Gitを導入するときにやったこと

目次

GitとGitHubをプロジェクトに導入する際にやったことをメモする。

はじめに

プロジェクトの初期から参画し、ある程度の舵取りができる機会に恵まれた。
プロジェクト参画時の状況としては以下の通り。

- Githubの採用は決定しているが、運用方法の詳細は決まっていない。
- Javaのチームで、人数は８人程度。
    - 開発環境はこれから構築する。
    - 設計担当が２名で、残りの６名がそれぞればらばらに製造を担当する。
    - メンバーはGitの経験はほとんどない。

ブランチ戦略

まず最初にブランチ戦略を決定する。

Gitでよく使われるブランチ戦略は、Git-Flow、Github-Flow、GitLab-Flowなどがある。
今回の導入には、Git-Flowを採用した。考慮した点は以下の通り。

- 数か月ごとにリリースがある。
  数か月ごとにリリースがあり、リリースされたものはエンドユーザに公開されて実践に即した動作テストに使われる。
  次のリリースまでの間は、次々と機能追加が実施される。
  つまり、リリース済みのブランチと作業中のブランチは明確に分離して管理する必要があった。
- チームメンバーが一つのシステムに対して、それぞれ個別に機能追加を担当する。
  多機能な画面が複数ある複雑なシステムを複数の人数で分担して作業することで、短期間に構築する必要があった。

この二つの要件を同時に満たすのがGit-Flowだった。
とくに、一度リリースしたものをmasterブランチに保存しておいて、別ブランチで実装作業を進めるという手順が明確に記述されているという点を評価した。
神経使って管理するべきブランチが２本になることでオペレーションがややこしくなるよう見えるが、ホットフィックスブランチやリリースブランチが完了する際にmasterとdevelopの両方に行うマージを一動作で実行するので手順の煩雑さは影響しない。

ツールの選定

使用するツールはSourceTreeを導入した。
コマンドラインにこだわる必要はない。

複数の人員で作業すると、必ずコンフリクトを起こす。
それはチームメンバーの技術レベルや習熟度などとは直接の関係はなく、だれでも起こしうる事故と解釈すべき。
その場合、修復作業をコマンドラインでやることは、不可能ではないがそれなりの学習コストがかかる。
同じ低い習熟度からスタートした場合、やはりGUIがあったほうがとっかかりを得やすい。

人にやらせる前に自分で実際にやってみる

本格的な作業用リポジトリを作る前に、自分で練習用のリポジトリを作って、実際に一通りの操作を経験しておく。

リポジトリの内容は、特別なにかのアプリケーションでなくてもいい。
例えば、公開用の手順書などを作る過程で、ドキュメント用のリポジトリを一つ作成する。

その同じリポジトリを複数のフォルダにチェックアウトし、それぞれのフォルダを別の担当者に割り振ったという体で、フィーチャーブランチやリリースブランチ、ホットフィックスブランチの開始から完了までの動作をファイル修正を交えて実施する。

事前に操作をして見ることで、チームメンバーにレクチャーする際の要点を洗い出すことができる。

ドキュメントを作る

自分でやってみる過程において、作業工程をテキストに記録しておく。そしてそれをもとに公開用の手順書をテキストにする。

作成したテキストは、閲覧・修正が容易な形で配布する。

Githubの採用は当初から決定していたので、テキストはasciidoc形式で記述することにした。
Githubでasciidoc形式のテキストを使うことで、コミット・プッシュするだけでファイルの公開まで可能となる。
そのままGitの管理下としてバージョン管理されるので、詳細な変更履歴はあえて作る必要はない。

アンチパターン

Git-Flow戦略に限らず、masterブランチを正しくメンテナンスすることは、
Gitを運用する上での命綱となる。

チーム内のフォロー

テキストを用意するだけでは不十分なので、実際の操作をチームメンバーの目の前で、何度でも実演して見せる必要がある。

操作に困ったらいつでも声をかけるようにアナウンスをし、実際にヘルプの要請がかかった場合はできる限り優先して対応する。

Git-Flow戦略の場合は、基本的に機能追加では master ブランチは手を付けない。
また、作業用である develop ブランチも、実作業は fearture ブランチを分岐して行うので直接編集することはない。
その上、ローカル開発環境にもリポジトリを作るので、何度コミットしてもプッシュしない限りは中央リポジトリには送信されないので、最悪の場合もでローカルのチェックアウトを削除すれば、全ての作業はなかったことになる。
つまり、本格的に破局的な失敗をする前に何段階もセーフティーネットがある。
実演の際には、それらの使い方を説明する。

とにかくはじめて使う場合は不安がつきまとうので、ぶっ壊しても大丈夫という安心感を演出する。

一度のレクチャーやひとつのドキュメント公開ではきっと対応し切れない。
チームメンバーから不明点や質問があがればそれはむしろ歓迎するべき。メンバーの環境でマウスを借りて実演してること。
個別に丁寧に対応することが、けっきょくは最も工数が少なくてすむ。個別に丁寧に対応していれば、やらかす前に声をかけてくれるようになるはず。