Rubyプログラマが
育つ仕組み

Rubyでの受託開発を10年回してみて

2017.11.2

(株) 万葉 大場寧子 @nay3

About Me

- 大場寧子（おおば　やすこ） @nay3
- プログラマ歴30年くらい
- Rubyプログラマ
- Railsエンジニア
- (株) 万葉 代表取締役

Family

(株) 万葉

- 2007年 4月2日に2名で設立
- Ruby on Rails を使った受託開発が中心
- おかげさまで、2017年4月で10周年を迎えることができました！
- 現在23名

We are hiring !

- 「たのしくチーム開発すること」にフォーカスしたい方
- 落ち着いた環境でプライベートも重視しながらスキル向上を図りたい方
- リモート勤務制度あり（研修中は東京勤務が必要）

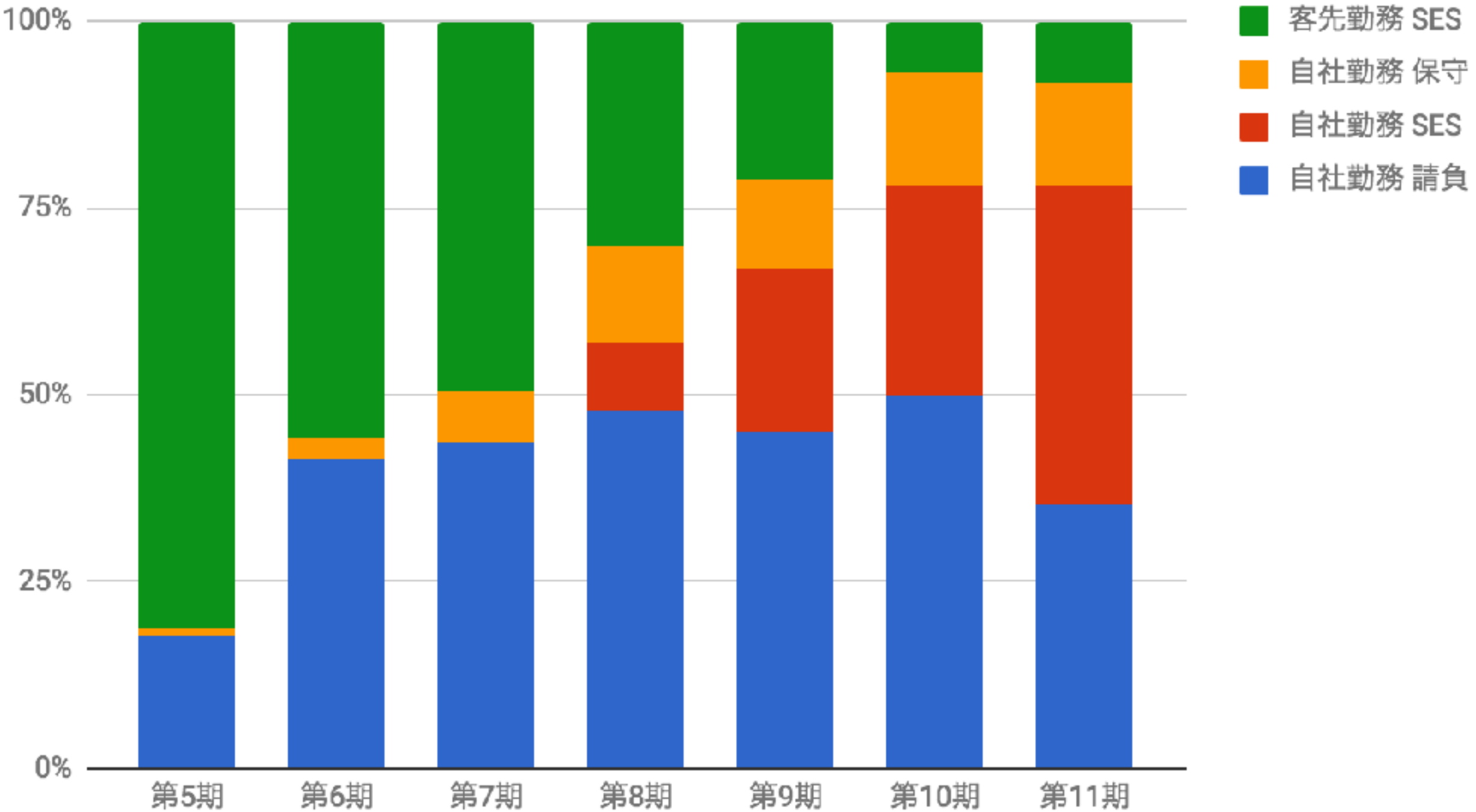
10年の中で
万葉が
変えてきたこと

https://www.flickr.com/photos/santheo/7004013586

- 仕事の形・内容
- チーム開発
- 組織の形
- 採用システム
- 教育システム

仕事の形

【 過去7年間の仕事形態の変化 】



仕事の内容

- Rails 中心
- Vue.js、React などが増えた
- 途中、スマフォアプリをたまに
- PHPを1度だけ受けた
- 最近 Go の案件もやったりしています

自社サービス開発を
やってみたい気持ちを
捨てた

受託の専門として
やっていく路線を選択

チーム開発

- 自分は理解が足りてなかった
- 入社した人々の奮闘によって特に育ってきた分野
- 日々研究

https://www.flickr.com/photos/schsboyssoccer/6125042131

組織の形

https://www.flickr.com/photos/freyaserendipity/8559076052

最初は古典的な
ピラミッド型の
中小企業を志向

ピラミッド型組織の課題

- チーム開発との相性が悪かった
  - チーム開発はフラット志向
  - 必要な情報を公開して個人が判断できるようにする基本原則が、情報を含めて統制するピラミッド型とそぐわない
- 受託では、案件チームはお客様の状況で柔軟に変動
  - 案件チーム主体の日々の生活と、固定的なグループごとに中間管理職に管理してもらう形が合わない

「ホラクラシー」

を参考にした

ホラクラシーとは

- ピラミッド型の組織とは違う新しい組織の形として提唱されている
- 生物の細胞のように、機能別に自立して働く仕組みを目指す
- 脳が個々の細胞の活動に直接指示を出さないように、ホラクラシーではサークルやロールといった機能が自分のことを自分で決める
- フラット構造と言われることもあるが、どちらかというと細胞構造（入れ子構造）が正しそう

現在のサークル/ロール

総務系の仕事の整理に使うとともに、広報、アサイン決定、採用書類選考、新人社員教育、情報システム、wiki整理などいろいろなロールに技術系の社員も就任

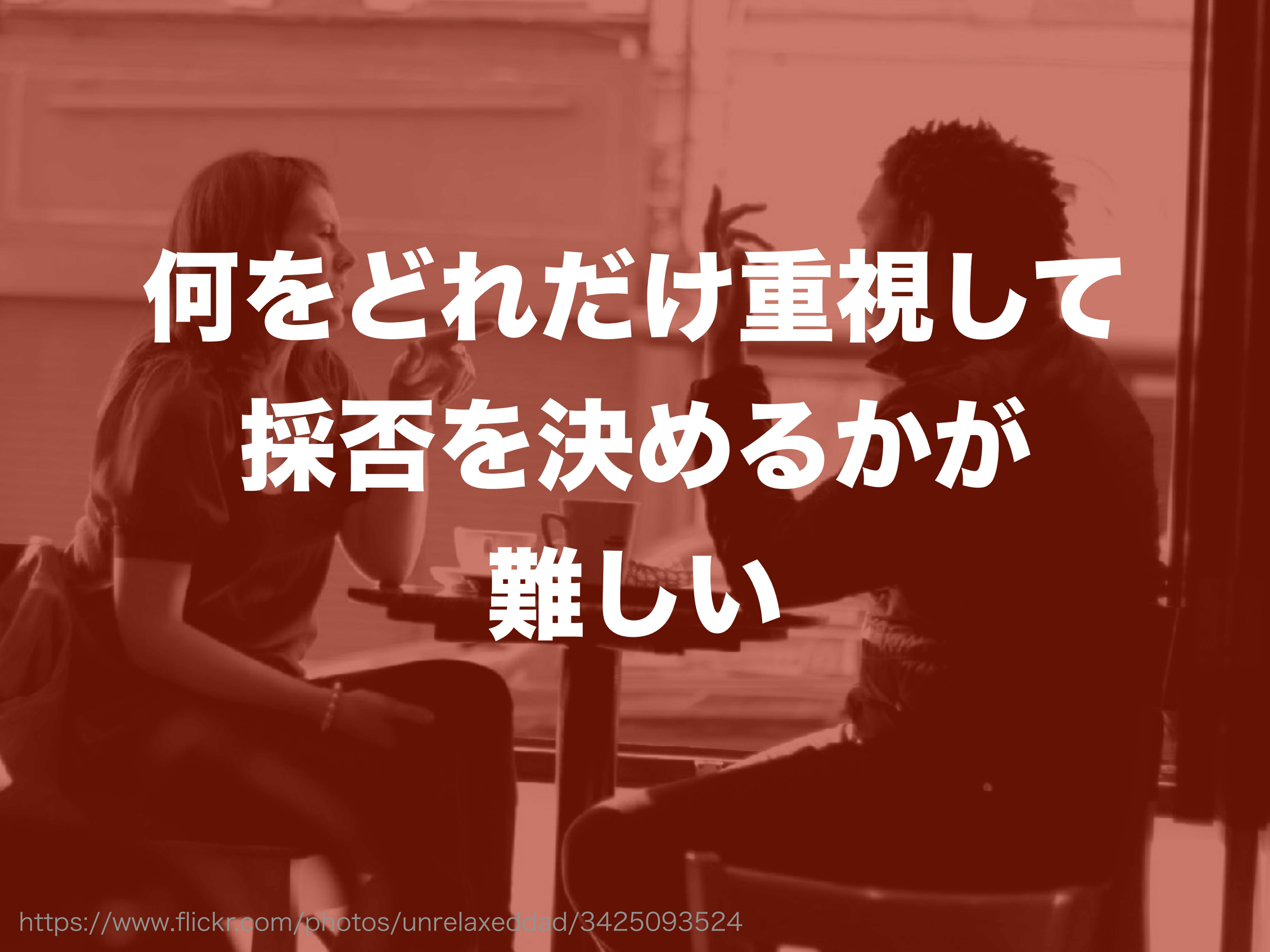
良くなってきた

- 厳格な運用ではなく、ゆるく取り入れている
- 担当者・担当範囲がはっきりして、物事を決めやすくなった
- 情報公開・レビューとの相性が良い
- チーム開発との相性が良い

採用
システム

https://www.flickr.com/photos/booboobook/6171761028

設立後まもなくから
面接でチェックしたい
項目を社内で共有



何をどれだけ重視して採否を決めるかが難しい

https://www.flickr.com/photos/unrelaxeddad/3425093524

面接官による採点から採否を算出する方法を確立

スムーズになり

成功が続くように

教育システム

https://www.flickr.com/photos/merchau/8548057127

受託開発は
技術者を
育てやすい
形態

色々な
開発案件

文化

チーム

社会のためにも
技術者を育てたい
気持ちがある
(少しずつですが)

万葉のフォーカス

- 成功するプロジェクトを増やしたい
- 「とりあえずRubyとRailsが書ける人」の頭数を増やしても成功するプロジェクトは増えない
- 「上手にRubyとRailsが書けて、チームで開発ができる人」を増やしたい

上手にRubyが書ける人

- Rubyは自由な言語
  - 自由には責任が伴う
    - 書き方を柔軟に自分で判断できると良い
- Rubyに合う設計、命名、実装
- 現実とRubyコードの間に折り合いをつけ続ける

上手にRailsが書ける人

- Rails のレールにうまく乗っかりながら目的達成まで持っていける
- 背景・近隣の知識（OOP、Ruby、Web、RDB、JSなど）がわかる
- バージョンアップの荒波やメンバー交代に耐え得る維持しやすいコードを書ける

知識に加えて
判断のうまさ
バランス感覚
が重要

実地で経験を積む必要がある

以前の新人研修

- メンターを任命する
- 会社のことを説明したら、すぐにプロジェクトに配属。メンターやプロジェクトメンバーが適宜OJTで教育。
- 試用期間中に Ruby Silver をとってもらう

人は育ってはいたけれ
ども、効率がイマイチ

新人のアサインされる案件の状況は毎回違う！

何を
やるのか

コードの
状態

リリース
時期

チーム
構成

新人のスキルや特性も毎回違う！

知識・経験

興味の方向

性格

コミュニケーション

得意分野

教える側が大変

案件の状況

案件の詳細を
把握するのは大変！

ほかのプロジェクトにいると
教育に参加しづらい

新人の特性

新人教育のたびに
ゼロベースで教え方を検討

教わる内容が不定

アサインされた
プロジェクトの個性や
担当したタスクに
左右される

受け入れチームが大変

https://www.flickr.com/photos/toff78/13006541975

会社が大変

- 受け入れの成否への不安、教育負荷への不安から、採用そのものにネガティブな雰囲気が生じやすい

- メンターや受け入れチームの負荷が高いので手配にとても気を使うし難しい

そこで

新人研修の
仕組みを改善する
有志プロジェクトが
発足！

研修制度を一新

https://www.flickr.com/photos/joemar/1573687605

従来システムを踏襲した点

- メンターを任命する
- 試用期間中に Ruby Silver をとってもらう
- メンティーごとの専用研修Slackチャンネルを作る

変更した点

研修期間を設け、

教育カリキュラムに

沿って課題を

進めてもらう

教育カリキュラムとは

- タスク管理アプリケーションを開発する課題
- 開発環境の構築、設計、個々の機能の実装など全部で24ステップで構成されている

各ステップの学習要素

環境構築

設計する習慣

ツールの
使い方

開発手順

Rails

デザイン

仕様はわざと少し曖昧に

- 現実の開発では、完全な仕様が存在することはほとんどない
- 自分で仕様を検討して決めるプロセスが必要になる仕掛けを入れている
- メンター向けには指導要領を用意

研修の仕組み

マスターデータ

カリキュラム
(ステップ1〜24)

新人ごとに

課題アプリケーション
レポジトリ (GitHub)

○○さん研修専用
Slackチャンネル

Step 1 のPR

Step 2 のPR

・
・
・

メンターが相談にのる

全社の有志も折々にレビュー

入社時のスキルによらず
同じカリキュラムで
研修を行い、
研修が終了してから
実案件にアサインする

スキルのある人は
早く実案件に
アサインされる

初心者はじっくり研修
に取り組める

とても良くなった！

https://www.flickr.com/photos/63259711@N04/5790081866

教える側がラク

- 研修課題が決まっているので、何をやってもらうかを都度考えないで済む
- 過去の研修の経験や資産を活かせる
- 標準カリキュラムを把握するのは案件の詳細を理解するよりずっと簡単
- 誰でも教える側になれるので、負荷がメンターに集中しない

教わる側が安心

- 網羅性のある全社で共通の学習内容
- 開発、PR、レビューのサイクルを通じて、万葉の開発スタイルを、実際にプロジェクトに参加する前に学べる
- 研修終了後、プロジェクトにスムーズに参加できる

受け入れチームが安心

https://www.flickr.com/photos/clickr88/5810111715

会社がラク

- 採用についての社内の雰囲気が明るくなった
- メンターや研修修了後の受け入れチームの選定がしやすくなった

- 注：純粋に研修のみの期間が生じるのでコスト的にはしっかりかかります

制度を変えてみて
はじめて分かったこと

研修のPRレビューは
思った以上に
素晴らしい効果がある！

みんなでレビューできる

- 研修経験者は当然くわしい
- メンター経験者も当然くわしい
- 毎度同じ課題だから、ほかの人も内容を推測しやすく、口をだしやすい
- メンターの負荷が軽減される
- メンティーの作業効率Up

文化のすりあわせができる

- 実案件に参加する前に、開発スタイルや文化のすり合わせができる
- レビューを通じて、人柄や価値観、こだわりなども分かる
- 新人・受け入れチーム双方の安心につながる

第三者も勉強できる

- 先輩社員のレビュー内容をみて知識を身につけたり、レビューの着眼点を学ぶことができる
- これまで学んだことの復習にもなる
- 最新のRailsの仕様を確認する機会も増える

レビュー自体が資産

- レビュー内容は将来の研修の資料として活用できる

新人教育制度は
自律的に
改善ループを回して
カリキュラムを
バージョンアップ

研修終了時/実案件へ
Joinして少ししてから
インタビューを実施

カリキュラムのバージョンアップ

① 研修修了後、メンティーにインタビューを実施

マスターデータ

課題の集合・メンター手引書
（ステップ1〜24）

② インタビュー結果や自分の意見を反映して、メンターが改善案をPR

Pull Request

③ 教育チームで、PRの差分をみて議論

教育MTG

④ MTGの成果を改めてPR、マージ

Pull Request

これまでのバージョンアップ

- 改善ループを通じて「この学習も必要だよね」というフィードバックが寄せられ、**学習内容の網羅性**が高まった
- 学ぶためのおすすめ**教材情報**が集まった
- 研修終了時期が新人ごとに異なり、アサイン計画が立てづらいので、過去にかかった時間の記録をつかって**予想終了時期を計算**するようにしてみている

新人は
自分が受けた
教育制度への
関心が高め

その新人が育って
メンターをすることで
教える効率 Up！
制度改良の効率も Up!

お知らせ

万葉の
教育カリキュラムを
公開します！

https://github.com/everyleaf/el-training

カリキュラムを
作るのは
手間がかかるので
もし良かったら
ご利用ください

PR歓迎です

※採用した新人に
読めてしまうし
会社ごとに
「正解」は違うので
指導要領は
公開していません

まとめ

- Ruby, Rails の受託開発会社としての10年で工夫してきたことを紹介
- 新人研修制度の刷新で採用・教育がスムーズになった
- 課題のPRレビューを全社で行うことで、技術だけでなく、文化、開発スタイル、価値観などを伝えることができる
- 教育カリキュラムを公開中