**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

-------------o0o------------



**TIỂU LUẬN MÔN**

**ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM**

**ĐỀ TÀI: QUẢN LÝ MÃ NGUỒN VÀ ĐẢM BẢO CHẤT LƯỢNG MÃ NGUỒN**

**Giảng viên hướng dẫn**: Phạm Thị Thương

**Nhóm thực hiện** : Nhóm 9

**Lớp**: KTPM - K17B

**Tên thành viên:** Trần Minh Long

Dương Văn Định

Nguyễn Văn Nghiệp

Nguyễn Văn Hiếu

Trần Đức Long

*Thái Nguyên, tháng 11 năm 2021*



Mục lục

[Chương 1:Tổng quan về đảm bảo chất lượng phần mềm 4](#_Toc99436336)

[1.1. Chất lượng phần mềm là gì? 4](#_Toc99436337)

[1.2. Chất lượng phần mềm là gì theo Roger Pressman 4](#_Toc99436338)

[1.3.Quản lý chất lượng phần mềm là gì? 5](#_Toc99436339)

[1.4. Mục tiêu của hệ thống quản lý chất lượng 5](#_Toc99436340)

[1.5. 10 yếu tố cơ bản của quản lý chất lượng 6](#_Toc99436341)

[1.6.Một số nguyên nhân gây ra lỗi phần mềm cần biết 6](#_Toc99436342)

[1.6. Phân loại yêu cầu phần mềm ứng với các yếu tố chất lượng phần mềm 9](#_Toc99436343)

[Chương 2: QUẢN LÝ MÃ NGUỒN VÀ ĐẢM BẢO CHẤT LƯỢNG MÃ NGUỒN 14](#_Toc99436344)

[2.1 Quản lý mã nguồn 14](#_Toc99436345)

[2.1.1 Giới thiệu chung về quản lý mã nguồn 14](#_Toc99436346)

[2.1.2 Mô hình hoạt động 15](#_Toc99436347)

[2.1.3 Lợi ích của việc quản lý mã nguồn 19](#_Toc99436348)

[2.1.4 Git & Github 21](#_Toc99436349)

[2.2. Đảm bảo chất lượng mã nguồn 26](#_Toc99436350)

[2.2.1Khái niệm chất lượng mã nguồn 26](#_Toc99436351)

[2.2.2. Tại sao chất lượng mã lại quan trọng 26](#_Toc99436352)

[2.2.3. Phân tích chất lượng mã: Mã tốt so với Mã xấu 27](#_Toc99436353)

[2.2.4. Làm thế nào để đo lường chất lượng mã nguồn 27](#_Toc99436354)

[2.2.5. Các chỉ số chất lượng mã nào sẽ sử dụng 29](#_Toc99436355)

[2.2.6. Cách cải thiện chất lượng mã nguồn 30](#_Toc99436356)

[2.2.7. phân tích chất lượng mã nguồn 31](#_Toc99436357)

[2.2.8. Công cụ phân tích sonarqube 32](#_Toc99436358)

[CHƯƠNG 3: XÂY DỰNG DEMO 40](#_Toc99436359)

[3.1. Quản lý mã nguồn 40](#_Toc99436360)

[3.1.1 thao tác với git 40](#_Toc99436361)

[3.1.2. triển khai trên github 43](#_Toc99436362)

[3.2. Đảm bảo chất lượng mã nguồn 54](#_Toc99436363)

[TÀI LIỆU THAM KHẢO 71](#_Toc99436364)

# Chương 1:Tổng quan về đảm bảo chất lượng phần mềm

## 1.1. Chất lượng phần mềm là gì?

Chất lượng được hiểu là tốt hay không tốt, bền hay không bền, sử dụng được lâu hay không? Đối với phần mềm, chất lượng là mức độ mà nó đáp ứng các yêu cầu của người dùng. Chất lượng phần mềm được đánh giá dựa trên nhiều tiêu chí khác nhau như:

Mức độ mà sản phẩm phần mềm đáp ứng các yêu cầu của khách hàng. Tức là khi sản xuất phần mềm, lập trình viên phải làm việc với khách hàng để làm rõ khách hàng muốn gì và yêu cầu chi tiết đối với phần mềm đó?

Mức độ mà phần mềm hoặc ứng dụng có thể đáp ứng nhu cầu của khách hàng hoặc người dùng cuối. Đây là liệu khách hàng có thỏa mãn được nhu cầu khi sử dụng hay không? Tuy nhiên, để đánh giá chính xác chất lượng của một phần mềm, khách hàng phải có khả năng phân tích những nhận xét, đánh giá nhất định.

## 1.2. Chất lượng phần mềm là gì theo Roger Pressman

Theo Roger Pressman, một kỹ sư phần mềm nổi tiếng của Mỹ cho biết. Chất lượng phần mềm được thể hiện khi nó có thể đáp ứng các yêu cầu chức năng, hiệu suất, các tiêu chuẩn nhất định.

Ở Việt Nam hiện nay, phần mềm có nhiều lỗ hổng lớn, nhưng chúng thường được phát hiện khi có sự cố lớn. Điều này khiến khách hàng dần mất lòng tin, tuy nhiên không phải nhà cung cấp phần mềm nào cũng kém chất lượng.

Có thể thấy chất lượng phần mềm đang là vấn đề được nhiều người quan tâm. Tuy nhiên, theo đánh giá của các chuyên gia trên thế giới, công nghiệp phần mềm tại các công ty ở Việt Nam vẫn còn nhiều hạn chế. Tuy nhiên, vẫn có một số đơn vị trong nước đã đáp ứng một số tiêu chuẩn nhất định như CMM / CMMI. Đây là một trong những tiêu chuẩn để đánh giá chất lượng và năng lực của phần mềm.

## 1.3.Quản lý chất lượng phần mềm là gì?

Phần mềm quản lý chất lượng, theo dõi và giám sát tất cả các quá trình từ khi tiếp nhận dự án. Kiểm tra phần mềm giám sát xem có lỗi hay không. Xem các kỹ thuật viên sản xuất phần mềm có “chạy” theo yêu cầu hay không? Cuối cùng, kiểm thử trước khi bàn giao được coi là nhiệm vụ chính của quản lý chất lượng phần mềm.

Đối với khách hàng, họ chỉ cần phần mềm có thể hoạt động theo đúng yêu cầu và thời gian theo thỏa thuận ban đầu. Tuy nhiên, là nhà phát triển phần mềm, không phải phần mềm nào cũng có thể được sản xuất đúng thời hạn và theo yêu cầu. Vì vậy, quá trình kiểm tra, thử nghiệm để phát hiện lỗi là một khâu rất quan trọng.

Như vậy mới đảm bảo đúng tiêu chí và thời gian giao hàng cho khách. Đòi hỏi người quản lý phải thực hiện tốt công việc tổ chức, vận hành, kiểm tra, thử nghiệm phần mềm. Không chỉ vậy, người quản lý còn phải tổ chức chặt chẽ để duy trì hoạt động của cả hệ thống công việc liên quan đến dự án. Tại các công ty chuyên gia công phần mềm luôn có một hệ thống quản lý chất lượng phần mềm hiệu quả. Bao gồm các quy trình được thực hiện trong suốt chu kỳ phát triển của dự án.

Hiện nay, có rất nhiều mô hình đưa ra các tiêu chuẩn và hướng dẫn thực hiện quản lý chất lượng phần mềm. Theo đó, ISO 9001-2000 và CMM / CMMi là hai mô hình quản lý chất lượng phổ biến nhất.

Nếu ISO 9001-2000 là tiêu chuẩn được sử dụng để quản lý chất lượng cho tất cả các ngành công nghiệp, thì CMM / CMMi tập hợp các thực hành quản lý chất lượng.

## 1.4. Mục tiêu của hệ thống quản lý chất lượng

Hệ thống quản lý chất lượng thường có hai mục tiêu:

* Xây dựng chất lượng ngay từ khi bắt đầu tiếp nhận các dự án phần mềm. Điều này giúp cho việc giám sát quản lý phần mềm chặt chẽ hơn. Yêu cầu tất cả các quy trình phải được thực hiện đúng cách, giữa người yêu cầu và người thực hiện.
* Chất lượng phần mềm được đảm bảo trong suốt quá trình phát triển.

## 1.5. 10 yếu tố cơ bản của quản lý chất lượng

* Tiêu chuẩn (Tiêu chuẩn)
* Lập kế hoạch (Lập kế hoạch)
* Đánh giá (Đánh giá)
* Thử nghiệm)
* Phân tích khiếm khuyết
* Quản lý cấu hình
* Bảo mật (Security)
* Đào tạo (Giáo dục / Đào tạo)
* Quản lý nhà cung cấp (Quản lý nhà cung cấp)
* Quản lý rủi ro

Để chất lượng phần mềm được đảm bảo, 10 yếu tố này cũng như các giai đoạn của phần mềm phải có mối liên hệ với nhau.

## 1.6.Một số nguyên nhân gây ra lỗi phần mềm cần biết

**Lỗi trong định nghĩa yêu cầu:**

* Lỗi này được coi là nguồn gốc của các lỗi phần mềm.
* Định nghĩa yêu cầu lỗi: Định nghĩa sai, ví dụ về công thức sai
* Định nghĩa không đầy đủ: Yêu cầu không rõ ràng
* Thiếu yêu cầu

**Yêu cầu không cần thiết:**

* Nhiều dự án có các yêu cầu không bao giờ được sử dụng.
* Xem thêm: P / O Trong Xuất Nhập Khẩu là gì, Po (Đơn Mua Hàng) Trong Xuất Nhập Khẩu là gì
* Tác động đến ngân sách, mức độ phức tạp, thời gian phát triển, v.v.

**Mối quan hệ giữa khách hàng và nhà phát triển**

* Hiểu sai các tài liệu yêu cầu
* Hiểu sai tài liệu khi thay đổi
* Sự hiểu lầm (bằng miệng) thay đổi trong quá trình phát triển
* Thông báo cho khách hàng về giải pháp dành cho nhà phát triển về các thay đổi yêu cầu và Trả lời câu hỏi của khách hàng về câu hỏi của nhà phát triển.
* Đôi khi khách hàng trình bày với tư cách là người dùng, lập trình viên trình bày theo cách nghĩ khác.

**Cố ý vi phạm các yêu cầu phần mềm**

Nhà phát triển sử dụng lại cùng một kết quả trước đó để tiết kiệm thời gian. Theo đó, các kỹ thuật viên thường sử dụng lại mã, họ chỉnh sửa những phần không cần thiết hoặc không sử dụng. Sau đó, chèn các “cải tiến” không thể chấp nhận được, có thể là cải tiến mã, sắp xếp / tìm kiếm mới…) do áp lực thời gian. Điều này có thể dẫn đến mất một số tính năng.

**Lỗi thiết kế logic**

* Thuật toán sai: Công thức sai, bảng quyết định sai, toán tử / toán hạng sai…
* Định nghĩa quy trình: các quy trình trong hệ thống không phản ánh chính xác quy trình kinh doanh.

Lưu ý: không phải lỗi nào cũng là lỗi phần mềm.

Đây là một lỗi thủ tục và không phải là một phần của hệ thống…

Lỗi khi xác định các điều kiện biên – nguồn lỗi phổ biến: Hãy cẩn thận với các giá trị ranh giới, ví dụ: “không nhiều hơn” “ít hơn”, “n lần trở lên;” “lần đầu tiên,” …

Thiếu trạng thái phần mềm: Nếu thứ hạng là> = O1 và RPI là số, thì…. Rất dễ bỏ lỡ các hành động dựa trên trạng thái phần mềm.

Bỏ qua các định nghĩa liên quan đến phản hồi hành động không hợp lệ trong phần mềm có mã để phát hiện các hành động không hợp lệ nhưng không thiết kế các hành động phản hồi phần mềm. Ví dụ: chuông báo thức, v.v.

**Lỗi lập trình**

Trong phần mềm, có rất nhiều thứ liên quan đến lỗi mã hóa, chẳng hạn như: Lỗi ngữ pháp, Lỗi logic (chương trình chạy; sai kết quả). Lỗi thời gian chạy (sự cố trong quá trình thực thi).

**Không tuân theo tài liệu và hướng dẫn mã hóa**

Đây cũng là một lỗi phổ biến ở các nhà phát triển phần mềm. Một số kỹ thuật thường không tuân theo các khuôn mẫu, không tuân theo các tiêu chuẩn mã hóa.

**Những tồn tại của quá trình thử nghiệm**

Là một phần của quy trình phát triển dự án phần mềm, tuy nhiên các đơn vị thường cắt xén! Nhiều công ty phần mềm không có kế hoạch kiểm thử, hoặc có nhưng chưa đầy đủ, chưa đầy đủ: Không kiểm thử tất cả các phần của ứng dụng, Không phát hiện lỗi trong tài liệu, báo cáo. Không thể phát hiện chính xác lỗi do mô tả lỗi không rõ ràng hoặc không đủ thời gian để sửa lỗi.

**Giao diện người dùng và lỗi thủ tục**

Giao diện không có hướng dẫn để người dùng thực hiện các hành động cần thiết với mỗi bước của quy trình. Điều này rất quan trọng, đặc biệt là đối với phần mềm phức tạp. Điều này đòi hỏi một quy trình gồm nhiều bước. Mỗi bước xử lý nhiều kiểu dữ liệu khác nhau cho phép kiểm tra các kết quả trung gian.

**Lỗi tài liệu**

* Các lỗi trong thiết kế tài liệu, hướng dẫn sử dụng, trợ giúp trực tuyến ..
* Liệt kê các chức năng không tồn tại: Đã có kế hoạch phát triển, nhưng bị hoãn lại và không có thời gian để sửa tài liệu.
* Thông báo lỗi vô nghĩa xuất hiện không phù hợp.
* Đặc tả: đặc tả lỗi, không đầy đủ, không nhất quán.
* Thiết kế: các lỗi cơ bản trong thiết kế phần mềm. Cài đặt (Code): lỗi lập trình, mã độc (mã độc).
* Hệ thống hỗ trợ: Ngôn ngữ lập trình kém, trình biên dịch bị lỗi…
* Kiểm tra không đầy đủ: kiểm tra không đầy đủ, kiểm tra kém, v.v.

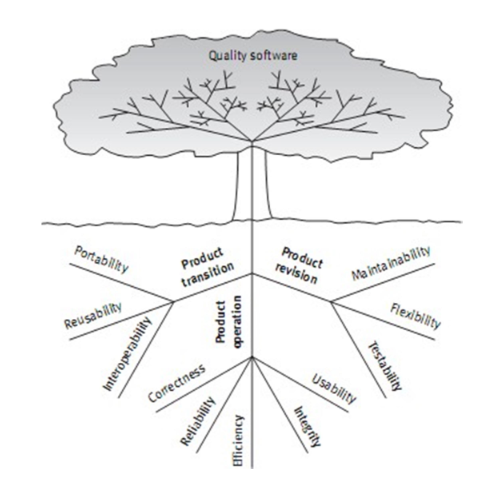
## 1.6. Phân loại yêu cầu phần mềm ứng với các yếu tố chất lượng phần mềm

Đã có nhiều tác giả nghiên cứu về các yếu tố chất lượng phần mềm từ các yêu cầu cả nó. Theo thời gian có thể quan niệm về việc đảm bảo chất lượng phần mềm có phần thay đổi, tuy nhiên mô hình các yếu tố đảm bảo chất lượng phần mềm của McCall ra đời vào những năm 70 của thế kỷ trước vẫn còn được nhiều người nhắc đến như là cơ sở tham chiếu các yêu cầu phần mềm. Sau McCall cũng có một số mô hình được quan tâm như mô hình do Evans, Marciniak do hay mô hình của Deutsch và Willis, tuy nhiên những mô hình này chỉ bổ sung hay sửa đổi một vài yếu tố chất lượng. Theo McCall, các yếu tố chất lượng phần mềm được chia làm ba loại :

- Các yếu tố hoạt động của sản phẩm bao gồm tính chính xác, tin cậy, hiệu quả, tính toàn vẹn, sử dụng được

- Các yếu tố rà soát bao gồm tính bảo trì, linh hoạt, có thể test được

- Các yếu tố chuyển giao bao gồm tính khả chuyển, có khả năng sử dụng lại, có khả năng giao tác.



Hình Cây mô hình yếu tố chất lượng phần mềm theo McCall

Chi tiết các thuộc tính được phân tích như sau :

1. Các yếu tố vận hành sản phẩm : Sự chính xác, độ tin cậy, tính hiệu quả, tính toàn vẹn và khả năng sử dụng được :

**- Sự chính xác** : Các yêu cầu về độ chính xác được xác định trong một danh sách các đầu ra cần thiết của hệ thống phần mềm, như màn hình hiển thị truy vấn số dư của khách hàng trong một hệ thống thông tin kế toán bán hàng…Các đặc tả đầu ra thường là đa chiều, một số chiều thông dụng là :

* Độ chính xác yêu cầu của các đầu ra này; chúng có thể bị ảnh hưởng bất lợi bởi các tính toán không chính xác hay các dữ liệu không chính xác.
* Tính đầy đủ của thông tin đầu ra; chúng có thể bị ảnh hưởng bất lợi bởi dữ liệu không đầy đủ.
* Up-to-dateness của thông tin (xác định bằng thời gian giữa sự kiện và việc xem xét hệ thống phần mềm.
* Độ sẵn sàng của thông tin (thời gian đáp ứng : được định nghĩa là thời gian cần thiết để có được các thông tin yêu cầu)
* Các chuẩn cho việc code và viết tài liệu cho hệ thống phần mềm.

**- Độ tin cậy :** Các yêu cầu về độ tin cậy giải quyết các lỗi để cung cấp dịch vụ. Chúng xác định tỷ lệ lỗi hệ thống phần mềm tối đa cho phép, các lỗi này có thể là lỗi toàn bộ hệ thống hoặc một hay nhiều chức năng riêng biệt của nó.

**- Tính hiệu quả :** Các yêu cầu về tính hiệu quả giải quyết vấn đề về các tài nguyên phần cứng cần thiết để thực hiện tất cả các chức năng của hệ thống phần mềm với sự phù hợp của tất cả các yêu cầu khác. Các tài nguyên phần cứng chính được xem xét ở đây là khả năng xử lý của máy tính (được đo bằng MIPS – triệu lệnh/giây; MHz – triệu chu kỳ/giây…); khả năng lưu trữ dữ liệu (dung lượng bộ nhớ, dung lượng đĩa – được đo bằng MBs, GBs, TBs…) và khả năng truyền dữ liệu (thường được đo bằng MBPS, GBPS ). Các yêu cầu này có thể bao gồm cả các giá trị tối đa tài nguyên phần cứng được sử dụng trong hệ thống phần mềm. Một yêu cầu khác về tính hiệu quả đó là thời gian giữa các lần phải sạc điện đối với các hệ thống nằm trên các máy tính xách tay hay các thiết bị di động.

- Các yêu cầu về tính toàn vẹn giải quyết các vấn đề về bảo mật hệ thống phần mềm, các yêu cầu này để ngăn chặn sự truy cập trái phép, để phân biệt giữa phần lớn nhân viên chỉ được phép xem thông tin với một nhóm hạn chế những người được phép thêm và thay đổi dữ liệu…

- Các yêu cầu về khả năng sử dụng được sẽ đưa ra phạm vi của tài nguyên nhân lực cần thiết để đào tạo một nhân viên mới và để vận hành hệ thống phần mềm.

1. Các yếu tố về rà soát sản phẩm : bảo trì được, linh động và kiểm tra được :

- Khả năng bảo trì được : Các yêu cầu về khả năng bảo trì được sẽ xác định người dùng và nhân viên bảo trì phải nỗ lực thế nào để xác định được nguyên nhân của các lỗi phần mềm, để sửa lỗi và để xác nhận việc sửa lỗi thành công. Các yêu cầu của yếu tố này nói tới cấu trúc modul của phần mềm, tài liệu chương trình nội bộ và hướng dẫn sử dụng của lập trình viên…

- Tính linh động : Các yêu cầu về tính linh động cũng bao gồm cả các khả năng và nỗ lực cần thiết để hỗ trợ các hoạt động bảo trì. Chúng gồm các nguồn lực (man- day) cần thiết để thích nghi với một gói phần mềm, với các khách hàng trong cùng nghề, với các mức độ hoạt động khác nhau, với các loại sản phẩm khác nhau…Các yêu cầu về yếu tố này cũng hỗ trợ các hoạt động bảo trì trở nên hoàn hảo, như thay đối và bổ sung vào phần mềm để tăng dịch vụ của nó và để thích nghi với các thay đổi trong môi trường thương mại và kỹ thuật của công ty.

- Khả năng test được : Các yêu cầu về khả năng kiểm tra được nói tới việc kiểm tra sự vận hành có tốt hay không của các hệ thống thông tin. Các yêu cầu về khả năng kiểm tra được liên quan tới các tính năng đặc biệt trong chương trình giúp người tester dễ dàng thực hiện công việc của mình hơn, ví dụ như đưa ra các kết quả trung gian. Các yêu cầu về khả năng kiểm tra được liên quan tới vận hành phần mềm bao gồm các chuẩn đoán tự động được thực hiện bởi hệ thống phần mềm trước khi bắt đầu hệ thống, để tìm hiểu xem có phải tất cả các thành phần của hệ thống phần mềm đều làm việc tốt hay không, và để có một bản báo cáo về các lỗi đã được phát hiện. Một loại khác của yêu cầu này là việc check các dự đoán tự động, được các kỹ thuật viên bảo trì sử dụng để phát hiện nguyên nhân gây lỗi phần mềm.

1. Các yếu tố về chuyển giao sản phẩm : tính lưu động (khả năng thích nghi với môi trường), khả năng tái sử dụng và khả năng cộng tác được :

- Tính lưu động : các yêu cầu về tính lưu động nói tới khả năng thích nghi của hệ thống phần mềm với các môi trường khác, bao gồm phần cứng khác, các hệ điều hành khác…Các yêu cầu này đòi hỏi các phần mềm cơ bản có thể tiếp tục sử dụng độc lập hoặc đồng thời trong các trường hợp đa dạng.

- Khả năng tái sử dụng : Các yêu cầu về khả năng tái sử dụng nói tới việc sử dụng các modul phần mềm trong một dự án mới đang được phát triển mà các modul này ban đầu được thiết kế cho một dự án khác. Các yêu cầu này cũng cho phép các dự án tương lai có thể sử dụng một modul đã có hoặc một nhóm các modul hiện đang được phát triển. Tái sử dụng phần mềm sẽ tiết kiệm tài nguyên phát triển, rút ngắn thời gian phát triển và tạo ra các moduls chất lượng cao hơn. Chất lượng modul cao hơn là dựa trên giả định rằng hầu hết các lỗi phần mềm đều được phát hiện bởi các hoạt động đảm bảo chất lượng phần mềm thực hiện trên phần mềm ban đầu, bởi những người sử dụng phần mềm ban đầu và trong suốt những lần tái sử dụng trước của nó. Các vấn đề về tái sử dụng phần mềm đã trở thành một phần trong chuẩn công nghiệp phần mềm (IEEE,1999).

- Khả năng cộng tác : Các yêu cầu về khả năng cộng tác tập trung vào việc tạo ra các giao diện với các hệ thống phần mềm khác. Các yêu cầu về khả năng cộng tác có thể xác định tên của phần mềm với giao diện bắt buộc. Chúng cũng có thể xác định cấu trúc đầu ra được chấp nhận như một tiêu chuẩn trong một ngành công nghiệp cụ thể hoặc một lĩnh vực ứng dụng.

# Chương 2: QUẢN LÝ MÃ NGUỒN VÀ ĐẢM BẢO CHẤT LƯỢNG MÃ NGUỒN

## 2.1 Quản lý mã nguồn

### 2.1.1 Giới thiệu chung về quản lý mã nguồn

Trong quá trình phát triển chương trình phần mềm, sau một thời gian mã nguồn chương trình sẽ ngày càng nhiều, rất khó để lập trình viên có thể kiểm soát được các chức năng đã thực hiện, cũng như quản lý tất cả mã nguồn đã viết ra. Đặc biệt đối với nhóm lập trình thì việc chia sẻ mã nguồn, quản lý công việc giữa các thành viên trong nhóm càng trở nên cấp thiết nhằm không bị chồng chéo công việc cũng như tăng tốc thực hiện dự án. Chính vì thế, việc áp dụng hệ thống quản lý phiên bản mã nguồn sẽ giúp giải quyết tất cả các vấn đề trên, giúp các thành viên trong dự án quản lý công việc, quản lý mã nguồn một cách hiệu quả.

Hệ thống quản lý phiên bản mã nguồn - Version Control System (VCS) là một hệ thống được sử dụng trong việc phát triển phần mềm, giúp lưu trữ và ghi nhận sự thay đổi của các tập tin, mã nguồn theo thời gian, nhờ đó hệ thống có thể theo dõi việc cập nhật sửa đổi mã nguồn của các thành viên trong suốt quá trình làm dự án.

Version control được hiểu như 1 công cụ giúp theo dõi sự thay đổi của toàn bộ cấu trúc chương trình, từ các file code cho đến các file hình ảnh, video. Với version control, ta có thể tạo ra nhiều phiên bản của các file ứng với từng thay đổi mà chính mình tạo ra. Các VCS sẽ lưu trữ tất cả các file trong toàn bộ dự án và ghi lại toàn bộ lịch sử thay đổi của file. Mỗi sự thay đổi được lưu lại sẽ được và thành một version (phiên bản).

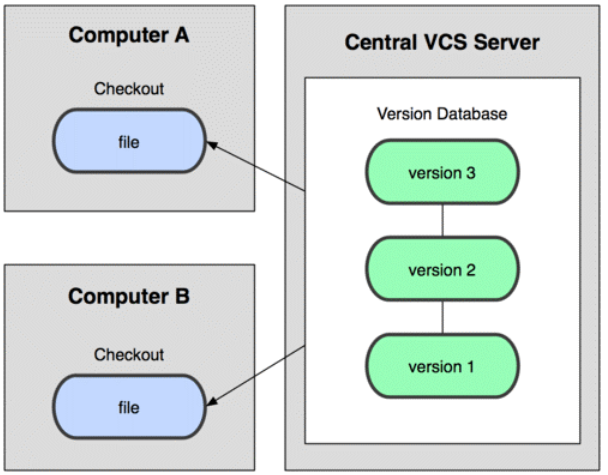
VCS nghĩa là hệ thống giúp lập trình viên có thể lưu trữ nhiều phiên bản khác nhau của một mã nguồn được nhân bản (clone) từ một kho chứa mã nguồn (repository), mỗi thay đổi vào mã nguồn trên local sẽ có thể ủy thác (commit) rồi đưa lên server nơi đặt kho chứa chính. Lập trình viên có thể xem lại danh sách các sự thay đổi của file như xem một dòng thời gian của các phiên bản. Mỗi phiên bản bao gồm: nội dung file bị thay đổi, ngày giờ sửa đổi, người thay đổi là ai, lý do thay đổi hay tên phiên bản

### 2.1.2 Mô hình hoạt động

#### 2.1.2.1 Mô hình quản lý mã nguồn tập trung (Centralized version control – CVC )

Mô hình sử dụng kiến trúc Client-Server. Server đóng vai trò là một kho dữ liệu (Repository) duy nhất lưu trữ tất cả các tập tin, mã nguồn, thay đổi… Các client sẽ kết nối đến server để lấy phiên bản mới nhất của mã nguồn về máy (check out) và cập nhật tất cả các thay đổi lên server (commit). Điển hình cho mô hình này có thể kể tới CVS (Concurrent Versioning System), Prefoce hay SVN (Subversion)

Trong hệ thống kiểm soát phiên bản tập trung (CVCS), một máy chủ hoạt động như một kho lưu trữ chính lưu trữ mọi phiên bản mã. Sử dụng kiểm soát nguồn tập trung, mọi người dùng cam kết trực tiếp với nhánh chính, vì vậy mô hình hoạt động này thường hoạt động tốt cho các nhóm nhỏ, vì các thành viên trong nhóm có khả năng giao tiếp nhanh chóng để không có hai nhà phát triển nào muốn làm việc trên cùng một đoạn mã đồng thời. Giao tiếp và kết nối liên tục mạnh mẽ là rất quan trọng để đảm bảo quy trình làm việc tập trung thành công.

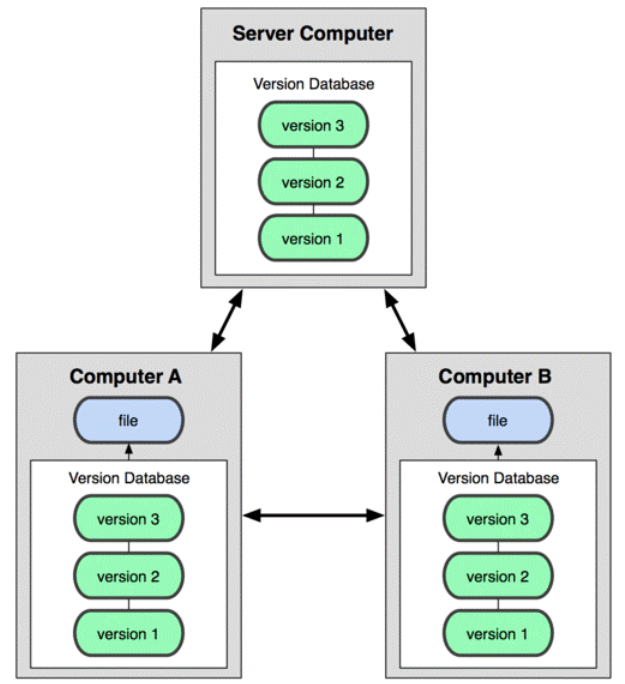


Các hệ thống kiểm soát mã nguồn nguồn tập trung, chẳng hạn như CVS, Perforce và SVN yêu cầu người dùng pull về phiên bản mới nhất từ máy chủ để tải xuống bản sao cục bộ trên máy của họ. Sau đó các lập trình viên viên sẽ push các commits đến máy chủ và giải quyết mọi xung đột hợp nhất trên kho lưu trữ chính.

Được triển khai dưới dạng client – server, quy trình làm việc tập trung cho phép khóa tệp để bất kỳ đoạn mã nào hiện đã được kiểm tra sẽ không thể truy cập được đối với người khác, đảm bảo rằng chỉ một nhà phát triển có thể đóng góp vào mã tại một thời điểm. Các thành viên trong nhóm sử dụng các nhánh để đóng góp vào kho lưu trữ trung tâm và máy chủ sẽ mở khóa các tệp sau khi hợp nhất.

* ***Ưu điểm của mô hình quản lý mã nguồn tập trung***
* Hoạt động tốt với các tệp nhị phân: Các tệp nhị phân, ví dụ các file dạng image hoặc văn bản, yêu cầu dung lượng lớn, vì vậy các nhà phát triển phần mềm chuyển sang hệ thống kiểm soát phiên bản tập trung để lưu trữ dữ liệu này. Với một máy chủ tập trung, các nhóm có thể kéo một vài dòng mã mà không cần lưu toàn bộ lịch sử trên máy cục bộ của họ. Người dùng của các hệ thống phân tán phải tải xuống toàn bộ dự án, điều này chiếm thời gian và không gian và ngăn họ làm sai khác. Nếu một nhóm làm việc với các tệp nhị phân thường xuyên, một hệ thống tập trung cung cấp cách tiếp cận hiệu quả nhất cho các nhà phát triển.
* Cung cấp tầm nhìn đầy đủ cho dự án: Với vị trí tập trung, mọi thành viên trong nhóm đều có toàn bộ khả năng hiển thị mã nào hiện đang được thực hiện và những thay đổi nào được thực hiện. Điều giúp các nhóm phát triển phần mềm hiểu được trạng thái của một dự án và cung cấp nền tảng cho sự cộng tác, vì các nhà phát triển chia sẻ công việc trong máy chủ trung tâm.
* Dễ hiểu và dễ sử dụng: Hệ thống quản lý tập trung dễ dàng sử dụng hơn phi tập trung, chính vì thế nó dễ dàng và phù hợp với lập trình viên ở mọi trình độ khác nhau. Điều này cũng giúp ít xảy ra các xung đột trong việc merge các mã nguồn lại với nhau.
* ***Nhược điểm của mô hình quản lý mã nguồn tập trung***
* Rủi do về dữ liệu: Do chỉ có một máy chủ lưu trữ dữ liệu và mọi thay đổi đều commit lên máy chủ nên khi xảy ra sự cố với máy chủ tập trung hoặc với máy tính cá nhân rất dễ bị mất dữ liệu. Trường hợp xấu nhất hoàn toàn có thể xảy ra đó là mất toàn bộ mã nguồn của dự án.
* Tốc độ chậm hơn và có thể ảnh hưởng tới tiến độ của dự án: Tốc độ vận hành của các nhóm phát triển phần mềm có ảnh hưởng trực tiếp đến tốc độ họ có thể gửi các tính năng và mang lại giá trị kinh doanh. Nếu các nhóm chậm phát triển, quá trình lặp lại và đổi mới bị đình trệ và các nhà phát triển có thể trở nên thất vọng với việc mất bao lâu để thấy những thay đổi của họ trong ứng dụng.
* Sự kết hợp với cùng vấn đề bị giảm: CVCS phù hợp với những dự án nhỏ và ít người. Bởi mỗi lập trình viên sẽ thực hiện một module của dự án và các lập trình viên khác không thể cùng thực hiện ( không thể cùng push). Điều này cũng ảnh hường nhiều tới tốc độ phát triển của dự án.

#### 2.1.2.2 Mô hình quản lý mã nguồn phân tán (Distributed version control – DVC)



Hệ thống kiểm soát phiên bản phân tán (DVCS) là một loại kiểm soát phiên bản trong đó cơ sở source code hoàn chỉnh - bao gồm cả lịch sử phiên bản đầy đủ của nó - được sao chép trên mọi máy tính của nhà phát triển.

Hệ thống kiểm soát phiên bản phân tán (Distributed Version Control System - DVCS) mang một bản sao cục bộ của kho lưu trữ hoàn chỉnh đến máy tính của mọi thành viên trong nhóm để họ có thể thực hiện Commit, phân nhánh và hợp nhất cục bộ. Máy chủ không phải lưu trữ tệp vật lý cho mỗi nhánh - nó chỉ cần sự khác biệt giữa mỗi commit. So với kiểm soát phiên bản tập trung, DVCS phân nhánh và hợp nhất quản lý tự động, tăng tốc hầu hết các hoạt động, cải thiện khả năng làm việc ngoại tuyến cho lập trình viên và không dựa vào một vị trí duy nhất để sao lưu mã nguồn.

Khi client lấy mã nguồn phiên bản mới nhất (clone) từ server về máy thì hệ thống sẽ sao chép luôn cả kho mã nguồn (repository), vì thế máy client lúc này sẽ đóng vai trò là kho mã nguồn (repository), mỗi kho con sẽ đồng bộ các thay đổi đến kho trên server chính. Ưu điểm của mô hình này là sự linh hoạt trong việc cập nhật lại mã nguồn trên server, client không cần lúc nào cũng phải duy trì kết nối đến server.

Git, Mercurial và Bazaar là một trong số những hệ thống quản lý mã nguồn phân tán phổ biến nhất.

* ***Ưu điểm của mô hình quản lý mã nguồn phân tán***
* Cung cấp quy trình làm việc và hệ thống phân cấp mạnh mẽ, các lập trình viên triển push các thay đổi mã vào kho lưu trữ của riêng họ và những người bảo trì thiết lập quy trình xem xét mã để đảm bảo chỉ có mã chất lượng hợp nhất vào kho lưu trữ chính.
* Đảm bảo sự phát triển ổn định của dự án, nhiều lập trình viên có thể tham gia đóng góp phát triển. Hệ thống tỏ ra cực kì hiệu quả với các dự án từ nhỏ tới lớn
* Các bản sao lưu đáng tin cậy: Mỗi máy tính khi pull project về đều có bản sao lưu ở trên chính máy tính đó. Có thể thực hiện các hoạt động commit ngay cả khi offline.
* Hợp nhất nhanh chóng và phân nhánh linh hoạt. Hỗ trợ rất nhanh việc hợp nhất mã nguồn, phân nhánh công việc cho từng team, từng thành viên làm việc.
* Phản hồi nhanh tróng và ít xảy ra xung đột mã nguồn
* Linh hoạt trong làm việc ngoại tuyến.
* **Nhược điểm của mô hình quản lý mã nguồn phân tán**
* Với dự án có lượng tài nguyên lớn ( các file ảnh, text lớn ) thì cần nhiều dung lượng lưu trữ hơn.
* Nếu dự án lớn, có nhiều tệp và file thời gian merge, lưu trữ có thể mất thời gian lớn
* Vẫn cần có một máy chủ trung tâm để kiểm soát lịch sử của các phiên bản trong dự án với nhiều thành viên
* Hoàn toàn có thể xảy ra conflict trong khi hợp nhất mã nguồn
* Cần phải trả phí hoặc duy trì máy chủ làm repository

### 2.1.3 Lợi ích của việc quản lý mã nguồn

* ***Lưu trữ mã nguồn tập trung***

Lợi ích chính của hệ thống quản lý mã nguồn là tạo ra môi trường lưu trữ tập tin, mã nguồn. Tuy nhiên không như cách lưu trữ truyền thống, hệ thống quản lý mã nguồn sẽ lưu trữ tập trung và giải quyết được các vấn đề dư thừa dữ liệu cũng như tìm kiếm nội dung lưu trữ.

Như ở ví dụ trên mã nguồn chương trình được lưu trữ trong nhiều tập tin nén khác nhau. Số lượng, dung lượng tập tin lưu trữ sẽ tăng trưởng theo thời gian thực hiện dự án.

Việc lưu trữ truyền thống như ví dụ trên sẽ làm dữ liệu trở nên dư thừa, tốn dung lượng lưu trữ cùng một tập tin giống nhau nhiều lần. Nhiều phiên bản khác nhau được đặt tên không theo qui cách cũng sẽ gây nhiều khó khăn trong việc tìm kiếm, xác định phiên bản, và không thể xác định được có thay đổi gì trong mã nguồn.

* ***Tạo môi trường làm việc nhóm***

Hệ thống quản lý mã nguồn sẽ tạo ra môi trường làm việc tập trung cho các thành viên tham gia dự án. Với cách làm truyền thống, mỗi người trong dự án làm việc độc lập lập, phải sao chép một bộ mã nguồn, phải thường xuyên liên hệ thông báo cho nhau mình đang chỉnh sửa phần nào trong mã nguồn, rất dễ xảy ra nhầm lẫn, sai sót, chỉnh sửa trùng hoặc xóa phần chỉnh sửa của nhau.

Hệ thống quản lý mã nguồn sẽ khắc phục tất cả các vấn đề tồn tại với cách làm truyền thống; tạo ra môi trường để mọi thành viên trong nhóm có thể tương tác, phối hợp với nhau, tránh việc thực hiện công việc trùng lặp, chồng chéo lẫn nhau. Người quản lý có thể quản lý mã nguồn, điều hành, tổng hợp công việc của nhóm. Lập trình viên tham gia các nhóm làm dự án có thể quản lý mã nguồn của mình, chia sẻ cũng như tiếp nhận mã nguồn từ đồng nghiệp.

* ***Lưu vết các thay đổi trong mã nguồn***

Không chỉ lưu trữ tập tin, mã nguồn, hệ thống còn cho phép lưu trữ các thay đổi trong quá trình thực hiện dự án. Các thay đổi được lưu lại không chỉ là tạo mới, xóa tập tin mà còn nội dung bị thay đổi trong tập tin mã nguồn.

Mỗi lần có các thay đổi trong mã nguồn dự án, hệ thống quản lý mã nguồn sẽ cung cấp thông tin về mục đích thay đổi, phần nào của mã nguồn bị thay đổi, người cập nhật… Các lần thay đổi sẽ được lưu trữ trong suốt quá trình làm dự án, cho phép quay ngược lại phiên bản trước đó nếu có phát sinh lỗi hoặc các vấn đề không mong muốn trong mã nguồn.

Có được một kho dữ liệu lịch sử, lưu vết các lần thay đổi trong mã nguồn sẽ giúp người quản lý theo dõi tiến trình làm dự án, giúp lập trình viên dễ tiếp cận, dễ tìm hiểu về mã nguồn khi được giao thực hiện dự án. Tất cả lịch sử thay đổi đều cố thể được đưa vào Hệ thống quản lý dự án như DevOps, JIRA… để quản lý tiến độ dự án cũng như thực hiện các phân tích khác.

### 2.1.4 Git & Github

#### 2.1.4.1 Tổng quan về Git

Git là một hệ thống quản lý phiên bản phân tán (Distributed Version Control System – DVCS) được phát triển bởi Linus Torvalds vào năm 2005, nó là một trong những hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay. Git cung cấp cho mỗi lập trình viên kho lưu trữ (repository) riêng chứa toàn bộ lịch sử thay đổi.

***Phân nhánh và hợp nhất (Branching and Merging)***

Git cho phép và khuyến khích lập trình viên làm việc tại các nhánh cục bộ có thể hoàn toàn độc lập với nhau. Việc tạo, hợp nhất và xóa các dòng phát triển đó chỉ mất vài giây. Điều này cho phép lập trình viên:

* Triển khai ý tưởng mới mà không sợ ảnh hưởng tới mã nguồn cũ trước đó
* Các branch được phân chia vai trò rõ ràng ( 1 nhánh chính productions và các nhánh phụ dùng để phát triển
* Dễ dàng thử nhiệm các tính năng, nếu không phù hợp, không hoạt động chỉ cần xóa bỏ chúng

***Nhỏ và nhanh (Small and Fast)***

Hầu như tất cả các hoạt động trên Git đều được thực hiện cục bộ, mang lại lợi thế về tốc độ rất lớn cho các hệ thống tập trung liên tục phải giao tiếp với máy chủ ở đâu đó.

Git được xây dựng để hoạt động trên nhân Linux nên rất hiệu quả trong việc xử lý luồng dữ liệu lớn. Git được viết bằng ngôn ngữ C, giảm chi phí thời gian chạy liên quan đến các ngôn ngữ cấp cao hơn.

***Là một hệ thống phân tán***

Phân tán là một trong những tính năng nổi bật nhất nhất của bất kỳ VCS tán nào, bao gồm Git, là nó được phân phối. Thay vì chỉ làm việc với một phần của dự án thì Git cho phép người dùng clone toàn bộ dự án về và thao tác trên đó.

***Tính toàn vẹn dữ liệu***

Mô hình dữ liệu mà Git sử dụng đảm bảo tính toàn vẹn của mật mã cho từng bit trong dự án của người dùng. Mọi tệp và commit đều được tổng kiểm tra và truy xuất bởi tổng kiểm tra của nó khi kiểm tra lại. Không thể lấy bất cứ thứ gì ra khỏi Git ngoài các bit chính xác mà người dùng đã đưa vào. Ngoài ra nhờ có các bản sao lưu dữ liệu local, dự án đảm bảo không bị mất mát dữ liệu.

***Git Free và Open Source***

Git là mã nguồn mở và hoàn toàn miễn phí đối với tất cả người dùng.

***Cơ chế hoạt động của git:***

* Git sẽ quản lý dữ liệu bằng tập hợp các "ảnh" (snapshot) của một hệ thống tập tin.
* Mỗi lần người dùng lưu lại trạng thái hiện tại của dự án, về cơ bản Git "chụp một bức ảnh" ghi lại nội dung của tất cả các tập tin tại thời điểm đó và tạo ra một tham chiếu tới "ảnh" đó.
* Để hiệu quả hơn, nếu như tập tin không có sự thay đổi nào, Git không lưu trữ tập tin đó lại một lần nữa mà chỉ tạo một liên kết tới tập tin gốc đã tồn tại trước đó.

#### 2.1.4.2 Một số thành phần của Git

* **Repository**

Trong hệ thống quản lý phiên bản, một repository là một cấu trúc dữ liệu giúp lưu trữ metadata cho một tập hợp các file hoặc cấu trúc thư mục. Đây chính là nơi chứa tất cả mã nguồn cho một dự án được quản lý bởi Git. Mỗi repo sẽ có hai cấu trúc dữ liệu chính đó là Object store và Index được lưu trữ ẩn trong thư mục .git

Có lại loại repository đó là local repository và remote repository.

* Local repository: Là repo được cài đặt trên máy tính của lập trình viên, repo này sẽ đồng bộ hóa với remote repo bằng các lệnh của git.
* Remote repository: Là repo được cài đặt trên server chuyên dụng, điển hình hiện nay là Github, Gitlab, Bitbucket
* **Branch**

Branch là cái dùng để phân nhánh và ghi lại luồng của lịch sử. Branch đã phân nhánh sẽ không ảnh hưởng đến branch khác nên có thể tiến hành nhiều thay đổi đồng thời trong cùng 1 repository. Các hoạt động trên mỗi branch sẽ không ảnh hưởng lên các branch khác nên có thể tiến hành nhiều thay đổi đồng thời trên một repository, giúp giải quyết được nhiều nhiệm vụ cùng lúc.

* **Commit**

Git Commit là một lệnh cam kết trong Git, được dùng để lưu lại những thay đổi trong repository bằng các ảnh chụp nhanh (snapshot). Có thể hiểu Commit giúp trình viên đánh dấu, lưu lại lại các thay đổi của mình với mã nguồn.

#### 2.1.4.3 Ưu điểm của Git

***Vì sao nên sử dụng git?***

Các dự án thực tế thường có nhiều lập trình viên làm việc song song. Vì vậy, một hệ thống kiểm soát phiên bản như Git là cần thiết để đảm bảo không có xung đột code giữa các lập trình viên.

Ngoài ra, các yêu cầu trong các dự án như vậy thay đổi thường xuyên. Vì vậy, một hệ thống kiểm soát phiên bản cho phép các nhà phát triển revert và quay lại phiên bản cũ hơn của code.

Cuối cùng, đôi khi một số dự án đang được chạy song song liên quan đến cùng một cơ sở code. Trong trường hợp như vậy, khái niệm phân nhánh trong Git là rất quan trọng.

***Các ưu điểm của Git:***

* Dễ sử dụng, thao tác nhanh, gọn, lẹ và rất an toàn.
* Sễ dàng kết hợp các phân nhánh (branch), có thể giúp quy trình làm việc code theo nhóm đơn giản hơn rất nhiều.
* Chỉ cần clone mã nguồn từ kho chứa hoặc clone một phiên bản thay đổi nào đó từ kho chứa, hoặc một nhánh nào đó từ kho chứa là bạn có thể làm việc ở mọi lúc mọi nơi.
* Deployment sản phẩm của bạn một cách không thể nào dễ dàng hơn.
* Sắp xếp công việc dễ dàng và đơn giản hơn
* Linh hoạt hơn khi sử dụng một lúc nhiều task
* Git cho phép lập trình viên hoạt động offline
* Git hoàn toàn miễn phí

Cách lưu trữ thông minh khiến việc merge nhanh và chính xác, tối ưu được việc quản lý phiên bản

#### 2.1.4.4 Github

GitHub là một dịch vụ cung cấp kho lưu trữ trực tuyến (Remote Repository) mã nguồn Git dựa trên nền web cho các dự án phát triển phần mềm. GitHub cung cấp cả phiên bản trả tiền lẫn miễn phí cho các tài khoản.

GitHub là một dịch vụ nổi tiếng cung cấp kho lưu trữ mã nguồn Git cho các dự án phần mềm. Github có đầy đủ những tính năng của Git, ngoài ra nó còn bổ sung những tính năng về social để các developer tương tác với nhau.

**Một số ưu điểm của Github**

* Dịch vụ miễn phí, mặc dù nó cũng có các dịch vụ trả phí.
* Tìm kiếm rất nhanh trong cấu trúc repos.
* Cộng đồng lớn và dễ dàng tìm thấy sự giúp đỡ.
* Cung cấp các công cụ thiết thực để hợp tác và tích hợp tốt với Git.
* Dễ dàng tích hợp với các dịch vụ của bên thứ ba khác.
* Nó cũng hoạt động với TFS, HG và SVN.

**Nhược điểm của Github**

Github không phải là hoàn toàn mở.

* Github có giới hạn về dung lượng, vì bạn không thể vượt quá 100MB trong một tệp duy nhất, trong khi kho lưu trữ được giới hạn ở 1GB trong phiên bản miễn phí.

#### 2.1.4.5 Các bước cơ bản để triển khai một dự án với Git và github

**Bước 1:** Tạo mới một repository trên Github dùng làm remote Repository, sau đó lấy thông tin kết nối tới repository.

**Bước 2:** Tạo Local repository với git bằng câu lệnh **git init**

**Bước 3:** Thêm toàn bộ các tài nguyên của dự án vào git với câu lệnh **git add .**

**Bước 4:** Thực hiện commit các file vừa thêm ở bước trên bằng câu lệnh **git commit – m “message”**

**Bước 5:** Thêm remote bằng câu lệnh **git remote add** để có thể push lên remote repository đã tạo ở bước 1.

Ví dụ: **git remote add origin https://github.com/user/repository.git**

**Bước 6:** Push commit từ local repository lên remote repository với câu lệnh: **git git push origin [name\_of\_your\_new\_branch]**

**Một số câu lệnh thường dùng khác trong quá trình sử dụng Git:**

git merge [branch\_name] : Dùng để merge branch hiện tại với branch\_name

git checkout [branch\_name] : Chuyển tới branch\_name

git log : Kiểm tra các commit đã commit trước đó

git pull : Lấy về thông tin từ remote và cập nhật vào các nhánh của local repo.

git fetch: Lệnh git fetch tải về dữ liệu từ Remote Repo nhưng không thực hiện merge ngay vào cách nhánh của local repository

git clone [repository\_link] : clone toàn repository remote về local

## 2.2. Đảm bảo chất lượng mã nguồn

### 2.2.1. Khái niệm chất lượng mã nguồn

Chất lượng mã xác định mã tốt (high quality) - và mã nguồn xấu (low quality).

Chất lượng, tốt, xấu - đều là chủ quan. Các nhóm khác nhau có thể sử dụng các định nghĩa khác nhau, dựa trên ngữ cảnh. Mã nguồn được coi là chất lượng cao có thể có ý nghĩa đối với một nhà phát triển phần mềm. Và nó có thể có nghĩa khác đối với một nhà phát triển ứng dụng web.

Vì lý do đó, chúng tôi giải thích chất lượng mã nguồn là gì, cách cải thiện chất lượng mã, các chỉ số chất lượng mã quan trọng là gì và các công cụ chất lượng mã có thể trợ giúp như thế nào.

### 2.2.2. Tại sao chất lượng mã lại quan trọng

Chất lượng mã nguồn rất quan trọng, vì nó ảnh hưởng đến chất lượng phần mềm tổng thể. Và chất lượng ảnh hưởng đến mức độ an toàn, bảo mật và đáng tin cậy của cơ sở mã của bạn.

Chất lượng cao là yếu tố quan trọng đối với nhiều nhóm phát triển ngày nay. Và nó đặc biệt quan trọng đối với những người đang phát triển các hệ thống quan trọng về bảo mật.

### 2.2.3. Phân tích chất lượng mã: Mã tốt so với Mã xấu

Mã nguồn tốt (chất lượng cao). Và đó là mã sạch. Nó đứng trước thử thách của thời gian.

Mã xấu ( chất lượng thấp). Nó sẽ không tồn tại lâu đâu.

Về cơ bản, mã được coi là tốt:

* Làm đúng công việc.
* Tuân theo một phong cách nhất quán.
* Dễ hiểu.
* Đã được ghi chép đầy đủ.
* Đã được kiểm tra.

**Kiểm tra là không đủ**

Lập trình viên không hoàn hảo. Kiểm tra và đánh giá mã thủ công sẽ không bao giờ tìm thấy mọi lỗi trong mã.

Một nghiên cứu về “ [Nguồn gốc và phương pháp loại bỏ lỗi phần mềm](https://www.ifpug.org/Documents/Jones-SoftwareDefectOriginsAndRemovalMethodsDraft5.pdf) ” cho thấy rằng các lập trình viên cá nhân có hiệu suất thấp hơn 50% trong việc tìm ra lỗi trong phần mềm của họ. Và hầu hết các hình thức kiểm tra chỉ đạt hiệu quả 35%. Điều này gây khó khăn cho việc xác định chất lượng.

**Lỗi mã hóa dẫn đến rủi ro**

Chất lượng của mã nguồn trong lập trình là quan trọng. Khi mã có chất lượng thấp, nó có thể gây ra [rủi ro về](https://www.perforce.com/blog/qac/3-best-practices-secure-software-development) an toàn hoặc bảo mật . Nếu phần mềm bị lỗi - do vi phạm bảo mật hoặc lỗ hổng an toàn - thì kết quả có thể rất thảm khốc hoặc gây tử vong.

**Chất lượng là trách nhiệm của mọi người**

Chất lượng là công việc của mỗi người. Người sản xuất. Người thử nghiệm. Người quản lý. Chất lượng cao phải là mục tiêu xuyên suốt quá trình phát triển.

### 2.2.4. Làm thế nào để đo lường chất lượng mã nguồn

Không có cách nào để đo lường chất lượng mã nguồn. Những gì đo lường có thể khác với những gì nhóm phát triển khác đo lường.

**Các khía cạnh chất lượng chính của mã nguồn để đo lường**

Dưới đây là năm đặc điểm chính để đo lường chất lượng cao hơn.

* **độ tin cậy**

Độ tin cậy đo xác suất hệ thống sẽ chạy mà không bị lỗi trong một khoảng thời gian hoạt động cụ thể. Nó liên quan đến số lượng lỗi và tính khả dụng của phần mềm.

Số lượng khuyết tật có thể được đo bằng cách chạy một [công cụ phân tích tĩnh](https://www.perforce.com/blog/qac/what-static-code-analysis) . Tính khả dụng của phần mềm có thể được đo bằng [thời gian trung bình giữa các lỗi (MTBF)](https://whatis.techtarget.com/definition/MTBF-mean-time-between-failures) . Số lượng lỗi thấp đặc biệt quan trọng để phát triển một cơ sở mã đáng tin cậy.

* **Khả năng bảo trì**

Khả năng bảo trì đo lường mức độ dễ dàng bảo trì của phần mềm. Nó liên quan đến kích thước, tính nhất quán, cấu trúc và độ phức tạp của codebase. Và việc đảm bảo mã nguồn có thể bảo trì dựa vào một số yếu tố, chẳng hạn như khả năng kiểm tra và khả năng hiểu.

Không thể sử dụng một số liệu duy nhất để đảm bảo khả năng bảo trì. Một số số liệu có thể xem xét để cải thiện khả năng bảo trì là số lượng cảnh báo theo phong cách và các biện pháp độ phức tạp Halstead. Cả người đánh giá tự động và con người đều cần thiết để phát triển các cơ sở mã nguồn có thể bảo trì.

* **Khả năng kiểm tra**

Khả năng kiểm thử đo lường mức độ phần mềm hỗ trợ các nỗ lực kiểm thử. Nó phụ thuộc vào mức độ bạn có thể kiểm soát, quan sát, cô lập và tự động hóa thử nghiệm, trong số các yếu tố khác.

Khả năng kiểm tra có thể được đo lường dựa trên số lượng trường hợp kiểm thử cần để tìm ra các lỗi tiềm ẩn trong hệ thống. Kích thước và độ phức tạp của phần mềm có thể ảnh hưởng đến khả năng kiểm tra. Vì vậy, việc áp dụng các phương pháp ở cấp mã nguồn - chẳng hạn như [độ phức tạp](https://www.perforce.com/blog/qac/what-cyclomatic-complexity) theo chu kỳ - có thể giúp cải thiện khả năng kiểm tra của thành phần.

* **Tính di động**

Khả năng di động đo lường mức độ khả dụng của cùng một phần mềm trong các môi trường khác nhau. Nó liên quan đến tính độc lập của nền tảng.

Không có một thước đo cụ thể nào về tính di động. Nhưng có một số cách có thể đảm bảo mã di động. Điều quan trọng là phải thường xuyên kiểm tra mã trên các nền tảng khác nhau, thay vì đợi cho đến khi kết thúc quá trình phát triển. Cũng nên đặt mức cảnh báo trình biên dịch của mình càng cao càng tốt - và sử dụng ít nhất hai trình biên dịch. Việc thực thi một tiêu chuẩn mã hóa cũng giúp cải thiện tính di động.

* **Khả năng tái sử dụng**

Khả năng tái sử dụng đo lường liệu các tài sản hiện có, chẳng hạn như mã nguồn có thể được sử dụng lại hay không. Nội dung được tái sử dụng dễ dàng hơn nếu chúng có các đặc điểm như tính mô đun hoặc khớp nối lỏng lẻo.

Khả năng tái sử dụng có thể được đo lường bằng số lượng phụ thuộc lẫn nhau. Chạy một trình phân tích tĩnh có thể giúp xác định những sự phụ thuộc lẫn nhau này.

### 2.2.5. Các chỉ số chất lượng mã nào sẽ sử dụng

Có một số chỉ số có thể sử dụng để định lượng chất lượng mã nguồn

* **Các chỉ số khiếm khuyết**

Số lượng khiếm khuyết và mức độ nghiêm trọng của các vấn đề đó là những thước đo quan trọng của chất lượng tổng thể.

Điều này bảo gồm:

* Xác định giai đoạn bắt nguồn của khiếm khuyết.
* Số lượng báo cáo lỗi mở.
* Thời gian để xác định và sửa chữa các vấn đề.
* Mật độ khiếm khuyết (ví dụ, số lượng khiếm khuyết trên mỗi dòng mã).
* **Chỉ số độ phức tạp**

Các chỉ số đo độ phức tạp có thể giúp ích trong việc đo lường chất lượng.

Độ phức tạp chu kỳ đo lường số lượng đường dẫn độc lập tuyến tính thông qua mã nguồn của chương trình.

Một cách khác để hiểu chất lượng là tính toán các thước đo độ phức tạp của Halstead. Các biện pháp này:

* Từ vựng chương trình
* Thời gian chạy của chương trình
* Độ dài chương trình được tính toán

### 2.2.6. Cách cải thiện chất lượng mã nguồn

Đo lường chất lượng giúp bạn hiểu được vị trí của mình. Sau khi đã đo lường, có thể thực hiện các bước để cải thiện chất lượng tổng thể.

Bốn cách bạn có thể cải thiện chất lượng mã:

1. Sử dụng một tiêu chuẩn mã hóa.

2. Phân tích mã nguồn - trước khi đánh giá mã nguồn.

3. Thực hiện theo các phương pháp hay nhất về đánh giá mã nguồn.

4. Refactor mã nguồn kế thừa (khi cần thiết)

**Cải thiện chất lượng mã nguồn:**

* **Sử dụng tiêu chuẩn mã hóa**

Sử dụng [tiêu chuẩn mã hóa](https://www.perforce.com/resources/qac/coding-standards) là một trong những cách tốt nhất để đảm bảo mã nguồn chất lượng cao.

Tiêu chuẩn mã hóa đảm bảo mọi người sử dụng đúng kiểu. Nó cải thiện tính nhất quán và khả năng đọc của cơ sở mã. Đây là chìa khóa để giảm độ phức tạp và chất lượng cao hơn.

Cách tốt nhất để sử dụng tiêu chuẩn mã hóa là:

* Đào tạo các nhà phát triển
* Giúp họ tuân thủ nó

Có thể thực hiện việc này bằng cách sử dụng [trình phân tích mã nguồn tĩnh.](https://www.perforce.com/blog/qac/how-enforce-coding-standards-automated-static-analysis)

* **Phân tích- Trước khi đánh giá mã nguồn**

Chất lượng nên được ưu tiên ngay từ khi bắt đầu phát triển. Không phải lúc nào thời gian cũng xa xỉ khi quá trình phát triển tiến triển. Đó là lý do tại sao điều quan trọng là phải phân tích mã *trước khi* bắt đầu xem xét mã. Và tốt nhất nên phân tích mã nguồn ngay sau khi nó được viết.

Trong [DevOps](https://www.perforce.com/solutions/devops) , phân tích mã diễn ra trong giai đoạn tạo. Máy phân tích tĩnh có thể được chạy qua mã ngay sau khi nó được viết. Điều này tạo ra một vòng phản hồi tự động, vì vậy các nhà phát triển có thể cải thiện mã trước khi nó chuyển sang giai đoạn xem xét mã.

Xét cho cùng, bạn phát hiện ra lỗi càng sớm thì việc giải quyết chúng càng nhanh, dễ dàng và rẻ hơn.

Cách tốt nhất để [cải thiện chất lượng là phân tích mã](https://www.perforce.com/webinars/how-improve-quality-static-code-analysis) tự động. Bằng cách chạy trình phân tích tĩnh trên mã sớm và thường xuyên, đảm bảo mã đến giai đoạn xem xét mã có chất lượng cao nhất có thể. Ngoài ra, có thể sử dụng máy phân tích tĩnh (chẳng hạn như [Helix QAC](https://www.perforce.com/products/helix-qac) và [Klocwork](https://www.perforce.com/products/klocwork) ) để theo dõi các chỉ số chất lượng chính.

* **Thực hiện theo các phương pháp hay nhất về đánh giá mã**

Việc xem xét mã thủ công vẫn rất quan trọng để xác minh mục đích của mã. Khi việc đánh giá mã được thực hiện tốt, chúng sẽ cải thiện chất lượng phần mềm tổng thể.

Cách tốt nhất để thực hiện [đánh giá mã là làm theo các phương pháp hay nhất](https://www.perforce.com/blog/qac/9-best-practices-code-reviews)  và tận dụng các công cụ tự động.

### 2.2.7. phân tích chất lượng mã nguồn

Việc phân tích và đo lường chất lượng có thể rất phức tạp, vì chất lượng có thể mang tính chủ quan. Có thể sử dụng một số số liệu để đánh giá khách quan mã, bao gồm cả độ phức tạp theo chu kỳ. Và có một số cách có thể giảm độ phức tạp và cải thiện chất lượng.

Mã hóa chất lượng có thể mất nhiều thời gian và nỗ lực hơn trong lần đầu tiên vượt qua. Nhưng bằng cách sớm giới thiệu chất lượng, bạn sẽ giảm chi phí bảo trì và sửa lỗi về lâu dài. Và bạn sẽ giảm nợ kỹ thuật của mình.

**Chọn đúng công cụ đánh giá chất lượng mã nguồn**

Sử dụng các [công cụ đánh giá chất lượng mã](https://www.perforce.com/solutions/static-analysis) nguồn phù hợp , bao gồm cả bộ phân tích tĩnh, là chìa khóa.

Các bộ phân tích tĩnh - chẳng hạn như [Helix QAC](https://www.perforce.com/products/helix-qac) và [Klocwork](https://www.perforce.com/products/klocwork) - giúp dễ dàng đảm bảo rằng mã của bạn có chất lượng cao. Bạn sẽ cải thiện chất lượng bằng cách:

* Áp dụng các tiêu chuẩn mã hóa.
* Phân tích mã tự động.
* Tuân theo các phương pháp hay nhất về mã hóa.
* Cấu trúc lại mã kế thừa.

### 2.2.8. Công cụ phân tích sonarqube

#### 2.2.8.1. Công cụ phân tích SonarQube

**SonarQube** (trước đây là Sonar) là một nền tảng nguồn mở được phát triển bởi SonarSource để kiểm tra chất lượng mã nguồn đó thực hiện các đánh giá một cách tự động. SonarQube sử dụng phương pháp phân tích tĩnh (static analysis of code) để phát hiện lỗi, mã nguồn thừa không có tác dụng và lỗ hổng bảo mật trên 20 ngôn ngữ lập 9 trình. Là một nền tảng mã nguồn mở giúp nhà phát triển có thể kiểm tra chất lượng mã nguồn của dự án.

**SonarQube** được viết bằng java nhưng hỗ trợ các ngôn ngữ khác nhau: PHP, Ruby, Java (bao gồm cả Android), C#, JavaScript, TypeScript, C/C++, Kotlin, Go, COBOL, PL/SQL, PL/I, ABAP, VB.NET, VB6, Python, RPG, Flex, Objective-C, Swift, CSS, HTML, và XML và hỗ trợ các cơ sở dữ liệu để lưu trữ kết quả: MySql, Postgresql.

**SonarQube** hỗ trợ trong dự án việc đánh giá mã nguồn theo các tiêu chuẩn của từng ngôn ngữ có trong dự án. Bên cạnh đó có thể thực hiện những việc sau:

• Phát hiện lỗi: phát hiện mã nguồn không dùng đến, các mã nguồn bị trùng lặp.

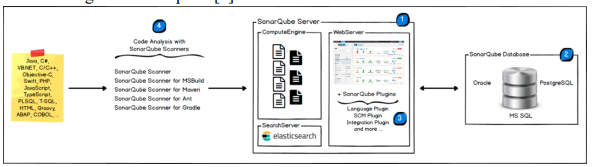
• Tính toán độ bao phủ của mã nguồn theo kiểm thử đơn vị (Unit test - Unittest coverage) – Đó là cách tính toán trong kiểm thử hộp trắng đảm bảo rằng tất cả các trường hợp trong mã nguồn đều được đảm bảo chạy qua khi chạy thực tế.

• Tính toán trong việc các mã nguồn được sử dụng mang tính tạm thời không được tối ưu để sử dụng lại và dễ bảo trì mã nguồn của hệ thống (Technical Debt) – Nghĩa là giải quyết vấn đề một cách nhanh chóng không theo quy chuẩn chỉ quan tâm đến kết quả. Giả sử đúng ra một giá trị phải khai báo là hằng số để sử dụng chung cho các trường hợp khác nhưng khi sử dụng lại dùng luôn giá trị chứ không khai báo biến hằng số. Ví dụ đoạn mã nguồn như sau: if (message == “Nhập thiếu thông tin”) {} thay vào đó có thể khai báo một biến là hằng số trong một lớp java như là Constant.java: public static final String MESSAGE = “Nhập thiếu thông tin”; Như vậy ở bất kỳ đâu cần sử dụng biến MESSAGE đều có thể sử dụng và khi thay đổi giá trị chỉ cần thay đổi ở một chỗ thay vì phải vào nhiều đoạn mã để thay đổi.

• So sánh chất lượng mã nguồn so với các lần kiểm tra trước.

• Phát hiện lỗ hổng bảo mật.

• Kiểm tra độ phức tạp của mã nguồn. Sonar gồm 4 thành phần:

Hình: 4 thành phần chính của SonarQube

➢ Một máy chủ SonarQube bắt đầu với 3 tiến trình chính:

* Máy chủ web cung cấp giao diện người dùng sử dụng và quản lý các phiên bản phân tích mã nguồn.
* Máy chủ tìm kiếm dựa trên Elasticsearch để sao lưu các tìm kiếm từ giao diện người dùng.
* Máy chủ phụ trách xử lý các báo cáo phân tích mã nguồn và lưu chúng vào cơ sở dữ liệu của SonarQube.

➢ Một cơ sở dữ liệu để lưu trữ:

* + Cấu hình của SonarQube (bảo mật, cài đặt plugin,...).
  + Ảnh chụp nhanh chất lượng của dự án, chế độ xem,...

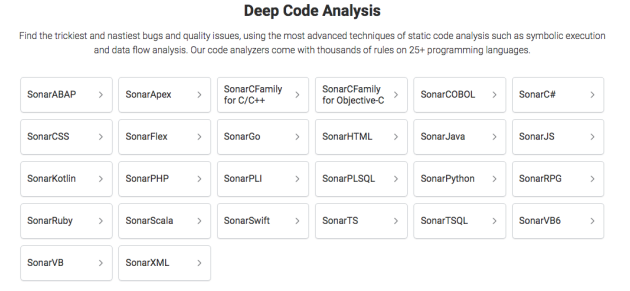
➢ Nhiều plugin SonarQube được cài đặt trên máy chủ, có thể bao gồm ngôn ngữ, SCM, tích hợp xác thực và quản trị.

➢ Một hoặc nhiều SonarScanners chạy trên máy chủ tích hợp liên tục và xây dựng để phân tích dự án.

**SonarQube Scanners** là các plugin sẽ được chạy ở phía người dùng. Giúp thu thập thông tin từ dự án, sinh các “thuộc tính" cho việc phân tích. Sau đó nó sẽ chạy bộ phân tích SonarQube dựa trên các “thuộc tính” này. Tương ứng với mỗi công cụ quản lý dựng (build management), sẽ có một Scanner tương ứng.

**SonarQube Plugins** bao gồm các plugin về ngôn ngữ lập trình, quản lý phiên bản mã nguồn hoặc thậm chí là các plugin cho các công cụ phân tích mã nguồn khác như PMD, FindBugs, jDepend, Android Lint, CheckStyle, WebDriver,…

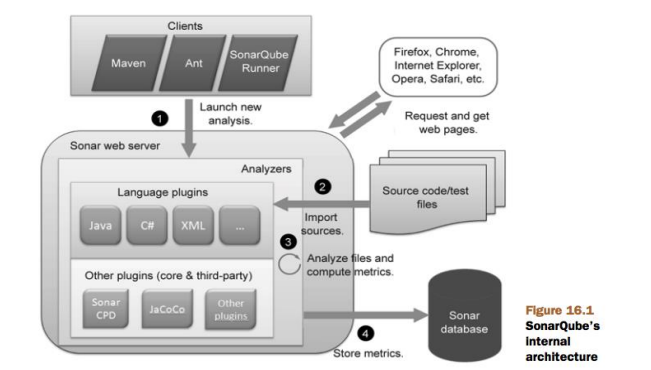
**SonarQube** hiện tại hỗ trợ cho 26 ngôn ngữ lập trình, do vậy sẽ có 26 plugin tương ứng để phân tích mã nguồn cho loại ngôn ngữ đấy.



Hình: Các ngôn ngữ SonarQube hỗ trợ

Một số kỹ thuật phân tích mã nguồn tĩnh bậc cao được sử dụng như: khớp mẫu (pattern matching), phân tích luồng dữ liệu, thực thi tượng trưng. SonarQube công bố các luật (rules) cho các ngôn ngữ này dựa theo các chuẩn nổi tiếng như CWE, SANS, OWASP, MISRA, CERT.

Cách thức hoạt động của SonarQube:



Hình: Cách thức hoạt động của SonarQube

• Sau khi các thông tin dự án được quét, thu thập, xử lý ở máy khách (client).

• **SonarQube Scanner** sẽ gửi những thông tin này ở dạng tệp nén lên máy chủ (server) thông qua các API mà máy chủ cung cấp. Ngay khi máy chủ nhận được yêu cầu xử lý này, việc đầu tiên là SonarQube sẽ nạp toàn bộ mã nguồn và các tệp đi kèm nhờ plugin ngôn ngữ. Sau mã nguồn được nạp xong, các bộ phân tích sẽ lần lượt quét qua mã nguồn và tạo ra chỉ số đo lường hoặc phát hiện các vấn đề. Các bộ phân tích được chia làm hai loại là sensor và decorator. Sensor là các bộ phân tích có thể tự tạo ra và cập nhật các chỉ số từ mã nguồn. Decorator là các bộ phân tích sử dụng chỉ số từ các sensor để phân tích sinh ra các chỉ số ở mức cao hơn.

• Trong toàn bộ quá trình, dữ liệu phân tích sẽ được lưu vào cơ sở dữ liệu ở các bước: sau khi nhập bởi plugin ngôn ngữ, sau khi sensor phân tích, sau khi decorator phân tích. Lúc này người dùng có thể kiểm tra kết quả phân tích trên giao diện web. Thay vì sử dụng trực tiếp các mã lỗi từ đã được định nghĩa sẵn từ các nguồn mở như CWE, OWASP,... SonarQube đã tạo ra những mã lỗi của riêng mình bằng cách tự tổng hợp lại từ nhiều nguồn khác nhau. Sau đó, các bộ lọc, bộ kiểm tra (luật) sẽ được viết ra dựa trên các mã lỗi riêng như vậy. Các luật của Java được triển khai tại org.sonar.java.checks. Để kiểm tra các lỗi có trong một tệp dữ liệu, SonarQube sẽ có hai phương pháp để thực hiện điều này:

• Sử dụng cú pháp cây và API cơ bản (Using syntax trees and API basics): Trước khi chạy bất kỳ quy tắc nào, SonarQube Java Analyzer phân tích tệp 12 Java đã cho và tạo ra cấu trúc dữ liệu tương đương: cú pháp cây (Syntax Tree). Mỗi cấu trúc của ngôn ngữ Java có thể được biểu diễn bằng một thành phần cụ thể cú pháp cây (Syntax Tree), chi tiết từng đặc điểm của nó. Mỗi cấu trúc này được liên kết với một giao diện riêng mô tả rõ ràng tất cả các đặc tính của nó. Khi tạo ra một luật, lớp IssuableSubscriptionVisitor sẽ được thực thi, nó cung cấp các phương thức hữu ích để làm rõ các vấn đề cũng như xác định chiến lược sẽ sử dụng khi phân tích một tệp. Nó dựa trên cơ chế đăng ký (subscription mechanism), cho phép chỉ định loại cây mà luật này sẽ phản ứng với. Các nút mà cần đảm bảo sẽ được duyệt được chỉ định bởi một phương thức là nodesToVisit(). Từ đó SonarQube sẽ có được các thông tin để bắt đầu xác định các lỗi có trong mã nguồn hay thực thi các luật.

• Sử dụng API (Using semantic API): Ngoài việc thực thi các luật dựa vào dữ liệu được cung cấp bởi cú pháp cây (syntax tree), thì SonarQube cung cấp càng nhiều thông tin hơn liên quan đến mã nguồn thông qua một mô hình ngữ nghĩa của mã. Tuy nhiên mô hình này hiện chỉ hoạt động với mã nguồn Java. Mô hình ngữ nghĩa này cung cấp thông tin liên quan đến từng ký hiệu được thao tác. Ví dụ, đối với một phương thức, API ngữ nghĩa sẽ cung cấp dữ liệu hữu ích như chủ sở hữu của phương thức, công dụng của nó, các loại tham số và loại trả về của nó, ngoại lệ mà nó có thể ném,...

#### 2.2.8.2. Cài đặt SonarQube

*Cấu hình yêu cầu:*

1. Tối thiểu 2GB RAM
2. Dung lượng ổ cứng phụ thuộc vào khối lượng code mà bạn sử dụng SonarQube phân tích
3. Ổ cứng có I/O tốt

# CHƯƠNG 3: XÂY DỰNG DEMO

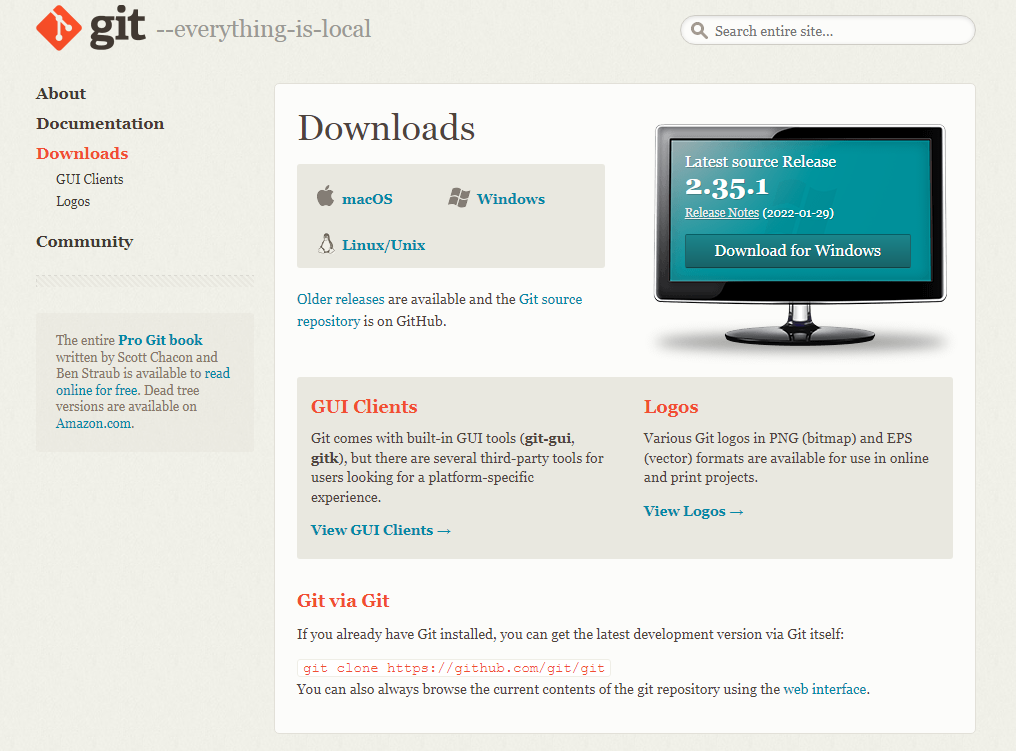
## 3.1. Quản lý mã nguồn

### 3.1.1 thao tác với git

Để thao tác với Github trên máy tính chúng ta cần tải 1 công cụ là Git.

Hướng dẫn tải và cài đặt git:

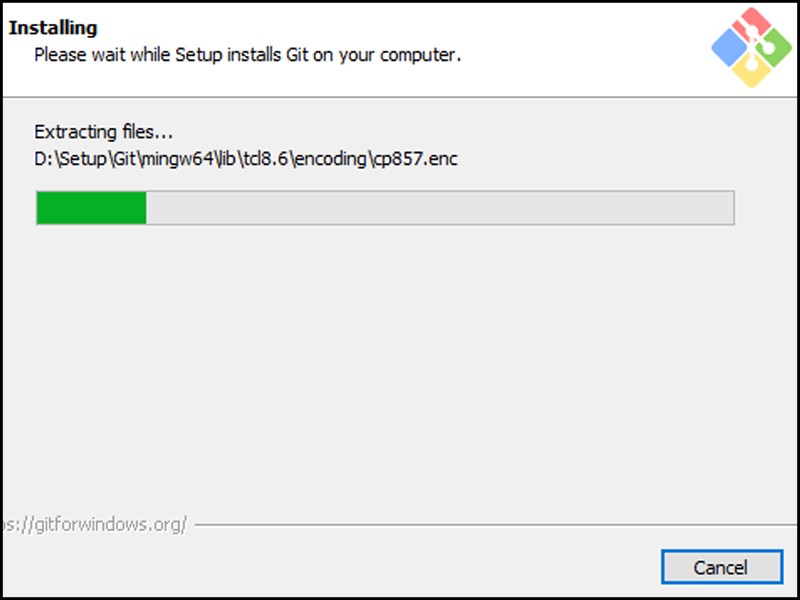
Bước 1: Truy cập địa chỉ [**https://git-scm.com/downloads**](https://git-scm.com/downloads) và chọn tải xuống:



Bước 2: Cài đặt trên window:

Sau khi tải về phiên bản của git, ta bắt đầu cài đặt.



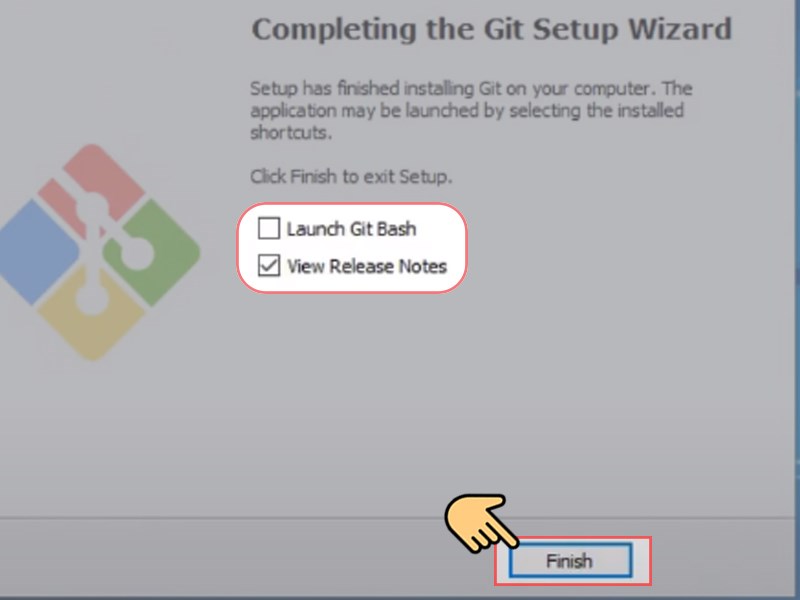


**Bước 3**: hoàn tất cài đặt

Cuối cùng bạn sẽ có 2 mục để bạn lựa chọn:

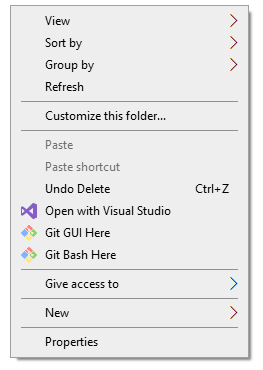
* Mục Lauch Git Bash: để mở cửa sổ dòng lệnh của Git lên ngay sau đó.
* Mục View Release Notes: để xem ghi chú về các thay đổi của phiên bản này so với các phiên bản trước đó.

Các bạn có thể lựa chọn tùy theo nhu cầu của bản thân. Và chọn Finish để hoàn tất cài đặt.



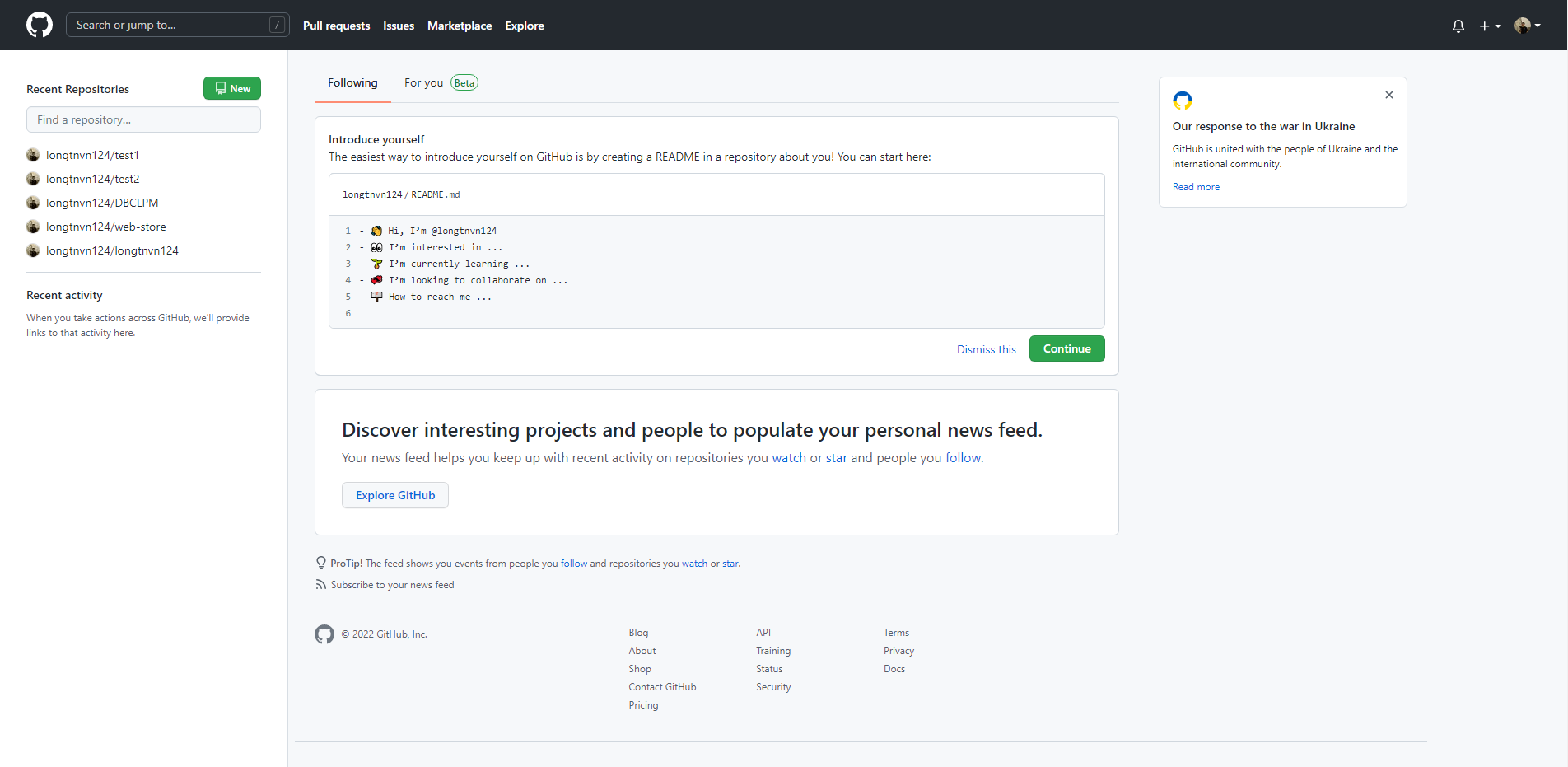
Bước 4: khởi chạy chương trình

Sau khi cài đặt xong, nhấn phải chuột vào một thư mục bất kỳ, một Context-Menu sẽ hiển thị, bạn có thể nhìn thấy các Menu-Item của Git, điều này chứng tỏ rằng bạn đã cài đặt Git thành công.



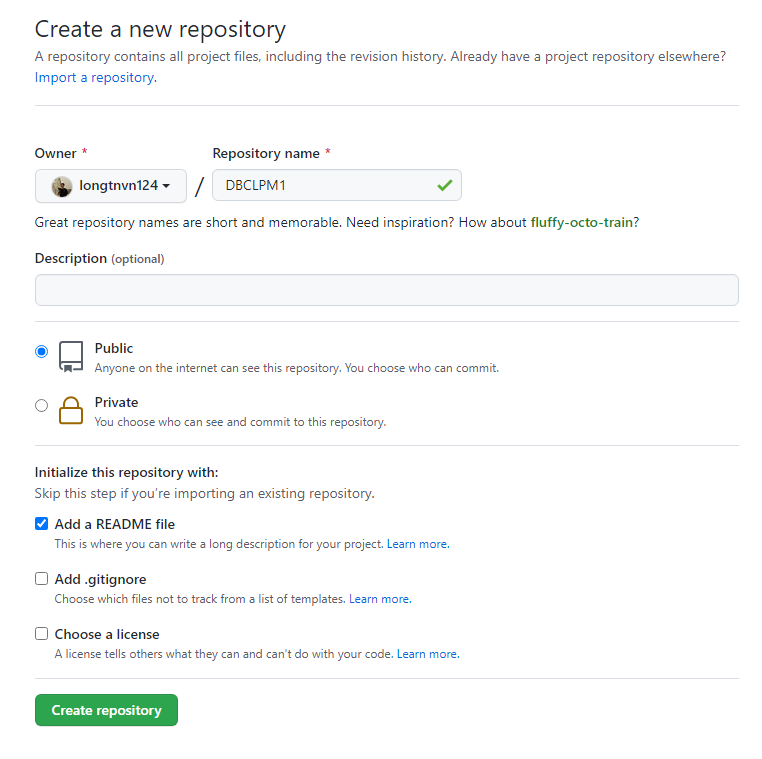
### 3.1.2. triển khai trên github

**Bước 1:** truy cập github.com



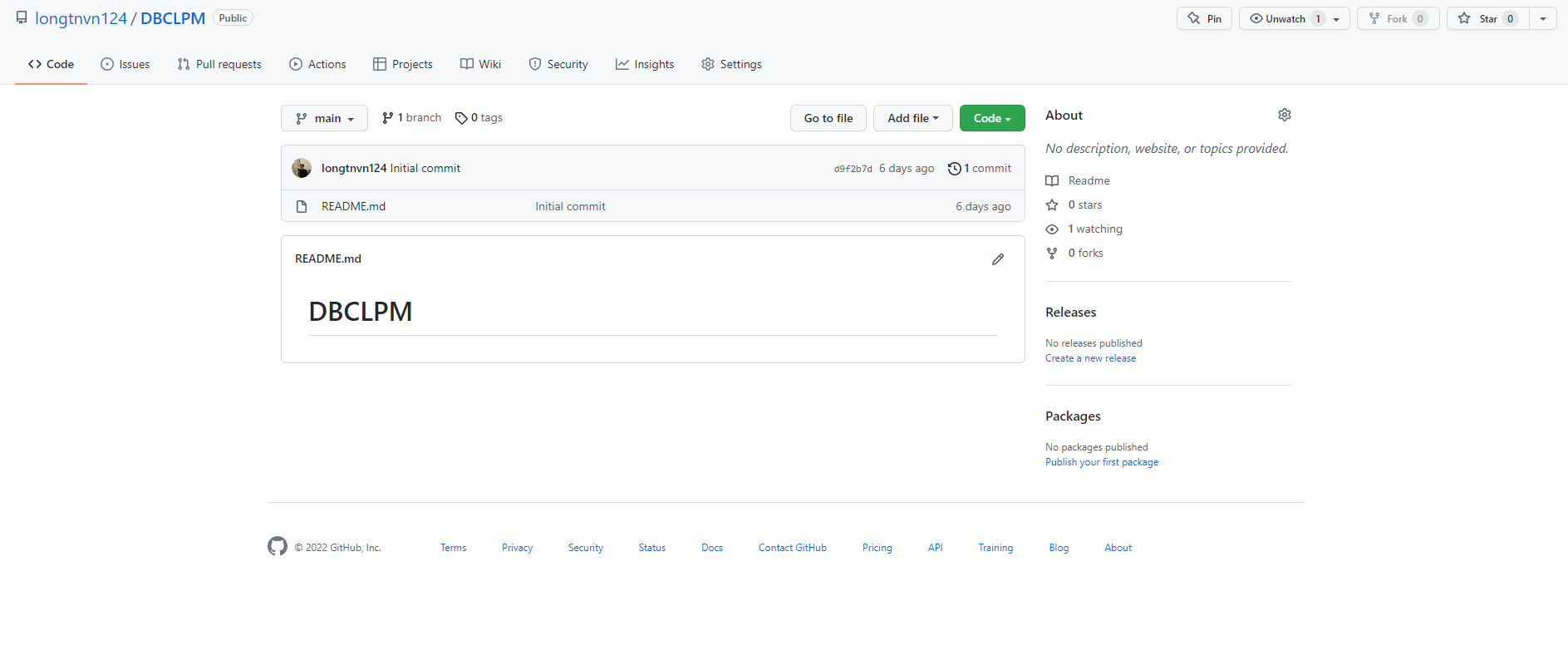
Giao diện chính của github

**Bước 2:** tạo repository mới



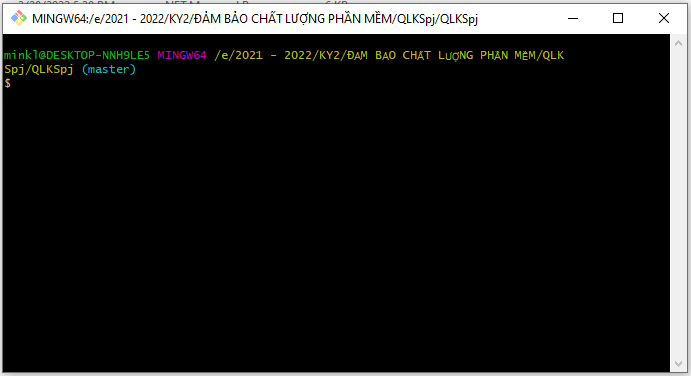
Sau khi chọn xong ta nhấn “Create repository”

Tiếp theo hiển thị repository mới tạo

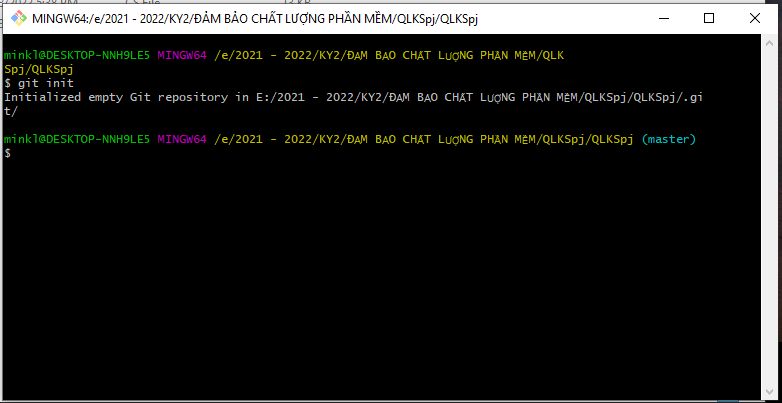


**Bước 3:** thực hiện viết lệnh trên git

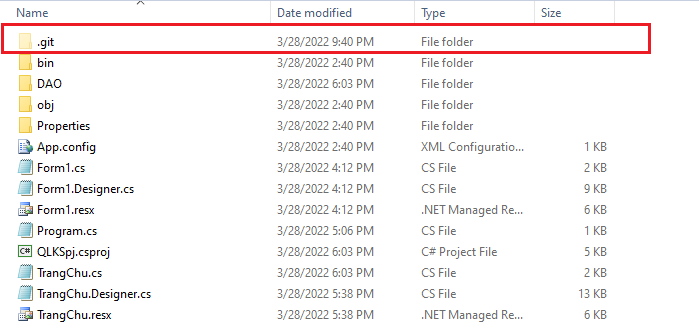
Đầu tiên, trong fodel chứa dự án chuột phải chọn “Git Bash Here”, Cửa sổ terminal hiện ra



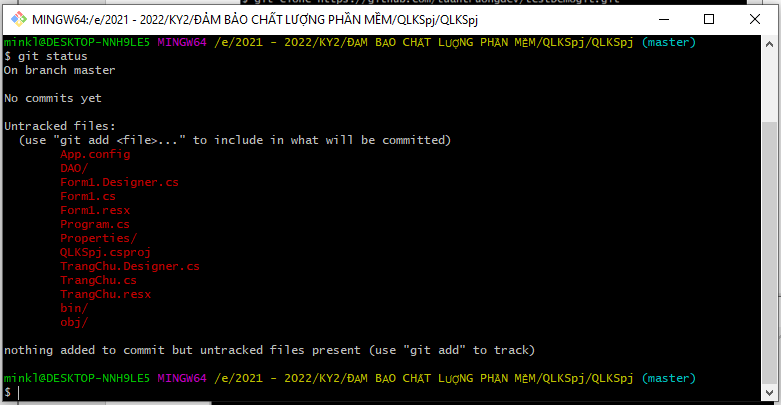
- thực hiện lệnh git init để tạo ra 1 file chứa Git Repo ở local:



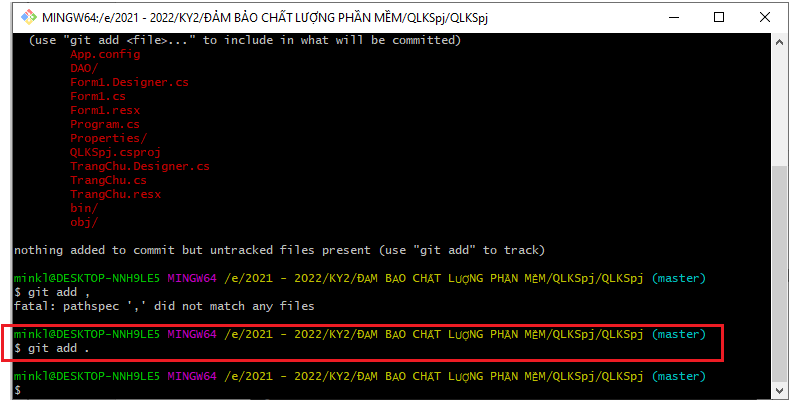
Lúc này trong project của ta có thêm 1 file mới “.git”



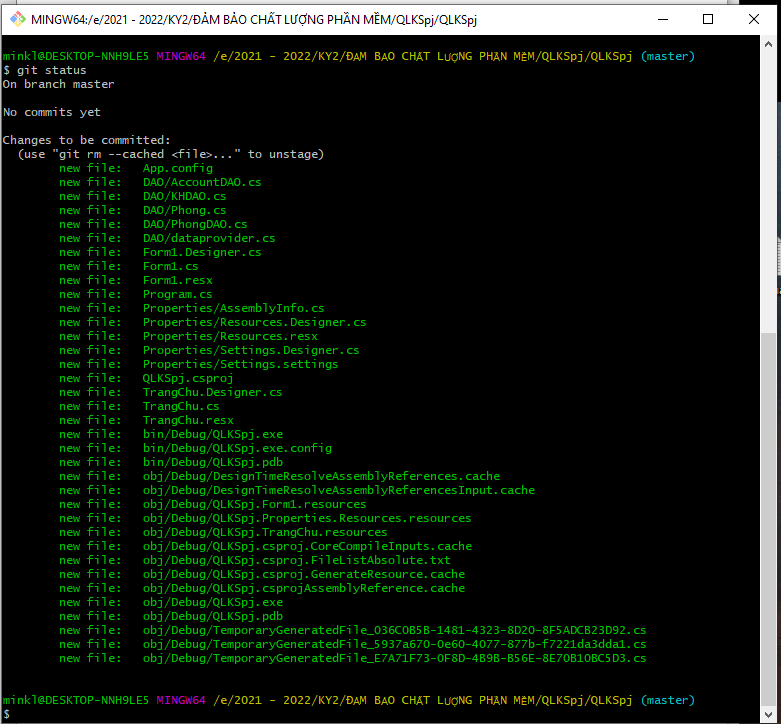
Thực hiện lệnh “git status” để kiểm tra tang thái file



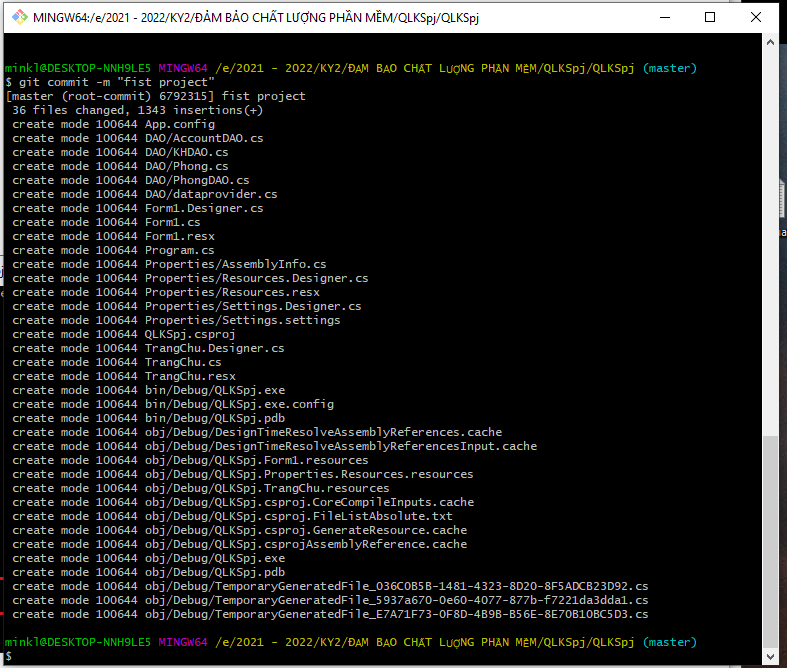
Thực hiện lệnh “git add .” để add toàn bộ file trong dự án



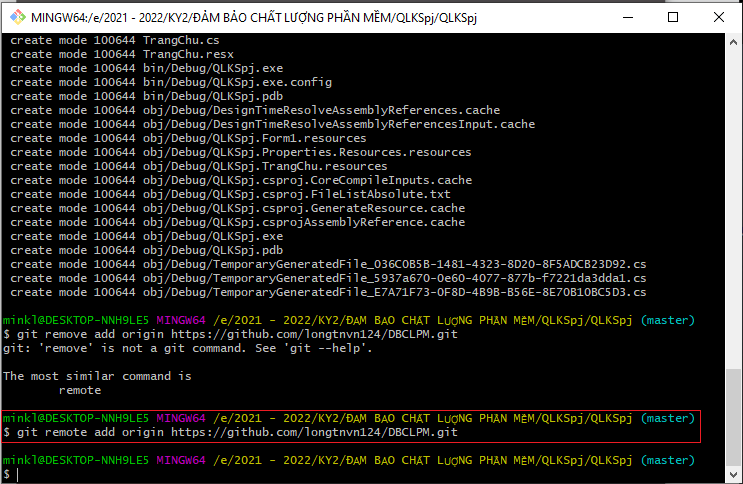
Thực hiện lệnh “git status” để kiếm tra trạng thái của file



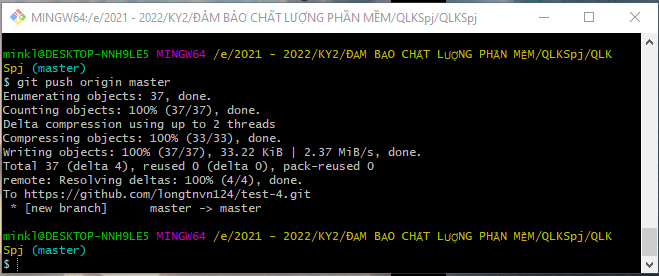
- thực hiện lệnh git commit -m “fist project”



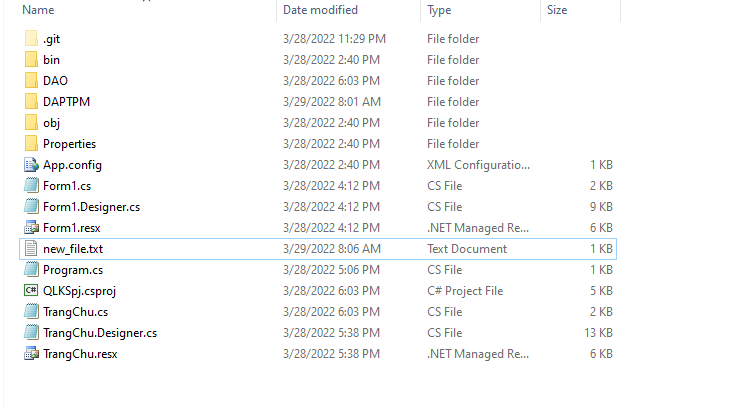
- Thực hiện lệnh: git remote origin https://github.com/longtnvn124/DBCLPM.git



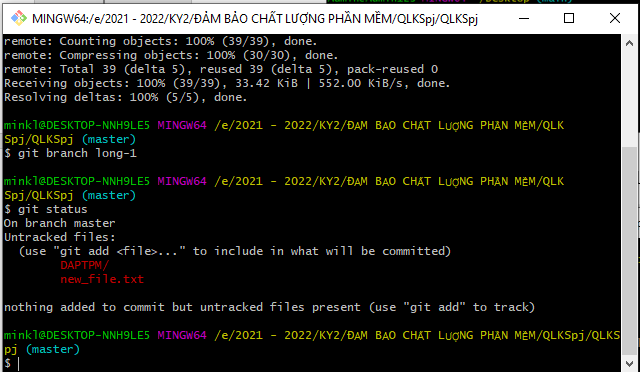
- Thực hiện lệnh: git push origin master

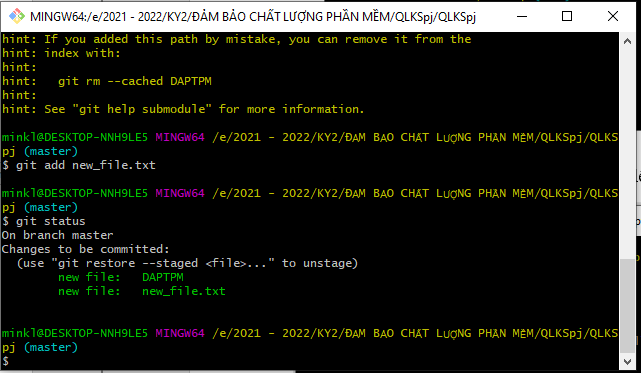


Tiếp theo ta tạo 1 file mới và thực hiện thêm 1 file vào git:

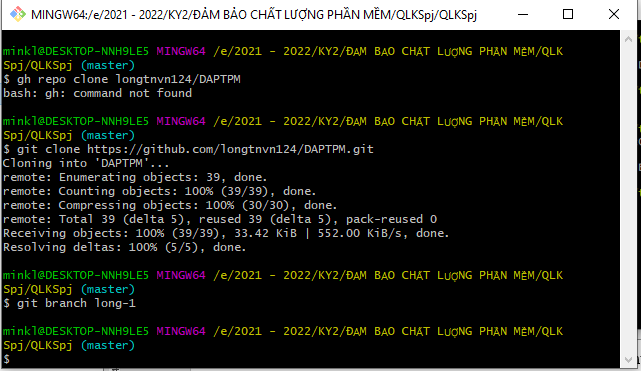


Tiếp theo ta add file và kiếm tra trạng thái

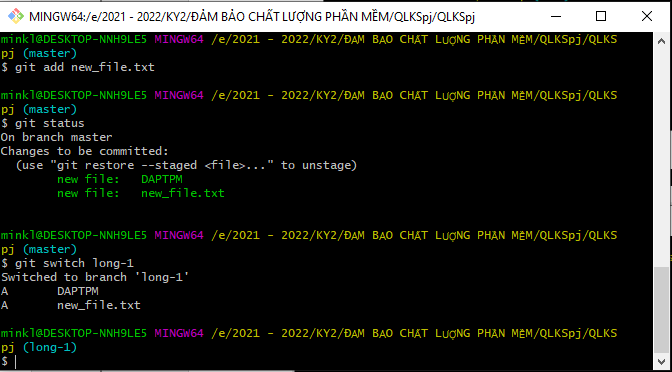




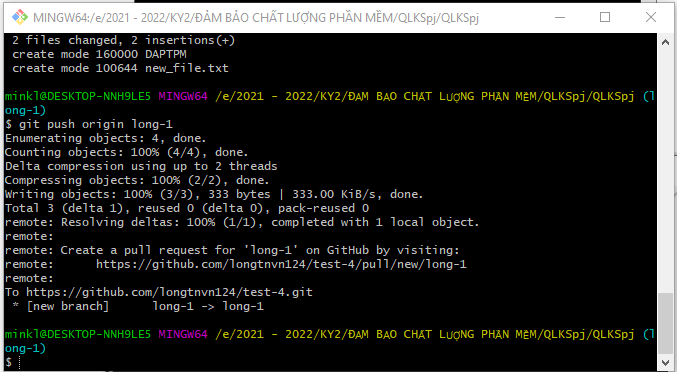
- Thực hiện lệnh : git branch long-1 để tạo 1 nhánh mới



- thực hiên lệnh: git switch long-1



- thực hiện lệnh : git push origin long-1

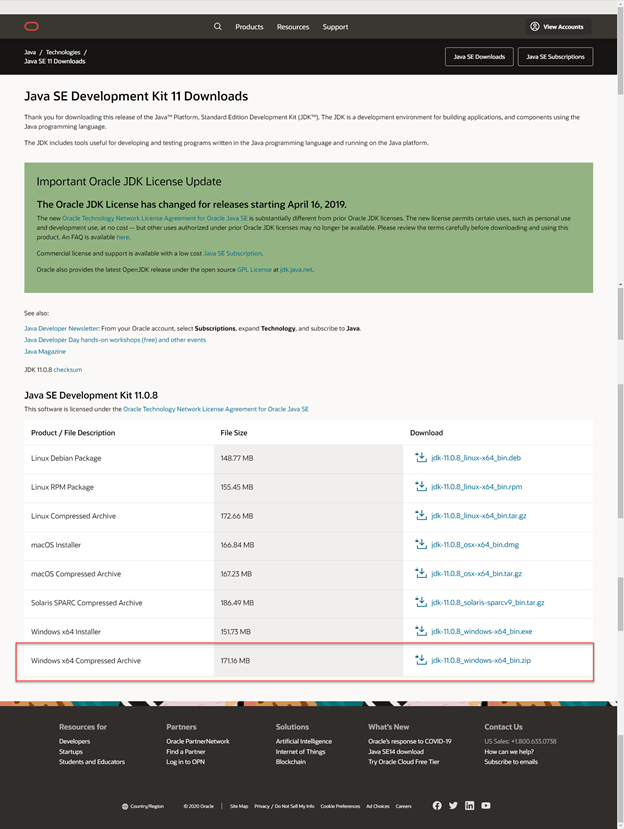


## 3.2. Đảm bảo chất lượng mã nguồn

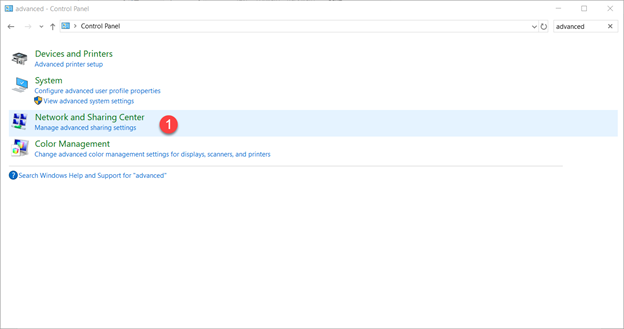
***Cài đặt SonarQube khi sử dụng Docker:***

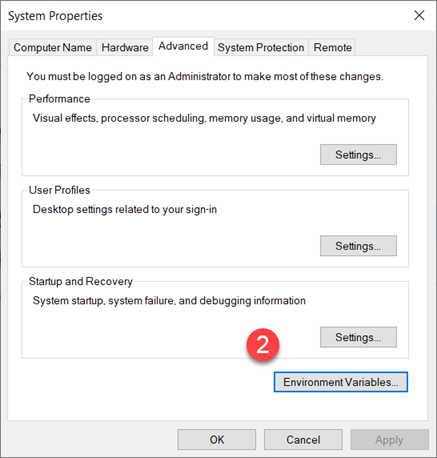
#### Bước 1: Cài đặt JDK 11

Tải [JDK 11](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html) và cài đặt vào ổ C:\Java

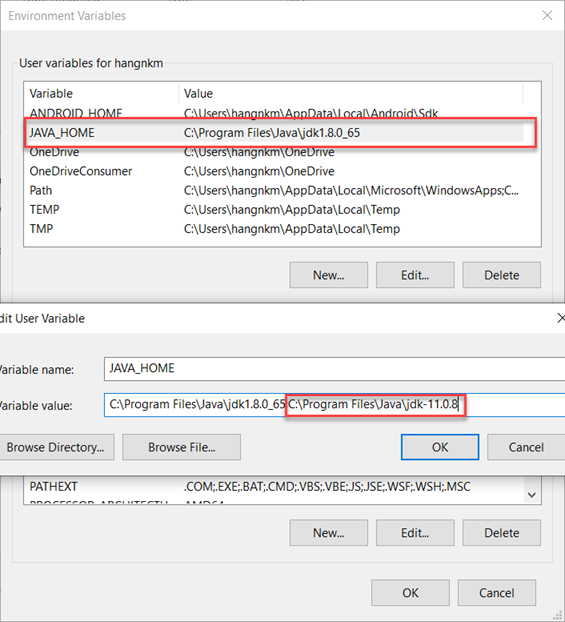
[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/tkg3d5j8r_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/tkg3d5j8r_image.png)

* Mở **Control Panel**, chọn **Advanced System Settings**

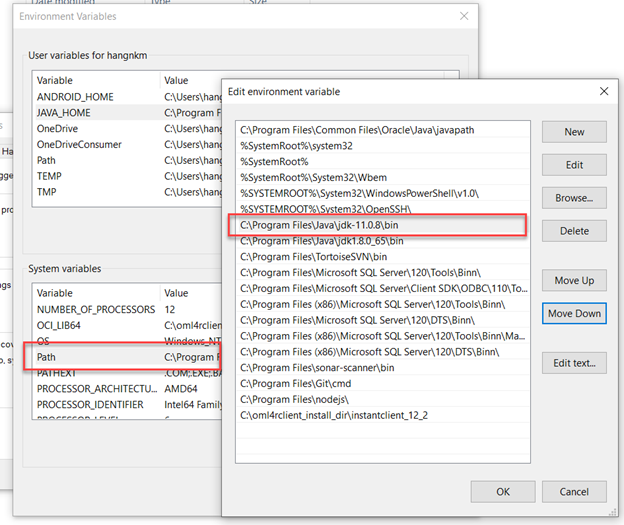
[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/gl92l4a5pt_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/gl92l4a5pt_image.png)

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/g9tn20skf8_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/g9tn20skf8_image.png)

* Set **JAVA\_HOME**: đường dẫn cài đặt JDK

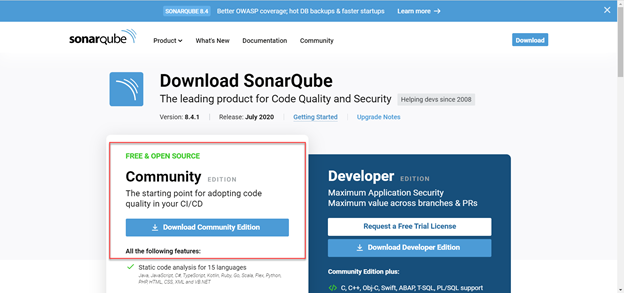
[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/daywappqf2_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/daywappqf2_image.png)

* Set Path: đường dẫn chứa thư mục bin của JDK

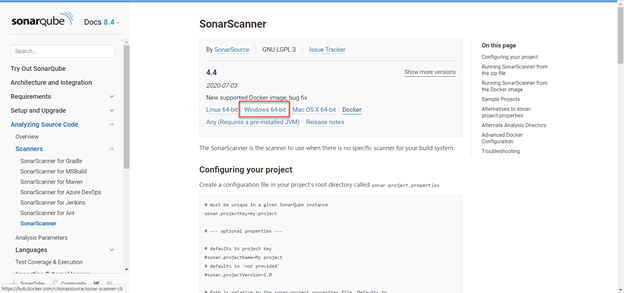
[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/xk3gs4b4ze_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/xk3gs4b4ze_image.png)

#### Bước 2: Tải các package

* Tải [SonarQube](https://www.sonarqube.org/downloads/) (bản Community)

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/8ei6xtaydh_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/8ei6xtaydh_image.png)

* Tải [SonarQube-Scanner](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/) (bản Windows)

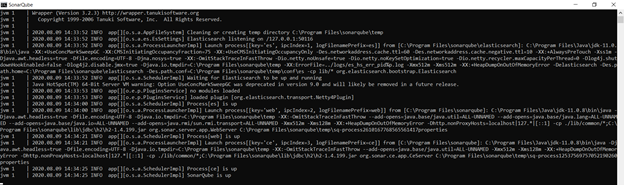
[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/ogcgu0m7bq_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/ogcgu0m7bq_image.png)

Các bạn giải nén tất cả và đổi tên thư mục thành sonarqube và sonar-scanner.  
Copy tất cả các thư mục trên vào Program Files của ổ C.

#### Bước 3: Start server của SonarQube

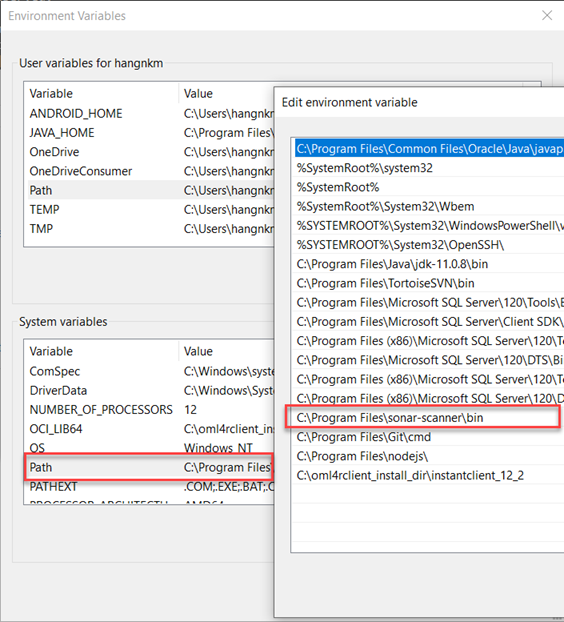
Mở file sonar.properties tại **C:\Program Files\sonarqube\conf**. Bỏ comment lệnh **#sonar.search.port=9001** và đổi lại thành **sonar.search.port=0**

Vào trong thư mục **C:\Program Files\sonarqube\bin\windows-x86-64**, tìm và chạy file **StartSonar.bat** để bật server sonarqube lên. Sau khi chạy thành công thì vẫn để cửa sổ chứ không được tắt.  
Thấy dòng chữ **SonarQube is up** là thành công.

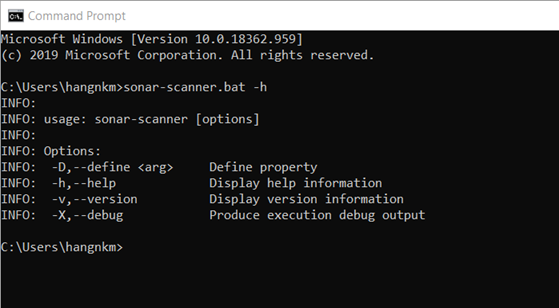
[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/w6474n6j6p_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/w6474n6j6p_image.png)

#### Bước 4: add sonarqube-scanner vào Path

Vào **Environment Variables** tương tự như bước 1. Tìm đến phần **Path**. Thêm đường dẫn chứa thư mục bin của sonar-scanner.

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/va7p2s4arg_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/va7p2s4arg_image.png)

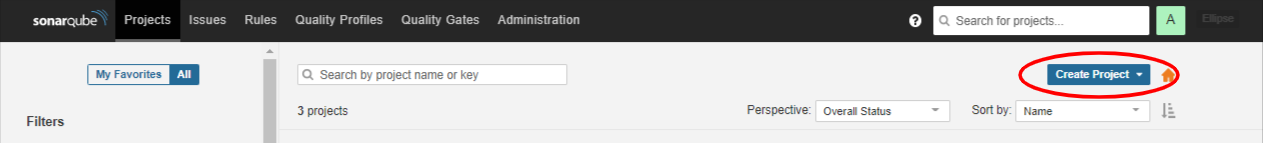
Sau khi hoàn tất mở cmd và gõ command **sonar-scanner.bat -h**. Nếu ra như sau thì là ok:

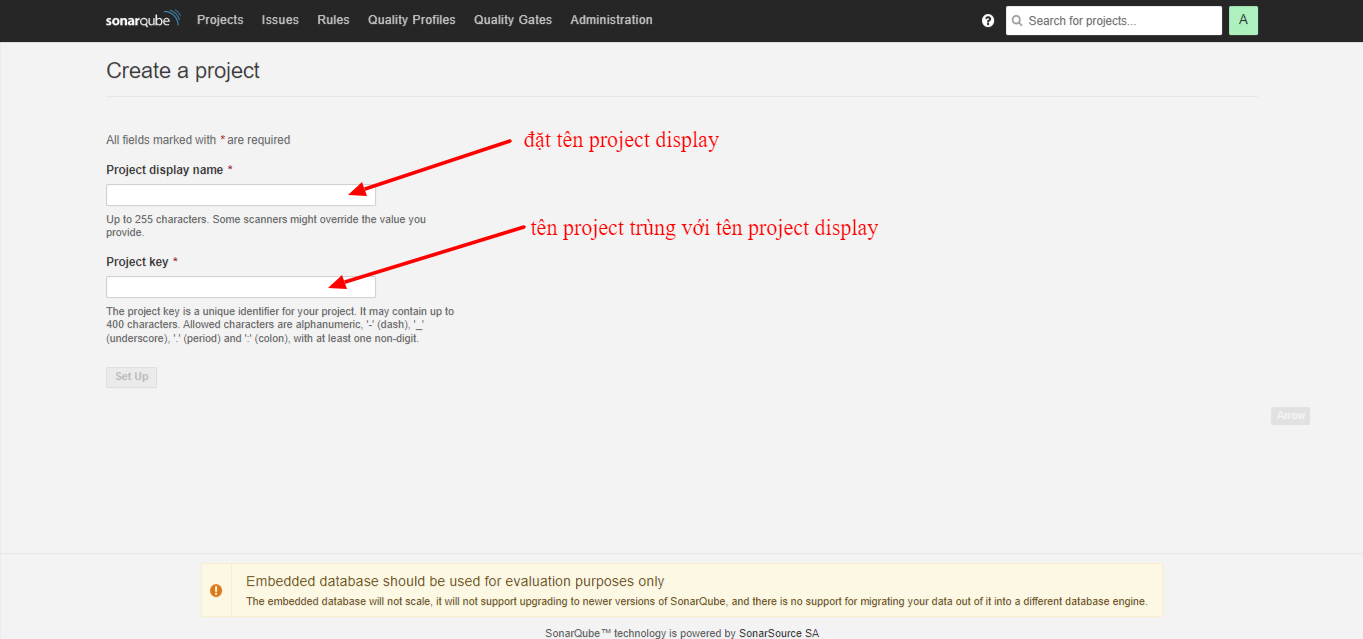
[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/kjq9lbzb0p_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/kjq9lbzb0p_image.png)

#### Bước 5: Tạo project và chọn ngôn ngữ cần test

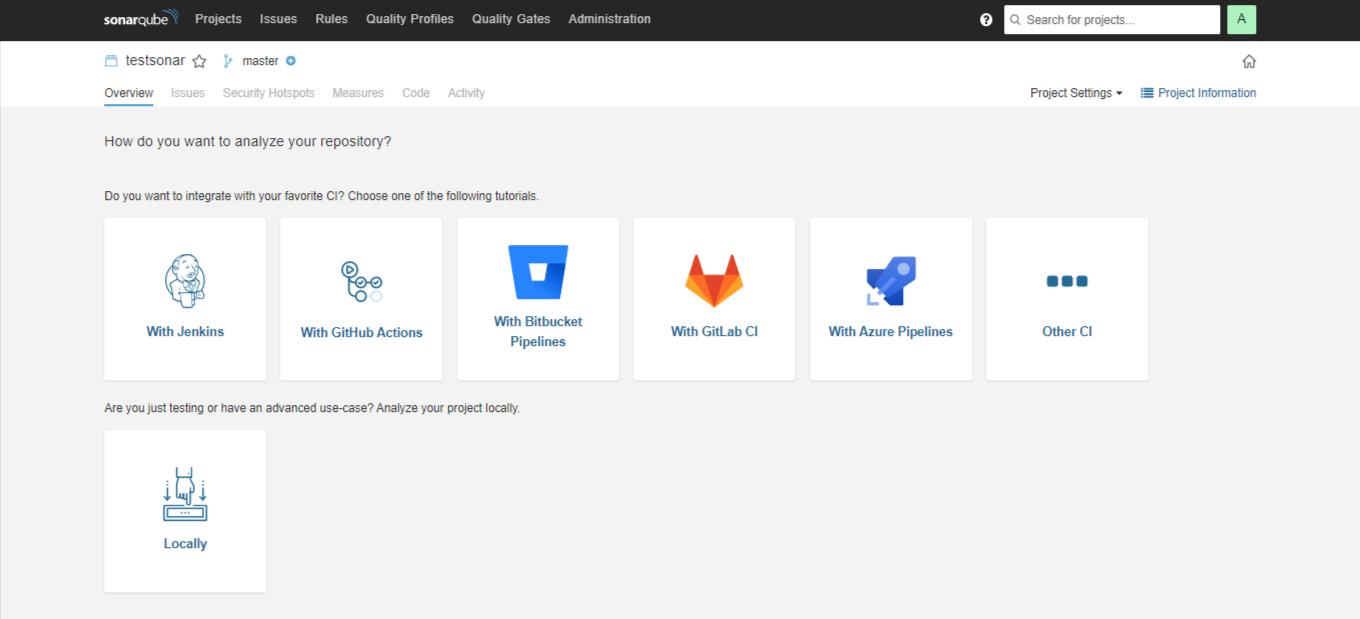
Gõ địa chỉ <http://localhost:9000/>. Đăng nhập ID/Password với **admin/admin**. Làm theo các bước sau:

Ấn vào Creat Project => Manually

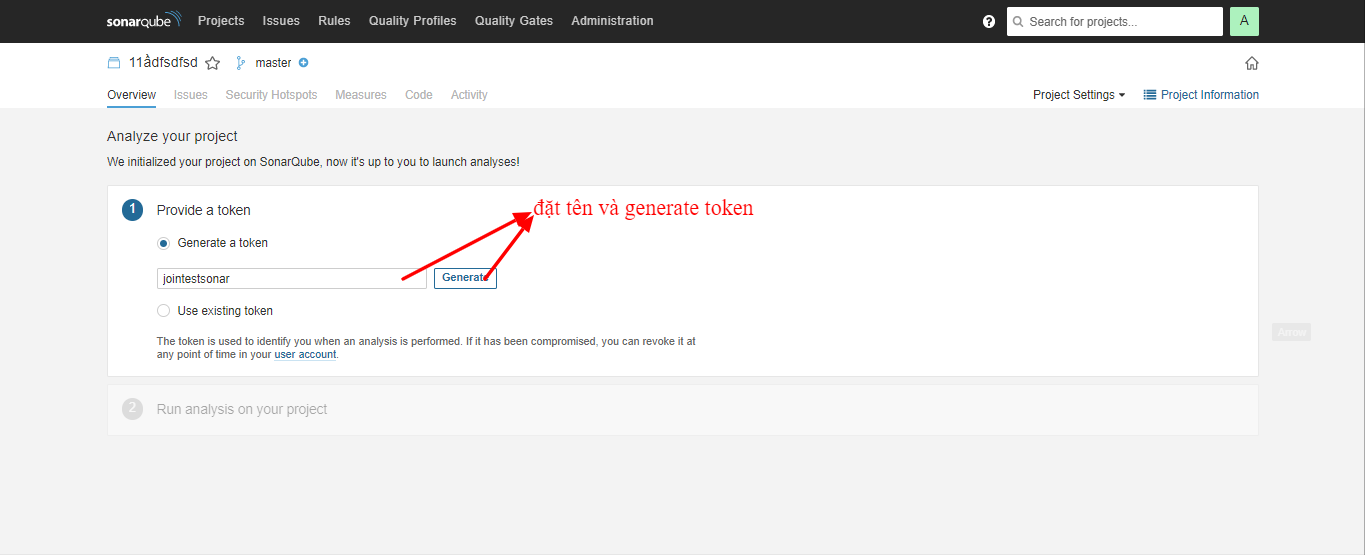
[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/nyec4us0k3_image.png)

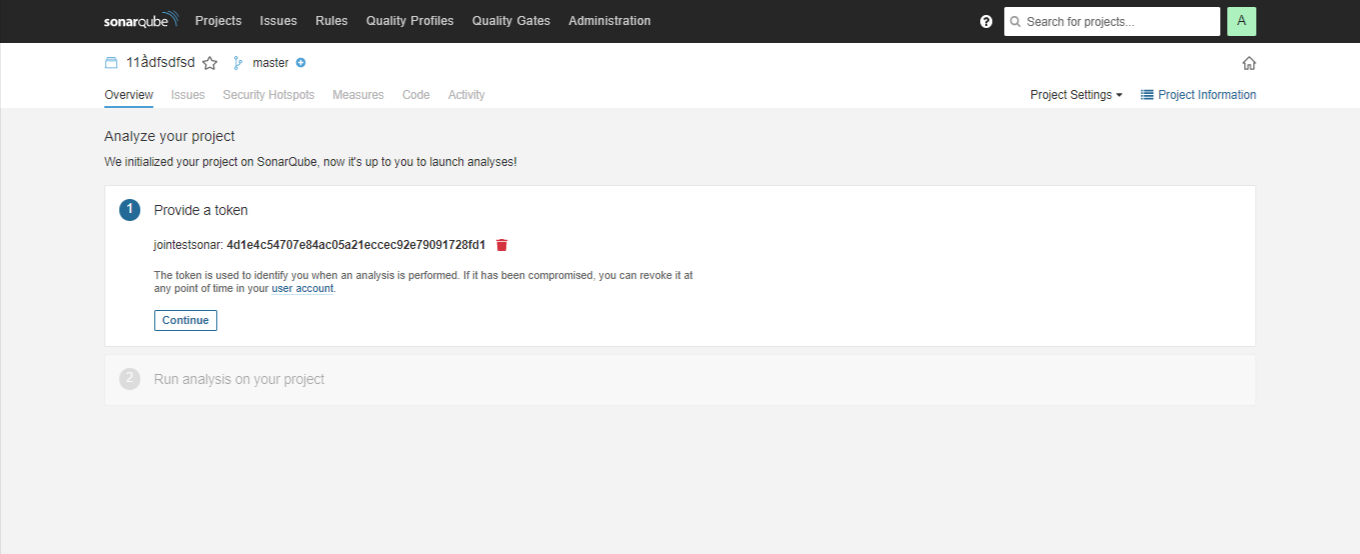
Đặt tên cho Project

#### Chọn nơi lưu trữ để phân tích dữ liệu

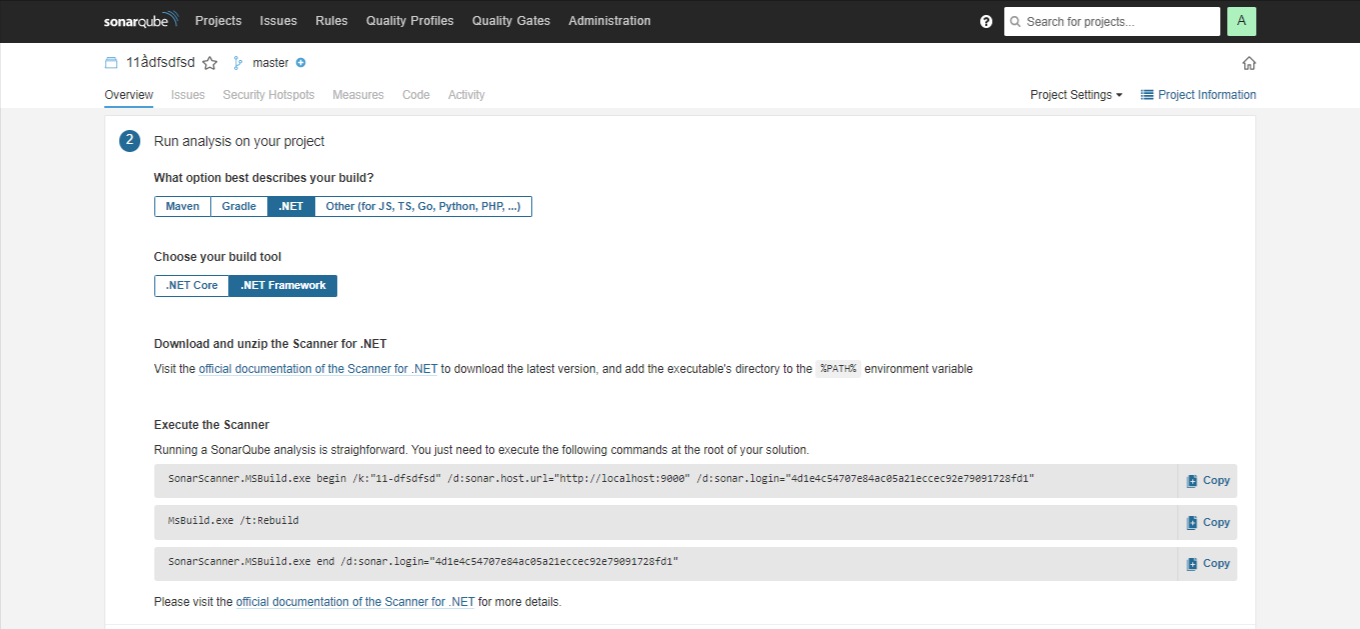


Generate token

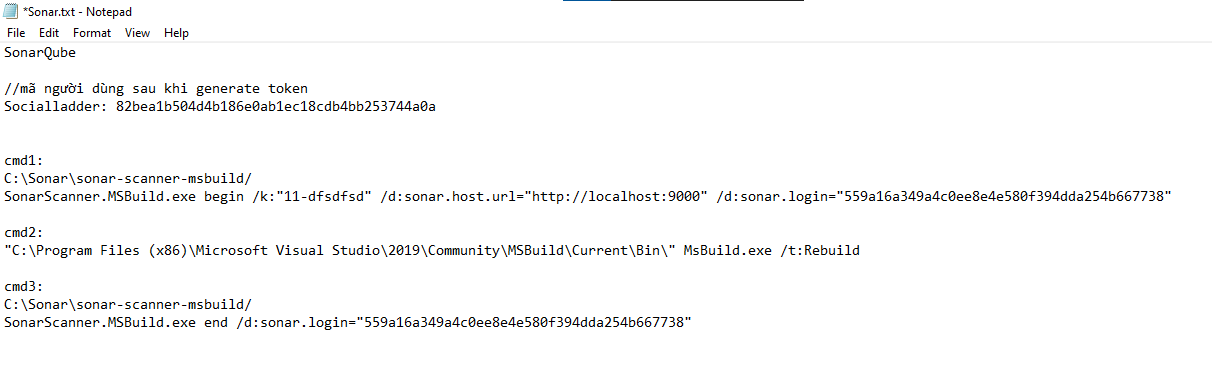




Chọn ngôn ngữ cần test

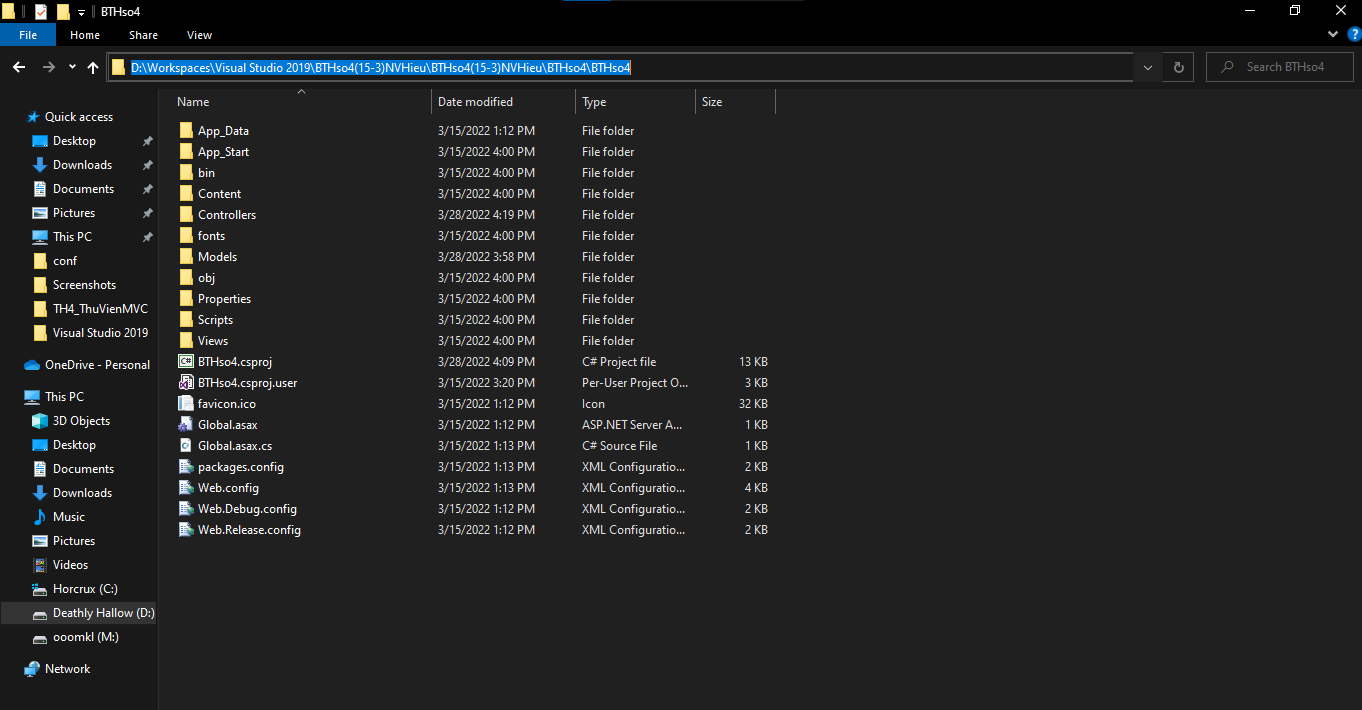


#### Bước 6: Config sonar trong project

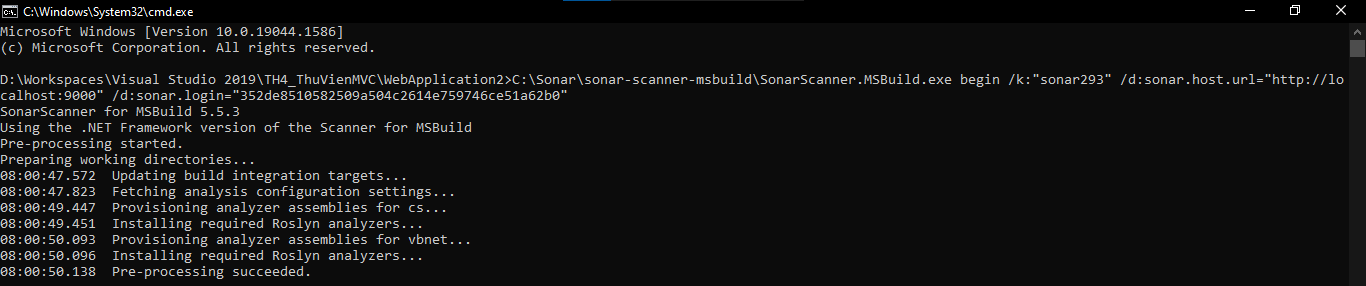
Tạo 1 file .txt có nội dung sau đây:  


#### Bước 7: Scan source

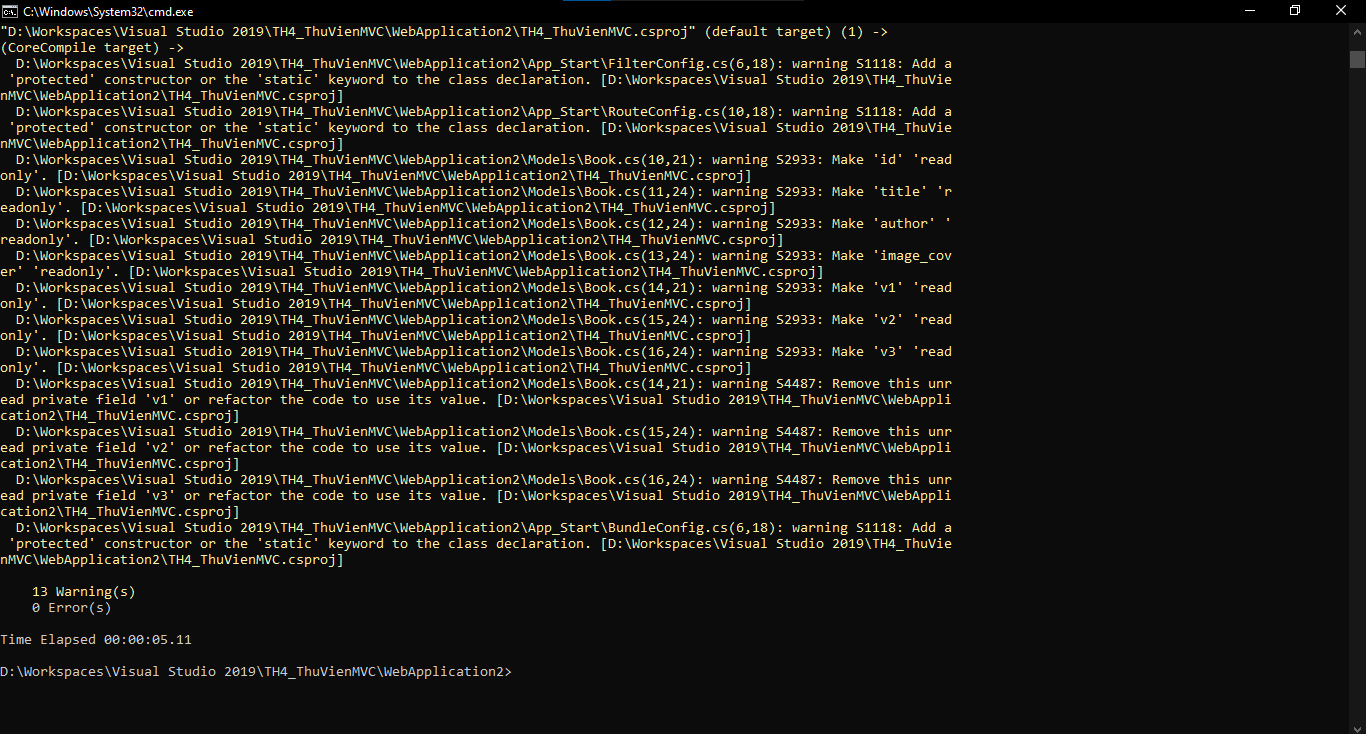
Tiếp theo mở Cmd ở đường dẫn project và thực hiện lần lượt các Command Prompt



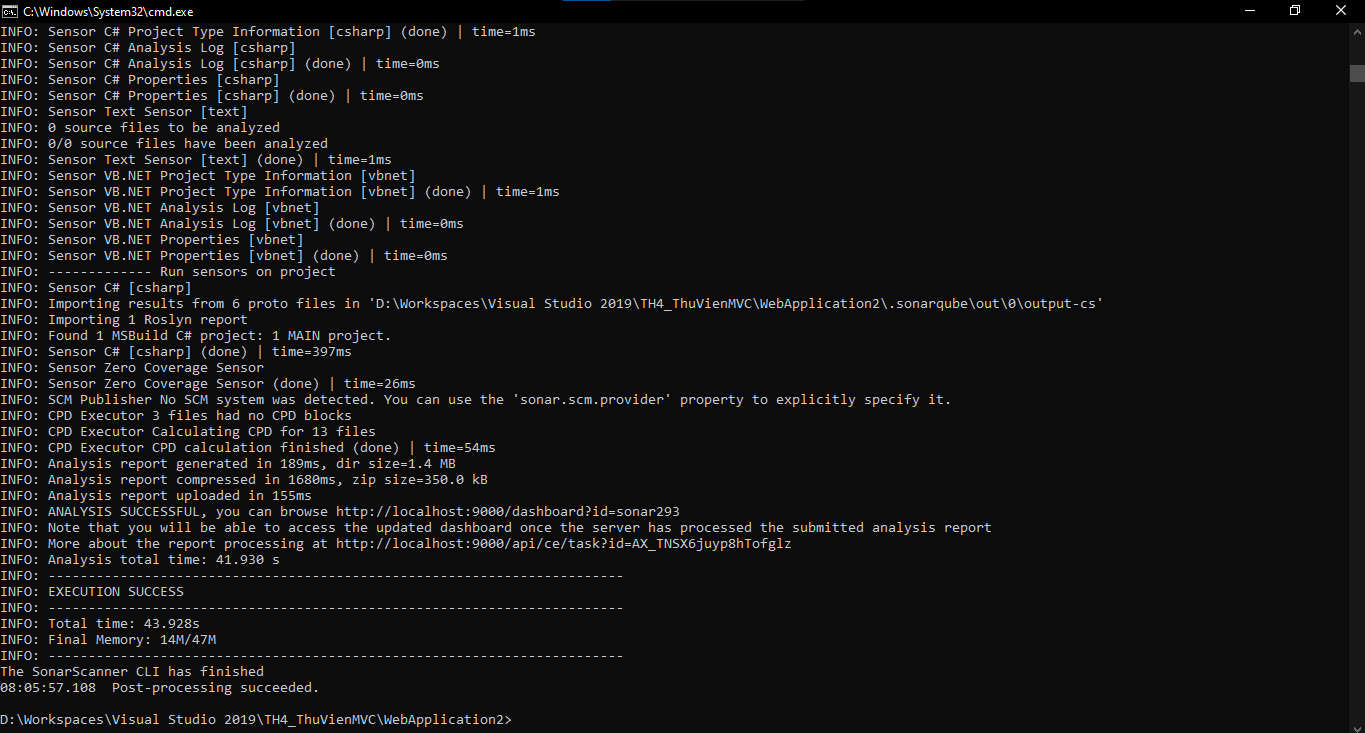
[Cmd1: bắt đầu chương trình SonarScanner.MSBuild.exe để build file .sonarqube](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/pm0a100gv5_image.png)



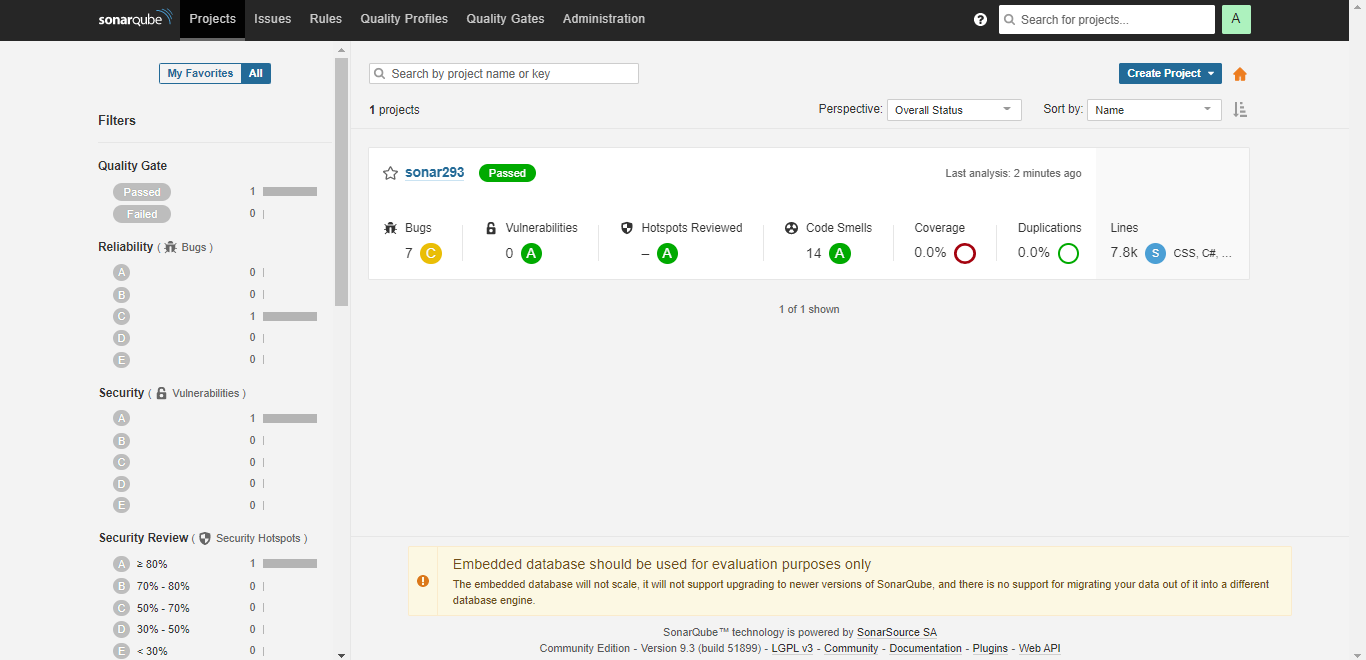
Cmd2: Rebuid file

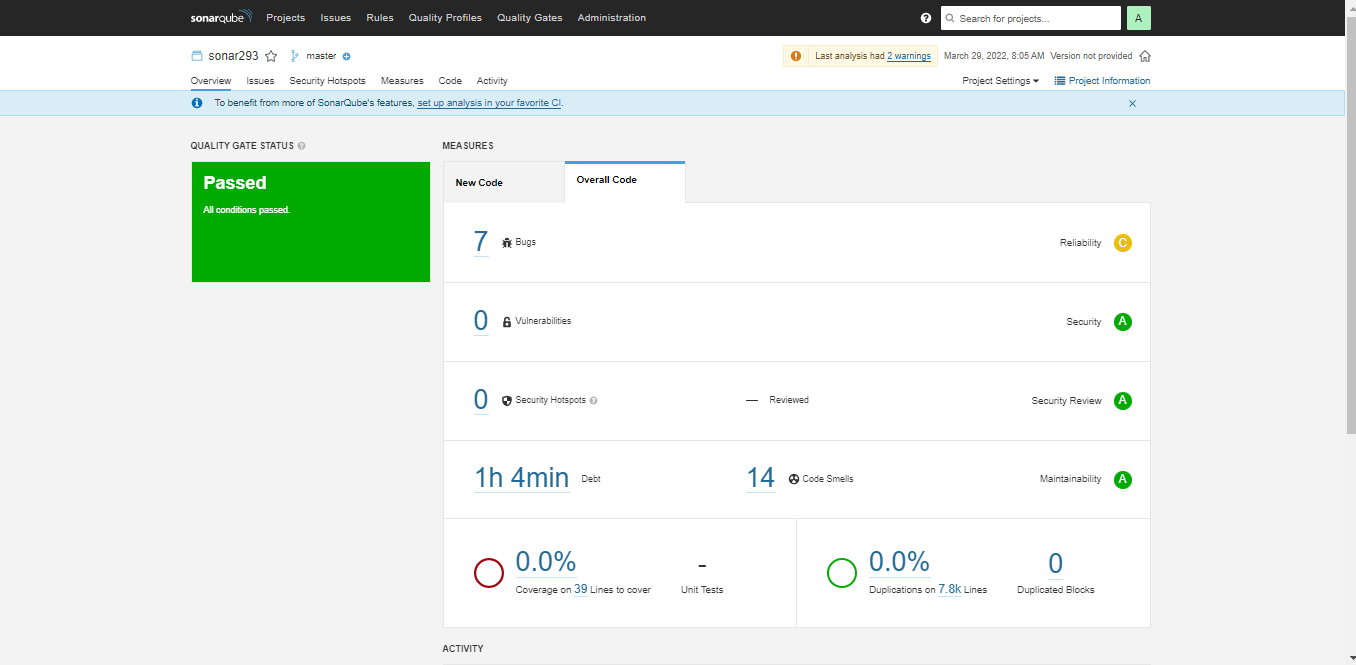


Cmd3: kết thúc chươn trình

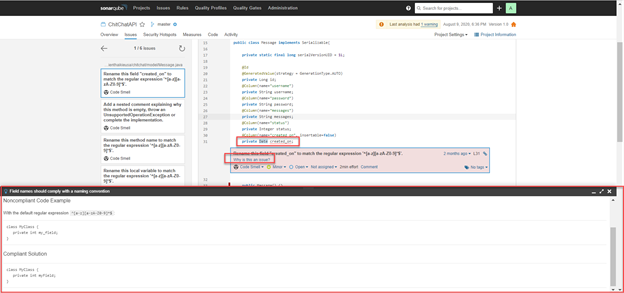


Refresh lại trang <http://localhost:9000/>, nhấn vào Projects, ta sẽ thấy báo cáo kết quả.

[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/6ict4gawpw_image.png)

[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/9zl0ewksx3_image.png)

Ở trang chi tiết của Project, ta có thể nhìn thấy rõ các Issues được phân loại theo **Bugs, Vulnerabilities, Security hay Code Smells**. Nhấn vào số lượng bugs ở mỗi loại, Sonar sẽ đưa ra cụ thể từng Issue nằm ở Class nào cũng như đề xuất giải pháp để sửa.

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/qygdljmclq_image.png)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/qygdljmclq_image.png)

# TÀI LIỆU THAM KHẢO

[1]: giáo trình đảm bảo chất lượng phần mềm

[2]:

[3]:

[4]:

[5]:

[6]: