4. Code reviews

VCS - PRODUCT DEVELOPMENT STANDARDS

Exported on 09/21/2020

Table of Contents

Tham khảo:

- https://google.github.io/eng-practices/review/reviewer/
- https://smartbear.com/learn/code-review/guide-to-code-review-process/
- https://phauer.com/2018/code-review-guidelines
- https://yalantis.com/blog/code-review-via-gitlab-merge-requests-code-review-must/
- https://viblo.asia/p/tao-pull-request-dung-cach-bWrZnwkrlxw

Code review là một phần thiết yếu trong quá trình phát triển phần mềm. Đây là một biện pháp làm tăng chất lượng sản phẩm và làm cho quá trình phát triển phần mềm hiệu quả hơn.

1 Tại sao cần code review?

Nếu không có code review, team dự án sẽ gặp phải một số vấn đề sau:

- Không có một thành viên nào trong team có cái nhìn toàn cảnh về code trong dự án do nó được viết bởi những người khác nhau.
- Code có chất lượng xấu có thể tồn tại trong thời gian dài, có khi là mãi mãi.
- Không ai chịu trách nhiệm về chất lượng code trong dự án (kể cả code do họ viết lẫn code của người khác viết)
- Các thành viên trong team khó có cơ hội để học hỏi kinh nghiệm và cải thiện kỹ năng code
- Bad code trở thành một điểm nghẽn trong testing phase: khó tìm ra bug, khó tìm root cause cũng như khó fix bug
- Khi có một thành viên mới tham gia vào dự án, code của họ có thể là một rủi ro lớn với dự án: khó tiếp nhận code cũ hoặc code mới không đảm bảo chất lượng.

Lợi ích của code review

- Ngoài việc viết test, code review cũng là một cách để giảm thiểu số lượng lỗi phát sinh. Đôi khi vì quá quen với những dòng code của chính mình, chúng ta khó phát hiện lỗi hơn là người khác.
- Thông qua code review, các thành viên trong nhóm sẽ nắm bắt được những gì đang diễn ra trong dự án, biết rõ ai đang phụ trách phần nào và người đó thực hiện việc như thế nào.
- Được học hỏi thêm nhiều kiến thức hữu ích. Việc được học hỏi đúng nơi (code của mình), đúng thời điểm (đang viết code) giúp chúng ta nhớ lâu hơn.
- Học cách giảng giải và góp ý cho người khác. Đây cũng là một cơ hội để chúng ta ôn lại & kiểm nghiệm lại kiến thức của mình.
- Thông qua việc học hỏi và trao đổi lẫn nhau, chúng ta có thể viết ra được những dòng code chất lượng. Hơn thế nữa, chúng ta có thể áp dụng những cái học được qua các dự án tiếp theo.

Khó khăn khi áp dụng code review

- Chúng ta sẽ tốn thời gian nhiều hơn bình thường vì phải thực hiện: xem xét code, sữa chữa và cập nhật code.
- Tuy nhiên đây là một sự đánh đổi chấp nhận được, để có những dòng code chất lượng hơn. Vì ngược lại, những dòng code không tốt có thể phát sinh ra lỗi và bạn sẽ tốn nhiều thời gian hơn để sữa những lỗi đó.

2 Nguyên lý chung của code review

- Code review phải là một phần trong quá trình phát triển phần mềm.
- Code review trên một phần code nhỏ nhưng phải có logic hoàn chỉnh như là một tính năng, một task, một bug fix hoặc một cải tiến.
- Sau khi code review xong thì mới đến phase test.
- Mọi developer trong team đều phải tham gia code review bất kể level của họ là gì (ngay cả junior cũng có thể review code của senior)
- Mọi code của tất cả developer đều được review bất kể level của họ.
- Reviewer không nên review liên tục quá 90 phút và quá 1000 dòng code. Chọn thời điểm review hợp lý. Hãy coi quá trình review là quá trình thảo luận và học hỏi lẫn nhau.

3 Phải review những cái gì

Khi review cần phải làm một checklist các thứ cần xem xét. Nhìn chung có các vấn đề sau:

- Mục đích:
  - Khi nhìn vào code, bước đầu tiên chính là hiểu được tại sao đoạn code này được viết lên, và tại sao phải tạo pull request này để thêm vào code gốc. Điều này quan trọng đối với reviewer ngang với tác giả. Cả hai phía đều cần phải hiểu được đoạn code này nhằm giải quyết vấn đề gì. Nếu có thể nắm được mục đích, reviewer có thể tiếp tục hướng đến một lần review code hoàn hảo. Nếu không, reviewer và tác giả cần một cuộc trao đổi về vấn đề này.
- Thiết kế:
  - Đây là bước thực sự bắt đầu nhìn vào code thực tế. Reviewer sẽ bắt đầu nhìn vào tổng thể các thay đổi, các điểm khác biệt so với code cũ, để biết được trọng tâm các thay đổi được cài đặt ở đâu.
  - Quan sát vị trí của các thay đổi trong toàn bộ cấu trúc của phần mềm từ đó đánh giá code có được thiết kế tốt và phù hợp với tổng thể dự án không?
- Chức năng:
  - Ý định ban đầu của tác giả là gì? Họ làm có đúng và tốt cho người dùng không (xem xét cả đối với người dùng sử dụng là khách hàng lẫn lập trình viên khác)
  - Bất kỳ thay đổi nào về mặt UI có hợp lý và tốt không?
  - Việc thay đổi code có dẫn đến rủi ro gì không? Đặc biệt cần chú ý tới các vấn đề trong lập trình đa luồng (performance, race condition, ...)
- Tính phức tạp:
  - Có cách nào để làm code đơn giản hơn?
  - Liệu một lập trình viên khác có thể dễ dàng hiểu và sử dụng lại được đoạn code này trong tương lai?
- Test:
  - Reviewer cần nhìn đến các phần unit test, để biết được phần mềm sẽ phản ứng thế nào với các thay đổi này. Nếu không có các bộ test mới, cần note lại để review kỹ hơn, hoặc yêu cầu tác giả giải thích vì sao việc tạo bộ test lại không cần thiết, hoặc quá tốn công sức để cài đặt.
  - Chạy thử test để kiểm tra code có chính xác không?
- Security:
  - Phương pháp sử dụng để authentication và authorization có phù hợp không?
  - Phần thay đổi của code có mở ra lỗ hổng bảo mật nào không?
  - Có dữ liệu nhạy cảm nào được lưu trực tiếp trong code thay vì lấy biến môi trường không?
- Library:
  - Có nên sử dụng thư viện này không? Liệu có thư viện nào tốt hơn?
  - Giấy phép (license) nguồn mở của thư viện này có phù hợp/có ảnh hưởng tới vấn đề thương mại hóa của sản phẩm không? (Chú ý phải tránh sử dụng license họ GPL)
- Name:
  - Đặt tên biến, class, method có rõ ràng không?
- Comment:
  - Comment có rõ ràng, đầy đủ và hữu ích không?
- Style:
  - Có tuân thủ đúng style guides không?
- Documentation:
  - Có cần phải update thêm thông tin trong các documentation liên quan không?

4 Rule cho tác giả

Đối với tác giả, điều quan trọng nhất là phải có tư duy **cởi mở và khiêm tốn** về những phản hồi họ sẽ nhận được.

- Hãy khiêm tốn:
    - Hãy lưu ý code của bất kỳ ai cũng đều có thể được cải tiến tốt hơn
    - Bạn không hoàn hảo
    - Bạn có thể gặp sai lầm
    - Cho dù bạn giỏi đến đâu, bạn vẫn có thể học hỏi được những điều mới để cải thiện tốt hơn.
    - Đừng cho rằng bạn là chuyên gia trong phát triển phần mềm nên không bao giờ bạn gặp sai lầm
- Bạn không phải là code của bạn (You are not your code):
    - Giá trị bản thân của bạn và code của bạn không liên quan đến nhau. Đừng đánh đồng hai thứ đó với nhau, hãy thoải mái khi ai đó đánh giá code của bạn
    - Lập trình là một kỹ năng - luôn luôn cải thiện bằng quá trình học hỏi không ngừng nghỉ.
- Reviewer và bạn có chung một mục tiêu: làm cho chất lượng phần mềm tốt hơn.
- Hãy nhớ về hiệu ứng IKEA (IKEA effect): cái gì chính bạn tự làm thì bạn luôn cho rằng nó có giá trị rất cao. Code của bạn cũng vậy. Bạn cho rằng code của bạn rất tốt nhưng với người khác thì khác đó.
- Mọi phản hồi đều sẽ mang lại những giá trị mới cho bạn.
- Code review là một quá trình đem lại rất nhiều giá trị thực tiễn và kinh nghiệm
- Code review là một loại thảo luận không phải là tranh luận. Có thể phản hồi là không hợp lý nhưng hãy luôn cư xử lịch sự và sẵn sàng thỏa hiệp.

5 Rule cho reviewer

Đối với reviewer, điều quan trong nhất phải chú ý là **cách thức phản hồi**. Điều này sẽ quyết định phản hồi của bạn có được chấp nhận hay không.

- Sử dụng I-Messages (thông điệp từ cái nhìn của bạn như tôi nghĩ, tôi cảm thấy, tôi ấn tượng, ...)
  - Đúng: Tôi thấy quá khó với tôi để hiểu đoạn code này.
  - Sai: Bạn viết code như ****
- Nói về code, không phải nói về coder
  - Đúng: Đoạn code này request quá nhiều lần tới server do vậy không hiệu quả.
  - Sai: Bạn request quá nhiều lần tới server do vậy không hiệu quả.
- Đặt câu hỏi, đừng áp đặt người khác
  - Đúng: Bạn nghĩ sao nếu thay tên biến này thành userId?
  - Sai: Thay tên biến này thành userID
- Luôn đề cập tới hành vi của tác giả không được nói về tính cách
  - Đúng: Tôi thấy bạn cần cần chú ý hơn tới việc viết test cho đoạn này.
  - Sai: Bạn là người cẩu thả khi viết test cho đoạn này.
- Chấp nhận rằng luôn có những giải pháp khác nhau để giải quyết một vấn đề:
  - Đúng: Tôi sẽ làm theo cách khác, nhưng giải pháp của bạn cũng ổn
  - Sai: Tôi luôn làm thế này và bạn cũng phải thế
  - Phân biệt giữa best practices và sở thích cá nhân. Nhiều khi phản ánh của bạn chỉ là sở thích cá nhân của bạn chứ không phải là best practices cho vấn đề đó.
- Đừng quá soi mói, đừng chỉ trích từng dòng code một.
  - Điều này có thể gây ra sự khó chịu với tác giả, giảm sự cởi mở của họ trong các phản hồi. Nghiêm trọng hơn có thể phá vỡ mối quan hệ giữa bạn và tác giả.
  - Điều này nguy hiểm hơn nhiều so với mấy lỗi nhỏ nhặt như đặt tên biến chưa hoàn hảo.
  - Thay vì đó nên góp ý từ từ tập trung vào flow và nghiệp vụ của code. Hãy nhớ giúp nhau dần tiến bộ!
- Tôn trọng và tin tưởng tác giả:
  - Không ai cố tình viết code xấu cả.
  - Tác giả đã viết code tốt nhất theo kiến thức và niềm tin của họ.
- Tuân thủ quy tắc OIR khi đưa ra phản hồi:
  - Observation (Quan sát): Method này có 100 dòng code
  - Impact (Ảnh hưởng): Điều này làm cho tôi khó đọc và hiểu được logic của method này
  - Request (Phản hồi): Tôi khuyên bạn nên tách thành những hàm con và đặt tên biến có ý nghĩa hơn.
- Trước khi đưa ra phản hồi, hãy tự hỏi:
  - Có đúng không? (ý kiến != sự thật)
  - Có cần thiết không? (tránh cằn nhằn, bình luận không cần thiết và làm những việc ngoài phạm vi)
  - Có tốt không? (không chê trách)
- Khiêm tốn! Bạn không hoàn hảo và bạn cũng có thể cải thiện.
- Đừng quên khen ngợi:
  - Nên khen ngợi những cái gì tốt rồi mới nêu những vấn đề.
  - Cuối cùng khi mọi thứ đã okie hãy nói: "Everything is good!"

6 Thực hiện code review trên Gitlab với project được tổ chức theo Git flow có tích hợp CI

"***Hãy coi code review là một văn hóa của team, không phải là một quá trình. Để review code hiệu quả, bạn và nhóm của bạn cần phải quyết định xem điều gì là quan trọng để bạn và phần mềm cùng được phát triển.***"

Giả sử bạn là một lập trình viên. Bạn có nhiệm vụ phát triển một feature hoặc fix một bug.

1. Theo Git flow, bước đầu tiên bạn sẽ thực hiện tạo một nhánh feature_ten_tinh_nang hoặc bugfix_ten_loi từ nhánh develop hiện tại.
2. Bạn code tính năng/bugfix đó, tự code tự test và commit tới nhánh feature hoặc bugfix cho tới khi tự cảm thấy hoàn thiện. Đến lúc này theo Gitflow bạn sẽ cần merge code lại vào nhánh develop. Thao tác này được gọi là Pull Request (PR). Chúng ta không thực hiện Pull Request tự động mà sẽ thực hiện trên giao diện Gitlab để thực hiện code review và kết hợp đồng thời với CI. Các bước lần lượt như dưới đây:
3. Chuẩn bị pull request
    - Kiểm tra tên branch phải thể hiện mục đích của việc update. Việc này giúp cho reviewer có thể nắm bắt nhanh chóng bạn đang làm gì.
        - Bạn nên tạo theo format sau: *<type>_<issue id>_<tên issue>* trong đó:
            - type: mục tiêu của branch như feature, bugfix
            - issue id: mã id mô tả trên jira có thể là bug hoặc user story. Chỉ cần biết id này thì reviewer có thể nắm được họ cần review cái gì.
            - tên issue: mô tả ngắn gọn mục đích của issue
        - Nếu bạn đặt tên branch không đúng có thể dễ dàng thay đổi bằng lệnh git sau:
            - Nếu đang ở tại branch muốn đổi tên: ***git branch -m newName***
            - Nếu đang ở branch khác: ***git branch -m oldName newName***
    - Đảm bảo rằng branch của bạn phải update dựa trên lastest source của branch cần merge (branch develop).
        - Việc này sẽ giúp tránh những conflict không cần thiết. Đôi khi task bạn đang làm mất vài ngày mới hoàn thành thì trong khoảng thời gian đó có tới hàng chục commit đã được merge. Bạn có đảm bảo rằng sẽ không có ai conflict với bạn không. Chính vì thế việc phát triển source code dựa trên lastest source là điều cần thiết. Để làm được điều này, bạn cần thiết phải rebase source code thường xuyên. Không nhất thiết phải rebase mỗi khi có một commit mới nhưng bạn phải chắn chắc mình không được để lỡ quá nhiều commit. Mặc dù việc này khá là phiền phức nhưng lợi ích của nó đem lại cao hơn nhiều.
            - Phát hiện conflict và resolve nó ngay lập tức. Việc này giúp tiết kiệm khá nhiều thời gian vì chỉ resolve dựa trên số ít commit.
            - Tăng tính khả dụng của source code, code của bạn có thể hoạt động ngay lập tức khi được merge.
        - Rebase source code như sau:
            - Commit source code ở branch của bạn
            - Di chuyển sang branch chính bằng `git checkout <branch>`, fetch (`git fetch`) và pull lastest source về (`git pull`)
                - git checkout develop
                - git fetch && git pull
            - Di chuyển về branch của bạn: `git checkout <my branch>`
            - `git rebase develop`
            - Giải quyết conflicts nếu có
    - Squash commit
        - Việc bạn push commit lên branch của bạn là điều hết sức bình thường, có thể cứ mỗi lần code xong cái gì bạn lại push lên branch, việc này tránh bị mất code nếu như có sự cố gì xảy ra. Tuy nhiên khi thực hiện xong, nhìn lại branch, thì hàng tá commit như thế có thể khiến mọi thứ trở nên rối tung lên. Vì khi merge vào branch chính thì nó sẽ merge từng commit của

bạn. Càng nhiều commit thì khả năng phát sinh conflict giữa các commit càng cao. Sau khi merge, nhìn vào graph trên Git thì thấy rất rối.

- Việc squash commit từ nhiều commit thành ít commit hơn hoặc thành 1 commit chẳng những giúp graph trở nên đẹp hơn mà còn có thể giúp phát hiện lỗi dễ dành hơn bởi vì càng ít commit càng dễ xác định được đâu là nguyên nhân gây lỗi.
- Cách thực hiện squash như sau:
    - Kiểm tra log commit: `git log --oneline`

```
871adf OK, feature Z is fully implemented      --- newer commit
0c3317 Whoops, not yet...
87871a I'm ready!
643d0e Code cleanup
afb581 Fix this and that
4e9baa Cool implementation
d94e78 Prepare the workbench for feature Z
6394dc Feature Y                               --- older commit
```

    - Như ví dụ trên chúng ta cần gom các commit cho feature Z từ d94e78 tới 871adf. Để thực hiện điều này thực hiện rebase như sau:
        - Chọn N commit cần gom:
            - `git rebase -i HEAD~N` (Trong ví dụ trên N=7 nên có thể chạy `git rebase -i HEAD~7`)
            - Nếu có quá nhiều commit (khó có thể đếm được thì có thể thay bằng commit-hash (hash cũ gần nhất với vị trí bạn muốn gom). Ví dụ trên: `git rebase -i 6394dc`
        - Một màn hình editor hiện ra show danh sách các commit cần gom theo thứ tự từ cũ tới mới. Giữ nguyên pick đầu tiên (cũ nhất), thay thế chữ pick ở các dòng tiếp theo bằng chữ s (squash)

```
pick d94e78 Prepare the workbench for feature Z     --- older commit
s 4e9baa Cool implementation
s afb581 Fix this and that
s 643d0e Code cleanup
s 87871a I'm ready!
s 0c3317 Whoops, not yet...
s 871adf OK, feature Z is fully implemented      --- newer commit
```

        - Save file đó lại. Một màn hình yêu cầu nhập tên commit mới. Ví dụ ở đây chúng ta nhập tên commit là `Feature Z`
        - Cuối cùng hãy kiểm tra lại log commit: `git log --oneline`

```
9bcw3r Feature Z
6394dc Feature Y
```

4. Trên giao diện GitLab, tạo một merge request
    - Chọn Source branch là branch cần merge. Target branch là develop. Sau đó tiếp tục nhập các title, description và assignee - người có quyền allow cho thao tác merge này, có thể assign cho chính bản

thân bạn.

Creating merge request

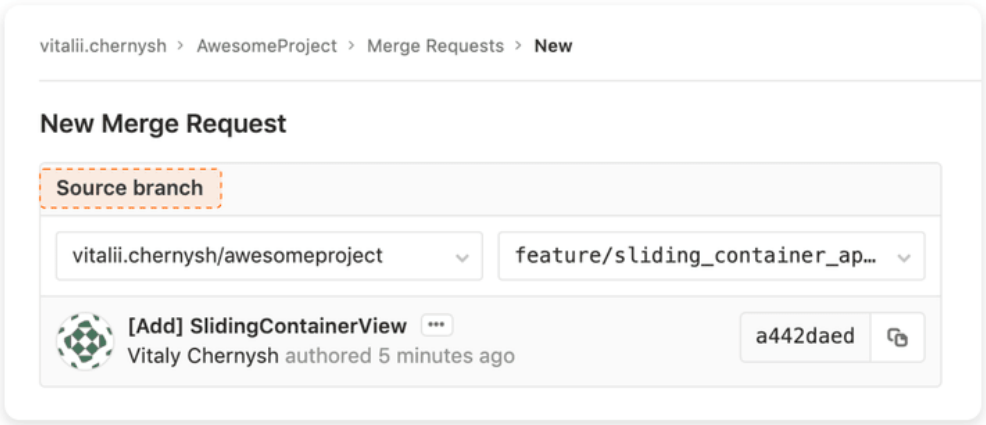
Step 1.

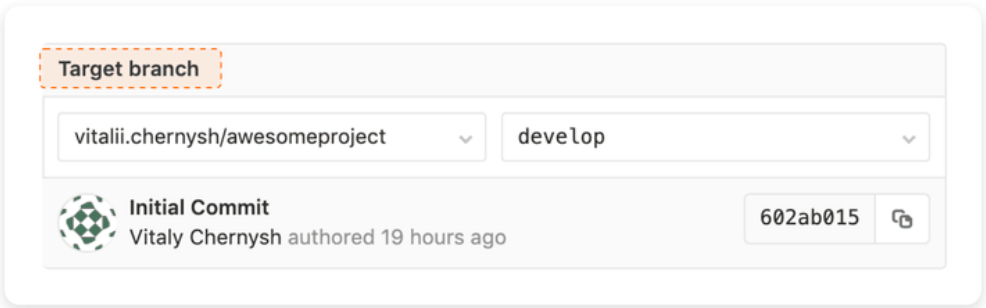
Click on the button

+ New merge request

Step 2.

Choose source and target branches





Step 3.

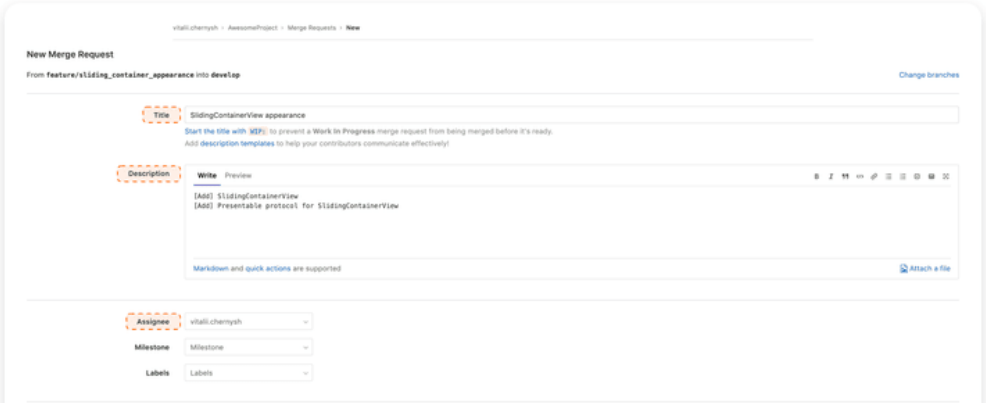
Click on the button

Compare branches and continue

Step 4.

Fill the fields:

• Title • Description • Assign to •



- Sau khi submit merge request. CI được trigger chạy unit test và code quality. Kết quả chạy CI được update vào Discussion của merge request.
- Bạn viết một comment trong Discussion và mentioned những người bạn muốn mời review code cho merge request này (Nhấn @ và chọn tên người review). Những người được mời sẽ nhận được email thông báo. Nên mời những ai và số lượng như nào thì do team tự thống nhất.
- Các reviewer khi nhận được email mời review sẽ truy cập vào gitlab để xem merge request này. Họ sẽ review theo những checklist quy định trước (xem mục Phải review những cái gì)
- Reviewer xem các thay đổi code, comment vào các đoạn code và thảo luận với tác giả thông qua discussion (Gitlab cung cấp giao diện trực quan cho các thao tác này)
- Bạn trao đổi thảo luận lại với các reviewer. Sửa code và commit lại vào chính source branch của bạn. Sau khi commit, các nội dung chỉnh sửa được tự động update vào giao diện merge request. CI lại được trigger chạy cho unit test và code quality - kết quả chạy được update vào discussion của merge request.
- Reviewer tiếp tục xem xét chỉnh sửa của bạn.
- Sau khi tất cả reviewer nhận xét "Everything is good!" hoặc "Looks good to me", "LGTM" tức là bạn đã hoàn thành bước code review.
- Bạn Accept merge request này và xóa branch cũ đi.
- CI tiếp tục được trigger chạy để build và deploy code từ nhánh develop tới hệ thống test. Lúc này các tester sẽ thực hiện test trên hệ thống test.