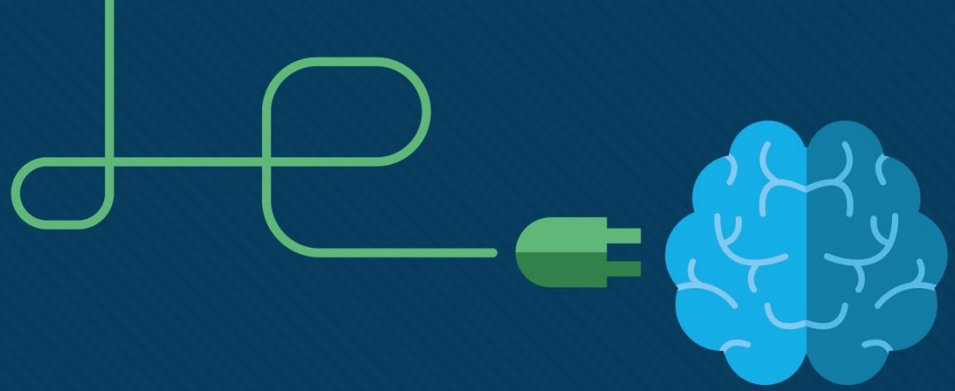
Bài 2: Qui trình thiết kế và phát triển phần mềm

NT521 - Lập trình an toàn và khai thác lỗ hổng phần mềm

DevNet Associate v1.0



Bài 02: Qui trình thiết kế và phát triển phần mềm

DevNet Associate v1.0

2.1. Phát triển phần mềm

Thiết kế và phát triển phần mềm: Giới thiệu

- Qui trình phát triển phần mềm thường được gọi là: "software development life cycle" (SDLC).
- SDLC không chỉ bao gồm việc lập trình, mà còn bao gồm: thu thập yêu cầu (gathering requirement), thiết kế (creating a proof of concept), kiểm thử (testing), và gỡ lỗi (fixing bugs).

Software Development Life Cycle (SDLC)

- SDLC là một qui trình phát triển phần mềm, bắt đầu từ việc "lên ý tưởng" cho đến kết thúc ở khâu "phân phối phần mềm". Qui trình gồm 6 giai đoạn. Mỗi giai đoạn có đầu vào là kết quả của giai đoạn ngay trước nó.

- Mặc dù mô hình Thác nước (Waterfall) vẫn được sử dụng rộng rãi, tuy nhiên nó dần được thay thế bởi các mô hình SDLC khác có tính linh động, thích ứng hơn. Mục đích các mô hình mới này nhằm tạo ra phần mềm tốt hơn, nhanh hơn và ít chi phí hơn.

- Các phương pháp phát triển phần mềm linh hoạt được gọi chung là "Agile Development".

SDLC: Các giai đoạn

- Thu thập và phân tích yêu cầu:
  - **“Các vấn đề hiện tại là gì? “Chúng ta muốn gì? “”.**
- Thiết kế:
  - **“Làm thế nào chúng xây được những gì chúng ta muốn?”**
- Phát triển:
  - **“Tạo những gì chúng ta muốn”.**
- Kiểm thử:
  - **“Chúng ta đã đạt được những gì chúng ta muốn chưa? “**
- Triển khai:
  - **“Hãy bắt đầu sử dụng những gì chúng ta đã xây dựng”**
- Bảo trì/Duy trì:
  - **“Hãy giữ cho sản phẩm của chúng ta ổn định”**

Thu thập và phân tích yêu cầu

- Giai đoạn phân tích và yêu cầu bao gồm việc khám phá tình hình hiện tại, nhu cầu và ràng buộc của các bên liên quan, cơ sở hạ tầng hiện tại, v.v. và xác định vấn đề cần giải quyết bằng phần mềm.
- Sau khi thu thập các yêu cầu, nhóm phân tích kết quả để xác định những điều sau:
    - Có thể phát triển phần mềm theo những yêu cầu này không và nó có thể được thực hiện với ngân sách không?
    - Có bất kỳ rủi ro nào đối với lịch trình phát triển không, và nếu có, chúng là gì?
    - Phần mềm sẽ được kiểm tra như thế nào?
    - Phần mềm sẽ được chuyển giao khi nào và như thế nào?
- Vào cuối giai đoạn này, phương pháp thác nước cổ điển đề xuất tạo tài liệu “Đặc tả yêu cầu phần mềm” (Software Requirement Specification - SRS), trong đó nêu rõ các yêu cầu và phạm vi phần mềm, đồng thời xác nhận điều này một cách tỉ mỉ với các bên liên quan.

Thiết kế và hiện thực

Thiết kế (Design)

- Trong giai đoạn Thiết kế, các kiến trúc sư và nhà phát triển phần mềm thiết kế phần mềm dựa trên SRS được cung cấp.
- Ở cuối giai đoạn, đội ngũ thiết kế tạo ra các tài liệu thiết kế mức cao - High-Level Design (HLD); và thiết kế Mức Thấp - Low-Level Design (LLD).

Hiện thực (implementation)

- Giai đoạn hiện thực còn được gọi là giai đoạn lập trình phần mềm (coding)
- Vì tất cả các thành phần và mô-đun được xây dựng trong giai đoạn này, nên đó là giai đoạn dài nhất của vòng đời.
- Vào cuối giai đoạn, mã chức năng (functional code) thực hiện tất cả các yêu cầu của khách hàng đã sẵn sàng để được kiểm tra.

Kiểm thử, Triển khai, Bảo trì

Kiểm thử (Testing)

- Thực hiện kiểm thử mã nguồn đã phát triển trong môi trường kiểm thử.
- Bao gồm các dạng kiểm thử: Functional testing, integration testing, performance testing and security testing.
- Quá trình kiểm tra tiếp tục cho đến khi tất cả các mã đều không có lỗi và vượt qua tất cả các bài kiểm tra. Vào cuối giai đoạn này, một phần mềm chất lượng cao, không có lỗi, đã sẵn sàng để đưa ra sử dụng (production).

Triển khai (Deployment)

- Phần mềm được cài đặt trên môi trường sản phẩm sẽ chạy thực tế (production environment)
- Vào cuối giai đoạn, người quản lý sản phẩm phát hành phần mềm cuối cùng cho người dùng cuối.

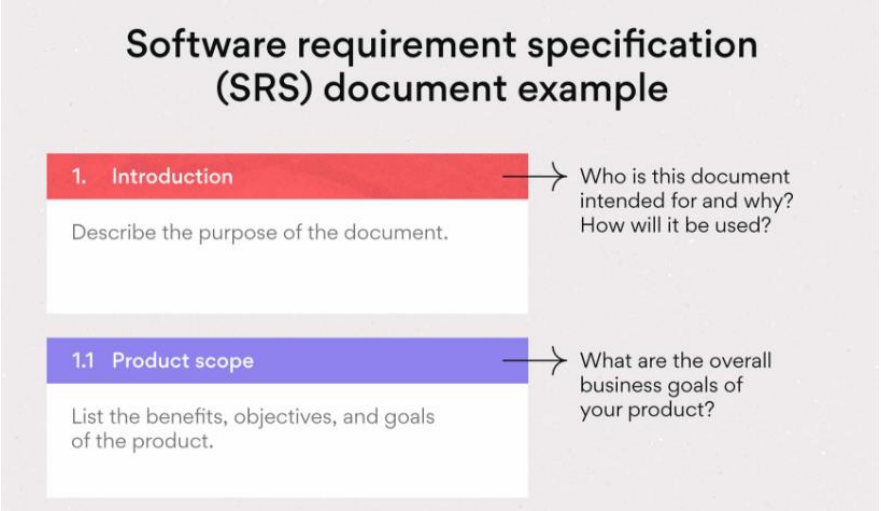
Duy trì/Bảo trì (Maintenance)

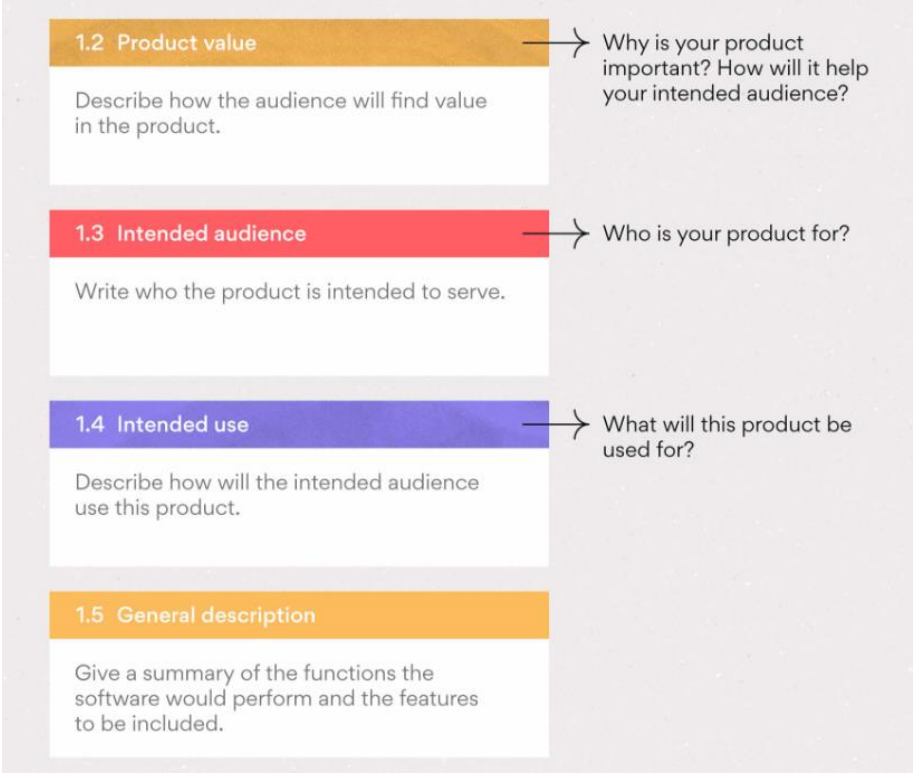
- Trong suốt giai đoạn Bảo trì, đội ngũ sẽ:
  - Cung cấp sự hỗ trợ cho khách hàng
  - Gỡ lỗi được tìm thấy trên sản phẩm phần mềm
  - Cải tiến phần mềm
  - Thu thập thông tin mới từ khách hàng
- Cuối cùng, nhóm sẽ làm việc trên vòng lặp tiếp theo của SDLC và phiên bản tiếp theo của phần mềm.

SDLC

Giai đoạn	Đầu ra	Chịu trách nhiệm
REQUIREMENT/PLANNING Lấy yêu cầu/Lên kế hoạch	Tài liệu Yêu cầu KH **Tài liệu đề xuất giải pháp (Proposal)** **Tài liệu Quản lý dự án (Project Plan)**	Sales, BA, PM, Customer
ANALYSIS Phân tích	**Tài liệu Đặc tả yêu cầu**	PM, BA, Senior, Software Architect
DESIGN Thiết kế	**Hồ sơ thiết kế, Flowchart, Diagram** …	Designer, BA
CODING Lập trình	Sản phẩm (Source Code)	Developers
TESTING Kiểm thử	Test case, Test plan, Bug Report …	QA, Tester
DEPLOYMENT & MAINTENANCE Cài đặt và bảo trì	Tài liệu Đặc tả hệ thống Tài liệu Hướng dẫn sử dụng Tài liệu Cấu hình và cài đặt	DevOps

Tài liệu đặc tả phần mềm (SRS)

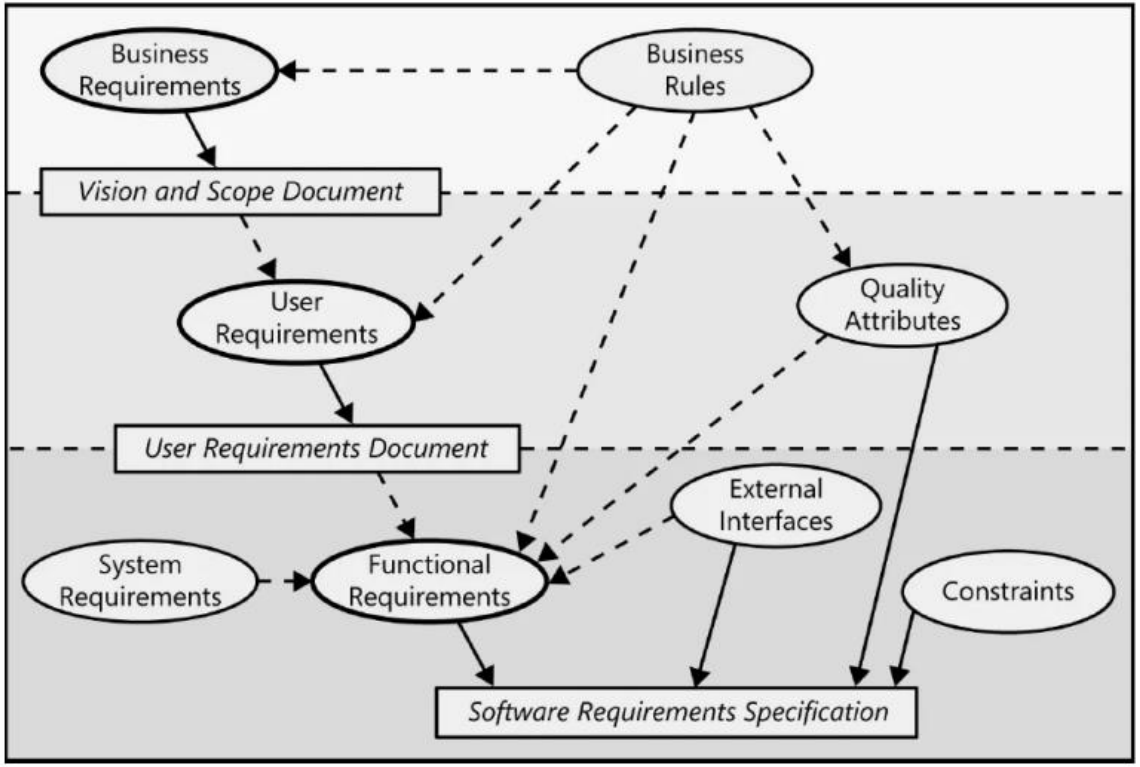




Tài liệu đặc tả phần mềm (SRS)

Table of Contents

Tài liệu đặc tả phần mềm (SRS)

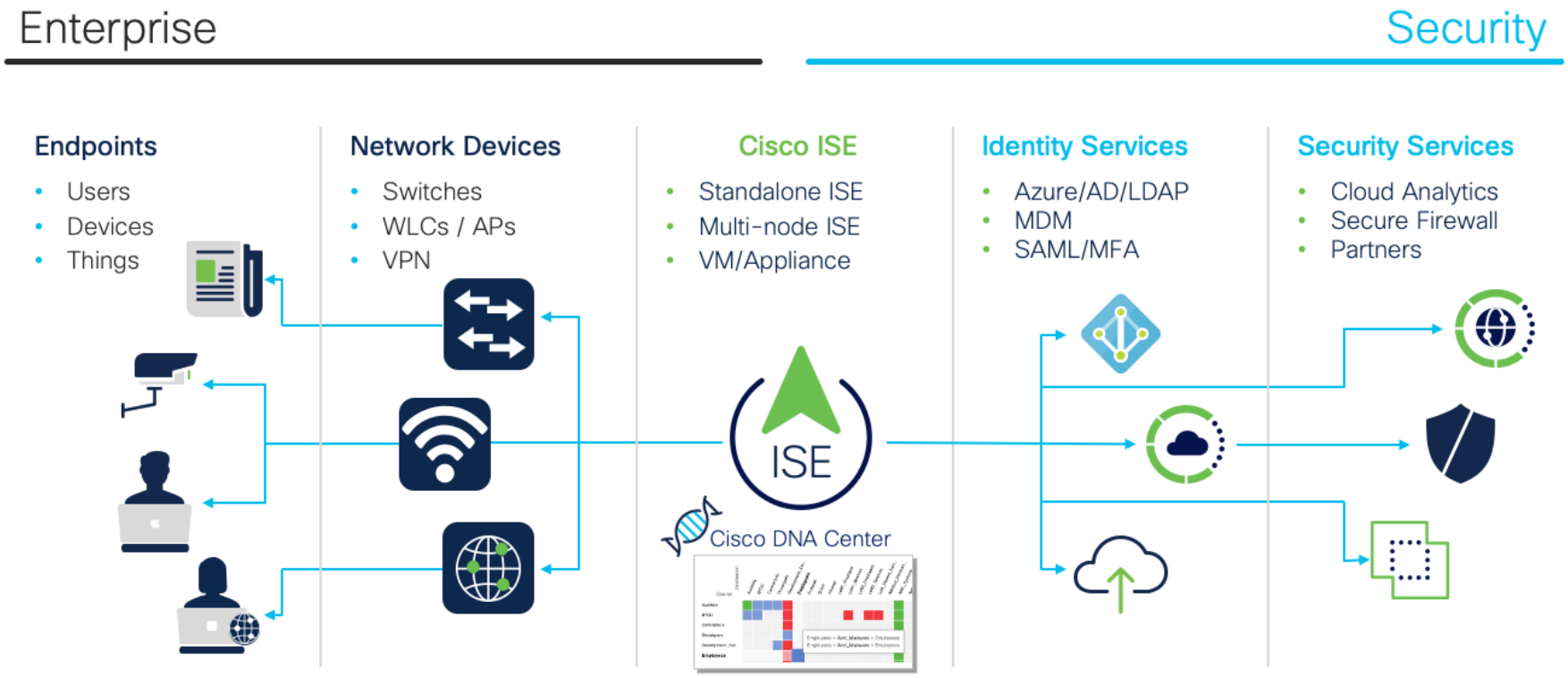


Mô hình mối quan hệ giữa một số loại thông tin yêu cầu. Nguồn: Software Requirements by Karl Wiegers Joy Beatty

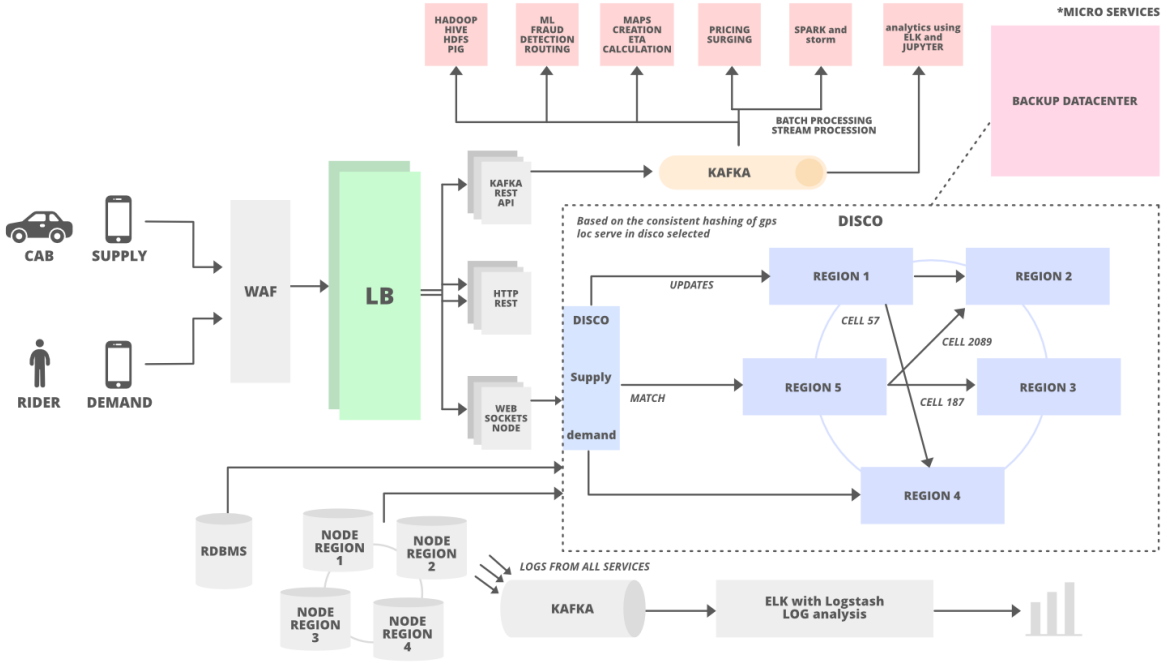
Thiết kế: High-Level Design (HLD)

- Thiết kế cấp cao hay HLD đề cập đến **hệ thống tổng thể**, một thiết kế bao gồm **mô tả về kiến trúc và thiết kế hệ thống**; và là một thiết kế hệ thống chung bao gồm:
  - Kiến Trúc Hệ Thống
  - Thiết kế cơ sở dữ liệu
  - Mô tả ngắn gọn về hệ thống, dịch vụ, nền tảng và mối quan hệ giữa các mô-đun.

Thiết kế: High-Level Design (HLD)



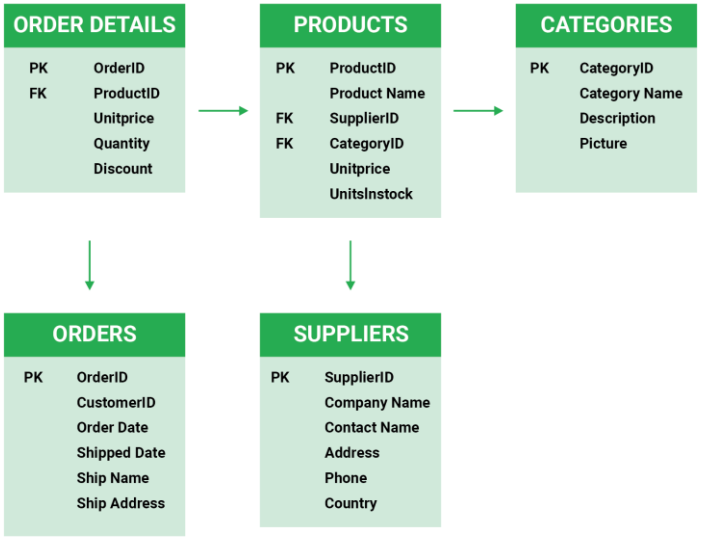
Thiết kế: High-Level Design (HLD)



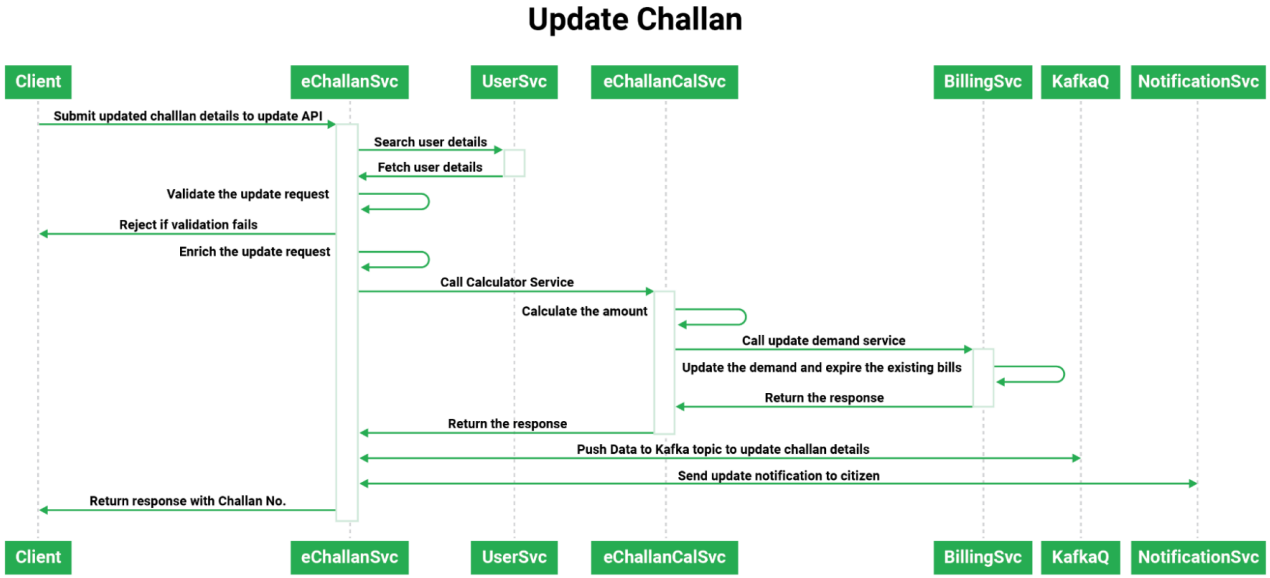
Thiết kế: Low-Level Design (LLD)

- Thiết kế cấp thấp (LLD) đề cập đến quá trình xác định **thiết kế chức năng, chi tiết của hệ thống** hoặc **thành phần** phần mềm. Nó liên quan đến việc xác định ***các mô-đun riêng lẻ, cấu trúc dữ liệu, thuật toán, giao diện và đầu vào/đầu ra*** của hệ thống. Nó bao gồm:
  - Data Flow Design
  - Data Structure Design
  - Algorithm Design
  - Component-Level Design
  - Architectural Design
- Mục đích của thiết kế cấp thấp là cung cấp một mô tả rõ ràng và chính xác về cách hệ thống sẽ hoạt động và cách các thành phần khác nhau của nó sẽ tương tác với nhau.

Thiết kế: Low-Level Design (LLD)



Thiết kế: Low-Level Design (LLD)

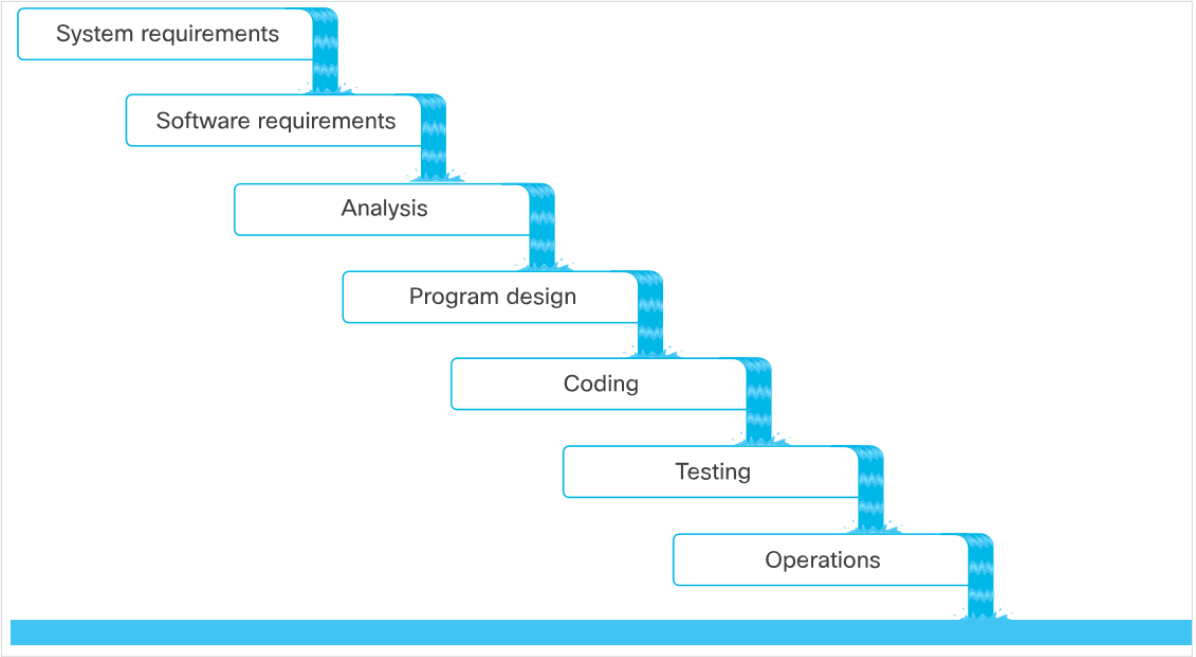


Phương pháp luận phát triển phần mềm

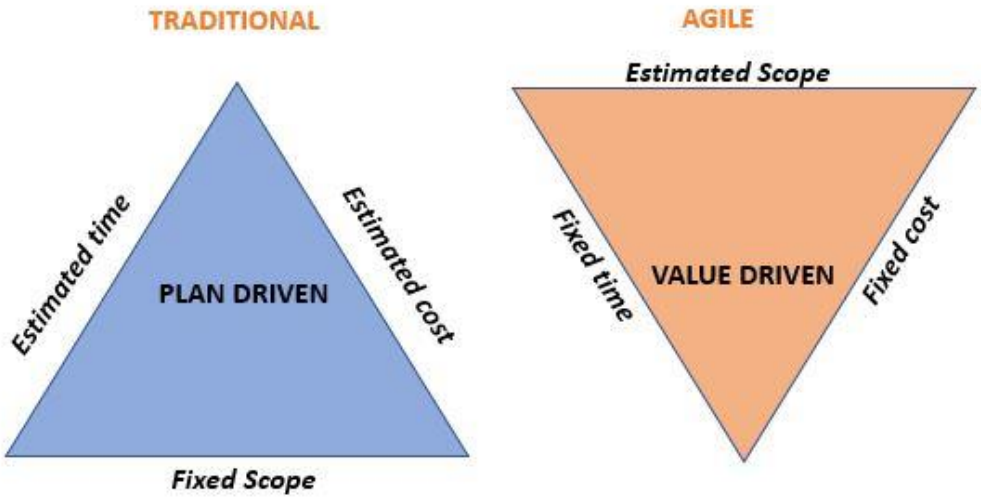
- Phương pháp luận phát triển phần mềm (software development methodology) thường được biết đến với thuật ngữ mô hình Chu kỳ Phát triển phần mềm (Software Development Life Cycle).
- Có 03 phương pháp phát triển phần mềm phổ biến:
  - Traditional SDLC: Waterfall
  - Agile SDLC: Agile scrum
  - Agile SDLC: Lean
- Việc sử dụng các phương pháp phát triển phần mềm, phụ thuộc vào:
  - Loại dự án (project)
  - Độ dài/thời gian thực hiện dự án (length of project)
  - Số lượng nhân lực của đội ngũ phát triển phần mềm (size of team).

Phương pháp phát triển phần mềm Waterfall

- Mô hình phát triển phần mềm Thác nước (Waterfall) nguyên bản được giới thiệu bởi Winston W. Royce.
- Mô hình nguyên bản của Waterfall bao gồm 7 giai đoạn:

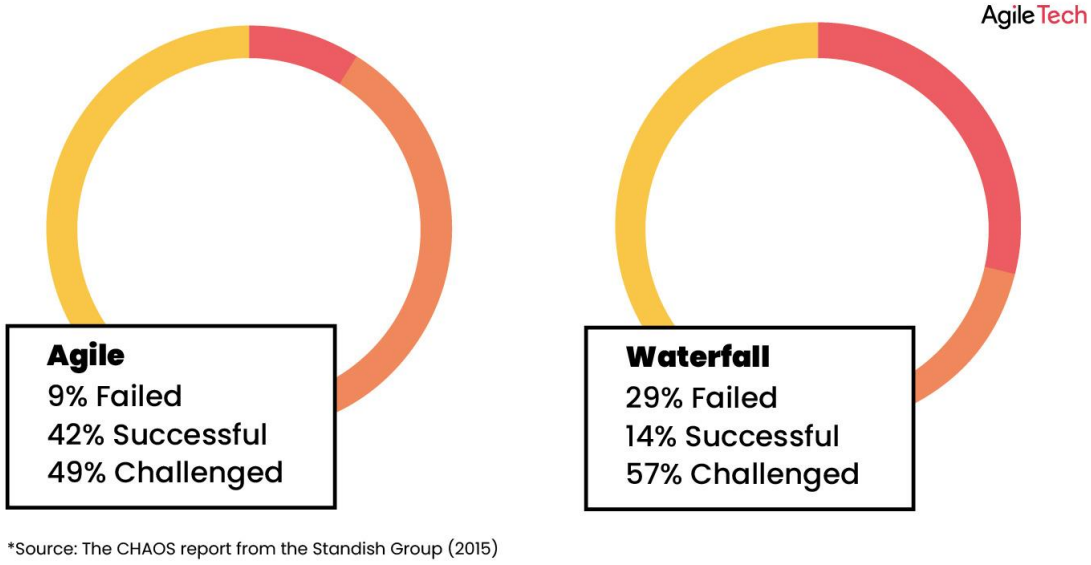


Phương pháp luận phát triển phần mềm



Phương pháp luận phát triển phần mềm

- Thống kê sự thành công của dự án phần mềm dựa trên 02 phương pháp.



Phương pháp phát triển phần mềm truyền thống

- Khi nào nên chọn mô hình phát triển phần mềm truyền thống (Plan-driven):
  - Các giai đoạn và yêu cầu được xác định rõ rang
  - Đưa ra sản phẩm mẫu chạy thử (protype) là một việc khó
  - Biết trước tất cả các yêu cầu trước khi bắt đầu dự án
  - Việc thay đổi không phải là ưu tiên
  - Các vai trò (role) được định nghĩa chat chẽ
  - Được khuyến nghị sử dụng trong các dự án phát triển với thời gian dài, chi phí cao
  - Tài liệu hóa chi tiết, toàn bộ quá trình phát triển
- Một số mô hình phát triển phần mềm truyền thống (Planning-driven)
  - Waterfall
  - Iterative Model
  - Rational Unified Process (RUP)
  - Spiral Model

Thảo luận

- Hãy liệt kê những thành phần cần thiết nhất của một quy trình phần mềm.
- Một quy trình phần mềm được một công ty đưa ra áp dụng trong toàn công ty được gọi là **organization process** và quy trình được một dự án trong công ty áp dụng được gọi là **project process**. Hãy giải thích sự khác nhau giữa hai loại quy trình này.

Phương pháp phát triển phần mềm Agile

- Phương pháp Agile là phương pháp phát triển phần mềm tập trung vào tính linh động và hướng khách hàng (customer-focused).
- Nhóm 17 nhà phát triển phần mềm đã đưa ra khái niệm "Manifesto for Agile Software Development", còn được gọi là Agile Manifesto, vào năm 2001. Theo Agile Manifesto, giá trị của phương pháp Agile bao gồm:
  - Chú trọng vào qui trình làm việc và tương tác của thành viên hơn là công cụ phần mềm hỗ trợ
  - Phần mềm hoạt động như yêu cầu hơn là chú trọng vào tài liệu thiết kế
  - Việc cộng tác với khách hàng được thực hiện qua các hợp đồng thoả thuận
  - Đáp ứng sự thay đổi so với việc tuân theo một kế hoạch
- Agile manifesto bao gồm 12 nguyên tắc khác nhau:

Agile Manifesto Principles			
Tập trung vào khách hàng	Hợp tác giữa các bên	Ưu tiên phần mềm hoạt động	Đơn giản
Chấp nhận sự thay đổi và thích ứng	Đội ngũ có động lực	Làm việc với tốc độ bền vững	Các nhóm tự tổ chức
Thường xuyên phân phối các phiên bản phần mềm hoạt động	Trao đổi trực tiếp	Môi trường linh hoạt, tốc độ (Agile)	Liên tục cải thiện

Các phương pháp Agile

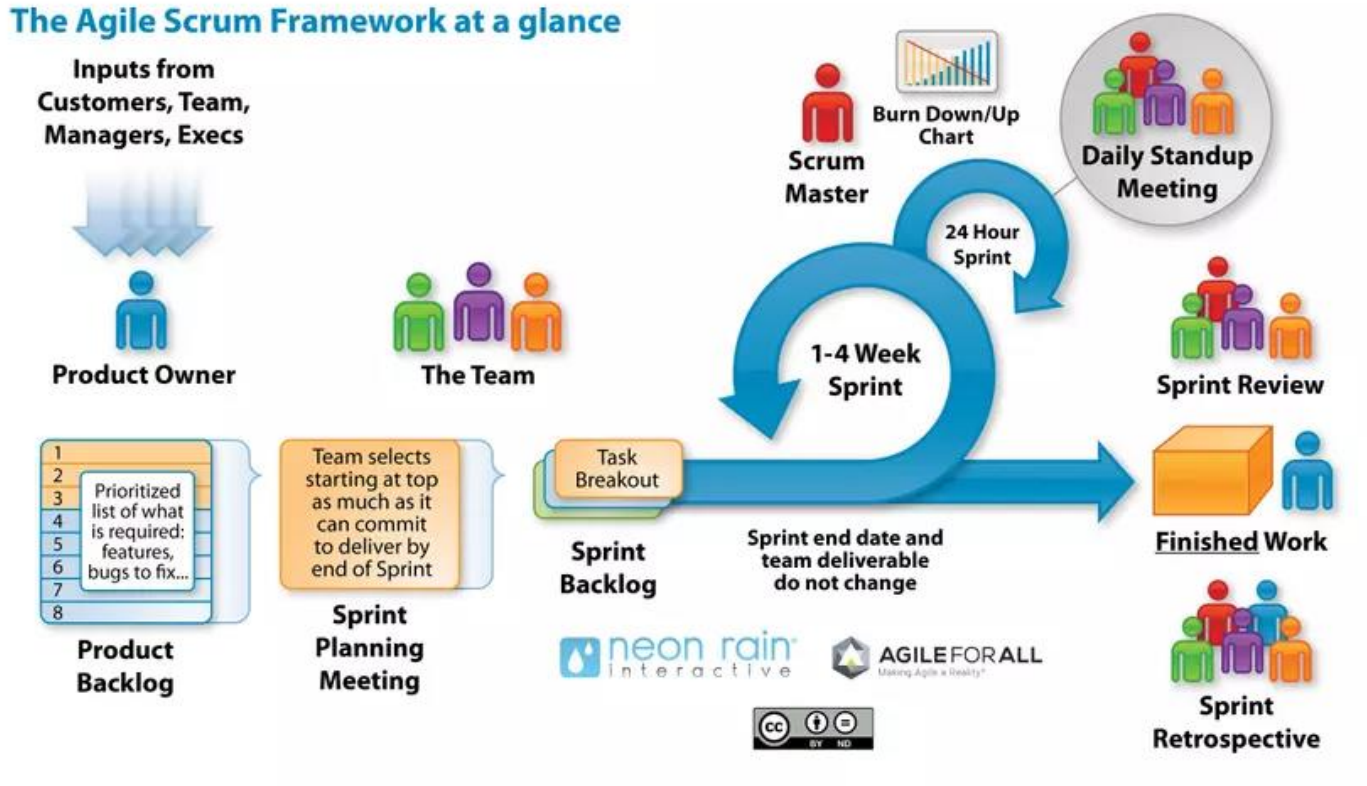
- Các phương pháp Agile phổ biến:
  - **Agile Scrum:** Scrum tập trung vào các nhóm nhỏ, tự tổ chức, nhóm họp hàng ngày trong thời gian ngắn và làm việc trong các cuộc chạy nước rút (sprint) lặp đi lặp lại.
  - **Lean:** Phương pháp Lean nhấn mạnh vào việc loại bỏ nỗ lực hao phí trong việc lập kế hoạch và thực hiện, đồng thời giảm tải nhận thức của lập trình viên.
  - **Extreme Programming (XP):** XP hướng tới giải quyết các loại vấn đề cụ thể về chất lượng mà các 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 dựng theo từng tính năng.

Các phương pháp Agile

- **Sprints**
  - Sprint là một khoảng thời gian cụ thể, thường từ 2-4 tuần, trong đó, mỗi nhóm đảm nhận nhiều nhiệm vụ (còn được gọi là User Story - câu chuyện của người dùng) mà họ cảm thấy có thể hoàn thành. Khi sprint kết thúc, phần mềm sẽ hoạt động và có thể phân phối được.
  - Khoảng thời gian của sprint được xác định trước khi quá trình bắt đầu và hiếm khi thay đổi.
- **Backlog**
  - Công việc tồn đọng (BackLog) bao gồm tất cả các tính năng của phần mềm, nằm trong danh sách ưu tiên.
- **User stories**
  - Câu chuyện người dùng (user story) là một tuyên bố đơn giản về những gì người dùng (hoặc một vai trò) cần và tại sao. Mỗi câu chuyện của người dùng phải đủ nhỏ để một nhóm có thể hoàn thành nó trong một lần chạy nước rút (sprint).
  - Mẫu gợi ý của 01 user story như sau:

As a <user|role>, I would like to <action>, so that <value|benefit>

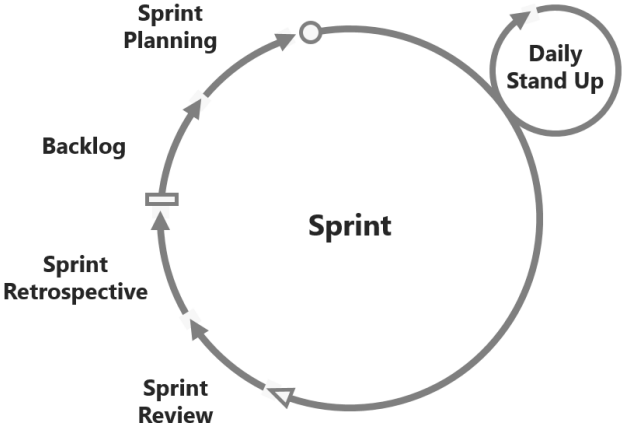
Các phương pháp Agile



Các phương pháp Agile

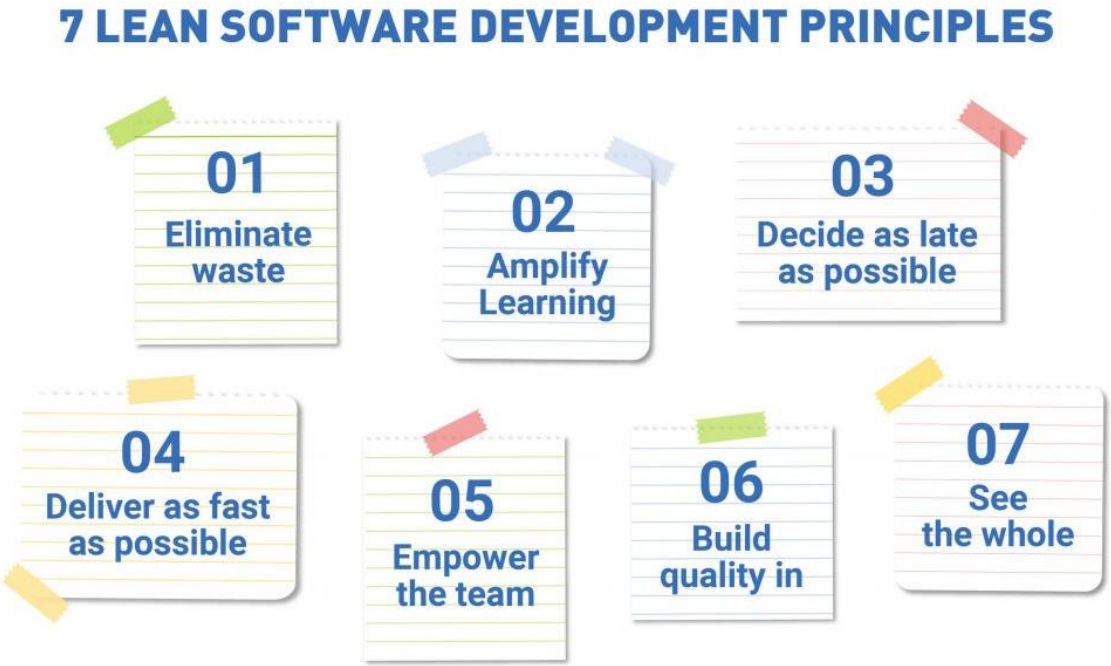
Nhóm Scrum (Scrum Teams)

- Nhóm Scrum có khả năng làm việc phối hợp (cross-functional), hợp tác, tự quản lý và tự trao quyền.
- Các nhóm scrum không được lớn hơn 10 cá nhân. Một Scrum Team điển hình thì nên trong khoảng từ 3-9 người, bao gồm cả ***Product Owner*** và ***Scrum Master***.
  - Product Owner – “What” Guy
  - Scrum Master – “How” Guy
- Scrum master nên có một cuộc họp trực tiếp hàng ngày với nhóm vào một thời gian cố định hàng ngày, không quá 15 phút.
- Mục đích là để "điểm lại" các nhiệm vụ (task) quan trọng đã hoàn thành, đang thực hiện hoặc sắp bắt đầu.



Phương pháp phát triển phần mềm Lean

- Phát triển phần mềm Lean dựa trên nguyên tắc **Lean Manufacturing**, tập trung vào việc giảm thiểu hao phí (waste) và tối đa hoá giá trị đối với khách hàng.



Phương pháp phát triển phần mềm Lean

- Qui tắc Lean bao gồm 7 bước, được nêu trong sách “Lean Software Development: An Agile Toolkit,” cụ thể như sau:
  - Giảm thiểu hao phí (Eliminate waste): xác định và triệt tiêu hao phí trong qui trình SDLC
  - Tăng cường việc học tập (Amplify learning): thông qua việc thực hành; thông qua sự tương tác và tích lũy kinh nghiệm từ những gì khách hàng muốn/thích.
  - Quyết định càng muộn càng tốt (Decide as late as possible)
  - Phân phối càng nhanh càng tốt (Deliver as fast as possible)
  - Trao quyền cho nhóm (Empower the team): Tạo sự hợp tác làm việc phối hợp từ bên trong nhóm hơn là từ người quản lý cấp trên
  - Xây dựng sự toàn vẹn từ bên trong (Build integrity in)
  - Tối ưu hóa toàn bộ (Optimize the whole): suy nghĩ cho thời gian dài hạn, ưu tiên Khách hàng > Công ty > Nhóm làm việc > Cá nhân

Phương pháp phát triển phần mềm Lean

Giảm thiểu hao phí (waste)

- Đây là nguyên tắc nền tảng cơ bản nhất của Lean.
- Có 07 dạng hao phí khác nhau trong phát triển phần mềm:
  - Hoàn thành công việc một phần (Partially done work)
  - Các quy trình gia tăng (Extra processes)
  - Các chức năng gia tăng (Extra features)
  - Hoán đổi nhiệm vụ công việc (Task switching)
  - Chờ đợi (Waiting)
  - Di chuyển (Motion)
  - Khuyết điểm (Defects)

Phương pháp phát triển phần mềm Lean

Tăng cường sự học hỏi với những sprint ngắn (Amplify Learning with Short Sprints)

- To be able to fine tune a software, there should be frequent short iterations of working software. This enables the following:
  - Developers learn faster
  - Customers can give feedback sooner
  - Features can be adjusted so that they bring customers more value

Đưa ra quyết định càng muộn càng tốt (Decide as Late as Possible)

- When there is uncertainty, it is best to delay the decision-making until as late as possible in the process. This is because it is better to base decisions on facts rather than opinions or speculations.

Phân phối sản phẩm càng nhanh càng tốt (Deliver as Fast as Possible)

Deliver As Fast as Possible	
Enables customers to provide feedback	Doesn't allow customers to change their mind
Enables developers to amplify learning	Makes everyone take decisions faster
Provides customers the required features	Produces less waste

Phương pháp phát triển phần mềm Lean

Empower the Team

- Each person must be allowed to make decisions in the area of their own expertise.

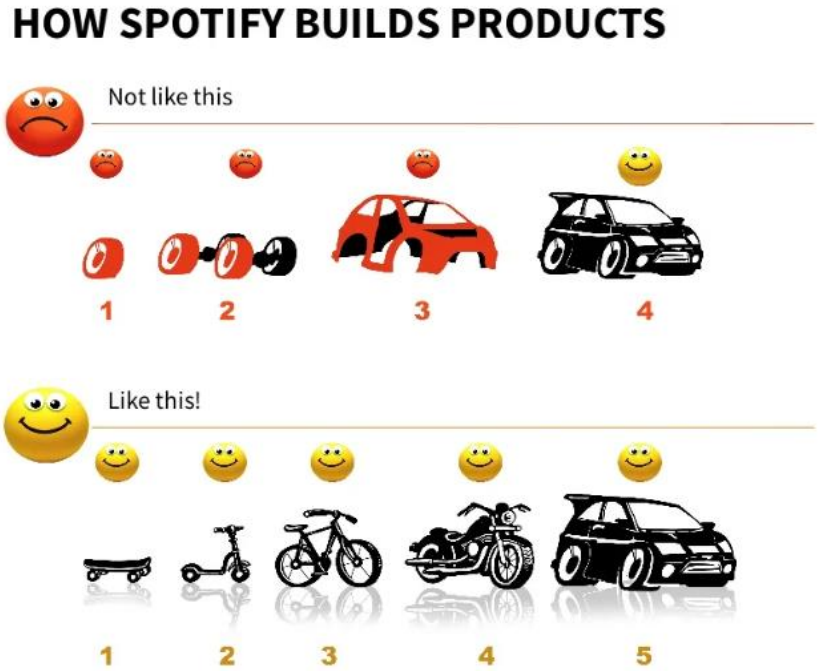
Build Integrity In

- Integrity for the software is when the software addresses the customer's needs as well as maintains the usefulness for the customer.

Optimize the Whole

- The software must be built cohesively. The value of the software will suffer if each expert only focuses on their expertise and doesn't consider the ramifications of their decisions on the rest of the software.

Xây dựng sản phẩm với sự toàn vẹn từ bên trong: Build Integrity In



- Khi khách hàng nói: “Đúng. Đây chính xác là những gì tôi cần, làm sao mà họ biết” – Tính toàn vẹn nhận thức (perceived integrity)
- Khi phần mềm có thể co giãn – phát triển dần dần tương ứng với nhu cầu của khách hàng và thời gian – Tính toàn vẹn khái niệm (Conceptual integrity)

Thảo luận

- Phương pháp Agile có phù hợp với những dự án có yêu cầu về mức độ an toàn và chính xác cao (critical systems) hay không? Giải thích lí do.

Bài tập

Giả sử bạn đang là trưởng dự án (project manager) cho một công ty chuyên về phát triển và cung cấp các dịch vụ phần mềm hỗ trợ nông dân trong việc quản lý sản xuất và phân phối sản phẩm nông nghiệp. Công ty có tên *Bạn Nhà Nông* hay *BNN*. Khách hàng tiềm năng của công ty là công ty trách nhiệm hữu hạn *Hai Lúa*. Công ty Hai Lúa có kế hoạch phát triển một hệ thống thương mại điện tử *hailua.com.vn* nhằm cung cấp các dịch vụ cho việc quảng bá, mua và bán các sản phẩm nông nghiệp ở Việt Nam. Mô hình hoạt động giống như ebay.com. Giám đốc công ty BNN phân công bạn phụ trách dự án này.

Dự án tương đối lớn với số thành viên khoảng 15-20 người và thời gian thực hiện từ 8 tháng đến 1 năm.

- **a.** Hãy nêu ra các điểm thuận lợi và bất lợi nếu áp dụng phương pháp Waterfall, Rational Unified Process (RUP), Lean, và Scrum.
- **b.** Hãy đưa ra các giả định (hay điều kiện) của dự án để mỗi phương pháp trên là phù hợp nhất với dự án.

3.2 Mẫu thiết kế phần mềm

Giới thiệu

- Mẫu thiết kế phần mềm - **Software design patterns** là các giải pháp để giải quyết các vấn đề thường gặp trong phát triển phần mềm.
- Các mẫu thiết kế không phụ thuộc vào ngôn ngữ lập trình.
- Mẫu thiết kế phần mềm được định nghĩa như sau:
  - Program to an interface, not an implementation.
  - Favor object composition over class inheritance.
- Các mẫu thiết kế phần mềm đã được chứng minh là hiệu quả, vì vậy sử dụng chúng có thể đẩy nhanh quá trình phát triển phần mềm.

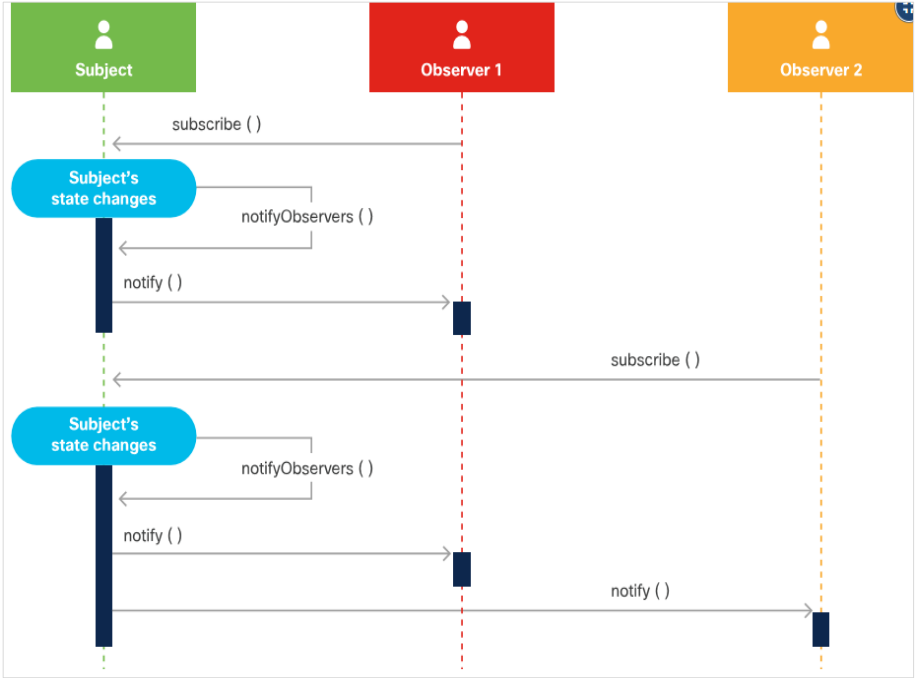
[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (known as the Gang of Four (GoF)), “Design Patterns - Elements of Reusable Object-Oriented Software” 1994

Các mẫu thiết kế ban đầu

- Gang of Four chia các mẫu thiết kế thành 3 nhóm chính:
  - Creational – Nhóm khởi tạo
  - Structural – Nhóm cấu trúc
  - Behavioral – Nhóm hành vi
- Liệt kê 23 mẫu thiết kế phần mềm.
- 2 mẫu thiết kế phổ biến nhất là:
  - Mẫu thiết kế Observer (thuộc nhóm Hành vi)
  - Mẫu Model-View-Controller (MVC)

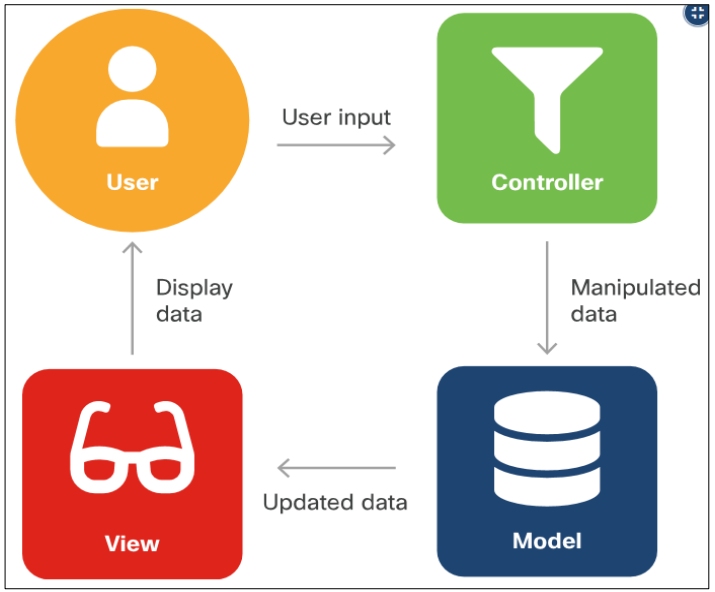
Mẫu thiết kế Observer Design Pattern

- **Mẫu thiết kế observer** là mẫu theo dạng đăng ký – thông báo (subscription - notification), cho phép các đối tượng nhận các sự kiện khi có thay đổi ở một đối tượng mà chúng đang quan sát.
- Hiện thực cơ chế đăng ký:
  - Đối tượng cần có khả năng lưu trữ các đối tượng đang theo dõi nó (các observer).
  - Đối tượng cần có phương thức để thêm và xóa các observer.
- Ưu điểm của thiết kế Observer: observer có thể nhận được theo thời gian thực khi có thay đổi xảy ra.
- Cơ chế đăng ký thường cho hiệu suất tốt hơn các lựa chọn khác, ví dụ cơ chế khảo sát (polling)



Model-View-Controller (MVC)

- Mẫu **Model-View-Controller (MVC)** nhằm đơn giản hóa việc phát triển các ứng dụng có giao diện người dùng.
- MVC trừu tượng hóa mã nguồn (code) và chức năng thành 3 thành phần khác nhau:
  - **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 dung và thay đổi nó để phù hợp với định dạng cho model hoặc view.
- Ưu điểm của MVC: mỗi thành phần có thể được build song song.



3.3 Hệ thống quản lý phiên bản

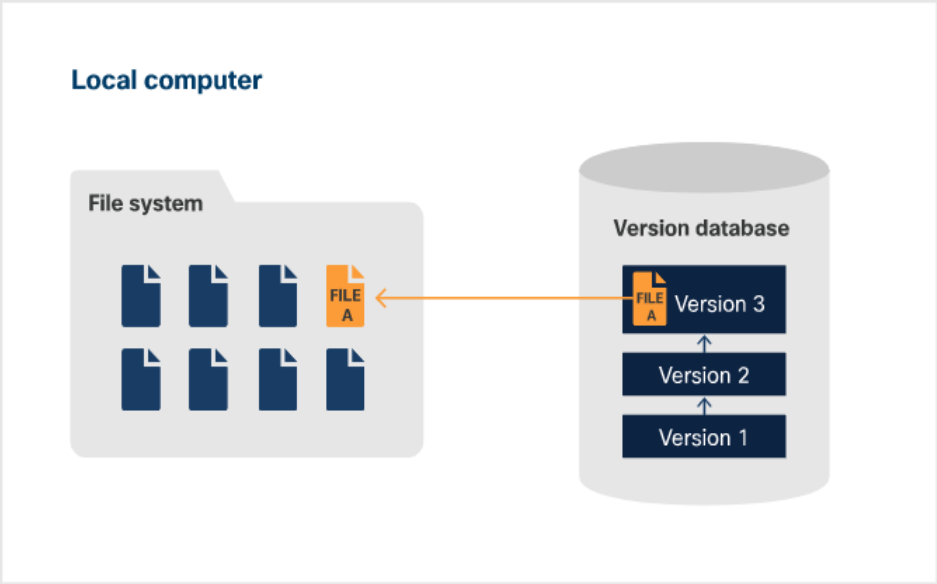
Các loại Hệ thống quản lý phiên bản

- Tên gọi: Version control, version control systems, revision control,source control.
- 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 (Accountability and visibility)
  - Làm việc độc lập
  - Đảm bảo an toàn
  - Làm việc từ bất kỳ đâu
- Có 3 loại Hệ thống quản lý phiên bản:
  - cục bộ - Local
  - Tập trung - Centralized
  - Phân tán - Distributed

Các loại Hệ thống quản lý phiên bản (tt)

Hệ thống quản lý phần mềm cục bộ - Local Version Control System (LVCS)

- LVCS sử dụng cơ sở dữ liệu đơn giản để lưu vết tất cả các thay đổi trên tập tin.
- Thông thường, hệ thống lưu trữ các khác biệt (delta) giữa 2 phiên bản của tập tin.
- Khi cần khôi phục phiên bản cũ, khác biệt được, các khác biệt được đảo ngược để quay lại phiên bản được yêu cầu.

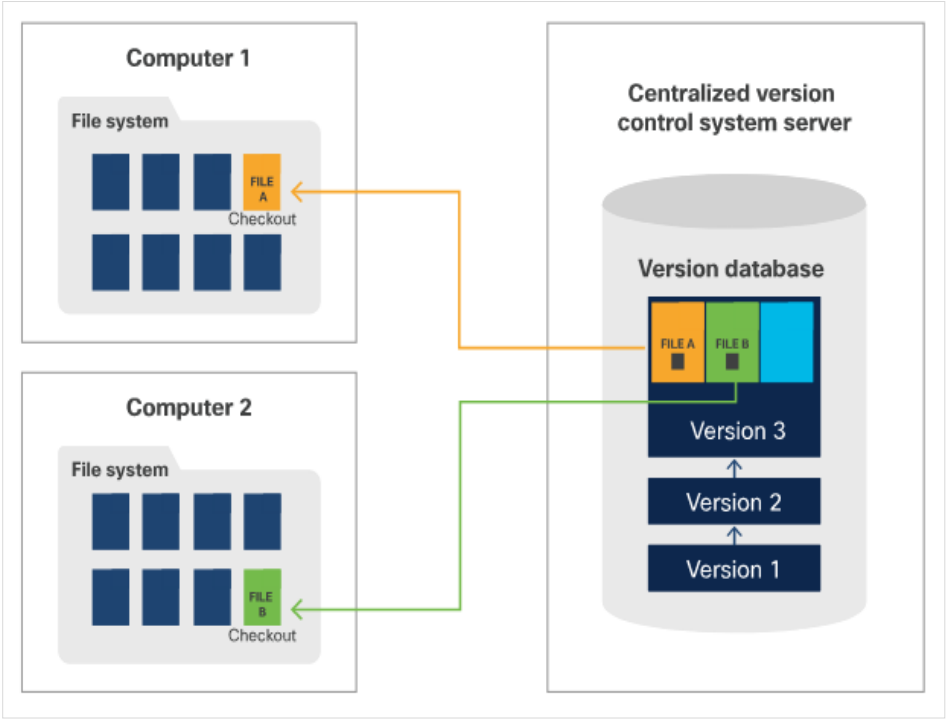


Các loại Hệ thống quản lý phiên bản (tt)

Hệ thống quản lý phiên bản tập trung

Centralized Version Control System (CVCS)

- CVCS sử dụng mô hình server-client.
- Kho lưu trữ được đặt trên 1 vị trí trung tâm, là 1 server.
- CVCS chỉ có 1 cá nhân được làm việc trên 1 file nhất định tại 1 thời điểm.
- Một cá nhân cần kiểm tra để khóa tệp và thực hiện các thay đổi cần thiết, sau đó check in khi hoàn thành.

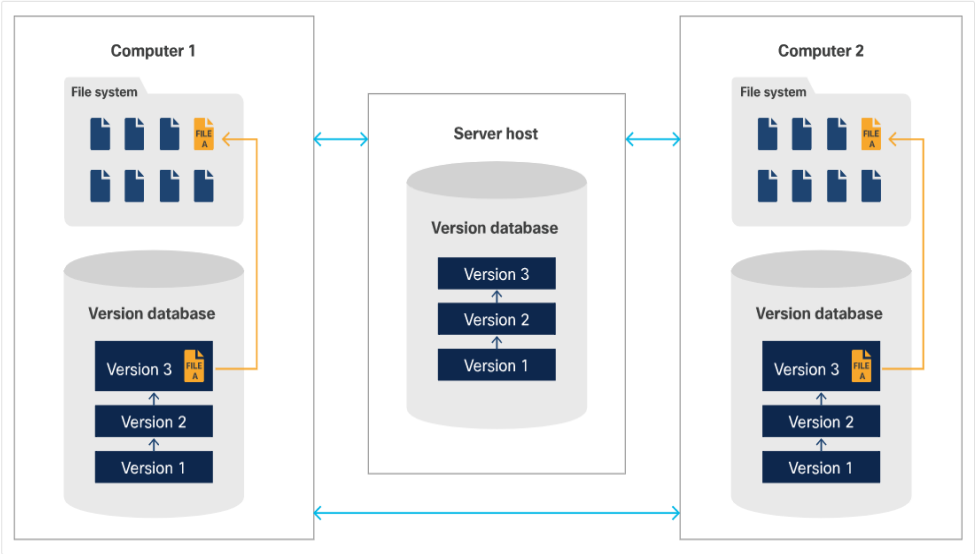


Các loại Hệ thống quản lý phiên bản (tt)

Hệ thống quản lý phiên bản phân tán

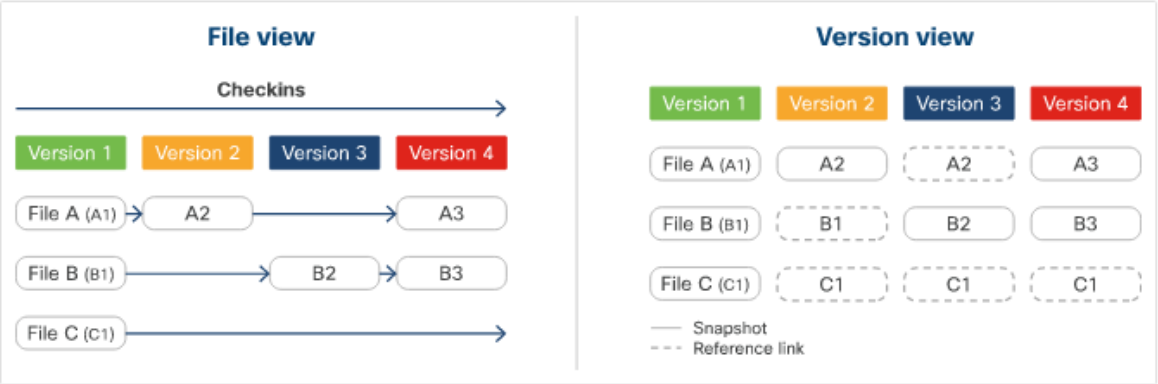
Distributed Version Control System (DVCS)

- DVCS là mô hình ngang hàng peer-to-peer.
- Kho lưu trữ nằm trên hệ thống client, nhưng thường được lưu trong dịch vụ hosting kho lưu trữ.
- Trong DVCS, mọi cái nhân có thể làm việc trên bất kỳ file nào, bất kỳ lúc nào, do chỉ có bản sao file ở cục bộ đang được sửa đổi. Do đó, không cần phải khóa file.
- Khi cá nhân đã thay đổi xong, họ đẩy file lên kho lưu trữ chính trong dịch vụ hosting kho lưu trữ, và hệ thống quản lý phiên bản kiểm tra xem có xung đột nào giữa các thay đổi hay không.



Git

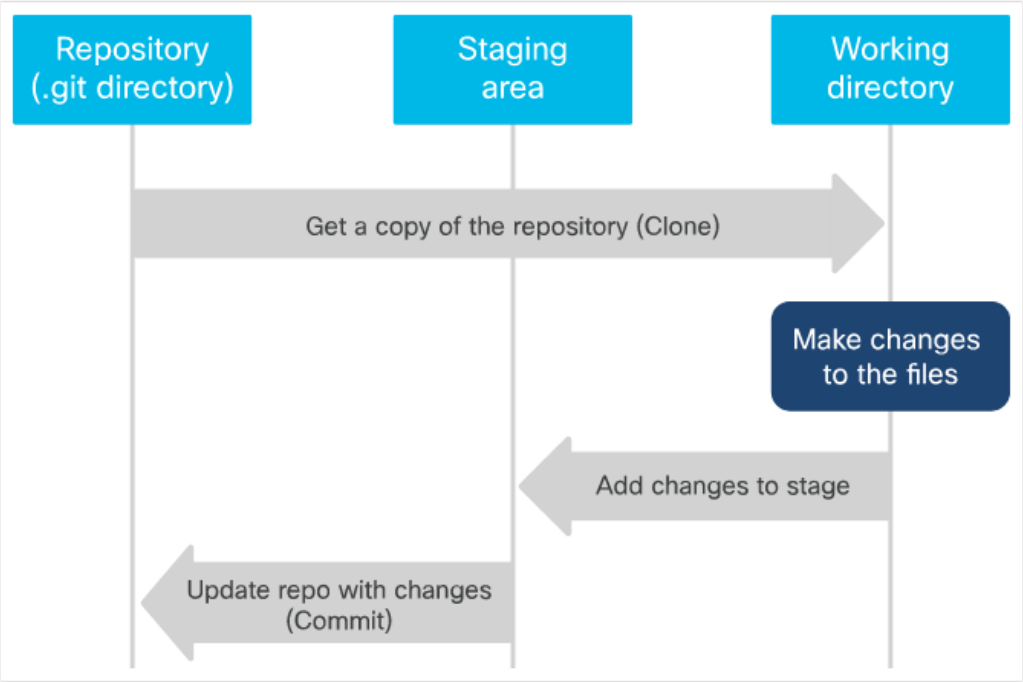
- Git là một hệ thống quản lý phiên bản phân tán mã nguồn mở, hiện đang được sử dụng rộng rãi trong phát triển phần mềm.
- Máy client cần cài đặt Git client, có phiên bản dành cho MacOS, Windows, và Linux/Unix.
- Điểm khác biệt quan trọng giữa Git và các hệ thống quản lý phiên bản khác là Git lưu trữ dữ liệu như các snapshot thay vì lưu trữ các khác biệt (delta giữa phiên bản hiện tại và trước đó).



- Nếu file không thay đổi, Git sử dụng link tham chiếu đến snapshot gần nhất trên hệ thống thay vì tạo ra các snapshot mới và giống nhau.

Git (tt)

- Git được tổ chức thành 3 giai đoạn và 3 trạng thái (gọi là 3s).
- 3 giai đoạn:
  - Kho lưu trữ (Repository (thư mục .git))
  - Thư mục đang làm việc (working directory)
  - Khu vực staging (staging area)
- 3 trạng thái:
  - Committed
  - Modified
  - Staged



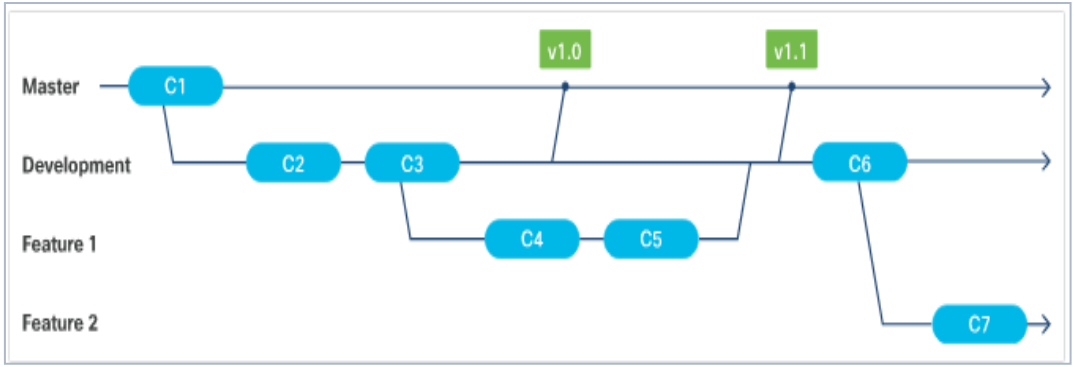
Kho lưu trữ cục bộ vs từ xa

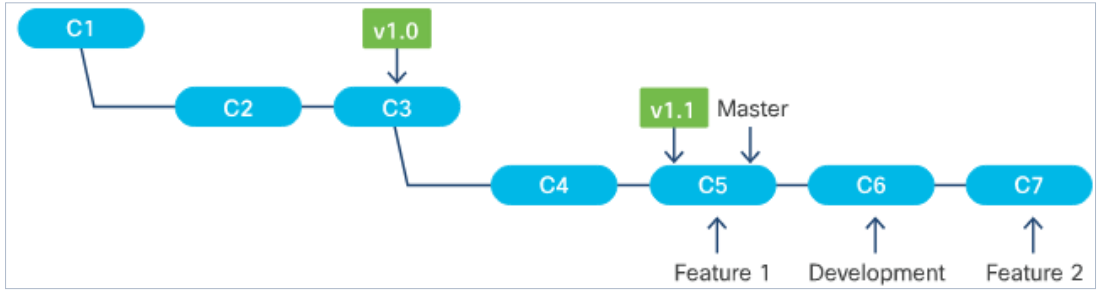
- Git có 2 loại kho lưu trữ: cục bộ (local) và từ xa (remote).
- Kho lưu trữ cục bộ nằm trên hệ thống file của 1 máy client, cũng là nơi thực thi các câu lệnh git.
- Kho lữu trữ từ xa được lưu trữ đâu đó bên ngoài máy client, thường là server hoặc dịch vụ hosting kho lưu trữ.
- Một khi lưu trữ từ xa cùng với Git tạo thành 1 hệ thống quản lý phiên bản phân tán (DVCS), vì kho lưu trữ từ xa sẽ chứa toàn bộ kho bao gồm mã nguồn đến lịch sử thay đổi file.
- Khi 1 máy client clone kho lưu trữ sẽ lấy về toàn bộ nội dung mà không cần khóa các file như trong CVCS.
- Sau khi kho cục bộ đã được clone từ kho lưu trữ từ xa hoặc kho lưu trữ từ xa đã được tạo từ kho cục bộ, 2 kho lưu trữ là độc lập với nhau đến khi các thay đổi trong nội dung được áp dụng trên các nhánh (branch) khác bằng câu lệnh Git thủ công.

Branching – Phân nhánh là gì?

- 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 1 kho lưu trữ được tạo, mã nguồn sẽ được tự động thêm vào nhánh (branch) Master.

- Các nhánh (branch) có thể là cục bộ hoặc từ xa, và có thể được xóa và có lịch sử, vùng staging và thư mục làm việc riêng.
- Việc tạo các nhánh Git rất đơn giản, việc chuyển đổi qua lại giữa các nhánh cũng nhanh chóng.
- Khi người dùng chuyển nhánh, mã nguồn trong thư mục làm việc và các file trên khu vực staging cũng thay đổi theo, nhưng thư mục của kho lưu trữ (.git) vẫn không đổi.





GitHub và các nhà cung cấp khác

- Git và GitHub không giống nhau.
- Git là 1 hiện thực của hệ thống quản lý phiên bản phân tán và cung cấp giao diện dòng lệnh (command), GitHub là một dịch vụ do Microsoft cung cấp để hiện thực dịch vụ hosting kho lưu trữ với Git.
- Bên cạnh việc cung cấp khả năng quản lý phiên bản và mã nguồn phân tán của Git, GitHub còn cung cấp các tính năng sau:
  - Đánh giá mã nguồn - code review
  - Tài liệu hóa - documentation
  - Quản lý dự án - project management
  - Theo dõi lỗi - bug tracking
  - Yêu cầu tính năng - feature requests
- GitHub giới thiệu khái niệm ‘pull request’, 1 cách để chuẩn hóa yêu cầu của người dùng để xem các thay đổi như mã nguồn mới, chỉnh sửa trong mã nguồn, etc., trong nhánh của người dùng để thêm vào nhánh chính của dự án hoặc các nhánh khác.

Các lệnh Git

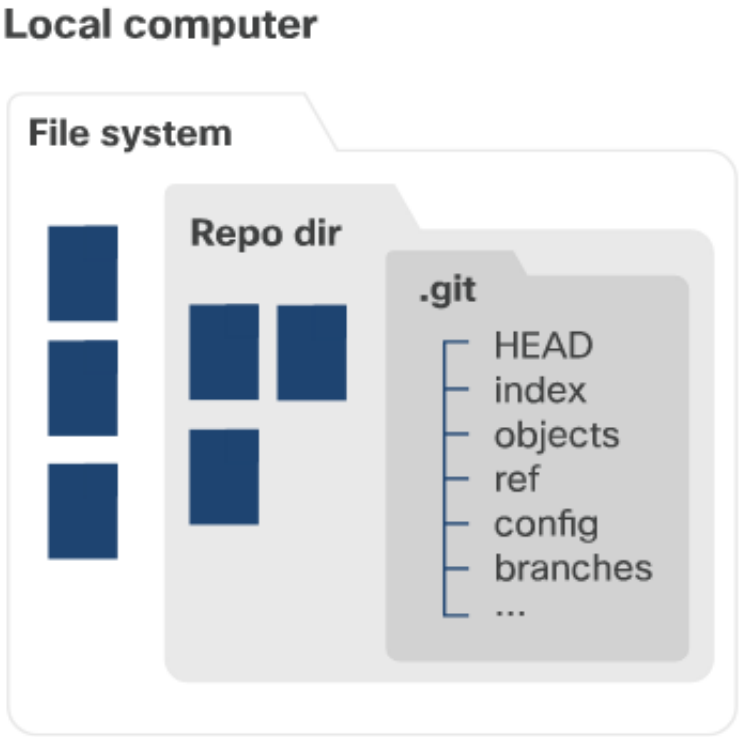
Thiết lập Git

- Để cấu hình Git, sử dụng option --global để thiết lập các cài đặt chung ban đầu.

  Lệnh: `git config --global key value`

Tạo kho lưu trữ Git mới

- Git cung cấp lệnh `git init` để tạo một kho lưu trữ Git rỗng, hoặc đưa 1 thư mục có sẵn thành kho lưu trữ Git.
- Khi có kho lưu trữ Git, 1 thư mục ẩn .git sẽ được tạo trong thư mục đó của kho lưu trữ.
- Thư mục .git là kho lưu trữ chứa các metadata như các file nén, lịch sử commit và khu vực staging. Bên cạnh đó, Git cũng tạo nhánh master.



Các lệnh Git (tt)

Lệnh: git init

- Để tạo kho lưu trữ Git từ một thư mục mới hoặc đã có sẵn, sử dụng lệnh:

  $ git init <project directory>

  Trong đó <project directory> là đường dẫn tuyệt đối hoặc tương đối đến thư mục mới hoặc đã có sẵn.

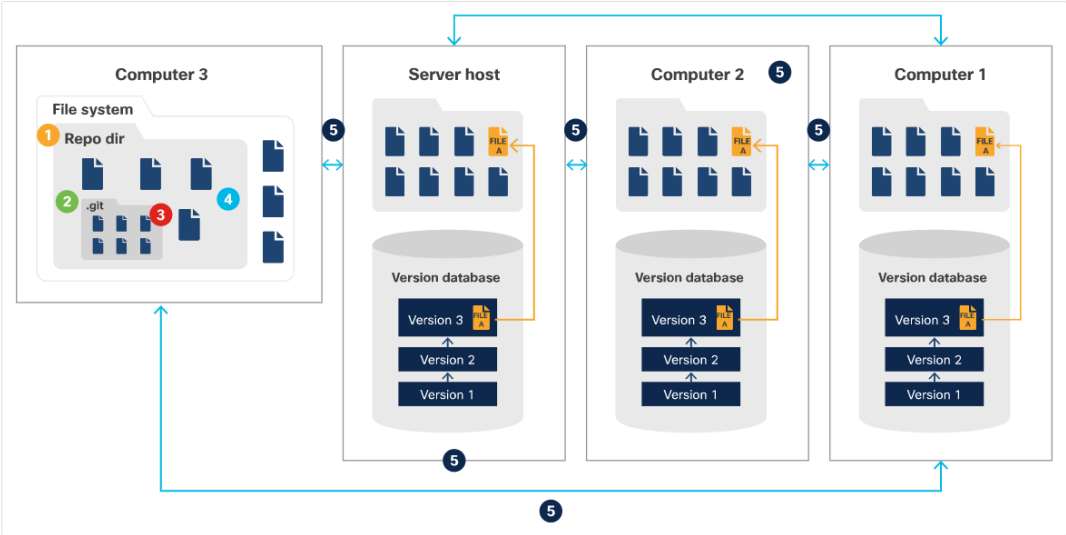
- Với tạo 1 kho lưu trữ Git mới, thư mục ở đường dẫn được cung cấp sẽ được tạo đầu tiên, sau đó là tạo thư mục .git.

Lấy về kho lưu trữ Git đã có

- **Lệnh**: git clone <repository> [target directory]

  Trong đó <repository> là vị trí của kho lưu trữ cần clone về.

- Git hỗ trợ 4 giao thức truyền chính để truy cập vào <repository>: Local, Secure Shell (SSH), Git, and HTTP.



Các lệnh Git (tt)

Xem các file đã thay đổi trong Thư mục làm việc

- Git cung cấp câu lệnh git status để xem danh sách các file có khác biệt giữa thư mục làm việc và nhánh cha.
- **Lệnh**: `git status`

So sánh thay đổi giữa các file

- Git cung cấp lệnh git diff, một công cụ so sánh các file.
- **Lệnh**: `git diff`
  - Khi dùng lệnh này, file không cần là file thuộc quản lý của Git.

Thêm và xóa file

Thêm file vào Khu vực Staging

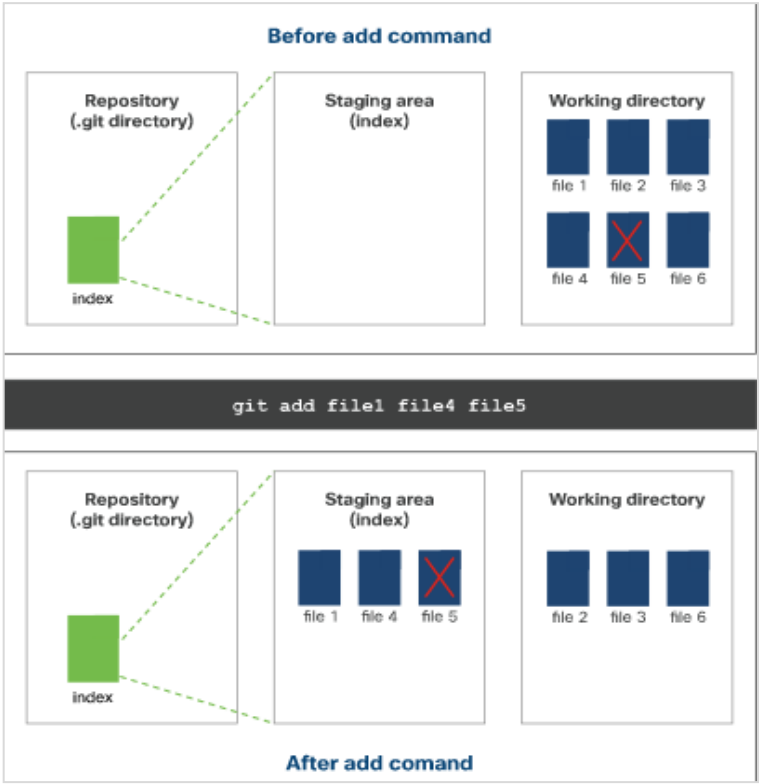
- **Lệnh**: `git add`
- Lệnh này có thể dùng nhiều lần trước khi kho lưu trữ Git được cập nhật (với lệnh commit).
- Chỉ các file được chỉ định trong lệnh git được thêm vào Khu vực staging
- Để thêm 1 file vào khu vực staging:

  `$ git add <file path>`

- Để thêm nhiều file vào khu vực staging, trong đó `<file path>` là đường dẫn tuyệt đối hoặc tương đối của file cần thêm.

  `$ git add <file path 1> ... <file path n>`

- Để thêm tất cả các file đã thay đổi vào khu vực staging: `$ git add`.



Thêm và xóa file (tt)

Xóa file trên kho lưu trữ Git

- Có 2 cách.
- **Cách 1**: Lệnh **git rm** dùng để xóa file trên kho lưu trữ Git và thêm vào khu vực staging.
  - **Lệnh**: git rm
  - Để xóa các file nhất định từ thư mục làm việc và thêm thay đổi này vào khu vực staging, sử dụng lệnh:
    $ git rm <file path 1> ... <file path n>

    Trong đó <file path> là đường dẫn tuyệt đối hoặc tương đối của file cần xóa trên kho lưu trữ Git.

Thêm và xóa file (tt)

- Để thêm các file nhất định cần xóa từ khu vực staging mà không xóa file đó trên thư mục làm việc, sử dụng lệnh sau:

```
$ git rm --cached <file path 1> ... <file path n>
```

Lệnh git rm sẽ không hoạt động với file đã ở khu vực staging với các thay đổi.

- **Cách 2**: Cách này là 1 quy trình gồm 2 bước. Bước 1 sử dụng lệnh hệ thống thông thường để xóa file và thêm file vào khu vực staging với lệnh Git.

```
$ rm <file path 1> ... <file path n>
$ git add <file path 1> ... <file path n>
```

Quy trình 2 bước này tương đương với lệnh `git rm <file path 1> ... <file path n>`. Sử dụng cách này không cho phép giữ file trong thư mục làm việc.

Cập nhật kho lưu trữ

Cập nhật Kho lưu trữ cục bộ với các thay đổi trong Khu vực staging

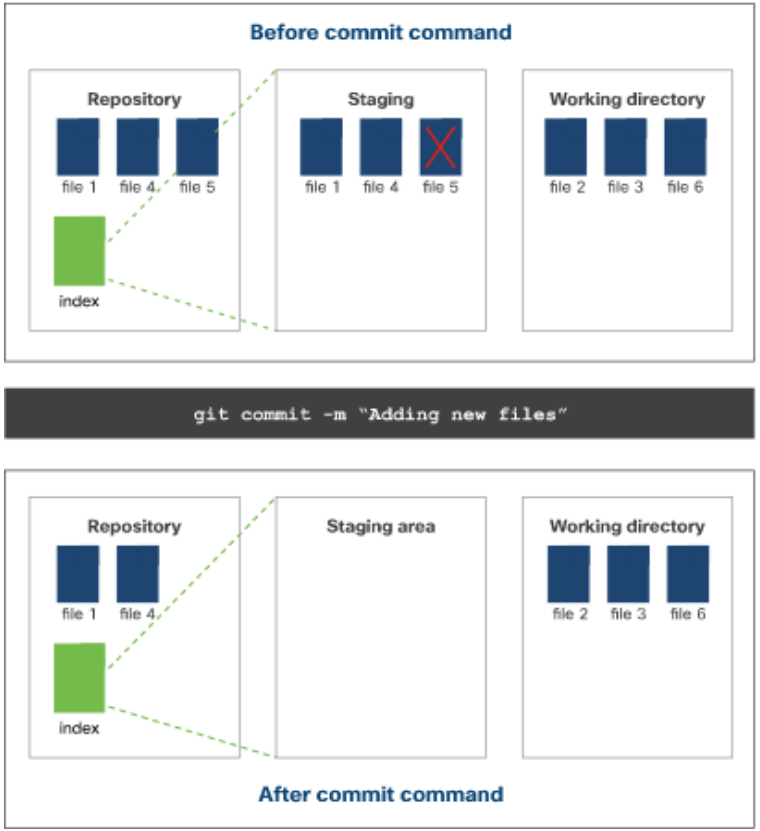
Lệnh: git commit

- Lệnh này kết hợp các thay đổi nội dung trong khu vực staging vào một commit duy nhất và cập nhật kho lưu trữ Git cục bộ.
- Để commit các thay đổi từ khu vực staging, sử dụng lệnh:

```
$ git commit
```

- Để commit các thay đổi từ khu vực staging với một thông điệp nào đó, sử dụng lệnh:

```
$ git commit -m "<message>"
```



Cập nhật kho lưu trữ (tt)

Cập nhật Kho lưu trữ từ xa

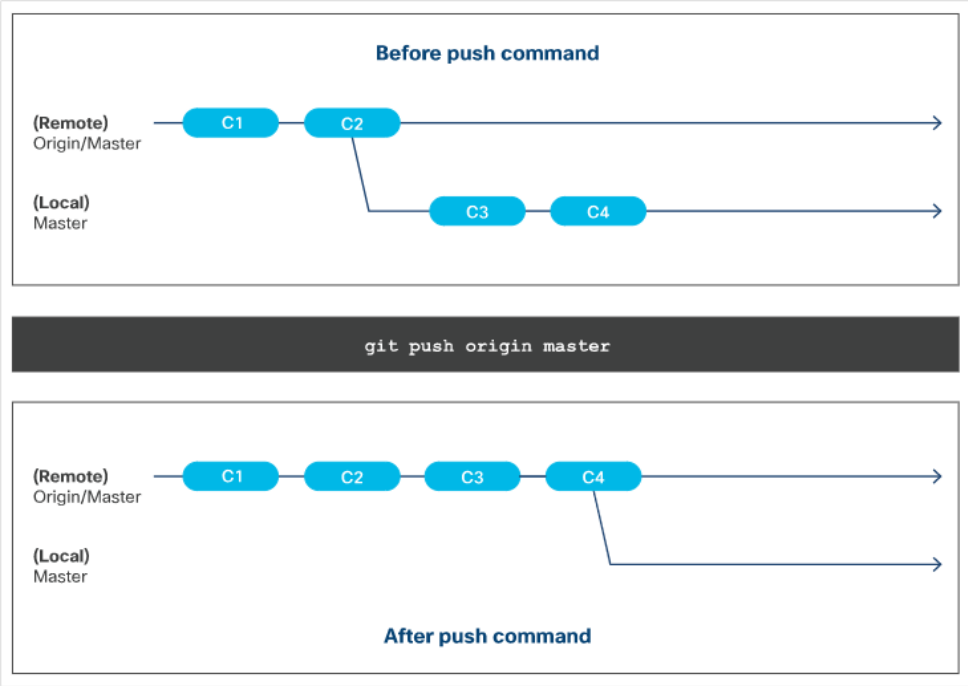
Lệnh: git push

- Lệnh này sẽ không thực hiện được nếu có xung đột khi thêm các thay đổi từ kho lưu trữ Git cục bộ đến kho lưu trữ Git từ xa.
- Để cập nhật nội dung từ kho lưu trữ cục bộ đến 1 nhánh cụ thể của kho lưu trữ từ xa, dùng lệnh:

```
$ git push origin <branch name>
```

- Để cập nhật nội dung kho lưu trữ cục bộ đến nhánh master của kho lưu trữ từ xa, dùng lệnh:

```
$ git push origin master
```



Cập nhật kho lưu trữ (tt)

Cập nhật bản sao cục bộ của Kho lưu trữ

- Bản sao cục bộ của Kho lưu trữ Git không tự động cập nhật khi có người dùng khác cập nhật Kho lưu trữ Git từ xa.
- Cập nhật bản sao cục bộ của Kho lưu trữ cần thực hiện thủ công.

Lệnh: `git pull`

- Khi thực hiện lệnh này, các bước sau sẽ xảy ra:
  - Kho lưu trữ cục bộ (thư mục .git) được cập nhật với commit, lịch sử file... và các thông tin khác mới nhất từ Kho lưu trữ Git từ xa.
  - Thư mục làm việc và nhánh được cập nhật với các nội dung mới nhất từ Bước 1.
  - Một commit được tạo ra ở nhánh cục bộ với các thay đổi ở Bước 1.
  - Thư mục làm việc được cập nhật với các nội dung mới nhất.

Cập nhật kho lưu trữ (tt)

- Để cập nhật bản sao cục bộ của Kho lưu trữ Git từ nhánh cha, sử dụng lệnh sau:

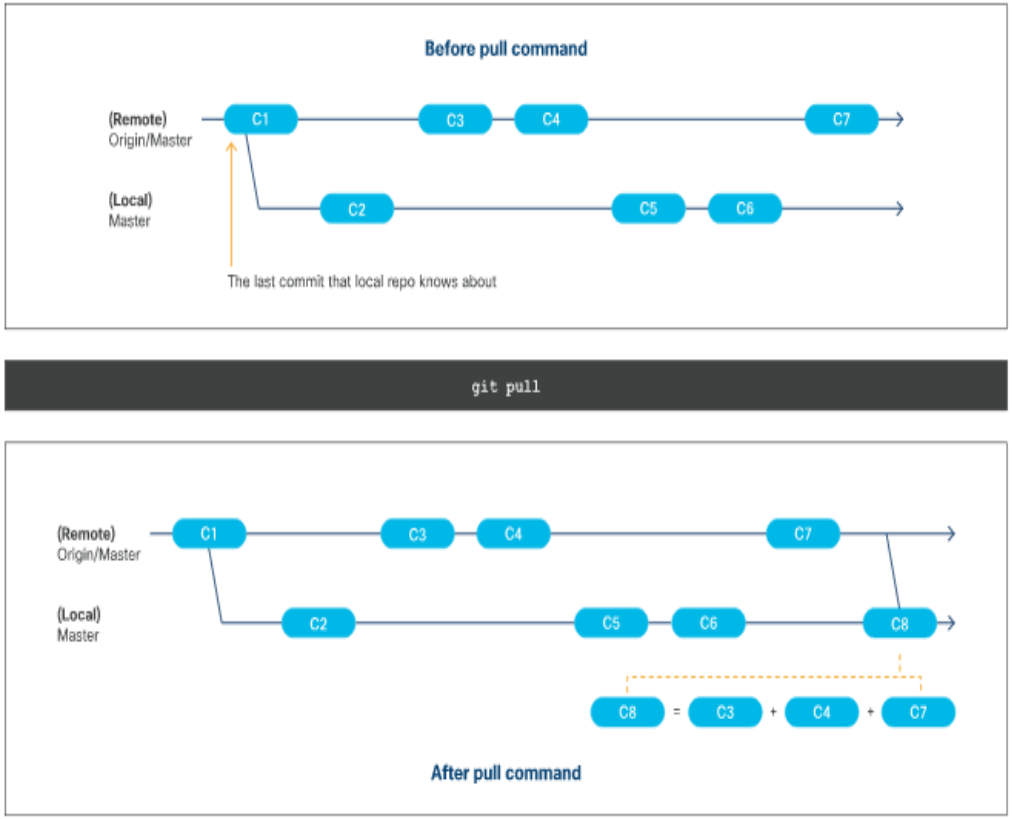
```
$ git pull
```

Hoặc

```
$ git pull origin
```

- Để cập nhật bản sao của kho lưu trữ Git từ 1 nhánh cụ thể, sử dụng lệnh:

```
$ git pull origin <branch>
```



Tính năng Phân nhánh (Branching)

Tạo và xóa một Nhánh

Cách 1: Lệnh git branch để liệt kê, tạo và xóa nhánh.

```
$ git branch <parent branch> <branch name>
```

Cách 2: Lệnh git checkout để chuyển nhánh bằng cách cập nhật thư mục làm việc với các nội dung của nhánh.

```
$ git checkout -b <parent branch> <branch name>
```

Deleting a Branch

- Để xóa 1 nhánh, sử dụng lệnh sau:

```
$ git branch -d <branch name>
```

Get a List of all Branches

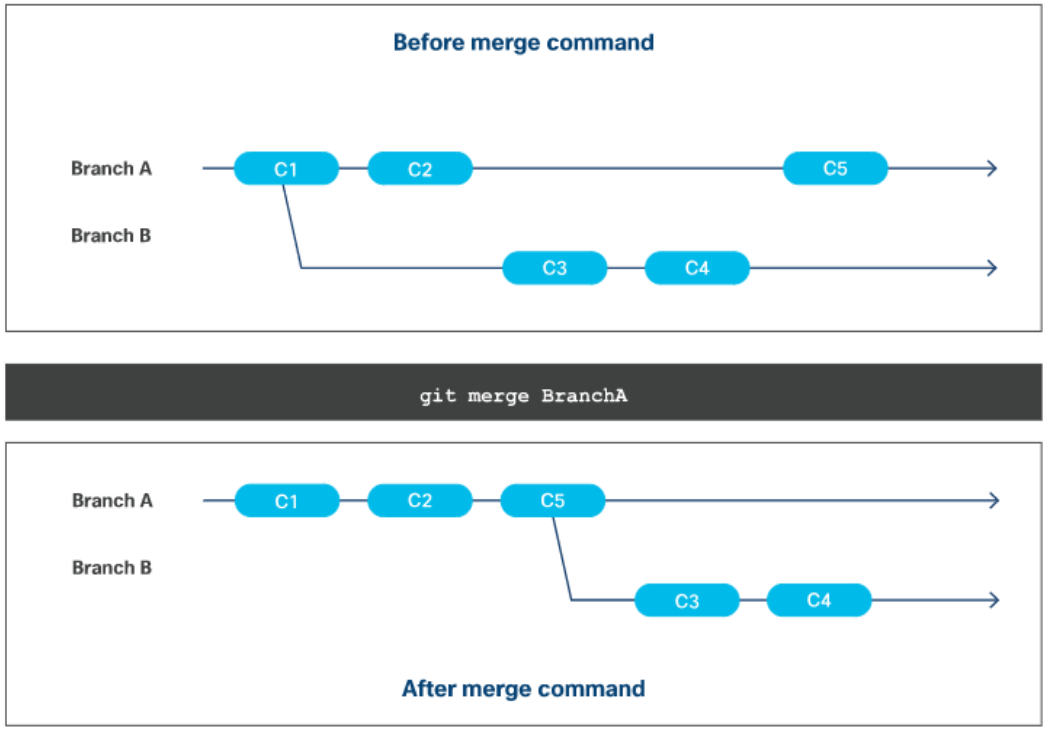
- Để liệt kê danh sách tất cả các nhánh cục bộ, sử dụng lệnh sau:

```
$ git branch
```
Or
```
$ git branch --list
```

Tính năng Phân nhánh (Branching) (tt)

Gộp nhánh

- Sau khi tạo, việc thay đổi các nhánh làm cho chúng có nội dung khác nhau.
- Khi Git gộp các nhánh lại, nó sẽ lấy các thay đổi/commit từ nhánh nguồn và áp dụng vào nhánh đích.
- Trong quá trình gộp nhánh, chỉ có nhánh đích bị thay đổi.
- Nhánh đích vẫn được giữ nguyên.



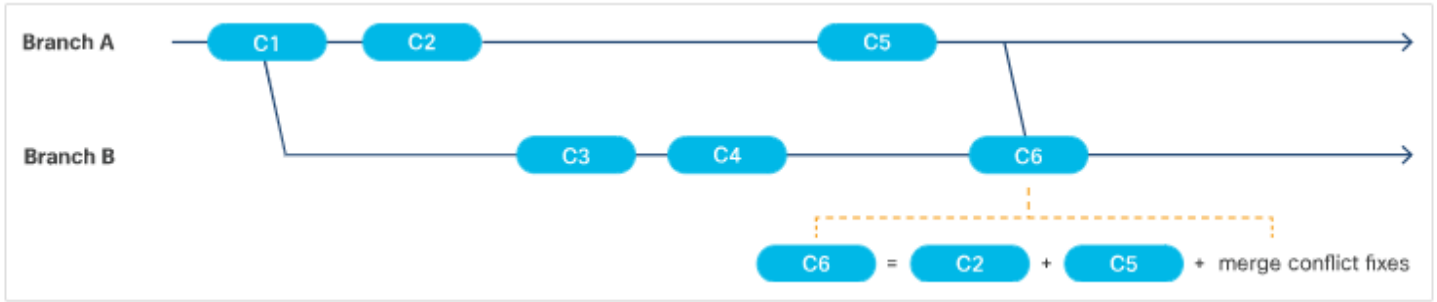
Tính năng Phân nhánh (Branching) (tt)

Gộp Fast-Forward

- Gộp fast-forward là khi giải thuật Git có thể áp dụng các thay đổi/commit từ nhánh nguồn đến nhánh đích một cách tự động và không có xung đột nào.

Xung đột gộp nhánh

- Một xung đột gộp nhánh xảy ra khi Git không thể thực hiện gộp fast-forward vì nó không biết cách tự động áp dụng các thay đổi từ các nhánh cùng trên các file.



Tính năng Phân nhánh (Branching) (tt)

Thực hiện gộp nhánh

- Git cung cấp lệnh git merge để gộp hai hay nhiều nhánh lại với nhau.
- **Lệnh**: git merge
  - Để gộp 1 nhánh vào nhánh/kho lưu trữ hiện tại của client, sử dụng câu lệnh sau:

```
$ git merge <branch name>
```

  - Để gộp 1 nhánh vào 1 nhánh không phải là nhánh/kho lưu trữ của client hiện tại, sử dụng lệnh sau:

```
$ git checkout <target branch name>
$ git merge <source branch name>
```

  - Để gộp 1 hoặc nhiều nhánh vào nhánh/kho lưu trữ hiện tại của client, sử dụng lệnh sau:

```
$ git merge <branch name 1>...<branch name n>
```

Các file .diff

File .diff là gì?

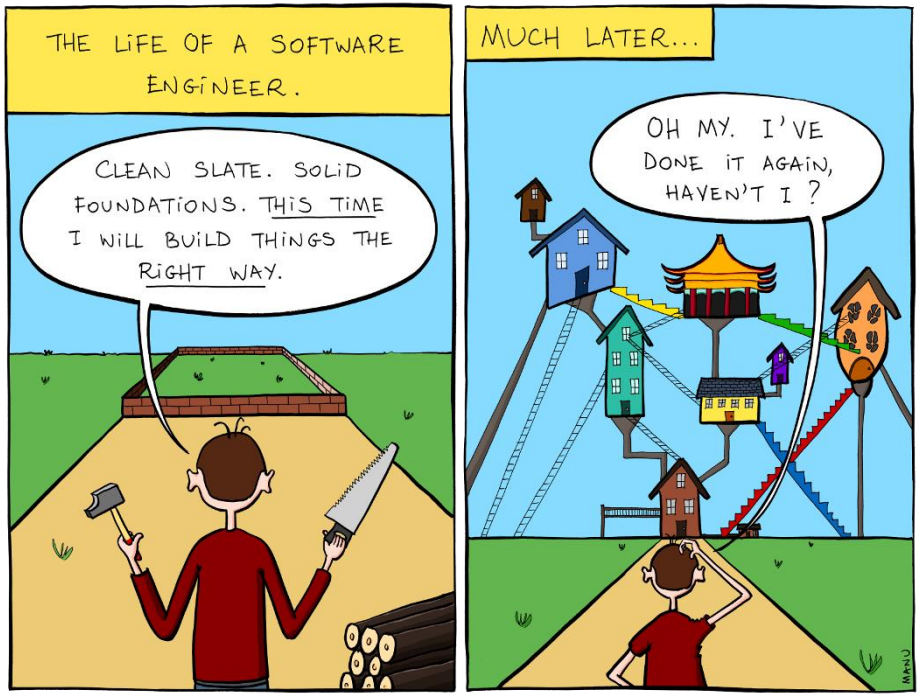
- Một file .diff được dùng để mô tả sự khác biệt giữa 2 phiên bản của 1 file đã bị thay đổi.
- Bằng cách sử dụng 1 số biểu tượng (symbols), file này có thể được các hệ thống khác đọc để phân tích xem các file đã được cập nhật như thế nào.
- Các symbols và ý nghĩa của chúng trong 1 file .diff như sau:

Symbol	Ý nghĩa
+	Chỉ ra dòng (line) đã được thêm.
-	Chỉ ra dòng (line) đã bị xóa.
/dev/null	1 file đã được thêm hoặc.
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.

3.4 Nền tảng lập trình

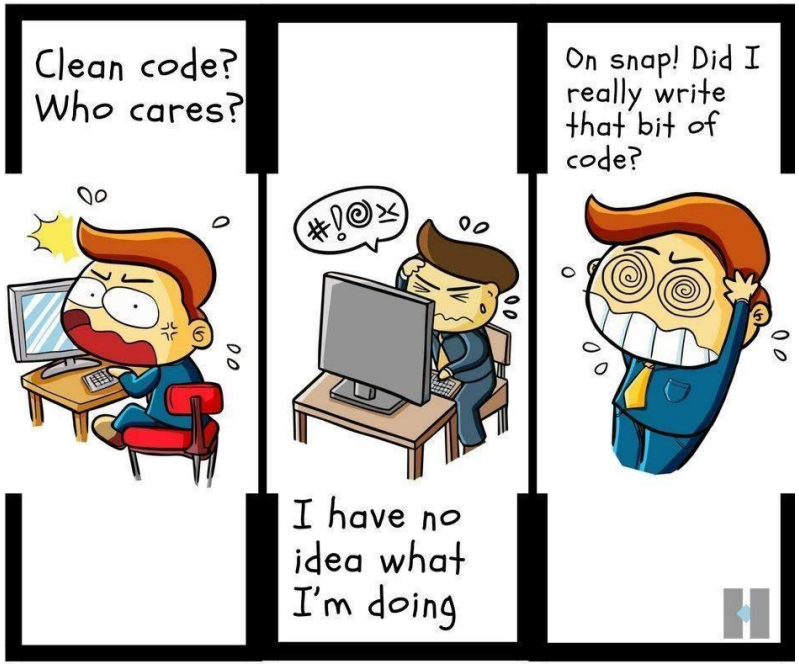
Methods, Functions, Modules, and Classes

- Khi quy mô và độ phức tạp của dự án ngày càng tăng và các nhà phát triển khác (và các bên liên quan) tham gia, các phương pháp qui định trước (disciplined method) và các phương pháp hay nhất (best practice) đóng vai trò cần thiết để giúp các nhà phát triển viết mã nguồn tốt hơn và cộng tác làm việc dễ dàng hơn.
- Thế nào là mã nguồn sạch (clean code)?



Mã nguồn sạch – Clean code

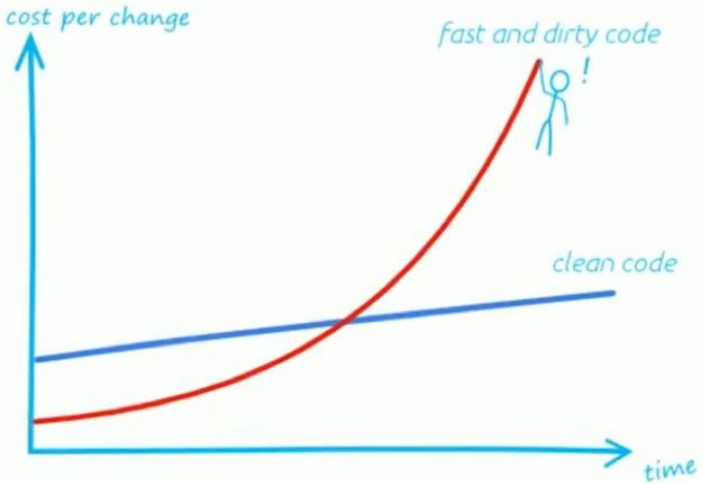
- Mã nguồn sạch - Clean code là kết quả của việc các lập trình viên khiến mã nguồn của họ dễ đọc và dễ hiểu ngay cả với các lập trình viên khác.
- Tuân theo một số quy tắc liên quan đến định dạng, tổ chức, tính trực quan của các thành phần, mục đích và khả năng tái sử dụng.
- Mã nguồn sạch nhấn mạnh các yếu tố: tính tiêu chuẩn, tổ chức phù hợp, tính mô-đun, cung cấp các dòng chú thích và các đặc điểm khác giúp mã nguồn dễ đọc.



Mã nguồn sạch – Clean code

Lý do các nhà phát triển muốn viết các mã nguồn sạch

- Mã nguồn sạch dễ hiểu hơn, nhỏ gọn hơn, tổ chức tốt hơn.
- Mã nguồn sạch được mô-đun hóa sẽ dễ kiểm tra với các phương pháp tự động hơn, ví dụ với các bộ khung kiểm thử đơn vị.
- Mã nguồn sạch được chuẩn hóa sẽ dễ quét và kiểm tra với các công cụ tự động.
- Bảo trì với chi phí thấp hơn
- Đơn giản là trông nó tốt hơn, đẹp mắt hơn.



Phương thức và Hàm

- Phương thức (Methods) và Hàm (Functions) là một khối các dòng mã (code) thực thi một tác vụ nào đó khi được thực thi.
- Chỉ dẫn về việc lập trình Mã thực thi của chương trình theo nguyên tắc đóng gói (encapsulation), (trong Phương thức hay Hàm):
  - Mã thực hiện một tác vụ rời rạc, ngay cả khi nó chỉ xảy ra một lần, có thể là một ứng cử viên cho việc đóng gói.
  - Mã tác vụ được sử dụng nhiều lần nên được đóng gói.
- Phương thức và Hàm đều được viết một lần và được dùng nhiều lần khi có nhu cầu.
- Nếu được sử dụng đúng, Phương thức và Hàm sẽ đơn giản hoá mã nguồn, và giảm thiểu sự tồn tại của lỗi (bug).
- VD: Cú pháp Hàm trong Python:

```
# Define the function
def functionName:
  ...blocks of code...
# Call the function
functionName()
```

Phương thức và Hàm

Đối số (Arguments) và Tham số (Parameters)

- Đối số và tham số giúp các phương thức và hàm trở nên linh hoạt hơn.
- Cú pháp của 1 hàm sử dụng đối số và tham số trong Python:

```
# Define the function
def functionName(parameter1,...,parameterN):
  # You can use the parameters just like local variables
  ...blocks of code...
# Call the function
functionName("argument1", 4, {"argument3":"3"})
```

Phương thức và Hàm

Lệnh Return

- Lệnh **return** nhằm trả về một giá trị bằng cách sử dụng từ khóa return theo sau là một biến hoặc biểu thức. Một lệnh return sẽ kết thúc hoạt động của hàm, và trả quyền điều khiển cho hàm gọi trước đó.
- Khi lệnh return được thực thi, giá trị của lệnh return được trả về và các dòng mã phía sau sẽ bị bỏ qua.
- Cú pháp của một hàm với câu lệnh return trong Python:

```
# Define the function
def functionName(parameter1,...,parameterN):
  # You can use the parameters just like local variables
  ...blocks of code...
  someVariable = parameter1 * parameter2
  return someVariable
# Call the function
myVariable = functionName("argument1", 4, {"argument3":"3"})
```

Phương thức và Hàm

Phương thức vs. Hàm

Phương thức	Hàm
Phương thức là các khối mã gắn liền với một đối tượng, thường dung trong lập trình hướng đối tượng.	Hàm là các khối mã độc lập.

Mô-đun

- Lập trình viên thường dùng mô-đun để chia các dự án mã nguồn lớn thành nhiều phần nhỏ hơn, từ đó mã nguồn sẽ đọc và dễ hiểu hơn.
- Các mô-đun gồm một tập các hàm và thường có các interface để mô-đun khác có thể tương tác với chúng.
- Một mô-đun được đóng gói thành 1 file và dự kiến sẽ hoạt động độc lập.
- Bên dưới là một mô-đun ví dụ với một tập các hàm, được lưu trong 1 file có tên circleClass.py.

```
# Given a radius value, print the circumference of a circle.
# Formula for a circumference is c = pi * 2 * radius

class Circle:

    def __init__(self, radius):
        self.radius = radius

    def circumference(self):
      pi = 3.14
      circumferenceValue = pi * self.radius * 2
      return circumferenceValue

    def printCircumference(self):
      myCircumference = self.circumference()
      print ("Circumference of a circle with a radius of " + str(self.radius) + " is " +
str(myCircumference))
```

Các lớp - Classes

- Trong hầu hết các ngôn ngữ lập trình hướng đối tượng (OOP), và trong Python, lớp (classes) là cách để gom nhóm dữ liệu và chức năng. Mỗi khai báo lớp định nghĩa 1 loại đối tượng mới.
- Lớp có thể có các biến lớp và biến đối tượng.
- Các lớp mới có thể được định nghĩa dựa trên các lớp đã định nghĩa trước đó, từ đó có thể kế thừa các thuộc tính, thành phần dữ liệu và các phương thức.
- Một lớp có thể được khởi tạo nhiều lần dưới dạng nhiều thực thể, và mỗi thực thể sẽ có các giá trị thuộc tính riêng cho nó.

***Lưu ý**: Khác với các ngôn ngữ OOP khác, Python không có cách định nghĩa các biến lớp 'riêng' và các phương thức nội bộ. Tuy nhiên, các phương thức và biến có 1 ký hiệu _ phía trước sẽ được xem là riêng và không được tham chiếu bên ngoài class.*

3.5 Đánh giá và kiểm thử mã nguồn

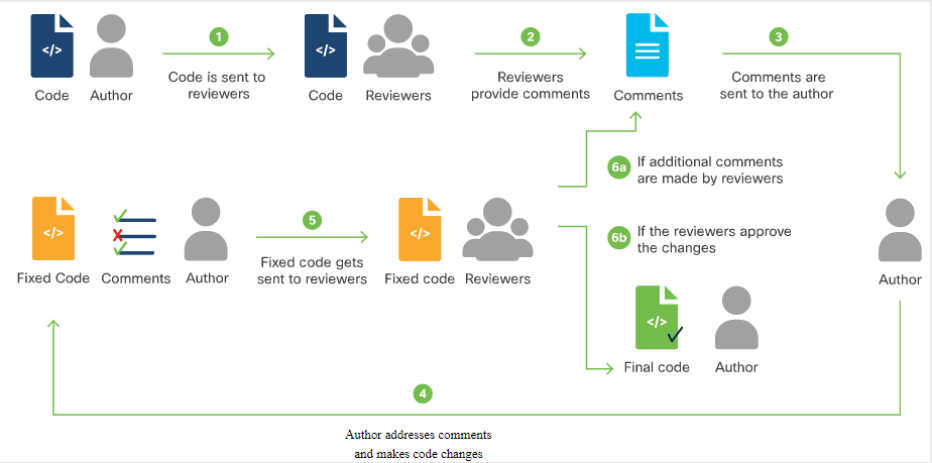
Đánh giá mã nguồn? Vì sao?

- Đánh giá mã nguồn - code review: lập trình viên xem xét các mã nguồn, 1 phần mã nguồn, hoặc các thay đổi nhất định trong mã nguồn và đưa ra phản hồi. Những lập trình viên như vậy được gọi là reviewers.
- Quá trình đánh giá mã nguồn chỉ diễn ra sau khi các thay đổi trong mã nguồn đã hoàn tất và kiểm tra xong.
- Mục tiêu của đánh giá mã nguồn nhằm đảm bảo mã nguồn cuối cùng:
  - Dễ đọc
  - Dễ hiểu
  - Tuân theo các phương pháp lập trình tốt nhất
  - Dùng đúng định dạng
  - Không có lỗi
  - Được chú thích và tài liệu đầy đủ
  - Sạch

Các loại đánh giá mã nguồn

Các loại đánh giá mã nguồn phổ biến nhất:

- **Đánh giá mã nguồn dạng formal**: Các lập trình viên tổ chức các buổi họp để đánh giá toàn bộ mã nguồn.
- **Đánh giá mã nguồn theo các thay đổi**: cũng được biết đến là đánh giá mã nguồn có công cụ hỗ trợ, đánh giá các mã nguồn bị thay đổi do lỗi, user story, tính năng, commit, v.v....
- **Đánh giá mã nguồn dạng Over-the-shoulder:** Một reviewer giám sát lập trình viên viết mã nguồn và đưa ra các phản hồi.
- **Gửi email**: các email tự động được gửi bởi các hệ thống quản lý mã nguồn khi có thay đổi.



Kiểm thử

- Kiểm thử phần mềm thường được chia làm 2 loại chính:
  - **Kiểm thử chức năng - Functional testing** kiểm tra xem phần mềm có hoạt động chính xác hay không. Phần mềm có hoạt động đúng như luồng logic đã định nghĩa, từ các cấp độ chi tiết thấp nhất với Kiểm thử Đơn vị, đến mức độ phức tạp nhất trong Kiểm thử Tích hợp?
  - **Kiểm thử phi chức năng - Non-functional testing** kiểm tra tính khả dụng, hiệu suất, tính bảo mật, khả năng phục hồi, tuân thủ các quy ước,... và nhiều vấn đề khác. Dạng kiểm thử này kiểm tra xem phần mềm có phù hợp với mục đích của nó, mang đến những giá trị lợi ích như mong đợi và giảm thiểu các rủi ro hay không?
- Lập trình viên xem các yêu cầu thiết kế như những bài kiểm thử và viết các phần mềm để vượt qua các bài kiểm thử đó. Đây gọi là Phát triển hướng kiểm thử (Test-Driven Development - TDD).

Kiểm thử đơn vị - Unit Testing

- Kiểm thử đơn vị - Unit Testing: kiểm thử chức năng một cách chi tiết từng đoạn mã nguồn nhỏ (từng dòng, từng khối, các hàm, các class, và các thành phần độc lập khác)
- Các framework kiểm thử là các phần mềm cho phép đưa ra các nhận định có thể kiểm thử được và xác định xem các nhận định đó có hợp lệ tại 1 thời điểm thực thi hay không.
- Ví dụ framework kiểm thử cho Python:

PyTest	unittest
• Tiện dụng, tự động thực thi các script bắt đầu bằng test_ hoặc kết thúc với _test.py, và bên trong các script đó tự động thực thi các hàm bắt đầu bằng ‘test_’ hoặc ‘tests_’ • Có thể kiểm tra đơn vị 1 đoạn mã bằng cách sao chép nó vào 1 tệp, sử dụng thư pytest, đặt tên các hàm cần kiểm tra cho phù hợp, lưu tệp với tên bắt đầu bằng ‘tests_’ và chạy với PyTest.	• Cú pháp khác so với PyTest. • Cần phân nhỏ class TestCase tích hợp sẵn và kiểm tra bằng cách ghi đè các phương thức mặc định hoặc thêm các phương thức mới với tên bắt đầu bằng ‘test_’.

Kiểm thử tích hợp - Integration Testing

- Kiểm thử tích hợp đảm bảo tất cả các đơn vị riêng biệt phù hợp với nhau để tạo thành ứng dụng hoàn chỉnh.
- Chạy mã nguồn với các hàm PyTest cho kết quả như sau:

***Lưu ý**: Sinh viên có thể chạy thử script này trên máy ảo với pytest. Tuy nhiên, hiểu được output và kiểm tra lỗi nằm ngoài phạm vi môn học.*

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.15+, pytest-3.3.2, py-1.5.2, pluggy-0.6.0
rootdir: /home/ubuntu/deploysample, inifile:
collected 1 item
test_sample_app.py F                                                     [100%]
================================== FAILURES ===================================
_____________________________ test_setconfig ______________________________
    def test_setconfig():
        setUp()
        set_config("TESTVAL")
>       assert get_config() == "ESTVAL"
E       AssertionError: assert 'TESTVAL' == 'ESTVAL'
E         - TESTVAL
E         ? -
E         + ESTVAL
test_sample_app.py:21: AssertionError
------------------------------ Captured log call -------------------------------
connectionpool.py          225 DEBUG    Starting new HTTP connection (1): localhost:80
connectionpool.py          437 DEBUG    http://localhost:80 "GET /get_config HTTP/1.1" 200 7
connectionpool.py          225 DEBUG    Starting new HTTP connection (1): localhost:80
connectionpool.py          437 DEBUG    http://localhost:80 "GET /config_action?dbhost=TESTVAL HTTP/1.1"
200 30
connectionpool.py          225 DEBUG    Starting new HTTP connection (1): localhost:80
connectionpool.py          437 DEBUG    http://localhost:80 "GET /get_config HTTP/1.1" 200 7
=========================== 1 failed in 0.09 seconds ===========================
```

Phát triển hướng kiểm thử - Test-Driven Development (TDD)

- Khi muốn kiểm tra để xác nhận thiết kế ứng dụng đáp ứng yêu cầu, nên viết các mã kiểm thử trước khi viết mã nguồn cho ứng dụng.
- Sau khi biểu đạt các yêu cầu trong các mã kiểm thử, có thể viết các mã nguồn ứng dụng đến khi nào vượt qua được các phép kiểm thử đã tạo trong mã kiểm thử.
- Mô hình cơ bản của TTD là một **quy trình 5 bước** được **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.

3.6 Các định dạng dữ liệu

Định dạng dữ liệu là gì?

- Rest APIs cho phép người dùng trao đổi thông tin với các dịch vụ hoặc thiết bị ở xa.
- 3 định dạng chuẩn phổ biến nhất để trao đổi thông tin với API ở xa gồm: XML, JSON, và YAML.
- Phân tích định dạng XML, JSON, hay YAML là yêu cầu thường xuyên phải thực hiện khi tương tác với các API. Ví dụ về các hoạt động trong mô hình hiện thực REST API như sau:
  - Chứng thực, thường dùng phương thức POST với cặp username/password và nhận một token dùng cho chứng thực các yêu cầu tiếp theo.
  - Thực hiện yêu cầu GET đến 1 đầu cuối (chứng thực khi cần thiết) để nhận trạng thái của 1 tài nguyên, với yêu cầu output dưới dạng XML, JSON, hay YAML.
  - Thay đổi XML, JSON, hay YAML được trả về.
  - Thực hiện POST (hoặc PUT) đến cùng điểm đầu cuối (cũng cần chứng thực) để thay đổi trạng thái của tài nguyên, cũng cần dưới dạng XML, JSON hay YAML và phân tích khi cần thiết để xem yêu cầu có thực hiện thành công hay không.

XML

- Viết tắt của E**x**tensible **M**arkup **L**anguage – ngôn ngữ đánh dấu mở rộng, là phương pháp gói các dữ liệu dạng văn bản trong các thẻ (tag) đối xứng để biểu thị ngữ nghĩa.
- XML là 1 biến thể của Structured, Generalized Markup Language (SGML), và là nền tảng của HyperText Markup Language (HTML). Các tệp XML kết thúc bằng đuôi ".xml".

Ví dụ 1 tệp XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Instance list -->
<vms>
  <vm>
    <vmid>0101af9811012</vmid>
    <type>t1.nano</type>
  </vm>
  <vm>
    <vmid>0102bg8908023</vmid>
    <type>t1.micro</type>
  </vm>
</vms>
```

XML (tt)

- **Phần thân XML:** ngoại trừ 2 dòng đầu của tệp XML, các phần còn lại được xem là phần thân.
- **Các tên thẻ (tag) người dùng tự định nghĩa:** Các thẻ XML là do người dùng tự định nghĩa. Nếu sử dụng XML cho ứng dụng tự phát triển, nên sử dụng các tên thẻ thể hiện rõ ý nghĩa của các thành phần dữ liệu, mối quan hệ và phân cấp của chúng.
- **Mã hóa ký tự đặc biệt:** dữ liệu dưới dạng XML là dạng văn bản đọc được.
- **XML Prologue:** XML prologue là dòng đầu tiên của tệp XML với thẻ <?xml ... ?>
- **Ghi chú trong XML:** các tệp XML có thể có các ghi chú (comment) với ký hiệu tương tự như trong các tệp HTML.
- **Các thuộc tính XML:** XML cho phép thêm các thuộc tính trong các thẻ tag để truyền tải thêm nhiều thông tin.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Instance list -->
<vms>
  <vm>
    <vmid>0101af9811012</vmid>
    <type>t1.nano</type>
  </vm>
  <vm>
    <vmid>0102bg8908023</vmid>
    <type>t1.micro</type>
  </vm>
</vms>
```

XML (tt)

- **XML Namespaces:**
  - Namespaces được định nghĩa bởi IETF và các cơ quan quản lý Internet, các tổ chức, và hệ thống namespace của họ thường được lưu trữ công khai trên web.
  - Namespaces được xác định bằng Uniform Resource Names (URIs) để có thể truy cập các tài liệu mà không cần quan tâm vị trí của nó ở đâu.
  - Ví dụ dưới đây sử dụng namespace với thuộc tính xmlns, xác định nội dung của 1 lời gọi hàm XML cần được diễn giải theo chuẩn NETCONF 1.0.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session>
    <session-id>4</session-id>
  </kill-session>
</rpc>
```

XML (tt)

- **Phân tích XML**
  - Trong ví dụ về XML Namespaces, cấu trúc tệp được thể hiện ở dạng một danh sách hay mảng 1 chiều (có tên ‘instances’) gồm nhiều đối tượng (mỗi đối tượng là 1 ‘instance’ bằng các thẻ). Mỗi instance chứa 2 cặp key-value chứa thông tin ID duy nhất và loại server.
  - Dưới đây là một cấu trúc dữ liệu trong Python có ý nghĩa tương đương:

```
vms [
  {
    {"vmid": "0101af9811012"},
    {"type": "t1.nano"}
  },
  {
    {"vmid": "0102bg8908023"},
    {"type": "t1.micro"}
  }
]
```

JSON

- JSON, hay JavaScript Object Notation, là một định dạng dữ liệu có nguồn gốc từ cách biểu diễn các dữ liệu phức tạp trong JavaScript.
- Các tệp JSON thường có đuôi ".json."
- Bên dưới là 1 ví dụ về một file JSON, chứa 2 giá trị chuỗi, 1 giá trị Boolean và 2 mảng:

```
{
  "edit-config":
  {
    "default-operation": "merge",
    "test-operation": "set",
    "some-integers": [2,3,5,7,9],
    "a-boolean": true,
    "more-numbers": [2.25E+2,-1.0735],
  }
}
```

JSON (tt)

- **Các kiểu dữ liệu cơ bản trong JSON:** bao gồm số, choỗi, Boolean và giá trị null.
- **Đối tượng JSON:** tương tự trong JavaScript, các đối tượng trong JSON có nhiều cặp key-value có thể nằm trong các dấu ngoặc nhọn ({}).
- **Danh sách (Lists) và Maps trong:** Trong các dạng dữ liệu này, mỗi cặp key-value không cần các dấu ngoặc riêng của chúng, mà dùng cho cả đối tượng. Các đối tượng JSON tổ hợp có thể lồng vào nhau tạo nên cấu trúc phức tạp. Dạng này cũng có thể biểu diễn các mảng dữ liệu hoặc đối tượng có thứ tự (list).
- **Không có hỗ trợ ghi chú (comment) trong JSON:** Không giống như XML và YAML, JSON không hỗ trợ bất kỳ phương pháp chuẩn nào để thêm các dòng ghi chú vào mã nguồn.
- **Khoảng trắng không quan trọng:** Khoảng trắng trong JSON không quan trọng, các tệp có thể sử dụng tab hoặc khoảng trắng để thụt lề, hoặc không dùng gì cả.

YAML

- YAML Ain't Markup Language (YAML) là tập cha của JSON, được thiết kế để dễ đọc hơn.
- Vì là tập cha của JSON nên các bộ phân tích YAML có thể phân tích các tệp JSON (nhưng ngược lại thì không).
- Do vậy, YAML tốt hơn JSON ở một số tác vụ, bao gồm việc nhúng trực tiếp mã JSON (có chứa dấu ngoặc kép) trong các file YAML.

```yaml
---
edit-config:
  a-boolean: true
  default-operation: merge
  more-numbers:
  - 225.0
  - -1.0735
  some-integers:
  - 2
  - 3
  - 5
  - 7
  - 9
  test-operation: set
...
```

YAML (tt)

- **Cấu trúc tệp YAML:** Các tệp YAML luôn bắt đầu bằng 3 dấu gạch ngang (--- nằm trên 1 dòng) và kết thúc bằng 3 dấu chấm (... cũng nằm trên 1 dòng).
- **Các kiểu dữ liệu YAML:** gồm số, chuỗi, booleans và null.
- **Đối tượng cơ bản:** các kiểu dữ liệu cơ bản được xem là các khóa.
- **Canh lề trong YAML và cấu trúc tệp YAML:** YAML sử dụng thụt lề để xác định phân cấp trong tệp.
- **Maps và Danh sách (Lists):** YAML dễ dàng biểu diễn các kiểu dữ liệu phức tạp như maps chứa nhiều cặp key-value và danh sách (list) có thứ tự.
  - Maps thường được biểu diễn trên nhiều dòng, bắt đầu bằng 1 nhãn khóa và một dấu hai chấm (:), theo sau đó là các thành phần, được thụt lề ở các dòng tiếp theo:

```
mymap:
  myfirstkey: 5
  mysecondkey: The quick brown fox
```

YAML (tt)

- Danh sách (lists hay arrays) được biểu diễn bằng các thành phần bắt đầu bằng dấu gạch ngang (-) và khoảng trắng, có thể thụt lề hoặc không.

```yaml
mylist:
  - 1
  - 2
  - 3
```

- Maps và danh sách (list) cũng có thể được biểu diễn khá tương tự như trong JavaScript hoặc Python, dưới dạng gọi là “flow syntax” – cú pháp luồng.

```yaml
mymap: { myfirstkey: 5, mysecondkey: The quick brown fox}
mylist: [1, 2, 3]
```

YAML (tt)

- **Chuỗi dài:** được biểu diễn với cú pháp ‘folding’, khi đó các dấu ngắt dòng sẽ được thay thế bằng khoảng trắng khi tệp được phân tích hay sử dụng, hoặc chuyển sang cú pháp không ‘folding’.

```
mylongstring: >
  This is my long string
  which will end up with no linebreaks in it
myotherlongstring: |
  This is my other long string
  which will end up with linebreaks as in the original
```

YAML (tt)

- **Ghi chú (comment):** Có thể thêm ghi chú trong YAML ở bất kỳ vị trí nào, ngoại trừ ở giữa chuỗi dài. Ghi chú bắt đầu bằng dấu thăng (#) và khoảng trắng.

```
# this is a comment
```

- **Nhiều tính năng khác của YAML:** YAML còn nhiều tính năng khác, thường được sử dụng trong ngữ cảnh ngôn ngữ cụ thể như Python, hay khi chuyển sang JSON hoặc các định dạng khác.

  Ví dụ, để ép 1 số được phân tích như 1 chuỗi, có thể dùng chuỗi !!str là 1 phần của “Failsafe” schema.

```
mynumericstring: !!str 0.1415
```

Parsing và Serializing

- Parsing – Phân tích định dạng: phân tích 1 thông điệp thành các thành phần và hiểu chức năng của chúng trong 1 ngữ cảnh nhất định.
- Serializing gần như ngược lại với parsing – phân tích định dạng.
- Các ngôn ngữ phổ biến như Python thường tích hợp các hàm phân tích định dạng (parsing) để nhận dữ liệu trả về từ các hàm I/O và tạo ra các cấu trúc dữ liệu có ý nghĩa tương đương chứa các kiểu dữ liệu hợp lệ.
- Mặt khác, chúng cũng có các serializers để chuyển các cấu trúc dữ liệu bên trong thành các thông điệp có ý nghĩa tương đương được định dạng như các chuỗi ký tự.

3.7 Tổng kết

Tóm tắt các nội dung cần nhớ...

- 6 giai đoạn của Quy trình phát triển phần mềm (SDLC): Requirements & Analysis, Design, Implementation, Testing, Deployment and Maintenance.
- 3 mô hình phát triển phần mềm phổ biến: Waterfall, Agile, và Lean.
- Mẫu thiết kế MVC đơn giản hóa việc phát triển các ứng dụng dựa trên giao diện người dùng dạng đồ họa.
- Quản lý phiên bản duy trì lịch sử thay đổi của tệp. Các loại hệ thống quản lý phiên bản: Cục bộ (Local), Tập trung (Centralized), và phân tán (Distributed).
- Git là một hiện thực mã nguồn mở của một hệ thống quản lý phiên bản phân tán, có 2 dạng kho lưu trữ: cục bộ và từ xa.
- Mã nguồn sạch là kết quả của quá trình điều chỉnh mã của lập trình viên để làm nó dễ đọc và dễ hiểu hơn đối với các lập trình viên khác.
- Đánh giá mã nguồn bao gồm đánh giá toàn bộ mã, một phần mã hoặc một thay đổi cụ thể trên mã nguồn để đưa ra các phản hồi.
- 3 chuẩn định dạng dữ liệu phổ biến nhất để trao đổi thông tin với các API từ xa: XML, JSON và YAML.
- Phân tích định dạng cần phân tích một thông điệp, chia thành các thành phần và hiểu mục đích trong 1 ngữ cảnh cụ thể. Serializing thì ngược lại.

Câu hỏi

- Hãy giải thích vì sao và bằng cách nào kiến trúc phần mềm ảnh hưởng đến các tính chất **portability** (chuyển đổi)**, scalability** (tính mở rộng), và **performance** (hiệu năng, tốc độ) của phần mềm. Các bạn có thể đưa ví dụ để giải thích.

Các khái niệm mới

- Software Development Life Cycle (SDLC)
- User experience (UX)
- Software Requirement Specification (SRS)
- Agile Scrum
- Lean
- Extreme Programming (XP)
- Feature-Driven Development (FDD)
- Sprints
- Backlog
- User stories
- Scrum Teams
- Model-View-Controller (MVC)
- Centralized Version Control Systems (CVCS)
- Distributed Version Control System (DVCS)
- Git
- Branching
- GitHub
- Arguments
- Parameters
- Object-Orient Programming (OOP)
- Formal Code Review
- Change-Based Code Review
- Over-the-Shoulder Code Review
- Test-Driven Development (TDD)
- Unit Testing
- Software Development Kits (SDKs)
- XML
- JSON
- YAML
- Application Programming Interfaces (APIs)
- REpresentational State Transfer (REST)
- Long Strings
- Parsing
- Serializing


CISCO