* Incremental development: đầu tiên tạo ra hệ thống đơn giản nhất có thể, ko cần dữ liệu thật, kết quả thật, nhưng phải là 1 khung xương vững chắc để phát triển hệ thống.
* Dựa trên khung xương này, ta sẽ thêm những code từng chút một cho đến khi đầy đủ
* Khi làm dự án nhỏ, nếu có sai sót thì có thể sửa nhanh chóng hoặc làm lại từ đầu, nhưng nếu là dự án lớn, thời gian sẽ rất lâu hơn và hậu quả có thể lớn hơn
* Cần cấu trúc, thiết kế tốt chương trình ngay từ đầu
* Vậy làm thế nào để có thiết kế tốt ngay từ đầu? Dự đoán? Cấu trúc dữ liệu? Giải thuật?
* Không thể tự mình xây mọi thứ mà phải kế thừa, dùng những cái đc xây sẵn (thư viện), nhưng cũng có thể tự mình viết lại những chức năng nếu nó là 1 trong các ưu tiên (về performance)
* Lên kế hoach cẩn thận ko có nghĩa là quá chi tiết. 1 kế hoạch tốt có thể cho phép thay đổi,phát triển dễ dàng
* Kĩ thuật không phải là những qui tắc, mà là công cụ.
* Cần biết trong trường hợp nào cần dùng công cụ nào
* 1 số thành công hay thất bại đc quyết định trc khi giai đoạn construction bắt đầu.
* Chapter 3:
* Fault Tolerance:
  + Kiến trúc cần chỉ ra hệ thống chịu đựng lỗi ntn?
  + Chịu đựng lỗi bao gồm các kỹ thuật phát hiện, phục hồi (nếu có thể), chịu đựng ảnh hưởng xấu
  + Hệ thống có thể xử lý lỗi theo các cách sau:
    - Hệ thống có thể quay lại và thử lại lần nữa nếu phát hiện lỗi. Điểm mà hệ thống quay lại là điểm mà ko có lỗi j xảy ra
    - Hệ thống có thể code phụ trợ nếu nó phát hiện lỗi trong code chính
    - Dùng giải thuật bỏ phiếu. Ví dụ, hệ thống có 3 class cùng làm 1 chức năng, ta có thể so sánh kết quả của 3 class này để lựa chọn
    - Hệ thống có thể thay giá trị lỗi = giá trị giả mạo # để những phần còn lại của hệ thống vẫn hoạt động tốt
* Tính khả thi
  + Về mục tiêu hiệu suất, tài nguyên có hạn, môi trương thực hiện
  + Kiến trúc phải chỉ ra hệ thống có thể thực hiện đc
* Overengineering
* Lựa chọn Buy-vs-Build
* Quyết định dùng lại
* Chiến lược thay đổi
* Chất lượng kiến trúc
  + 1 kiến trúc tốt thể hiện qua các lí do tại sao chọn lựa cái này mà ko phải cái kia, chức năng các class trong hệ thống, những thông tin của class cần đc giấu, …
  + Kiến trúc tốt thì phải tự nhiên, và dễ dàng
  + Mỗi sự thay đổi trong tương lai phải gắn với những khái niêm, phạm vi của kiến trúc ?
  + Mục tiêu của kiến trúc phải đc khẳng định rõ.
  + Kiến trúc phải chỉ ra động lực cho những quyết định. Phải có lí do cho những quyết định
  + Kiến trúc tốt thì độc lập với ngôn ngữ lập trình, môi trường.
  + Khi thiết kế kiến trúc, chỉ làm những việc cần thiết, chứ ko làm dư, làm những công việc của những giai đoạn #
  + Kiến trúc cần chỉ ra những khu vực (những thứ, chức năng) rủi ro. Lí do. Cách giảm thiểu rủi ro đó
  + Cần nhìn kiến trúc dưới nhiều góc độ khác nhau
* Danh sách câu hỏi:
  + **Kiến trúc:**
  + Cách tổ chức đã rõ ràng chưa?
  + Các khối chức năng chính đã được xây dựng tốt chưa? Bao gồm nhiệm vụ, và giao diện với các khối khác
  + Kiến trúc đã thực hiện hết các yêu cầu chưa? Liệu có quá nhiều khối hay quá ít khối cùng thực hiện 1 yêu cầu?
  + Đã mô tả và đánh giá những class quan trọng
  + Đã mô tả và đánh giá việc thiết kế dữ liệu
  + Đã chỉ ra cách thiết kế db
  + Đã quan tâm đến những nguyên tắc kinh doanh chưa?
  + Đã thiết kế giao diện người dùng chưa?
  + Để những thay đổi ở giao diện người dùng ko ảnh hưởng đến phần còn lại của chương trình, ta đã chuẩn hóa nó chưa?
  + Cách xử lý I/O đã đc mô tả và đánh giá chưa?
  + Đã ước lượng tài nguyên chưa? Đã có cách quản lý tài nguyên chưa?
  + Bảo mật ntn?
  + Tài nguyên cho mỗi class, mỗi khối đã đc chỉ rõ?
  + Kiến trúc đã chỉ ra cách mở rộng dự án chưa?
  + Interoperability
  + Quốc tế hóa/ Địa phương hóa
  + Cách thức xử lý lỗi đã đc chỉ ra và đánh giá chưa?
  + Đã có phương pháp chịu đựng lỗi chưa?
  + Hệ thống có khả thi hay ko?
  + Overengineering
  + Có những chức năng cần tận dụng từ cái có sẵn ko? hay là tự làm hết
  + Kiến trúc đã mô tả cách tạo ra code có thể sử dụng lại chưa?
  + Kiến trúc có khả năng thích nghi với các thay đổi tương lai ko?
  + **Chất lượng kiến trúc**
  + Kiến trúc đã xét đến tất cả yêu cầu chưa?
  + Có phần nào dư hay ko?
  + Kiến trúc có độc lập với ngôn ngữ, với môi trường ko?
  + Động lực để đưa ra quyết định là gì?
  + Mình có hài long với kiến trúc này chưa?
* 3.6
* Danh sách câu hỏi:
* Đã xác định dự án mình thuộc loại nào?
* Tất cả yêu cầu đã đc mô tả tốt chưa, đã đủ ổn định chưa
* Kiến trúc đã tốt?
* Những rủi ro đã đc mô tả?

Chapter 4:

4.3

* Nếu ngôn ngữ thiếu những đặc tính mà mình muốn, hãy tự tạo ra nó
* Tự tạo ra những qui tăc, qui ước, chuẩn khi lập trình

4.4

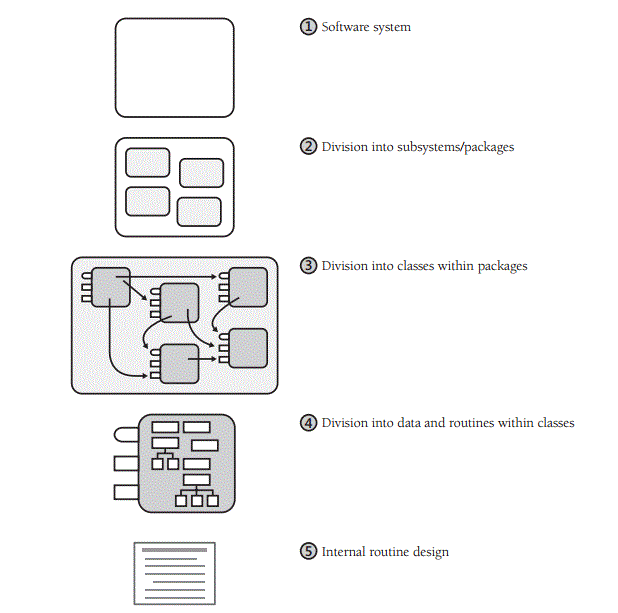
* Danh sách câu hỏi về cách lập trình
* Đã định nghĩa các qui ước về tên, chú thích, cách trình bày?
* Đã định nghĩa những cách thức như xử lý lỗi, quan tâm đến bảo mật ntn, class interface, performance, …
* Đã biết công nghệ mình sử dụng đang ở giai đoạn nào? Cần điều chỉnh cách thức cho phù hợp
* Đã xác định đc cách sử dụng ngôn ngữ để thực hiện qui tắc lập trình?
* Đã định nghĩa qui trình nhất quán chưa?
* Có cần vừa lập trình vừa test hay ko?
* Có cần viết testcase trc khi code?
* Có cần viết unit test?
* Có cần dùng debugger để coi quá trình chạy chương trình?
* Integration test?
* Có cần xem xét code của người trong nhóm?
* Đã cọn công cụ quản lý phiên bản?
* Đã chọn ngôn ngữ, phiên bản của ngôn ngữ?
* Có dùng fw ko? Nếu có thì chọn fw nào?
* Sử dụng những đặc tính ko chuẩn của ngôn ngữ?
* Những công cụ khác: trình soạn thảo, refactor, debugger, …
* Cần lập trình sử dụng ngôn ngữ, chứ ko lập trình theo ngôn ngữ

Part 2

Chapter 5:

5.1

* Thiết kế phần mềm là khái niệm, là sự chế tạo 1 mô hình, chuyển yêu cầu kĩ thuật thành phần mềm
* 1 thiết kế cấp cao tốt (kiến trúc) sẽ cung cấp 1 cấu trúc có thể chứa những thiết kế nhỏ hơn
* **Design is a wicked problem**
* Không có thiết kế nào ko bị lỗi, mình phải thưc hiện cái thiết kế đó, xem xét những lỗi xảy ra, và làm lại thiết kế khác tốt hơn ???
* **Importance of Managing Complexity**
* Nên tổ chức chương trình mà ở 1 thời điểm, ta chỉ cần tập trung vào 1 phần
* Do đó mục tiêu là giảm lượng logic chương trình mà ta phải nghĩ ở 1 thời điểm. Tương tự như trò tung hứng.
* Ở cấp kiến trúc, để giảm sự phức tạp, ta có thể chia nhỏ hệ thống thành nhiều hệ thống nhỏ.
* Mục tiêu khi giải quyết vấn đề (ko chỉ ở lập trình) là chia vấn đề phức tạp thành những phần đơn giản
* **How to attack Complexity**
* Sự phức tạp bắt nguồn từ:
  + 1 giải pháp phức tạp cho 1 vấn đề đơn giản
  + 1 giải pháp đơn giản (sai) cho 1 vấn đề phức tạp
  + 1 giải pháp không phù hợp, phức tạp cho 1 vấn đề phức tạp
* Do cuộc sống càng ngày càng phức tạp,nên sự phức tạp trong lập trình là tất yếu.
* 2 phương pháp để quản lý sự phức tạp:
  + Giảm sự phức tạp loại essential phải giải quyết ở 1 thời điểm
  + Đừng để phức tạp accidental sinh sôi
* **Desirable Characteristics Of a Design**
* 1 thiết kế tốt sẽ đạt đc những đặc tính sau đây, cái này có thể mâu thuẫn cái kia, nên cần sự cân bằng
  + **Giảm thiểu sự phức tạp**: tránh tạo ra những thiết kế độc đáo, thông minh, vì nó thường khó hiểu. Thay vào đó hãy tạo ra thiết kế đơn giản, dễ hiểu. Thiết kế phải cho phép ta có thể tập trung vào 1 phần mà ko cần quan tâm đến những phần #
  + **Dễ bảo trì**: Xây dựng chương trình mà khi 1 PG # đọc vào thấy dễ hiểu, ko cần doc
  + **Loose Coupling:** Giảm thiểu các sự kết nối giữa các phần trong hệ thống 1 cách tối đa
  + **Khả năng mở rộng:** mở rộng hệ thống mà ko va chạm, ảnh hưởng đến các kiến trúc nền tảng. Thay đổi 1 phần mà ko ảnh hưởng đến phần #.
  + **Khả năng sử dụng lại:** Sử dụng lại những phần khác
  + **High fan-in:** là số lượng các class dùng 1 class nào đó. Hệ thống nên có những class tiện ích tốt.
  + **Low-to-meium fan-out:** fan-out là số lượng class mà 1 class sử dụng. Số đó ko nên nhiều, nên nhỏ hơn 7, nếu nhiều thì sự phức tạp sẽ tăng lên.
  + **Khả năng di chuyển (portability):** Khả năng di chuyển hệ thống của mình qua 1 môi trường #.
  + **Leanness:** Thiết kế hệ thống mà ko có phần nào dư ra.
  + **Phân tầng (Stratification):** Phân tầng hệ thống trên sự phân tích, để ta có thể nhìn hệ thống ở 1 cấp độ mà ko quan tâm đến cấp độ khác. Ví dú như ta viết 1 hệ thống mới dựa trên 1 hệ thống cũ chất lượng kém, ta có thể viết 1 lớp giao diện với code cũ,, che đi phần cũ kém chất lượng đó.
  + **Kĩ thuật chuẩn:** Sử dụng những kĩ thuật thông dụng, dễ hiểu để ko làm hệ thống khó hiểu
* **Levels of Design**
* 1 số kỹ thuật thiết kế đc áp dụng cho mọi cấp độ của hệ thống, 1 số chỉ áp dụng ở 1số cấp độ.



* **Level 1: Software system**
* **Level 2: Division into subsystems or packages**
* Các hệ thống phụ có thể là DB, UI, business rules, phần báo cáo, log, …
* Chia hệ thống thành những hệ thống phụ và