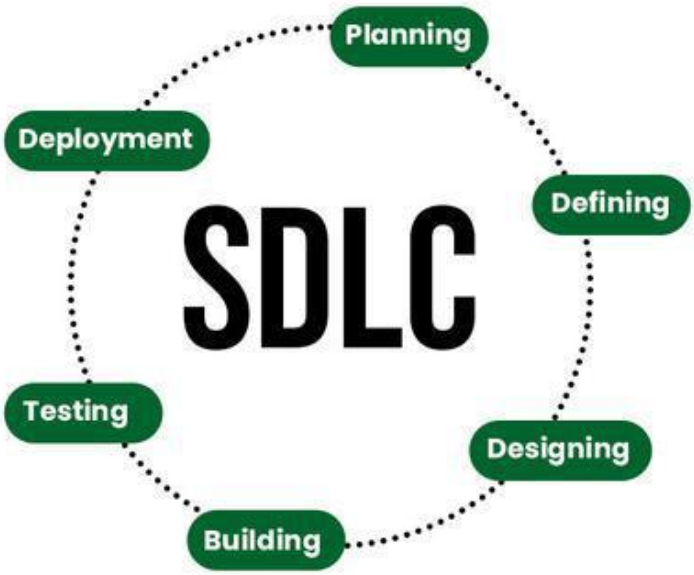
Vòng đời phát triển phần mềm
(Software Development Lifecycle)

Lê Phú Trường
Mã số sinh viên: 22110245
Email: 22110245@student.hcmus.edu.vn

Khoa Toán – Tin học
Trường Đại học Khoa học tự nhiên
Đại học Quốc gia thành phố Hồ Chí Minh

1. Giới thiệu về Software Development Lifecycle (SDLC)

a) **Định nghĩa:** Software Development Life Cycle (SDLC) là một phương pháp luận có cấu trúc được các nhóm phần mềm sử dụng để thiết kế, phát triển, thử nghiệm và triển khai phần mềm chất lượng cao một cách hiệu quả. Nó xác định một loạt các giai đoạn được tổ chức tốt bằng việc phân tích yêu cầu, lập kế hoạch, thiết kế, phát triển, thử nghiệm, triển khai và bảo trì nhằm đảm bảo phần mềm đáp ứng được kỳ vọng của khách hàng, giảm thiểu rủi ro và nằm trong giới hạn ngân sách và thời gian.



Hình 1: Các giai đoạn thực thi của SDLC

Bằng cách tuân theo SDLC, các nhóm phát triển có thể tạo ra các giải pháp phần mềm có thể bảo trì, tiết kiệm chi phí và kịp thời. Các mô hình phổ biến trong SDLC bao gồm các

mô hình Waterfall, Agile và Spiral. Quy trình này rất quan trọng để đảm bảo thành công của dự án và chất lượng phần mềm.

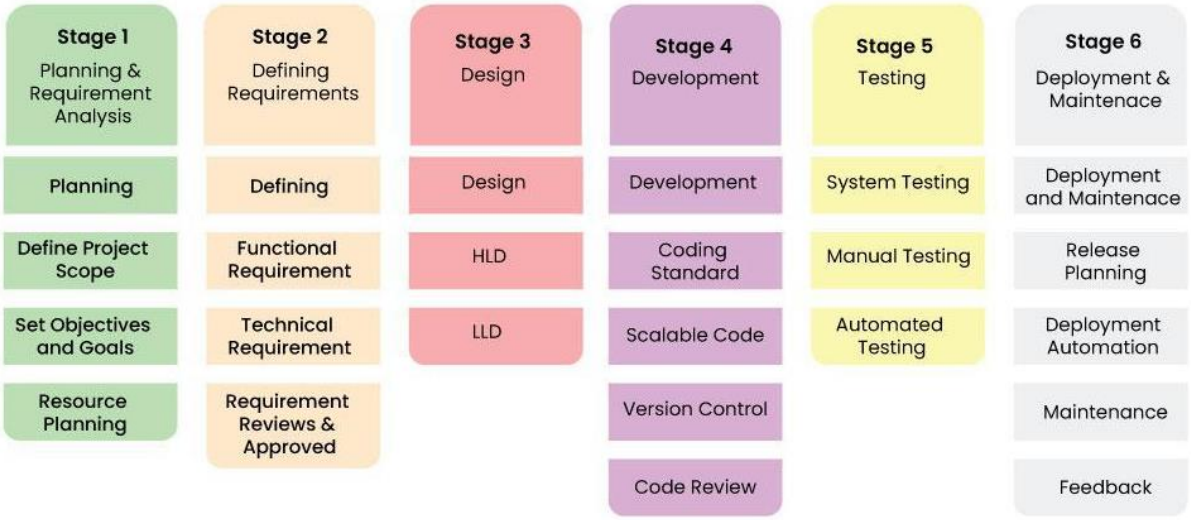
b) **Tầm quan trọng của SDLC:** Phát triển phần mềm có thể là một thách thức để quản lý do các yêu cầu liên tục thay đổi, nâng cấp công nghệ thường xuyên và nhu cầu hợp tác liên chức năng hiệu quả. Phương pháp Vòng đời phát triển phần mềm (SDLC) giải quyết những thách thức này bằng cách cung cấp một khuôn khổ có cấu trúc đảm bảo quản lý có hệ thống trong suốt quá trình phát triển. SDLC xác định các giai đoạn, hoạt động và sản phẩm cụ thể, cung cấp lộ trình rõ ràng cho tất cả các bên liên quan. Khuôn khổ này giúp điều chỉnh các mục tiêu phát triển với kỳ vọng của khách hàng, đảm bảo rằng mọi người tham gia đều đồng ý về các yêu cầu của dự án và có kế hoạch để đạt được chúng.

c) **Lợi ích chính của việc sử dụng SDLC:** Tăng khả năng hiển thị và minh bạch: SDLC tăng cường khả năng hiển thị trong suốt quá trình phát triển cho tất cả các bên liên quan, đảm bảo giao tiếp rõ ràng và hiểu biết chung về các mục tiêu ở mọi giai đoạn.

- *Cải thiện lập kế hoạch và lên lịch dự án:* Bằng cách xác định các hoạt động và mục tiêu ở mỗi giai đoạn, SDLC tạo điều kiện cho việc ước tính, lập kế hoạch và lên lịch hiệu quả hơn, cho phép quản lý dự án tốt hơn.
- *Nâng cao quản lý rủi ro:* SDLC giúp xác định và giảm thiểu rủi ro ngay từ đầu dự án, giảm khả năng xảy ra các vấn đề bất ngờ và sự chậm trễ.
- *Hiệu quả về chi phí và thời* gian: Phương pháp tiếp cận có cấu trúc cho phép ước tính chi phí chính xác hơn, giảm tổng chi phí dự án và thời gian cần thiết để giao hàng.
- *Phân phối phần mềm có hệ thống:* Bản chất có tổ chức của SDLC đảm bảo rằng phần mềm được phát triển một cách có hệ thống, cải thiện khả năng đáp ứng kỳ vọng của khách hàng và đạt được sự hài lòng cao hơn của khách hàng.
- *Cải thiện quan hệ khách hàng:* Bằng cách tăng tính minh bạch và cung cấp thông tin cập nhật thường xuyên về tiến độ, SDLC củng cố mối quan hệ với khách hàng, dẫn đến sự hợp tác và tin tưởng tốt hơn.
- Giảm rủi ro dự án: Thông qua một quy trình có cấu trúc, SDLC giảm thiểu rủi ro liên quan đến các yêu cầu thay đổi và nâng cấp công nghệ.
- *Giảm chi phí sản xuất:* Cấu trúc rõ ràng của SDLC cho phép kiểm soát tốt hơn ngân sách của dự án, giảm lãng phí và chi phí không cần thiết, cuối cùng dẫn đến chi phí sản xuất thấp hơn.

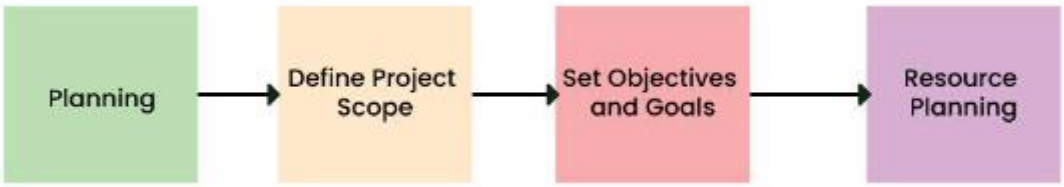
2. Các giai đoạn của Software Development Lifecycle

Vòng đời phát triển phần mềm (SDLC) là một quy trình có hệ thống để xây dựng phần mềm đảm bảo chất lượng và tính chính xác của phần mềm. Quy trình này được chia thành nhiều giai đoạn khác nhau, mỗi giai đoạn có các nhiệm vụ và mục tiêu cụ thể:



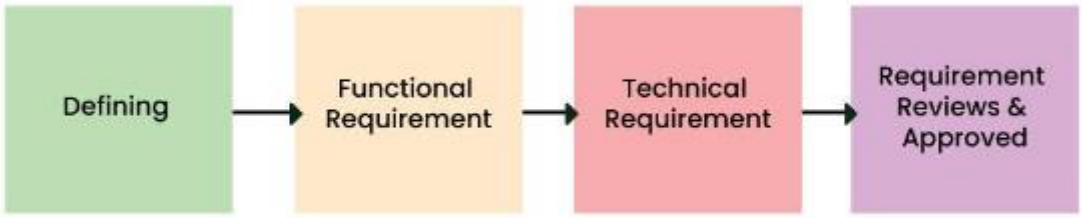
Hình 2: Chi tiết các giai đoạn chính của SDLC

i. *Planning*: Giai đoạn lên kế hoạch là nền tảng của SDLC. Giai đoạn này bao gồm việc xác định mục tiêu, phạm vi và ràng buộc của dự án, đồng thời đặt ra kỳ vọng về thời gian, chi phí và nguồn lực. Lên kế hoạch phù hợp là rất quan trọng để ngăn ngừa sai sót sau này trong dự án đồng thời cũng xác định rủi ro và chuẩn bị các chiến lược để giảm thiểu. Trong giai đoạn này, tính khả thi của dự án được đánh giá để đảm bảo đáp ứng cả mục tiêu kinh doanh và nhu cầu của người dùng.



Hình 3: Sơ đồ tinh chỉnh của giai đoạn Planning

ii. *Requirement Gathering and Analysis:* Trong giai đoạn này, nhóm dự án thu thập các yêu cầu từ các bên liên quan, khách hàng và nghiên cứu thị trường. Điều này đảm bảo rằng mọi người đều hiểu rõ về mục tiêu và chức năng của phần mềm. Output quan trọng của giai đoạn này là ***Software Requirement Specification (SRS)***, trong đó nêu rõ mọi tính năng và chức năng của phần mềm. SRS đóng vai trò là điểm tham chiếu trong suốt dự án.



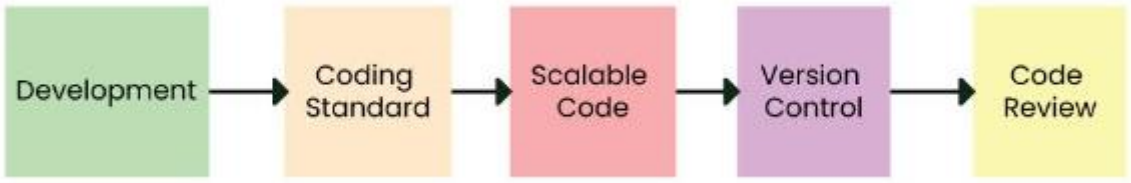
Hình 4: Sơ đồ tinh chỉnh của giai đoạn Requirement Gathering and Analysis

iii. *Design:* Giai đoạn thiết kế sử dụng SRS làm thư mục tham khảo để phát triển kiến trúc phần mềm. Giai đoạn này được chia thành hai phần:

- High-level design: Tập trung vào kiến trúc hệ thống, modules và luồng dữ liệu.
- Low-level design: Chi tiết các chức năng cụ thể, bao gồm cơ sở dữ liệu, giao diện và thuật toán chi tiết. Output của phần này là ***Design Document Specification (DDS)***, được các bên liên quan xem xét trước khi tiến hành phát triển.

Hình 5: Sơ đồ tinh chỉnh của giai đoạn Design

iv. *Implementation and Coding:* Giai đoạn này là nơi developers biên dịch các thông số kỹ thuật thiết kế thành mã thực tế. Nhóm phát triển xây dựng phần mềm bằng cách tuân thủ các tiêu chuẩn và giao thức mã hóa. Đây thường là giai đoạn tốn nhiều thời gian nhất, yêu cầu developers sử dụng nhiều ngôn ngữ lập trình khác nhau (như Python, Java hoặc C++) tùy thuộc vào yêu cầu của dự án. Code được tạo trong giai đoạn này phải phù hợp với các thông số kỹ thuật thiết kế được nêu trong giai đoạn trước.



Hình 6: Sơ đồ tinh chỉnh của giai đoạn Implementation and Coding

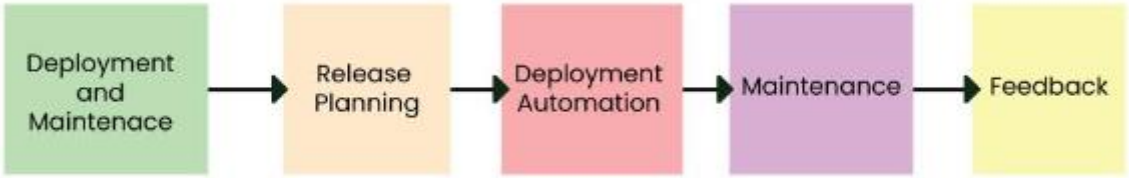
v. *Testing:* Sau khi mã hóa hoàn tất, phần mềm được kiểm tra nghiêm ngặt để đảm bảo đáp ứng các yêu cầu đã xác định và không có lỗi. Testing bao gồm unit testing, integration testing, system testing và acceptance testing. Bất kỳ lỗi nào được tìm thấy đều được sửa và phần mềm được kiểm tra lại để đảm bảo chức năng hoạt động trơn tru.

Mục tiêu là xác minh rằng sản phẩm hoạt động như mong đợi và đáp ứng các tiêu chuẩn chất lượng được xác định trong SRS.

Hình 7: Sơ đồ tinh chỉnh của giai đoạn Testing

vi. *Deployment:* Sau khi thử nghiệm thành công, phần mềm được triển khai vào môi trường sản xuất, giúp người dùng cuối có thể sử dụng. Một số dự án có thể bao gồm ***User Acceptance Testing (UAT)*** trước khi triển khai cuối cùng. Điều này cho phép các bên liên quan và người dùng xác minh hiệu suất của sản phẩm trong bối cảnh thực tế. Nếu có bất kỳ vấn đề nào phát sinh trong quá trình UAT, chúng sẽ được giải quyết trước khi phần mềm được phát hành cho tất cả người dùng.

vii. *Maintenance:* Giai đoạn bảo trì bao gồm hỗ trợ, cập nhật và cải thiện phần mềm sau khi ra mắt hoặc sửa mọi lỗi phát sinh sau khi triển khai, giải quyết phản hồi của người dùng và thêm các tính năng hoặc cải tiến mới. Theo thời gian, khi yêu cầu của người dùng thay đổi, phần mềm có thể cần được cập nhật để duy trì tính phù hợp.



Hình 8: Sơ đồ tinh chỉnh của giai đoạn Deploymen and Maintenance

3. Software Development Life Cycle Models

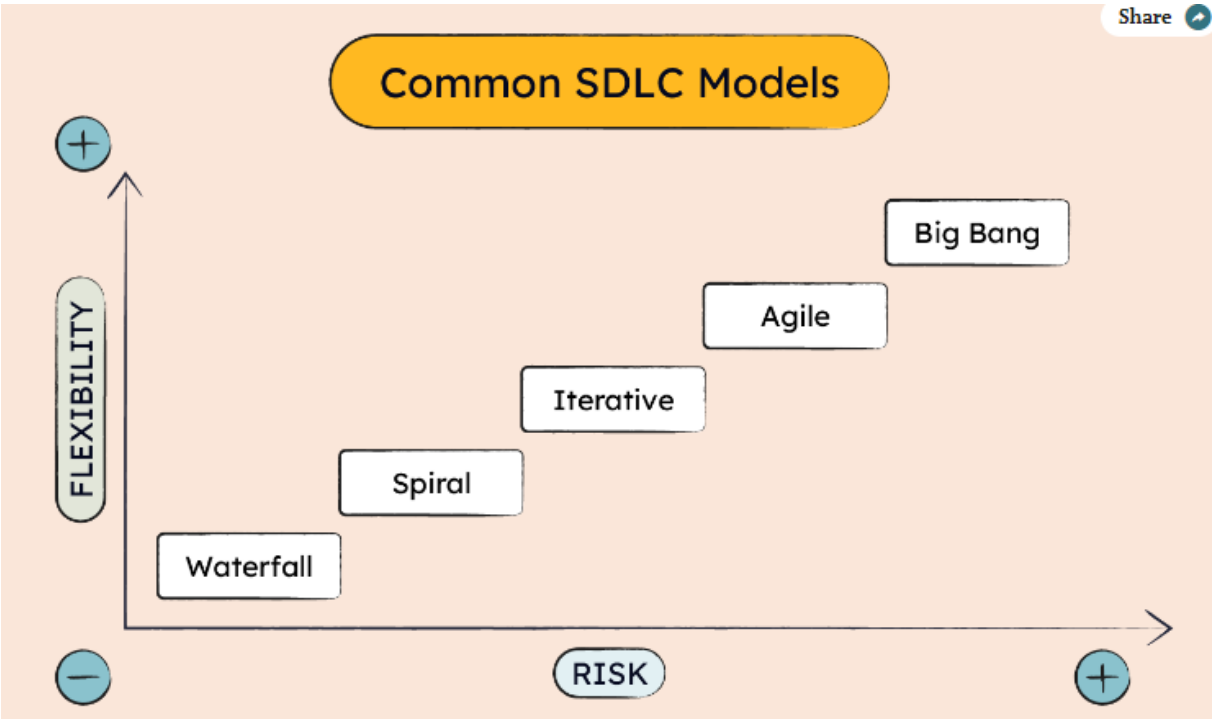
Trong lĩnh vực phát triển phần mềm, có rất nhiều mô hình SDLC đã được công nhận, với hơn 50 mô hình đang được sử dụng. Mặc dù không có mô hình nào là hoàn hảo, mỗi mô hình đều có những ưu điểm và nhược điểm riêng, phù hợp cho các dự án phát triển phần mềm hoặc đội ngũ cụ thể. Dưới đây là những mô hình phổ biến nhất:

- *Waterfall Model:* Mô hình Waterfall là mô hình cơ bản và lâu đời nhất trong SDLC. Các giai đoạn phát triển diễn ra tuần tự, với mỗi giai đoạn phụ thuộc vào kết quả của giai đoạn trước. Sau khi hoàn thành một giai đoạn, không thể quay lại thay đổi, dẫn đến tính linh hoạt thấp.
  - **Ưu điểm**: Mô hình này dễ quản lý và có sản phẩm cụ thể ở cuối mỗi giai đoạn.
  - **Nhược điểm**: Khó thay đổi nếu đã hoàn thành một giai đoạn, phù hợp cho các dự án nhỏ và yêu cầu rõ ràng.

- *Agile Model:* Agile được thiết kế để thích ứng với các yêu cầu thay đổi nhanh chóng. Mô hình này chia SDLC thành nhiều chu kỳ phát triển ngắn, trong đó các thay đổi nhỏ và tăng dần được đưa ra sau mỗi chu kỳ.
    - **Ưu điểm**: Giúp xử lý vấn đề trong các dự án phức tạp sớm và nhanh chóng, có sự tham gia của khách hàng liên tục để thu thập phản hồi.
    - **Nhược điểm**: Sự phụ thuộc quá nhiều vào phản hồi của khách hàng có thể dẫn đến thay đổi phạm vi quá mức hoặc chấm dứt dự án sớm.
- *Iterative Model:* Mô hình Iterative bắt đầu với một tập hợp nhỏ các yêu cầu và cải thiện dần qua các phiên bản lặp lại. Mỗi chu kỳ lặp sẽ tạo ra một phiên bản phần mềm chưa hoàn chỉnh nhưng có thể triển khai được, và các yêu cầu mới được bổ sung qua từng chu kỳ.
    - **Ưu điểm**: Dễ quản lý rủi ro, có thể thay đổi yêu cầu giữa các chu kỳ.
    - **Nhược điểm**: Các chu kỳ lặp đi lặp lại có thể gây ra thay đổi phạm vi và ước lượng sai tài nguyên.
- *Spiral Model:* Mô hình Spiral kết hợp giữa chu kỳ lặp của Iterative và luồng tuần tự của Waterfall, với sự ưu tiên cho việc phân tích rủi ro. Mỗi chu kỳ của mô hình này sẽ xem xét rủi ro, xây dựng các nguyên mẫu và cải thiện dần phần mềm.
    - **Ưu điểm**: Phù hợp cho các dự án lớn, phức tạp, cần nhiều thay đổi.
    - **Nhược điểm**: Có thể tốn kém cho các dự án nhỏ với phạm vi hạn chế.
- *V-shaped Model:* Mô hình V-shaped là một biến thể của mô hình Waterfall, trong đó mỗi giai đoạn phát triển có một giai đoạn kiểm thử tương ứng. Quy trình phát triển và kiểm thử diễn ra tuần tự, hình thành cấu trúc giống hình chữ V.
    - **Ưu điểm**: Mỗi giai đoạn đều đi kèm với kiểm thử, giúp đảm bảo chất lượng.
    - **Nhược điểm**: Tính linh hoạt thấp, gặp khó khăn khi thay đổi yêu cầu giữa chừng.
- *Big Bang Model:* Mô hình Big Bang không yêu cầu kế hoạch cụ thể hay tài liệu hóa rõ ràng. Các nhà phát triển nhảy ngay vào quá trình viết mã, và các yêu cầu được đưa ra khi cần. Do đó, mô hình này có rủi ro cao nhưng phù hợp cho các dự án nhỏ hoặc dự án học thuật.
    - **Ưu điểm**: Phù hợp cho các dự án nhỏ với ít tài nguyên hoặc các dự án học thuật.
    - **Nhược điểm**: Thiếu sự rõ ràng trong giai đoạn yêu cầu, dẫn đến nguy cơ phát triển lệch hướng.
- *Lean Model:* Phương pháp tinh gọn cho phát triển phần mềm được lấy cảm hứng từ các nguyên tắc và thực hành sản xuất tinh gọn. Các nguyên tắc tinh gọn khuyến

khích tạo ra luồng công việc tốt hơn và phát triển văn hóa cải tiến liên tục. Bảy nguyên tắc tinh gọn là:

- Eliminate waste.
- Amplify learning.
- Make decisions as late as possible.
- Deliver as fast as possible.
- Empower your team.
- Build integrity in.
- Build holistically.



Hình 9: Xếp hạng các mô hình SDLC được sử dụng rộng rãi dựa trên tính linh hoạt

Tài liệu tham khảo

[Gee24] GeeksforGeeks. (2024, August 29). *Software Development Life Cycle (SDLC)*. GeeksforGeeks. https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/

[AmaWe] *What is SDLC? - Software Development Lifecycle Explained - AWS*. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/sdlc/

[Alt24] Altvater, A. (2024, August 27). *What is SDLC? Understand the software development life cycle*. Stackify. https://stackify.com/what-is-sdlc/

[Synop] *What is the Software Development Life Cycle (SDLC) and how does it work? | Synopsys*. (n.d.). https://www.synopsys.com/glossary/what-is-sdlc.html

[Tutsp] *SDLC - overview*. (n.d.). https://s.net.vn/h50j

[Cla24] Clark, H. (2024, June 3). *The Software Development Life Cycle (SDLC): 7 phases and 5 models*. The Product Manager. https://shorturl.at/G17WP