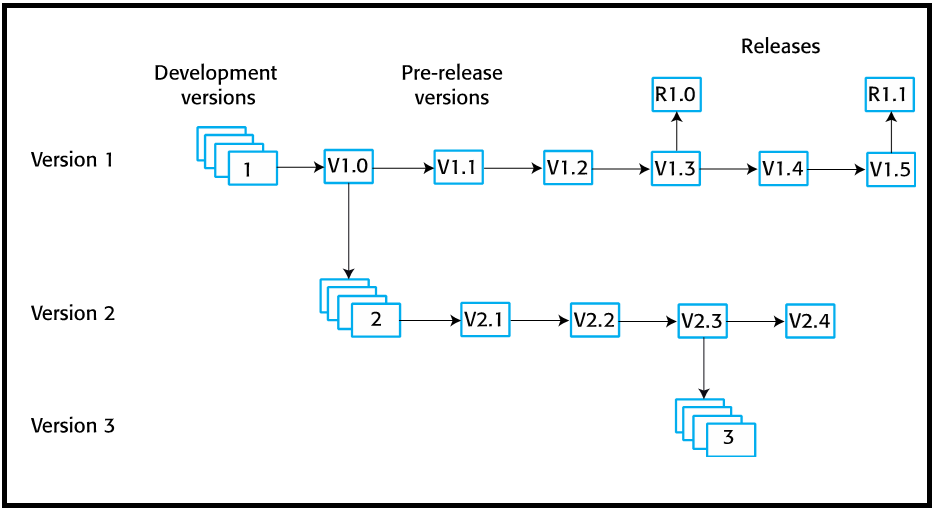
* + 1. **Configuration management (CM)**: trong quá trình developing, cần phải tracking các versions của mỗi software component bằng một Configuration management.
       1. CM là tên được đặt cho một process chung của việc quản lý sự thay đổi của phần mềm.
       2. Mục tiêu của CM là hỗ trợ việc tích hợp hệ thống để tất cả các devs có thể truy cập vào source code, documents một các có kiểm soát, biết được sự thay đổi của các tài nguyên này, thay đổi lúc nào, ai là người thay đổi, thay đổi với mục đích gì …
       3. Biên dịch dịch và liên kết các thành phần để tạo ra hệ thống (Chapter 25)
       4. Các activities:
          1. Version management
          2. System integration: support để các devs xác định được phiên bản của các thành phần để tạo ra phiên bản của hệ thống. Các description này sẽ được dùng để xây dựng nên hệ thống 1 cách tự động bằng các biên dịch và liên kết các thành phần cần thiết
          3. Problem tracking
    2. Host-target development: việc tạo ra phần mềm không phải lúc nào cũng là trên 1 máy tính hay cùng một môi trường, mà thường là chúng ta phát developing trên 1 máy tính và execute ở nhiều máy tính khác nhau



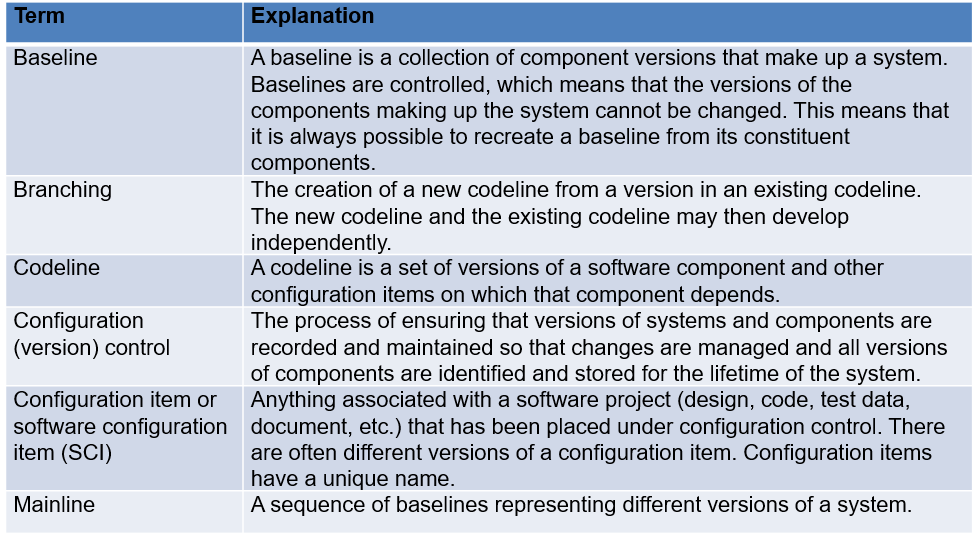
1. **Open Source development**
   1. Định nghĩa:
      1. Là một cách tiếp cận để phát triển phần mềm, trong đó mã nguồn của hệ thống phần mềm là mở và được published để các tình nguyện viên được mời vào để cùng phát triển
      2. Linux là một ví dụ của OSD, một hệ điều hành mã nguồn mở được sử dụng làm server system hay desktop environment, hay Java với Apache web server và hệ cơ sở dữ liệu MySQLation
2. **Configuration Management**
   1. Định nghĩa:
      1. Software system thường được thay đổi trong quá trình phát triển và sử dụng
      2. **Configuration Management (CM)** sẽ quan tâm đến các policies (chính sách), các processes và các công cụ để quản lý sự thay đổi của hệ thống phần mềm
      3. CM rất quan trọng vì chúng ta sẽ rất dễ ràng mất đi khả năg tracking các sự thay đổi của component được tích hợp vào hệ thống
      4. CM rất quan trọng cho các dự án để control các thay đổi được thực hiện bởi các devs khác nhau
   2. Các hoạt động của CM:
      1. Version management:
         1. Tracking sự thay đổi giữa các version của các component trong hệ thống, đảm bảo rằng các sự thay đổi này không làm ảnh hưởng một cách tiêu cực đến các component khác
      2. System building:
         1. Là một process mà chúng ta sẽ lắp ghép các component vào với nhau, các thư viện hay dữ liệu và biên dịch chúng thành một hệ thống chạy được
      3. Change management:
         1. Tracking các yêu cầu thay đổi từ phía khách hàng và sự thay đổi hệ thống tới từ các devs, tầm quan trọng của những thay đổi này và sự ảnh hưởng của nó đến hệ thống và quyết định xem những thay đổi này có nên được tiến hành hay không
      4. Release Management:
         1. Chuẩn bị cho việc release sản phẩm và tracking version của hệ thống đã được release để khách hàng sử dụng

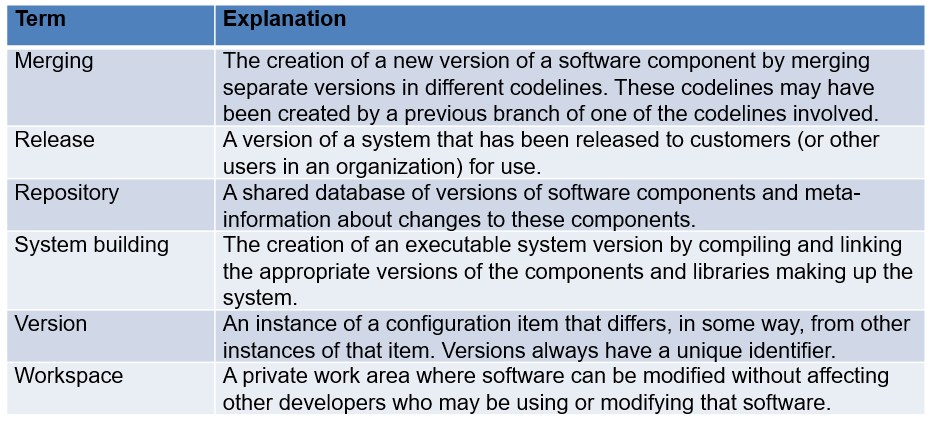


* 1. Agile development và CM:
     1. Agile development, khi mà components và systems được thay đổi rất nhiều lần, thì CM tools là cực kỳ quan trọng
     2. Các version được lưu ở một shared repository và các devs sẽ kéo nó về máy của mình
     3. Các devs sẽ thực hiện các sự thay đổi trên code có trên máy của họ, build, chạy và test các code này. Khi devs đã hoàn thiện những thay đổi này, họ sẽ đưa những component đã được thay đổi này lên shared repository
  2. Development phases:
     1. Là giai đoạn mà development team sẽ thực hiện những thay đổi của hệ thống, có thể là thêm các chức năng mới hoặc sửa lỗi, tăng hiệu năng của phần mềm …
     2. Release phases là nơi mà phần mềm được release để khách hàng sử dụng
  3. Multi-version systems:
     1. Đối với các hệ thống lớn, thường bao giờ cũng có nhiều hơn một version của hệ thống, sẽ có nhiều version ở các giai đoạn khác nhau của việc phát triển
     2. Có thể có nhiều nhóm cùng tham gia vào việc phát triển các phiên bản khác nhau

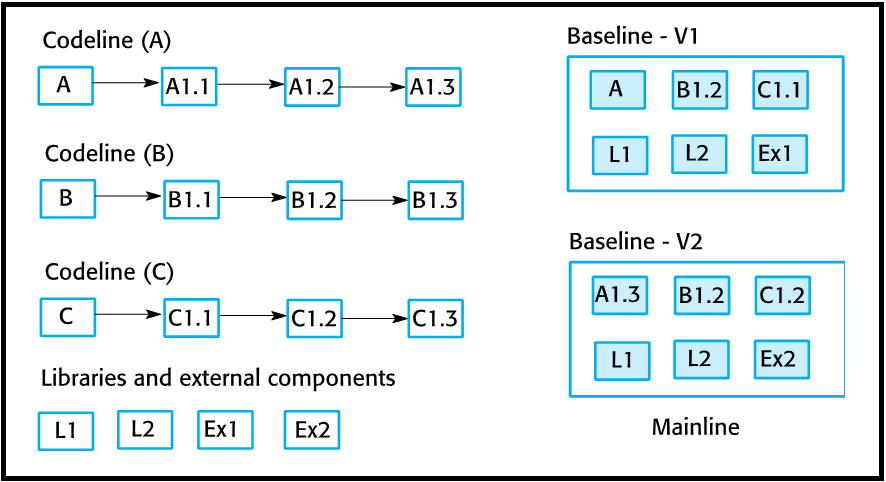


* 1. Các thuạt ngữ của CM



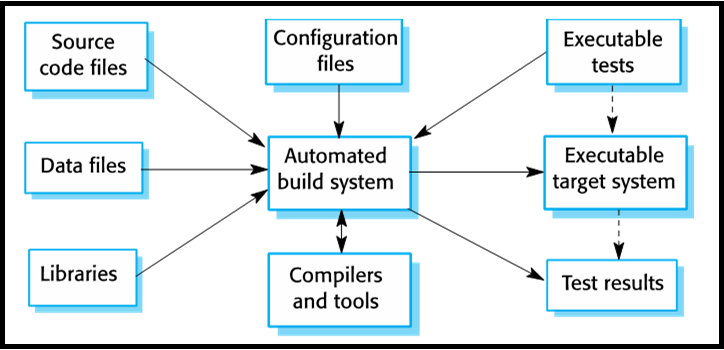


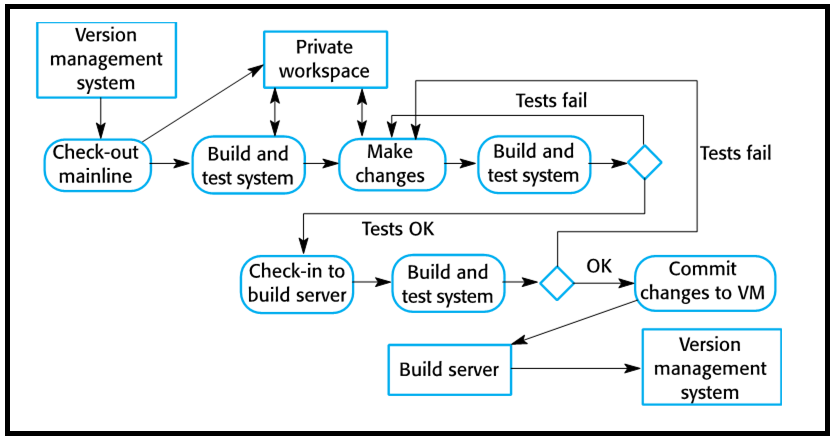
1. **Version management (VM)**
   1. Định nghĩa:
      1. Process tracking các phiên bản khác nhau của các thành phần trong hệ thống hoặc các configuration items và cả hệ thống sử dụng các components này
      2. Mục đích là để đảm bảo các sự thay đổi thực hiện bởi các devs khác nhau không làm ảnh hưởng tiêu cực đến cả hệ thống hay các components khác
      3. Có thể coi VM là process của việc quản lý codelines và baselines
   2. Codelines và Baselines:
      1. **Codelines** là một tập hợp các phiên bản source code với trình tự từ cũ tới mới, version sau được phát triển từ các phiên bản trước đó. Codelines thường được apply cho các components của hệ thống
      2. **Baseline:** là một definition của một hệ thống. Một baseline sẽ xác định các component versions được sử dụng trong hệ thống, cùng với đó là đặc tả về các thư viện sử dụng, các configuration files ..



* 1. Version control system:
     1. VC là một nơi để chúng ta xác định, lưu trữ và kiểm soát các quyền truy cập các phiên bản khác nhau của components. 2 loại VCS phổ biến:
        1. Phân tán: git …
        2. Tập trung: Subversion ...

1. **System building:**
   1. Định nghĩa:
      1. Là process của việc tạo ra một hệ thống chạy được hoàn chỉnh bằng cách kết hợp các components, các thư viện, các configuration files …
      2. System building tools và VM tools phải được communicate với nhau vì building process sẽ checkout các components từ repository của VMS
   2. Build platform:
      1. Development system bao gồm bộ biên dịch, source code editor …
      2. Build server: dùng để build các phiên bản chạy được của hệ thống
      3. Target environment: nên tảng mà hệ thống được executed





1. **Change management**
   1. Change