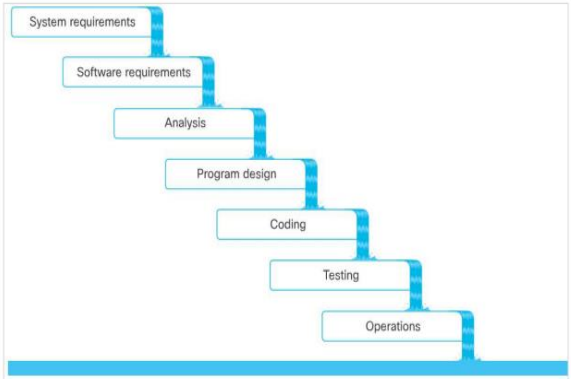
Ôn tập Lập trình an toàn và khai thác lỗ hỏng hệ thống

Chương 2 – 3

- Quy trình phát triển phần mềm **Software Development Life Cycle** (SDLC) gồm 6 giai đoạn:
  - Thu thập và phân tích yêu cầu -> tài liệu Đặc tả yêu cầu phần mềm – Software Requirement Specification (SRS).
  - Thiết kế -> tài liệu Thiết kế mức cao (High-level Design – HLD) và thiết kế mức thấp (Low-level Design – LLD).
  - Phát triển, coding -> mã chức năng (function code) hiện thực yêu cầu.
  - Kiểm thử.
  - Triển khai.
  - Duy trì và bảo trì.
- Các phương pháp phát triển phần mềm linh hoạt gọi là "**Agile Development**".
- Phương pháp phát triển phần mềm truyền thống:
  - **Mô hình thác nước Waterfall**
    - Mô hình nguyên bản gồm 7 giai đoạn.
    - Một giai đoạn bắt đầu khi giai đoạn trước đó kết thúc.
    - Không có sự quay lui. (waterfall cải tiến cho phép quy lui)
    - Dự án phù hợp: có yêu cầu xác định rõ ràng, đầy đủ và cố định hoặc ít thay đổi, nhân lực có kinh nghiệm, nắm vững công nghệ.



▪ **Ưu điểm:**
- Đơn giản, nổi tiếng, dễ hiểu, dễ sử dụng, dễ quản lý.
- Kế hoạch rõ ràng,tài liệu được hoàn thành sau mỗi giai đoạn.
- Các yêu cầu được cung cấp sớm cho người dùng.
- Người quản lý dự án (PM) lập kế hoạch, kiểm soát chặt chẽ.

▪ **Nhược điểm:**
- Không linh hoạt, khó thích ứng khi có thay đổi.
- Thời gian phân phối sản phẩm lâu.
- Không phù hợp với các dữ án kéo dài và tiếp diễn lâu
- Nhiều nguy cơ và không chắc chắn.
- Lãng phí nguồn nhân lực.

  - **Mô hình Xoắn ốc – Spiral**
    - Các hoạt động được tổ chức thành các chu trình xoắn ốc thay vì tuần tự, có quản lý rủi ro.
    - Mỗi chu trình xoắn ốc là 1 giai đoạn của chu trình, gồm 4 phần:
      - Xác định mục tiêu của chu trình.
      - Đánh giá giảm thiểu rủi ro.
      - Phát triển và đánh giá hệ thống.
      - Lên kế hoạch cho chu trình kế tiếp.
  - **Mô hình RUP (Rational Unfied Process)**

- Thể hiện các giai đoạn theo thời gian, các hoạt động cụ thể trong quy trình.
  Mỗi giai đoạn chia làm nhiều vòng lặp, mỗi vòng lặp như một quy trình waterfall.
- Gồm 4 giai đoạn:
  - Inception – khởi đầu.
  - Elaboration – chi tiết hóa.
  - Consutruction – xây dựng.
  - Transition – chuyển giao.

- Ưu điểm:
  - Độ đảm bảo cao, phù hợp với phần mềm chú trọng vào an toàn hệ thống.
  - Thích hợp cho các hệ thống phần mềm lớn.
  - Có sẵn tài liệu cho nhân viên mới.
  - Đảm bảo làm việc theo quy trình rõ ràng.
- Nhược điểm:
  - Chi phí tài liệu.
  - Không linh hoạt trước những thay đổi.
  - Cản trở sự đổi mới.

- Phương pháp phát triển phần mềm Agile tập trung vào tính linh động và hướng tới khách hàng.
    - **Scrum**: tập trung vào các nhóm nhỏ, tự tổ chức, họp hàng ngày trong thời gian ngắn và làm việc trong các sprint độc lập. (Sprint là một khoảng thời gian cụ thể, thường là 2 – 4 tuần) , trong đó nhiều nhóm đảm nhận nhiều nhiều nhiệm vụ (còn được gọi là user story).
    - **Lean**: phương pháp Lean nhấn mạnh vào việc loại bỏ hao phí trong việc lập kế hoạch, tối đa hóa giá trị đối với khách hàng và thực hiện, đồng thời giảm tải nhận thức của lập trình viên.
        - Quy tắc Lean gồm 7 bước:
            - Giảm thiểu hao phí.
            - Tăng cường việc học tập.
            - Quyết định càng muộn càng tốt.
            - Phân phối càng nhanh càng tốt.
            - Trao quyền cho nhóm.
            - Xây dựng toàn vẹn từ bên trong.
            - Tối ưu hóa toàn bộ.
        - Ưu điểm:
            - Phù hợp với môi trường năng động, hỗn loạn.
            - Hỗ trợ các yêu cầu phát sinh, thay đổi nhanh chóng.
            - Tránh chi phí tài liệ.
            - Trao quyền mọi người.
        - Nhược điểm:
            - Cần người có năng lực cao, tinh thần tự giác cao.
            - Khó khăn trong các hệ thống quan trọng về an toàn.
            - Vấn đề ở các dự án lớn.
    - **Extreme Programming (XP):** XP hướng tới giải quyết các loại vấn đề cụ thể về chất lượng cuộc sống mà nhóm phát triển phần mềm phải đối mặt.
    - **Feature-Driven Development (FDD):** FDD quy định rằng việc phát triển phần mềm phải được tiến hành theo mô hình tổng thể, được chia nhỏ, lập kế hoạch, thiết kế và xây dựn theo từng tính năng.

- So sánh truyền thống và Agile:

Đặc điểm	Truyền thống (Waterfall)	Agile (Scrum)
Mục tiêu	Tăng sự thành công của dự án qua việc lặp lại những kết quả tốt của các dự án đã thành công	Đáp ứng sự thay đổi của khách hàng một cách tốt nhất, hiệu chỉnh phù hợp với từng khách hàng và dự án, chuyển giao nhanh
Quy trình	Chặt chẽ, các bước được định nghĩa rõ ràng	Quy trình không chặt chẽ, được hiểu ngầm
Tài liệu	Tài liệu, biểu mẫu nhiều, rõ ràng	Ít viết tài liệu, dựa vào hiểu ngầm
Kế hoạch	Kế hoạch chi tiết, áp dụng xuyên suốt dự án	Có kế hoạch nhưng ở mức độ cao, không chi tiết
Quản lý	Dựa vào kế hoạch, so sánh kế hoạch với thực tế, chỉ tương tác với khách hàng khi cần thiết	Trông đợi vào sự hợp tác giữa các thành viên trong nhóm, tăng sự chủ động của thành viên, đánh giá tiến độ dựa trên khả năng đáp ứng yêu cầu của khách hàng, tương tác với khách hàng thường xuyên
Các dự án phù hợp	Lớn, nhiều người Yêu cầu về độ an toàn cao	Nhỏ, ít người (<20 người) Dự án có thay đổi nhiều và nhanh Dự án ngắn

- Mẫu thiết kế:
    - **Mẫu thiết kế Observer** là mẫu theo dạng đăng ký – thông báo, cho phép đối tượng nhận các sự kiện khi có thay đổi ở một đối tượng mà chúng đang quan sát. Ưu điểm: có thể nhận được thông báo theo thời gian thực khi có thay đổi xảy ra.
    - **Mẫu thiết kế Model-View-Controllder (MVC)** nhằm đơn giản hóa việc phát triển các ứng dụng có giao diện người dùng. Ưu điểm: mỗi thành phần có thể được build song song.
        - **Model:** cấu trúc dữ liệu của ứng dụng và có nhiệm vụ quản lý dữ liệu, logic và các quy tắc của ứng dụng, nhận input từ controller.
        - **View:** biểu diễn trực quan của dữ liệu.
        - **Controller:** đóng vai trò trung gian giữa model và view, nhận input từ người dùng và thay đổi nó để phù hợp với định dạng cho model hoặc view.
- Hệ thống quản lý phiên bản: là phương pháp để quản lý các thay đổi trong các tập tin để lưu trữ lịch sử thay đổi.
    - Lợi ích:
        - Cho phép phối hợp làm việc.
        - Hỗ trợ truy vết và trực quan.
        - Làm việc độc lập.
        - Đảm bảo an toàn.
        - Làm việc từ bất cứ đâu.
    - Có 3 loại hệ thống quản lý phiên bản:
        - **Cục bộ - Local Version Control System (LVCS).**
            - Sử dụng CSDL đơn giản đẻ lưu vết tất cả các thay đổi trên tập tin.
            - Thông thường, hệ thống lưu trữ các khác biệt giữa 2 phiên bản của tập tin.
            - Khi cần khôi phục phiên bản cũ, các khác biệt được đảo ngược để quay lại phiên bản được yêu cầu.
        - **Tập trung – Centralized Version Control System (CVCS).**
            - Sử dụng mô hình Client – Server.
            - Kho lưu trữ được đặt trên một vị trí trung tâm là server.
            - Chỉ cho phép 1 cá nhân được làm việc trên 1 file nhất định tại 1 thời điểm.
            - Một cá nhân cần kiểm tra khóa tệp và thực hiện các thay đổi cần thiết, sau đó hoàn thành mở khóa tệp.
        - **Phân tán – Distributed Version Control System (DVCS).**
            - Là mô hình peer – to – peer.
            - Kho lưu trữ nằm trên hệ thống client nhưng thường được lưu trong dịch vụ hosting kho lưu trữ.
            - Mọi cá nhân được làm việc trên bất kỳ file nào, bất kỳ lúc nào, không cần phải khóa file.
            - Khi cá nhân lưu trữ xong có thể đẩy file lên khi lưu trữ chính trong dịch vụ hosting kho lưu trữ, hệ thống quản lý hiên bản kiểm tra xem có xung đột không.
- **Git** là hệ thống quản lý phiên bản mã nguồn mở, hiện đang được sử dụng rộng rãi trong phát triển phần mềm. Git lưu trữ các snapshot thay vì lưu trữ các khác biệt giữa các phiên bản.
    - **3 giai đoạn (stage):**
        - Kho lưu trữ Repository.
        - Thư mục đang làm việc working directory.
        - Khu vực staging (staging area)
    - **3 trạng thái (stage):**
        - Committed
        - Modified
        - Staged
    - **2 loại khi lưu trữ:**
        - Kho lưu trữ cục bộ: nằm trên hệ thống file của 1 máy client (nơi thực thi các câu lệnh git).
        - Kho lưu trữ từ xa: lưu đâu đó bên ngoài máy client, thường là server hoặc dv hosting. (**DVCS**)

- **Phân nhánh** cho phép người dùng có thể làm việc độc lập trên mã nguồn mà không ảnh hưởng đến mã nguồn chính trong kho lưu trữ. Khi một kho lưu trữ được tạo, mã nguồn sẽ được tự động thêm vào nhánh Master.
- **Git** và **Github** không giống nhau:
    - **Git** là một hiện thực của hệ thống quản lý phiên bản phân tán và cung cấp giao diện dòng lệnh.
    - **Github** là một dịch vụ do Microsoft cung cấp để hiện thực dịch vụ hosting kho lưu trữ với Git.
      Một số tính năng khác của Github:
        - Đánh giá mã nguồn
        - Quản lý dự án
        - Theo dõi lỗi
        - Yêu cầu tính năng
- Các lệnh git:
    - Thiết lập git: **git config – loabl key value**
    - Tạo kho lưu trữ git mới: **git init**
    - Tạo kho lưu trữ từ một thư mục có sẵn: **git init <project directory>**
    - Lấy về kho lưu trữ đã có: **git clone <repository>**
    - Xem các file đã thay đổi trong thư mục làm việc: **git status**
    - So sánh các file: **git diff**
    - Thêm file vào khu vực staging: **git add <file path>**
    - Xóa file: **git rm <file path>**
    - Cập nhật kho lưu trữ cục bộ: **git commit -m "<message>"**
    - Cập nhật kho lưu trữ từ xa: **git push**
    - Cập nhật bản sao kho lưu trữ cục bộ: **git pull**
- Git diff

Symbol	Ý nghĩa
+	Chỉ ra dòng (line) đã được thêm (có ở file sau nhưng không có ở file trước)
-	Chỉ ra dòng (line) đã bị xóa (có ở file trước nhưng không có ở file sau)
/dev/null	1 file đã được thêm.
or "blank"	Thêm các dòng ngữ cảnh bên cạnh các dòng đã bị thay đổi.
@@	Ký hiệu trực quan chỉ định bắt đầu khối thông tin tiếp theo. Trong các thay đổi của 1 file, có thể có nhiều khối như vậy.
index	Hiển thị các commit đang so sánh.

- **Phương thức (Methods) và hàm(Function):**
    - Phương thức: là các khối mã gắn liền với một đối tượng, thường dùng trong lập trình hướng đt.
    - Hàm: là các khối mã độc lập.
    - Lập trình theo nguyên tắc đóng gói:
        - Mã code một tác vụ nào đó ngay cả khi nó chỉ xảy ra một lần, cũng có thể được đóng gói.
        - Mã code sử dụng nhiều lần nên được đóng gói.
    - **Đối số (Arguments) và tham số (parameters)**: giúp các phương thức và hàm trở nên linh hoạt hơn.
- **Mã nguồn sạch – clean code** là kết quả của việc các lập trình viên khiến mã nguồn của họ dễ đọc và dễ hiểu. Tuân theo các quy tắc về định dạng, cách tổ chức, tính trực quan của các thành phần, mục đích và khả năng tái sử dụng.
- **Đánh giá mã nguồn – code review:** lập trình viên (reviewer) xem xét toàn bộ hoặc 1 phần mã nguồn, hoặc các thay đổi nhất định trong mã nguồn và đưa ra phản hồi.
    - **Các loại đánh giá mã nguồn:**
        - Đánh giá mã nguồn dạng formal: các lập trình viên họp để đánh giá toàn bộ mã nguồn.
        - Đánh giá mã nguồn theo các thay đổi: đánh giá mã nguồn có công cụ hỗ trợ.
        - Đánh giá mã nguồn dạng Over-the-shoulder: 1 reviewer giám sát 1 coder.
        - Gửi email

- **Kiểm thử phần mềm** được chia thành 2 loại chính:
    - Kiểm thử chức năng – Functional testing
    - Kiểm thử phi chức năng – Non-functional testing
- **Test-driven Development (TDD) – phát triển hướng kiểm thử**: lập trình viên có thể xem xét các yêu cầu thiết kế như những bài kiểm thử và viết các phần mềm đẻ vượt qua các bài kiểm tra đó.
    - **Kiểm thử đơn vị - unit testing:** kiểm thử chức năng một cách chi tiết.
    - **Kiểm thử tích hợp – integration testing:** đảm bảo tất cả các đơn vị riêng biệt phù hợp với nhau để tạo thành một ứng dụng hoàn chỉnh.
    - Mô hình cơ bản của TDD là quy trình 5 bước lặp đi lặp lại:
        - Tạo phép kiểm thử mới.
        - Chạy phép kiểm thử để tìm các lỗi không mong muốn.
        - Viết mã ứng dụng để vượt qua phép kiểm thử mới.
        - Chạy phép kiểm thử để tìm lỗi.
        - Tái cấu trúc và cải thiện mã ứng dụng.
- Các định dạng dữ liệu:
    - **RestAPIs** cho phép người dùng trao đổi thông tin với các dịch vụ hoặc thiết bị ở xa.
        - **XML Extensible Markup Language:** ngôn ngữ đánh dấu mã rộng, là phương pháp gói dữ liệu dạng văn bản trong các thẻ tag đối xứng để biểu diễn ngữ nghĩa.
        - **JSON JavaScript Object Notation:** là một định dạng dữ liệu có nguồn gốc từ cách biểu diễn cá dữ liệu phức tạp trong JavaScript.
        - **YAML Ain't Markup Language:** là tập cha của JSON được thiết kế dễ đọc hơn.
- **Parsing – phân tích định dạng:** phân tích một thông điệp thành các thành phần và hiểu chức năng của chúng trong một ngữ cảnh nhất định.
- **Seriaizing** gần như ngược lại với parsing, chuyển đổi các cấu trúc dữ liệu bên trong thành các thông điệp có ý nghĩa tương đương được định dạng như các chuỗi ký tự.

Design Flaw	Security bug
- Lỗi trong **thiết kế** phần mềm. - Cho phép user thực hiện các hành động vốn không được thực hiện. - Biện pháp: dùng security design concepts, requirements và threat modeling.	- Vấn đề trong **hiện thực** phần mềm, trong lập trình. - Cho phép user dùng ứng dụng một cách độc hại hoặc không chính xác. - Biện pháp: đánh giá mã nguồn, kiểm thử security, training và hướng dẫn về secure code.

Chương 4 – 5

- Nguyên tắc **Pushing Left / Shifting Left**: cần đảm bảo security ngay từ khi bắt đầu dự án.
- **Software Assurance Maturity Model (SAMM):** framework hỗ trợ hình thành và hiện thực chiến lược bảo mật phần mềm phù hợp với các rủi ro mà tổ chức phải đối mặt. Hỗ trợ:
    - đánh giá các phương pháp bảo mật phần mềm hiện có của tổ chức.
    - thiết lập chương trình đảm bảo bảo mật phần mềm cân bằng trong các vòng lặp phát triển.
    - demo các giải pháp cái tiến cụ thể đối với chương trình bảo mật.
    - định nghĩa và đo lường các hoạt động liên quan đến bảo mật trong tổ chức.
- Làm sao để tránh Insecure Design:
    - **Tạo các user story tốt**
        - Cần các yêu cầu chức năng và phi chức năng về bảo mật.
        - User story: mô tả ngắn gọn, dễ hiểu một chức năng của phần mềm.
        - Trong user story cần ghi những flaw có thể xảy ra của phần mềm.
    - **Yếu tố security trong quy trình phát triển:** Phát triển phần mềm an toàn: an toàn phần mềm ngay từ đầu và có kiểm thử bảo mật.
    - **Bảo mật và tách biệt các layer, thư viện:** Đảm bảo các layer (ứng dụng và mạng) tách biệt rõ ràng và sử dụng các thư viện design pattern an toàn.
- Security througn obscurity – bảo vệ mã nguồn.
- **Threat Modeling – Mô hình hóa mối đe dọa**: quy trình xác định các mối đe dọa tiềm ẩn đối với ứng dụng và đảm bảo các biện pháp giảm thiểu thích hợp được áp dụng.
    - **Quy trình**:
        - Đưa ra các câu hỏi và đảm bảo đã cân nhắc các trường hợp xấu xảy ra.
        - Đánh giá khả năng, ảnh hưởng -> loại bỏ các concern không thể xảy ra, không ảnh hưởng.
        - Đánh giá các concern còn lại ở mức độ cao, trung bình, thấp
        - Đưa ra kế hoạch giảm thiểu, loại bỏ hoặc chấp nhận.
    - **Phân loại:**
        - **Stride:** tập trung vào các vấn đề xung quanh chức thực, phân quyền, CIA và thoái thác trách nhiệm.
            - **Spoofing:** truy cập bất hợp pháp và sử dụng thông tin chứng thực của user khác.
            - **Tampering:** thay đổi dữ liệu độc hại hoặc trái phép.
            - **Repudiation:** phủ nhận đã thực hiện hành vi độc hại, tranh thoái thác trách nhiệm và khả năng nhận ra các mối đe dọa thoái thác trách nhiệm.
            - **Information Disclosure:** lộ thông tin với những user không được phép truy cập.
            - **Denial of service (DOS):** từ chối dịch vụ với các user hợp lệ, các mối đe dọa với tính sẵn sàng và tin cậy của hệ thống.
            - **Elevation of Privilege:** các user không có quyền cố gắng lấy quyền để chiếm hệ thống, xâm nhập hiệu quả và trở thành 1 phần của hệ thống tin cậy.
        - **PASTA – Process for Attack Simulation and Threat Analysis**: tập trung vào các yêu cầu kinh doanh, chức năng, sơ đồ dữ liệu (gồm 7 bước).
        - **TRIKE**: phương pháp mô hình hóa mối đe dọa mã nguồn mở, dùng khi đánh giá bảo mật từ góc nhìn quản lý rủi ro.
        - **VASTA** – Visual, Agile, Simple threat: mô hình hóa mối đe dọa tự động.
        - **DREAD**: nhằm đánh giá, phân tích và tìm xác suất xảy ra rủi ro bằng cách đánh giá các mối đe dọa ở các phương tiện khác nhau.
        - **OCTAVE**: xác định, đánh giá, quản lý rủi ro cho các tài sản CNTT
- **Secure Code**:
    - HTTP methods: method nào không sử dụng thì disable để giảm khả năng tấn công.
    - Identity – quản lý định danh.
    - Authentication và Authorization – chứng thực và phân quyền truy cập.
    - Session Management – quản lý phiên làm việc.
    - Bounds Checking.
    - Error Handling, Logging and Monitoring – Xử lý lỗi, ghi log và giám sát.

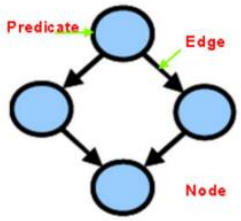
- Thiết lập **Secure coding baselines:**

Secure coding baselines là các yêu cầu secure code tối thiểu và danh sách checklist để team dự án chuyển sang giai đoạn tiếp theo. Cũng là một phần trong các điều kiện cần được đảm bảo để sản phẩm có thể được phân phối.

- **Training về Mã nguồn an toàn:**
    - Mục đích: thông báo cho đội ngũ phát triển biết về các phương pháp secure code sẽ được áp dụng. Thay vì các rule về lập trình an toàn đơn thuần, tốt hơn hết nên đưa ra các case study hoặc ví dụ về mã nguồn có lỗ hổng trong một ngữ cảnh nào đó.
    - Có 2 cách quét mã nguồn:
        - Quét mã nguồn tĩnh với IDE plugin
        - Quét mã nguồn hàng ngày để tạo ra các báo cáo quét hàng ngày.
    - Công cụ phân tích mã nguồn:
        - **Phân tích mã nguồn tĩnh**: phương pháp thực hiện trên mã nguồn "tĩnh" (không thực thi) của phần mềm với các công cụ phân tích mã nguồn tĩnh để phát hiện các lỗ hổng tiềm ẩn.
            - Xem xét mã nguồn trước khi thực thi.
            - Phân tích nhanh hơn so với phân tích động.
            - Có thể tự động hóa quản lý chất lượng mã nguồn.
            - Có thể tự động hóa việc tìm bug hay các vấn đề bảo mật ở giai đoạn đầu.
            - Một số công cụ phân tích mã nguồn tĩnh được tích hợp sẵn trong các IDE.
- Kiểm thử an toàn: **White-box testing** và **Black-box testing**.
- **DevOps** là sự kết hợp giữa phát triển (development) và vận hành (operations) phần mềm.
    - Lợi ích của DevOps:
        - Cải thiện sự hợp tác, hỗ trợ chức năng và khắc phục lỗi nhanh hơn.
        - Tăng tính linh hoạt, nhanh chóng và tin cậy.
        - Bảo mật cơ sở hạ tầng và bảo vệ dữ liệu.
        - Bảo trì và cập nhật nhanh hơn.
        - Chuyển đổi các dự án với chiến lược số hóa.
        - Tăng tốc độ, năng suất của team kinh doanh và CNTT.
- Java Security:
    - **Static application security testing (SAST):** giúp xác định các vấn đề trong mã nguồn thông qua tìm kiếm trong mã nguồn trước khi biên dịch. Ngăn ngừa một số vấn đề:
        - Lộ tài nguyên.
        - Các khóa (keys) hay định danh (credentials) được code cứng.
        - Vấn đề về sự ổn định.
    - **Dynamic application security testing (DAST):** xác định các thông tin không thể tự động xác định từ bên trong mã nguồn.
- **Anchore Engine** là dự án mã nguồn mở, cung cấp dịch vụ để kiểm tra, phân tích và chứng nhận các image container. Anchore Engine được cung cấp dưới dạng một image container docker.
- **SonarQube** là một công cụ đánh giá mã nguồn tự động để phát hiện bug, lỗ hổng và "code smell" trong mã nguồn.
- **Insecure Deserialization**: khi kẻ tấn công có thể thay thế dữ liệu đã serialize bằng dữ liệu độc hại nào đó, có thể gây ra hậu quả nghiệm trọng.
    - **Serialization:** chuyển dữ liệu từ dạng object sang dạng có thể truyền hoặc lưu trữ trong stream hoặc file.
    - **Deserialization:** đưa dữ liệu về dạng object ban đầu sau khi truyền hoặc khi cần sử dụng.
    - Giải pháp: hạn chế serialization và deserialization khi có thể, nếu cần sử dụng object đã serialize, chỉ sử dụng dữ liệu từ các nguồn tin cậy.
- **Race condition** xảy ra khi các hoạt động của phần mềm thực thi dựa trên thời gian, và các tiểu trình cần được thiết lập đồng bộ, tuy nhiên không phải như vậy.

Chương 6

- **Quy trình kiểm thử phần mềm – Software Testing Life Cycle STLC** bao gồm các giai đoạn:
  - Requirenment analysis – phân tích yêu cầu.
  - Test planning – lập kế hoạch.
  - Test case development – thiết kế kịch bản.
  - Test execution – thực hiện kiểm thử.
  - Test cycle closure – đóng chu trình kiểm thử.
- **Kiểm thử hộp trắng**: thiết kế test case dựa vào cấu trúc nội tại bên trong của đối tượng cần kiểm thử. Đảm bảo tất cả các câu lệnh, các biểu thức điều kiện bên trong chương trình đều được thực hiện ít nhất 1 lần.
  - Các kỹ thuật kiểm thử hộp trắng:
    - **Basic Path Testing – Kiểm thử đường cơ sở** là phương pháp thiết kế test case đảm bảo rằng tất cả các đường dẫn độc lập (independent path) trong một code module đều được thực thi ít nhất 1 lần => cho biết lượng test case tối thiểu.

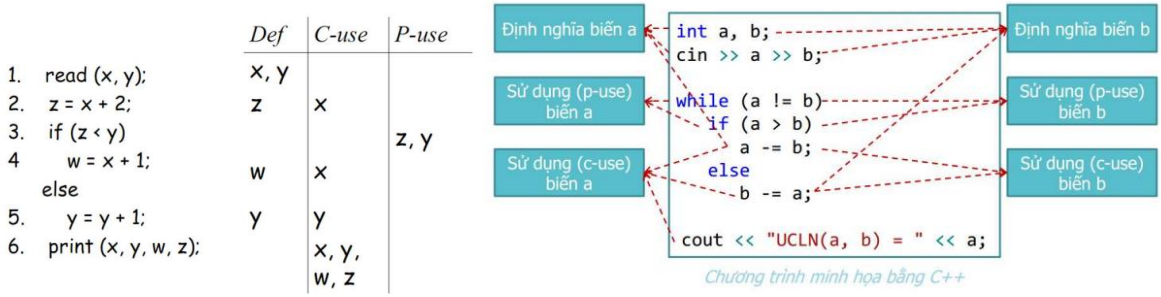


Xây dựng đồ thị luồng điều khiển:
- **Edge**: đại diện cho một luồng điều khiển.
- **Node:** đại diện cho một hoặc nhiều câu lệnh xử lý.
- **Predicate node:** đại diện 1 biểu thức điều kiện.

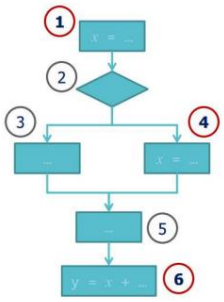
    - **Control-flow / Coverage Testing – Phủ kiểm thử** là kỹ thuật thiết kế các test case đảm bảo cover được tất cả các câu lệnh, biểu thức điều kiện trong code module cần kiểm thử.
      Có 4 tiêu chí đánh giá độ bao phủ:
      - **Method Coverage (phương thức):** tỷ lệ % các phương thức trong chương trình được gọi thực hiện bởi các kịch bản kiểm thử.
      - **Statement Coverage (câu lệnh):** tỷ lệ % các câu lệnh trong chương trình được gọi thực hiện bởi các test case.
      - **Decision/Branch Coverage (biểu thức điều kiện):** tỷ lệ % các biểu thức điều kiện trong chương trình được ước lượng giá trị trả về (true, false) trong các test case.
      - **Condition Coverage (biểu thức điều kiện đơn):** tỷ lệ % các biểu thức điều kiện đơn trong biểu thức điều kiện phức của chương trình được gọi ước lượng giá trị trả về (true, false) trong test case.

      Các cấp bao phủ kiểm thử:

      - **Phủ cấp 0**: kiểm thử những gì có thể kiểm thử được, phần còn lại để người dùng phát hiện và báo lại sau.
      - **Phủ cấp 1**: bao phủ câu lệnh, các câu lệnh được thực hiện ít nhất 1 lần.
      - **Phủ cấp 2**: bao phủ nhánh, tại các điểm quyết định thì các nhành đều được thực hiện ở cả 2 phía true và false.
      - **Phủ cấp 3**: bao phủ điều kiện, các điều kiện con đều được thực hiện ít nhất 1 lần.
      - **Phủ cấp 4**: kết hợp phủ nhánh và điều kiện.
    - **Data-flow Testing – Kiểm thử luồng dữ liệu** là kỹ thuật thiết kế test case dựa vào việc khảo sát sự thay đổi trạng thái trong chu kỳ sống của các biến trong chương trình. Kiểm thử luồng dữ liệu lựa chọn các đường dẫn để kiểm thử dựa trên vị trí định nghĩa (define) và sử dụng (use) các biến trong chương trình.



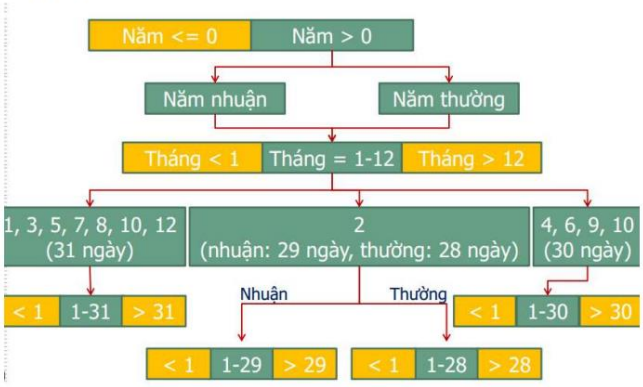
- DEF(1) = {x, …}
- DEF(4) = {x, …}
- USE(6) = {x, …}
- (1, 2, 3, 5, 6) và (4, 5, 6) là các đường dẫn DU của biến *x*.
- (1, 2, 4, 5, 6) không là đường của biến *x* vì nó được định nghĩa lại ở câu lệnh 4.
- Các cặp DU là (1, 6) và (4, 6)



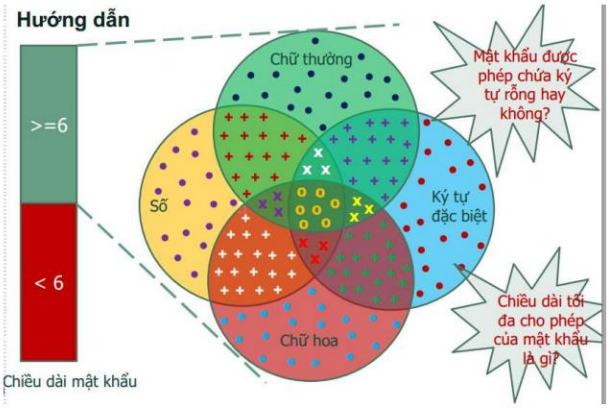
- **Kiểm thử hộp đen** còn được gọi là kiểm thử dựa trên đặc tả vì thông tin duy nhất làm cơ sở để kiểm thử hộp đen là bảng đặc tả yêu cầu chức năng của từng thành phần phần mềm. Tester tập trung vào những gì làm được (WHAT) và không quan tâm nó làm như thế nào (HOW).
  - **Ưu điểm:**
    - Không cần truy cập mã nguồn.
    - Tách biệt khung nhìn của người dùng và lập trình viên.
    - Nhiều người có thể tham gia kiểm thử.
  - **Nhược điểm:**
    - Kiểm thử khó đạt kết quả cao, khó thiết kế test case.
    - Khó kiểm thử phủ hết được mọi trường hợp.
    - Không có định hướng kiểm thử rõ ràng.
  - **Quy trình kiểm thử hộp đen:**
    - Phân tích, xem xét các đặc tả chức năng của phần mềm.
    - Thiết kế kịch bản kiểm thử.
    - Thực thi kịch bản kiểm thử.
    - So sánh kết quả đặt được với kết quả mong muốn trong từng test case.
    - Lập báo cáo kết quả kiểm thử.
  - **Các kỹ thuật kiểm thử hộp đen:**
    - **Phân vùng tương đương EP:** VD: xếp loại 0 <= ĐTB <= 5 yếu, 5 <= ĐTB <= 7 trung bình.
      - Phân chia một tập các điều kiện kiểm thử thành các tập con có các giá trị tương đương nhau và kiểm thử các tập con này.
      - Nếu một giá trị đại diện trong nhóm đúng thì các giá trị còn lại trong nhóm đúng và ngược lại.
      - Phù hợp với các bài toán có giá trị đầu vào là một miền xác định.
      - Giảm thiểu số lượng test case.

- *Quiz 01:*
  - Chương trình kiểm tra một ngày tháng nhập vào có hợp lệ hay không? Người cần nhập vào năm, tháng, ngày của ngày muốn kiểm tra.
  - Ví dụ 12/12/2017 là ngày hợp lệ, 32/12/2017 là ngày không hợp lệ.
  - Sử dụng phương pháp phân vùng tương đương thiết kế các test case để kiểm thử chương trình trên.

- *Hướng dẫn:*



- *Quiz 02:*
  - Chức năng đăng ký tài khoản của một ứng dụng học tập yêu cầu mật khẩu phải có chứa chữ hoa, chữ thường, số, ký tự đặc biệt và chiều dài tối thiểu là 6.
  - Sử dụng kỹ thuật phân vùng tương đương viết các test case kiểm tra ô nhập liệu mật khẩu.



  - **Phân tích giá trị biên BVA** là kỹ thuật kiểm thử dựa trên các giá trị tại biên giữa các phân vùng tường đương, bao gồm các trường hợp hợp lệ và không hợp lệ.
    - Hiệu quả nhất trong trường hợp các đối số đầu vào (input variables) độc lập với nhau và mỗi đối số đều có một miền giá trị hữu hạn.
    - Một số kỹ thuật kiểm thử giá trị biên:
      - Standard BVA: số test case là 4n + 1

        =>

      - Robust testing: số test case là 6n + 1

        <=

      - Worst-case testing: các biến sẽ được kiểm tra đồng thời tại biên để dò lỗi, không kiểm thử tại các giá trị không hợp lê. Số test case là 5^n.
      - Robust worst-case testing: tương tự Worst-case testing nhưng kiểm tra thêm tại các giá trị không hợp lệ. Số test case là 7^n.

❖Giả sử biến x có miền giá trị [min,max]

→ Các giá trị được chọn để kiểm tra

Min	-	Minimal
Min+	-	Just above Minimal
Nom	-	Average
Max-	-	Just below Maximum
Max	-	Maximum

❖Mở rộng của Standard BVA

❖Kiểm thử cả hai trường hợp:

- Input variable hợp lệ (clean test cases)

  → Kiểm thử tương tự như Standard BVA trên các giá trị (min, min+, average, max-, max)

- Input variable không hợp lệ (dirty test cases)

  → Kiểm thử trên 2 giá trị: min-, max+ (nằm ngoài miền giá trị hợp lệ)

    - Bản quyết định
    - Kiểm thử dịch chuyển trạng thái

- **Kiểm thử Fuzzing**
  - Là một kỹ thuật dựa trên việc gửi các giá trị ngẫu nhiên hay rác tới một ứng dụng hoặc một tính năng/hàm của chương trình để quan sát hành vị bất thường của ứng dụng. (Fuzzing là kiểm thử hộp đen nhưng nó cũng có thể là hộp trắng nếu có mã nguồn).
  - Là một kỹ thuật phát hiện lỗi của phần mềm bằng cách tự động hoặc bán tự động sử dụng phương pháp lặp đi lặp lại thao tác sinh dữ liệu, sau đó chuyển cho hệ thống xử lý.
  - Các chương trình và framework được dùng để tạo ra kỹ thuật fuzzing được gọi là **fuzzer**.
  - Các kỹ thuật fuzzing:
    - **Kỹ thuật sinh mẫu**: biến đổi ngẫu nhiên, biểu diễn cấu trúc ngữ pháp, thuật toán định thời.
    - **Kỹthuậtphântíchđộng:** dynamic symbolic execution,cover feedback,dynamic taint analysis.
    - **Kỹ thuật phân tích tĩnh** bao gồm: control-flow analysis, data-flow slices.
  - **Ưu điểm:**
    - Tìm thấy những lỗ hỏng nghiêm trọng nhất về bảo mật hoặc những khiếm khuyết phần mềm.
    - Có thể tìm ra những lỗi không tìm được khi kiểm thử bị hạn chế về thời gian và nguồn lực.
    - Kết quả kiểm thử fuzzing hiệu quả hơn các phương pháp khác.
  - **Nhược điểm**
    - Chỉ riêng Fuzzing thì không thể xử lý hết các mối đe dọa anh ninh tổng thể hoặc các lỗi.
    - Kém hiệu quả với các lỗi mà không gây treo chương trình.
    - Không cung cấp nhiều kiến thức về hoạt động nội bộ của các phần mềm.
    - Chương trình có các đầu vào phức tạp đòi hỏi tốn thời gian tạo ra 1 fuzzer đủ thông minh.

Chương 7

- **Vulnerability – Lỗ hổng**: một lỗi trong bảo mật của hệ thống có thể cho phép kẻ tấn công sử dụng hệ thống 1 cách khác với thiết kế của hệ thống.
- **Exploit – khai thác/cách thức khai thác**
    - Động từ: lợi dụng 1 lỗ hổng để khiến hệ thống phản ứng lại theo 1 cách khác với thiết kế ban đầu.
    - Danh từ: công cụ, tập các lệnh hay mã nguồn dùng để lợi dụng 1 lỗ hổng. Tên gọi khác là **Proof of Concept (POC).**
- **Fuzzer**: một công cụ hoặc ứng dụng có chức năng thử tất cả hoặc 1 loạt các đầu vào không mong muốn cho một hệ thóng. Mục đích của fuzzer là xác định hệ thống có bug hay không để tránh bị khai thác mà không cần biết tất cả về hoạt động bên trong của hệ thống.
- **0 – day:** một cách thức khai thác một lỗ hổng chưa được tiết lộ công khai.

  Một lỗ hổng zero – day là một lỗi, điểm yếu hoặc bug có trong phần mềm, firmware hoặc phần cứng đã tiết lộ công khai nhưng chưa được vá. Các nhà nghiên cứu đã tiết lộ về lỗ hổng, và các nhà sản xuất và phát triển đã biết về vấn đề bảo mật đó nhưng chưa có bản vá hoặc bản cập nhật chính thức để giải quyết.
- **N – day**: một khi lỗ hổng đã được công bố công khai và các nhà sản xuất hay phát triển đã có các bản vá cho nó, lỗ hổng trở thành lỗ hổng đã biết hay còn gọi là n – day.
- **4 nhóm thanh ghi**:
    - **General purpose – mục đích chung**: dùng để thực hiện các phép tính toán thông thường (EAX, EBX, ECX). Một thanh ghi mục đích chung khác là ESP hay con trỏ stack.
    - **Segment**: thanh ghi CS, DS, SS registers, 16 bits, dùng để theo dõi các segment và cho phép tương thích ngược với các ứng dụng 16 bits.
    - **Control – điều khiển**: điều khiển hàm trong bộ xử lý. Một trong các thanh ghi quan trọng nhất là Extended Instruction Pointer (EIP) chứa địa chỉ của lệnh mã máy tiếp theo sẽ được thực thi.
    - **Khác:** các thanh ghi thêm. VD Extended Flags EFLAGS lưu các giá trị tương ứng với kết quả của các phép kiểm tra.

eax	Lưu giá trị trả về của hàm
esp Stack pointer	Định nghĩa stack frame, lưu động. Thành ghi 32 bit có thể bị thay đổi bởi nhiều lệnh (Push, Pop, Call, Ret), luôn trỏ đến địa chỉ thấp nhất trong stack.
ebp Base pointer	Định nghĩa stack frame, cố định. Thanh ghi 32 bit thường được dùng để tham chiếu đến các tham số và biến cục bộ trong stack frame.
eflags	Chứa các condition codes cho các lệnh nhảy có điều kiện
eip Instruction pointer	Lưu địa chỉ lệnh cần thực hiện tiếp theo, được lưu lại trên stack dưới tác dụng của lệnh CALL. Bất kỳ câu lệnh nhảy nào cũng có thể thay đổi trực tiếp giá trị eip.

- Quản lý bộ nhớ:
    - Có 3 dạng section khác nhau:
        - **.text** chứa các lệnh thực thi của chương trình, vùng read-only.
        - **.data** và **.bss** cho các biến toàn cục, trong đó, .data dành cho các biến static và đã khởi tạo, .bss dành cho các biến chưa khởi tạo.
    - **Stack** cấu trúc dữ liệu thoe dạng Last In First Out (LIFO), phát triển từ địa chỉ cao xuống địa chỉ thấp nếu có dữ liệu được thêm vào stack.
    - **Heap** cấu trúc dữ liệu cho cấp phát động, phát triển từ địa chỉ thấp lên địa chỉ cao nếu có dữ liệu được thêm vào.
    - Value 0x1A2B3C4D
        - Big Endian: 1A 2B 3C 4D
        - Little Endian: 4D 3C 2B 1A

Chương 8 – tràn số nguyên và chuỗi định dạng

- **Interger Overflow – lỗ hỏng tràn số nguyên** xảy ra khi một phép toán số học cố gắng tạo một giá trị số nằm ngoài phạm vi có thể hiểu được biểu diễn bằng một số chữ số nhất định – cao hơn giá trị lớn nhất hoặc thấp hơn giá trị nhỏ nhất có thể biểu diễn.

  => Hiểu đơn giản hơn là lỗ hỏng trong việc máy tính lưu trữ và tính toán các con số gây ra sai lệch và khiến cho chương trình chạy sai với mong muốn.

  - Cách phòng tránh:
    - Cấp phát kiểu dữ liệu đủ lớn cho giá trị lớn nhất và nhỏ nhất để lưu được tất cả các giá trị có thể tính ra.
    - Sắp xếp các phép toán cẩn thận, kiểm tra giá trị các biến.
    - Cẩn thận khi ép kiểu.
    - Đánh giá, kiểm thử mã nguồn, ứng dụng.
    - Lựa chọn ngôn ngữ hạn chế tràn số nguyên như python hoặc dùng các thư viện IntergerLib hoặc SafeInt cho C/C++
- Format String – lỗ hỏng chuỗi định dạng

Ký hiệu	Kiểu định dạng
d	Số nguyên (integer) - 4 bytes
u	Số nguyên không dấu – 4 bytes
x	Dạng hexan – 4 bytes
s	Kiểu chuỗi (string) – con trỏ
c	Ký tự - 1 byte

Ký hiệu	Độ dài	Kiểu định dạng
hh	1 byte	Ký tự
h	2 byte	Short int
l	4 byte	Long int
ll	8 byte	Long long int