

從零開始 Code Review，兩年實戰經驗分享！

Java 技術堆疊於 2019-05-19 10:32:00 發佈 3985 收藏 12

作者：wenhx

<http://www.cnblogs.com/wenhx/p/5641766.html>

前幾天看了《Code Review 程式設計師的寄望與哀傷》，想到我們團隊開展 Code Review 也有 2 年了，結果還算比較滿意，有些經驗應該可以和大家一起分享、探討。

我們為什麼要推行 Code Review 呢？我們當時面臨著程式碼混亂、Bug 頻出的狀況。

當時我覺得要有所改變，希望能提高產品的程式碼質量，改善開發團隊面臨的困境。並且我個人在開發上有很多經驗，也希望這些知識能夠在團隊內傳播。

各種考慮後，我們最後認為推行 Code Review 能改善或解決我們面臨的很多問題。

這篇文章的目的不是告訴大家怎麼在一個團隊內推行 Code Review，首先因為我個人僅在一家公司內推行過，並沒有很多經驗。

其次每家公司、每個團隊的情況都不太一樣，應該根據公司或團隊的實際情況選擇恰當的方案，並根據成員的反饋來及時調整，推動 Code Review 的實施。

所以，本文是介紹我們公司是如何實施 Code Review 的，我們是如何解決我們遇到的問題的，希望我們的經驗能給大家帶來些幫助。

行文倉促，如有遺漏或錯誤，歡迎指正。關注 Java 技術堆疊微信公眾號，在後台回覆關

鍵字：**Java**，可以獲取更多棧長整理的 Java 技術乾貨。

一、流程和規則

經過簡單的對比、試用，我們最後採用了 Git Flow+Pull Request (PR) 模式來做 Code Review。 (PR 模式詳情可參見 Git 工作流指南：Pull Request 工作流)

Pull Request(PR)簡單的說就是你沒有權限往一個特定的倉庫或分支提交程式碼，你請求有權限的人把你提交的程式碼從你的倉庫或分支合併到指定的倉庫或分支。

由於 PR 需要有權限的人確認，所以非常適合在這個過程中做 Code Review，是否接受或者拒絕就取決於 Code Review 的結果。

在支援 PR 模式的軟體裡，每一個 PR 都有一個新增程式碼的對比 (diff) 介面。

程式碼稽核者可以線上瀏覽請求合併的新增程式碼，並針對有疑問的程式碼行新增評論，通過這種方式來實現 Code Review。

評論可以被所有有權限查看倉庫的人看到，每個人都可以回覆任何人的評論，有點像論壇裡某個帖子的討論。

這種模式是事後稽核，也就是程式碼已經提交到了中心倉庫，Review 過程中頻繁的改動會造成歷史簽入記錄的混亂。

當然 Git 可以採用更改歷史記錄來解決這個問題，由於容易誤操作，我們一般只在基礎類庫這類要求比較嚴格的項目上實施。

我們所瞭解到的支援 PR 模式的軟體都採用 Git 作為原始碼版本控制工具，所以我們的原始碼版本控制工具也遷移到了 Git。

由於 Git 太靈活了，因此誕生了很多的 Git 流程，用來規範 Git 的使用。

常見的有集中式工作流、功能分支工作流、Gitflow 工作流、Forking 工作流、Github 工作流。

我們對 Git Flow 做了些調整，調整後的流程被命名為 Baza Flow，定義見後文。

根據 Baza Flow，我們大部分倉庫只定義了 2 個主幹分支，master 和 develop。(例外，我們有一個倉庫有 3 個開發小組同時進行開發，定義了 4 個主幹分支，目前還比較順暢，再多估計主幹分支之間的合併就比較繁瑣了。)

master 對應生產環境程式碼，所有面向生產環境的發佈來源都是 master 分支的程式碼。

develop 則對應本地測試環境的程式碼。

絕大多數情況下，QA（測試）只測試 develop 分支和 master 分支的程式碼。

由於開發人員都在一個團隊內，所以我們沒有採用基於倉庫的 PR，採用的是基於分支的 PR。

我們對主幹分支的操作權限做了限制，只有特定的人才能操作，develop 分支是項目開發 Leader 和架構師，master 分支是 QA。

有權限往主幹分支合併的成員會按照約定的規則來執行合併，不會合併沒有完成稽核的 PR。

上面這點其實蠻重要的，所以我們會對有權限合併的人有特別的約定，在什麼情況下才能合併程式碼。（見後文 PR 的說明）

PR 的發起人要主動的推動 PR 的稽核，Leader 也會密切關注 PR 稽核的進度，在需要的時候及時介入。

我們組態了 CI 伺服器（什麼是 CI）只編譯特定的分支，通常是 develop 和 master 分支。

所有的程式碼合併到了主幹分支之後，都會自動觸發編譯和本地測試環境的發佈，QA 無需依賴開發人員編譯的程式碼來測試，也無需自己手工操作這些，保證了開發人員和測試人員的相互獨立。

我們本地測試環境的發佈包含了資料庫和站點的發佈，全自動的，發佈完成以後就是一個可用的產品，有時間這部分也可以分享一下。

我們還使用了 Scrum 裡面一個很重要的概念：完成定義。

就是我們規定了我們一個任務的完成被定義為：程式碼編寫完成，經過自測，提交的 PR 經過稽核並且合併到主幹分支。

也就是說，所有的程式碼被合併到了主幹分支之後任務才算是完成，而被合併到主幹分支必須要經過 Code Review，這是強制的。

由於我們的託管軟體對於 Pull Request 的限制，我們對 Git Flow 做了改動，改動的地方有：

1. 每一個大功能我們會建立一個單獨的 feature 分支，項目開發人員基於這個單獨的 feature 分支建立自己的任務分支。

比如，對於 CS 2 項目來說，啟動的時候分支的建立是：master -> develop -> feature/v2。

開發人員應該基於這個大特性分支 feature/v2 來建立自己的任務分支，比如建立 XXXX，可以用一個單獨的分支 feature/v2-xxxx。

完成這個任務以後，立即向上游分支（feature/v2）提交 pull request。然後從 feature/v2-xxxx 建立自己的下一個任務分支，比如 YYYY 編輯 feature/v2-yyyy。

請注意，合併到上游分支的功能必須相對獨立而且是可用的，分支任務工作量 0.5-1 個工作日，不宜超過 2 個工作日，超過 2 個工作日不向上游合併，需要向團隊解釋。

程式碼經過 Review 以後，可能會進行必要的修改，修改在原分支修改，修改完畢程式碼合併進上游分支，原分支會定期刪除。

項目組成員在收到合併成功的通知後，請自行從上游大特性分支向下合併到自己當前的開發分支。

提交 pull request 後建立新任務分支的時候務必知會一下相關配合同事（比如前端的同事），讓他們在新的分支上繼續開發。

2. 對於小功能，預計在 0.5-1 個（不超過 2 個）工作日工作量的開發任務，直接基於 develop 分支建立特性分支即可。

3. 在各個分支遇到的 bug，請基於該分支建立一個 Bug 分支。

如果在缺陷跟蹤管理系統上有對應的項，命名請使用缺陷跟蹤管理系統的 ID，比如 BAZABUG-1354 比如這個 Bug 的分支命名就是 bugfix/BAZABUG-1354。

如果在缺陷跟蹤管理系統上沒有對應的項，命名請簡短的說明修改內容，比如“JX 9df2b01 引用 bootstrap css 虛擬路徑重寫，避免出現字型無法找到的問題”，分支命名可以是 bugfix/miss-font。

完成修改以後提交並推送到中心倉庫然後立即向上游分支提交 pull request。

4. 發起 pull request 以後，請將 pull request 的連結在 IM 上發給程式碼稽核者，以此通知對方及時進行稽核。

二、執行

我們在團隊內部提倡質量優先，開發團隊不能為了進度犧牲質量，並在團隊內部達成了共識。

所以，無論進度有多麼緊迫，Code Review 的過程都一定會做。

所有的問題一定會被提出，只是會根據進度的緊迫程度，以及問題的大小，改動成本，決定問題是現在解決，還是加一個 TODO，並記錄在缺陷跟蹤管理系統內，以防日後遺忘。

多數情況下，我們都會要求立即解決，哪怕因此造成了發佈的推遲。

我們深知，其實多數情況下，現在不解決，日後不知道猴年馬月才能解決。

我們在團隊內推行 Code Review 的過程中沒有遇到太多阻力。


原因大概有兩點，首先管理層方面瞭解之前遇到的各種問題，也迫切希望能有所改善，所以從一開始就是支援的態度。

其次，絕大部分開發人員覺得在這個過程中能自己能學習到東西，並沒有牴觸，遇到很好的意見時大家都還是很高興的。


最後，慢慢的形成了一種氛圍，整個團隊都會自覺的維護它。

附一張我們稽核的對話圖，這位童鞋嘗試對系統內部散落各地發業務郵件的程式碼做一個整理，用一套模式來處理，調整了 3 版才定調，然後修改了很多細節才通過了合併，前後大概用一個多星期時間：

 16:34
已合并。

 16:35
这一路真漫长 🤔

 16:35
👍 👍 现在的代码很漂亮了呀。

 16:36
学到了很多 🍌 🍌

表面上看來 Code Review 會延緩項目的進度，但是在我們 2 年多的執行過程中，大多數時候沒感覺到延緩。

原因是，雖然程式碼合併的週期變長了，但是由於程式碼質量提高了，導致 Bug 變少了，由於 Bug 引起的返工問題也變少了，因此整體的進度其實並沒有延緩。

我個人認為對一個成熟的團隊其實做 Code Review 反而會加快整體的項目進度，但是手頭上沒有統計資料支撐我的觀點。（對於軟體開發的度量，歡迎有心得的同學告知我）

我們每個分支有權限合併的人都不止一個，這樣可以保證有人請假不在的時候，程式碼仍然可以被其他同事稽核通過之後合併。

半年前，我們團隊加入了很多新成員，剛加入的新同事對規範、項目、產品的熟悉程度都不高，導致了有一段時間，我們遇到了 PR 稽核週期變長的問題。

加上之前遇到的一些問題，我們總結了一個說明，目的是減輕 Code Review 對開發人員工作的負擔，加快 PR 稽核通過的過程。

說明如下：

Pull Request 的說明

任務完成才能提交 PR。

PR 應該在一個工作日內被合併或者被拒絕。

PR 在有嚴重問題（包括但不限於架構問題、安全問題、設計問題），太多問題，或者任務無效的情況下會被拒絕。

嚴禁一個 PR 裡面有多個任務，除非它們是緊密關聯的。

PR 提交之後只允許針對 Review 發現問題再次提交程式碼，除非有充足的理由，嚴禁在同一个 PR 中再次提交其它任務的程式碼。

提交 PR 時候有一個描述框，內容會自動根據 Commit 的 message 合併而成。

切記，如果一次提交的內容包含很多 Commit，請不要使用自動生成的描述。

請用簡短但是足夠說明問題的語言（理想是控制在 3 句話之內）來描述：

你改動了什麼，解決了什麼問題，需要程式碼審查的人留意那些影響比較大的改動。

特別需要留意，如果對基礎、公共的元件進行了改動，一定要另起一行特別說明。

稽核人員邀請原則：

1. 在建立 PR 時，Reviewers（稽核人）一欄裡主要填寫“必需稽核人”。只有這些人稽核都通過，才允許合併。
2. 除了“必需稽核人”外，還有一些其它稽核人，我們可以在 Description 裡做為“邀請稽核嘉賓”@進來。
3. 主幹分支間的合併，如 Develop => Master，或 Master => Develop 等，則需要把整個團隊（開發+QA）都列為“必需稽核人”。

必須稽核人的列表由團隊決定，可能包括以下人選：

團隊 Leader

前端架構師（如果有前端程式碼改動）（可以授權）

後端架構師（如果有後端程式碼改動）（可以授權）

產品架構師

對此 PR 解決的問題比較熟悉的（之前一直負責這部分業務的同事）

此 PR 解決的問題對他影響比較大（比如認領的任務依賴此 PR 的同事）

其它稽核人，包括但不限於：

需要知悉此處程式碼改動的人但又不必非要其稽核通過的同事

可以從這個 PR 中學習的同事

可以授權指的是，根據約定，Bug 修復之類的改動，或者影響較小的改動，前端架構師和後端架構師可以授權團隊內的某個資深開發人員，由這個資深開發人員代表他們進行稽核。

主幹分支之間的合併，大型 Feature 的合併，前端架構師和後端架構師需要參與。

上述稽核人關注的視角不太一樣：

團隊 Leader 關注你是否完成了任務，前後端架構師關注是否符合公司統一的架構、風格、質量，產品架構師從整個產品層面來關注這個 PR。

熟悉此問題的同事可以更好的保證問題被解決，確保沒有引入新問題。

被影響的同事可以及時瞭解他受到的影響。

團隊 Leader 或者產品架構師如果覺得 PR 邀請的稽核者不足或者過多，必須調整為合適的人員，其它同事可以在評論中建議。

三、收穫

我們團隊實施 Code Review 收穫不少，總結出來大概有以下幾點：

1、 短期內迅速提高了程式碼質量。

原因有幾個，大家知道自己的程式碼會被人稽核之後寫得會比較認真。

理論上程式碼質量是由整個團隊內最優秀的那個人決定的。

大家也能在 Review 的過程中學習到其它同事優秀的編碼。

2、 Bug 數量迅速減少。

但是這個我們沒有資料統計比較，比較遺憾。

我和 QA 聊過，他給我的資料是在我們的一個新項目每 2 週一次的大發佈，平均只會發現 1~2 個 Bug。

這點提高了整個團隊的幸福感，大家不用經常被火燒眉毛。

3、 團隊成員對項目的熟悉程度會比較均衡。

新同事通過參與 Code Review 能很快熟悉團隊的規範。

程式碼不會只有個別人瞭解、熟悉，Bug 誰都能改，新功能誰都能做。

對公司來說避免了人員的風險，對個人來說比較輕鬆（誰都能來幫你），可以選自己喜歡的任務做。

4、 改善團隊的氛圍

Review 的過程中會需要非常多的溝通，多溝通能拉近團隊成員的距離。

並且無論等級高低，大家的程式碼都是要經過 Review 的，可以在團隊內營造一個平等的氛圍。

每個成員都可以審查別人的程式碼，這很容易激發他們的積極性。

亮一下我們的資料:

我們從 2014 年 1 月 17 日開始第一個 PR 的提交，到 2016 年 7 月 5 日一共發出了 6944

個 PR，其中 6171 個通過，739 個拒絕。日均 11.85 個 PR，最多的一天提了 55 個 PR。

這些 PR 一共產生了 30040 個評論，平均每個 PR 有 4.32 個評論，最多的一個 PR 有 239 個評論。

參與上述 PR 評論的同事一共有 53 位，平均每位同事發出了 539 個評論，最多的使用者發出了 5311 個評論，最少的發了 1 個（剛推行 Code Review 就離職的同事）。

需要說明一下，只有簡單的問題會通過評論來提出。比較複雜的，比如涉及到架構、安全等方面的問題，其實都會面對面的溝通，因為這樣效率更高。

四、總結

雖然有合適的工具支援會更容易實施 Code Review，但它本身並不特別依賴具體的工具，所以前文並沒有具體指明我們用了什麼工具，除了 Git。

原因是基於分支的 PR 流程依賴於大量建立分支，而 Git 建立一個分支非常的簡單，所以 PR 模式+Git 是一個很好的搭配。

我們在切換到 Git 之前，也做 Code Review，採用的是提交程式碼以後把 commit 的 Id 發給相關同事來審查的流程。

稽核通過以後會在缺陷跟蹤管理系統裡面評論，QA 同事沒見到稽核通過的評論就認為任務沒有完成，拒絕進行測試。

雖然沒有現在這樣直接方便，但是也還是做起來了。

PR 稽核的過程中，新加入的團隊成員常見的問題是不符合程式碼規範之類的，其實是可以通過原始碼檢查工具來解決的，這部分我們一直在計畫中 ((√ □ √))，並沒有開始實施。