layout: page title: Description permalink: /desc/

Osaka×Github Meetup!!

content

- 自己紹介
- github wiki
- git flow
- 最後に - Jekyll + gh-pagesでの静的ページ

伝えること

- 1つのプロジェクトの設計から開発終わりまでのフローをGitHubを利用してやるには
- なぜGitHubの機能を使用しているか、過去の経験とともに話す
- 15分くらいで終わる内容で

自己紹介

スライド1：経歴書

- 竹内宏佑
- 使用言語 -> Python, Haskell, Swift, Java
- 中学くらいからJavaScriptのプログラミングを初める
- 大学で情報科学を専攻
- 1年の後半からプログラミングの仕事をするようになる
- 3年に友人と、バイラルメディアサイト「Whats」の構築 -> 500強UUのサイトに
- 1年後の夏に、単独でサンフランシスコのデータ分析系のベンチャーでインターン
- その後、株式会社WarranteeでCTOを務め、現在に至る -> これが一年前

スライド2：Warranteeロゴ

1年前に、現在のWarranteeに入社した当時は、私は社内のプロジェクトマネジメントに何度も憤りました。 会社ではモバイルアプリケーション、そのバックエンドAPIがメインで開発されていましたが、 適当なGitのブランチ管理、まっさらなドキュメントしかなく、入って私がすぐできることといえば、 ひたすら社内で開発されたコードを読みふけることだけでした。

スライド3：プロマネのジレンマ

現在日本の多くのベンチャーでは、システムの多くがRuby/Railsで書かれているため、(Rubyを愛するGitHubさん、ごめんなさい。私は心からPythonとHaskellが大好きです) 大半のプロジェクトはうまくモジュール化されておらず、多くのコードが依存しあっているため、 入社当初のプログラマが私と同じような状態に陥っていると思います。

これはもちろん、プログラミング言語や、コードを書くプログラマのリファクタリング能力にもよると思いますが、 如何せんベンチャー業界で特別優秀なエンジニアを取得することは困難です。

これはアプリケーションに混在するコードだけの話でなく、そもそもアプリケーションの機能自体にも同じことがいえます。

しかし、プログラミング言語の選択や、プログラマのコード管理技術の向上に比べて、アプリケーションの機能(いわゆるインターフェース)は、どの会社のものを見ても、意外と綺麗にモジュール化(Restful)されていると思います。(ここはRuby on Railsが社会に与えた恩恵でもあると思います)

スライド4：ossドキュメンテーション

すこし話がずれますが、みなさんが日常でGitHubを利用する機会といえば、やはりオープンソースのプロジェクトを検索する時だと思います。 ベンチャー会社で限られたエンジニアリングのパワーを発揮するためには、できるだけプロジェクトのコアな部分以外(画像のアップローダやメールの送信など)は自力ではなく、世界の優秀なハッカーたちが作ってくれたオープンソースライブラリを使いまわして開発をすすめていかなければなりません。

GitHubでホスティングされているオープンソースで人気の高いものの殆どは、READMEというライブラリのドキュメントがしっかり書かれています。これは初めてライブラリを見たユーザーに対して、ライブラリが何に使えるのかやどのようにして使うことができるのかなどを伝えるためです。

スライド5: GitHub Wiki

したがって、オープンソースのライブラリを使う最初のステップでは、私がWarranteeに入ったころにプログラムを読みふけっていたように、ライブラリの中身まで読んで理解する必要はありません。私はこのオープンソースの特性を社内でのプロジェクトマネジメントでも利用したいと思い、GitHub Wikiの利用を始めました。

GitHub Wiki

スライド1: WarranteeでのGitHub Repositoryの使用

Warranteeでは、すべてのアプリケーションをGitHubのリポジトリで管理しています。私がWarranteeに入ってから現在まで、モバイルアプリケーションでユーザーが入力したデータの処理を行うモバイルバックエンドAPIの開発を中心に行ってきたので、今回はAPIのプロジェクトを例に話していこうと思います。

スライド2: APIの概要

もともとAPI自体は入社当時から開発されていましたが、基本的な機能が何かは知りませんでした。WarranteeのAPIは単純で、ユーザーの登録/ログインや、ユーザーが購入した家電製品の購買情報などを登録する機能がメインですが、それ以外にも細かい機能がたくさんあります。

まず私が作成したのは、APIの機能一覧のWikiを作成することです。

GitHubのWikiはMarkdownと呼ばれるHTMLやTexのような単純な文章構造を記述できるフォーマットで書くことができ、さらにサイドバーなども作成することができます。

次に、APIの各機能の詳細な使用方法を、自分で定めた統一のフォーマットにしたがってひたすら書き続けました。(すべての機能のドキュメントを書き終えるのに1週間近くはかかりました)

APIの開発とWikiを使ったドキュメント作成が終わったあと、私はiOSのモバイルプログラムを書き始めました。 この時、GitHubのWikiにAPIの使用方法がすべて記述してあったため、過去に自分が書いたAPIのプログラムを見なおして「ユーザーの個人情報を取得するときのAPIの使い方ってどうだっけ？」と迷う必要はありませんでした。つまり、モバイルのプログラムを書く際に、APIのプログラムを見なおして時間を浪費することはありませんでした。

スライド3: 外注時のドキュメント提供

Wikiにドキュメントを書いた恩恵はこれだけではありませんでした。

数カ月前、私はAPIとモバイル以外のアプリケーションのプログラムを書く必要になり、モバイルのプログラムを外注先にお願いすることになりました。

外注先に依頼するとき、追加してほしい機能だけではなく、現在の機能でどのようにAPIを使用しているかを外注先に伝えないと、普段から自社のアプリケーションを開発しているわけでもない外注先のエンジニアにとって、突然プログラムを読み始めるのは大変です。

そこで私は、GitHubにはリポジトリごとに外部のユーザーを追加することができるため、ドキュメントのみ記述されているリポジトリを新規作成して、外注先のユーザーにAPIの仕様を共有しました。

APIのドキュメントは早急に作成したものであるため、一部間違いがあることもありましたが、その点を他の会社の開発者から指摘されてもWikiですぐに修正して、そのまま共有することができました。

Git Flow

スライド1: Wikiの作成とその後の開発フロー

APIのドキュメントをWikiに書き終えた後は、次にAPIに実装する機能をまずWikiに書く癖をつけました。 そうすることによって、APIを書き始める前に、APIの設計は初めて見る人にとって使いやすいかを考えることができるからです。

スライド2: Git Flow

APIの設計をWikiに書くと、続いて実装していきます。Git FlowはVincent Driessennが開発したGitブランチの綺麗な管理モデルで、私のようにこれから実装する機能や修正する機能が事前に決まっている場合はとても有効です。

Gitではいろいろなプログラムの新規開発やバグの修正を自分の作業しているブランチにコミットしますが、Git Flowという拡張機能をインストールすると、ブランチが「feature, develop, release, master, hotfix」に別れます。

- *feature*: 新規機能開発中のブランチ
- *develop*: 開発済み(未検証)のブランチ
- *release*: 開発から運用に移行するまでの検証・リリース準備ブランチ
- *master*: 運用で使用されている本番用のプログラムを管理 (バグや未実装があってはいけない)
- *hotfix*: 運用で見つかったバグを即座に修正するためのブランチ

スライド3：新しい機能の開発例

WarranteeのAPIでは、ユーザーが登録した家電を中古売却査定に出すための機能がありますが、こちらを実装する手順を元に、Git Flowについて簡単に解説していきます。

まず、Git Flowを既存のGitプロジェクトに追加する場合は、以下のコマンドを実行します。

{% highlight sh %} $ git flow init #> branch: develop {% endhighlight %}

スライド4：Featureブランチの作成

次に、設計に沿ってAPIの機能を実装していきますが、この機能は新しいAPIの機能として実装するため、featureブランチを作成します。

{% highlight sh %} $ git flow feature start sales-order #> branch: feature/sales-order {% endhighlight %}

今回は中古売却の査定を依頼するための機能を作成しますので、sales-orderというブランチをfeatureとして作成しました。

スライド5：コミット

開発中は、モジュールを追加・修正するごとに細かくコミットしていきます。

{% highlight sh %} $ git add --all $ git commit -m 'ham spam egg...' $ git push origin feature/sales-order {% endhighlight %}

スライド6：Featureブランチの終了

新しい機能が完成したら、最後にfeatureブランチを終了して、developブランチに統合します。

{% highlight sh %} $ git flow feature finish sales-order #> branch: develop {% endhighlight %}

このあとreleaseブランチで運用前の検証などを行い、完成品をmasterに統合するのですが、この箇所は普段の開発の中で多発する作業でもないので、今回は短い発表時間ということもあり割愛させていただきます。

GitHub Pages

最後に、GitHubのよく知られた便利機能を紹介しておしまいにします。

スライド1：Jekyll

この発表資料はJekyllという簡単で頻繁に更新しないようなページを作成するためのツールを用いて作りました。Jekyllはベースのデザインを HTML, CSS, JSで作成しておくと、コンテンツの文章などをREADMEでも使われているMarkdownによって書くことができ、すべてのページに対して同じデザインが適応されます。

スライド2：GitHub Pages

Jekyllで作成したこのページはGitHubのリポジトリで保管していますが、GitHub Pagesを設定することによって、ご覧のように公開された状態でページを見ることができます。 もちろん自分のドメインを設定することも可能です。

GitHub PagesはGitHubのそれぞれのリポジトリごとに設定することができるため、今回はせっかくGitHubの講演会ということもあり、GitHub Pagesでスライドを公開してみました。

Osaka×Github Meetup | green-latte

Fin

ご清聴、ありがとうございました！