モブプログラミングのすすめ

@tanukiti1987

今日の発表について

今回はモブプログラミングについて、
その実践の仕方や心構えについて私なりの考えを踏まえ話をします

ただし、それらに正解はなく、**チームにより最適解は異なります**

モブプロに少しでもポジティブなチームの背中をそっと押せれば
幸いです 🙏

せっかくなので一緒にコラボレーションしましょう！

LiveShare URL: XXX

markdown 側にコメントや質問をつけたり、
アンケートページでは、賛同できそうな選択肢に 「👍」を書き足してみたりしてください 😊

モブプログラミングとは

モブプログラミングとは、3 名以上のチームメンバーが同時に同じタスクに取り組むソフトウェア開発手法の一つ

ペアプログラミングとも似ていますが、目的や特徴がやや異なります

ペアプログラミングとの違い

- **コラボレーションの形式**: モブプロは必ずしもエンジニアだけでなくてよい。集合知を活かし、意思決定をスムーズにするなら、POやデザイナーなども含めてよい
- **学習と知識の共有**: ペアプロは 2 人の間で知識共有をする。とりわけ経験が異なるペアには効果が高い。モブプログラミングでは、チーム全体のスキルアップ効果があるやドメイン知識の平滑化などが起きる

ドライバーとナビゲーター

ペアプログラミングでは各役割を以下のように呼びます

- **ドライバー**: 実際にコードを書く人
- **ナビゲーター**: 実装の方向性を検討し、実装に関する意思決定をする人

モブプロではナビゲーターが複数人います

ペアプロには流用できそうなラリーから来た語源ですが、
モブプロの場合、ナビゲーターが複数人いると、誰の言う事を聞いていいか分からなくなりますね 😊

タイピストとその他のモブ

今回は参考資料* にもあったとおり、

- ドライバー → **タイピスト**
- ナビゲーター → **その他のモブ**

と呼ぶことにします

モブプログラミング、良いですか？

ここでアンケート 📝

モブプログラミング、どうですか？

- 積極的に取り入れているよ！
  - A: 🙌
- 取り入れたいが、なかなか実行できないな
  - A: 🙌🙌🙌🙌🙌🙌🙌🙌
- 取り入れてみたが、ちょっと合わなかった
  - A: 🙌
- モブプロとは心の距離が遠くなってます
  - A: 🙌

モブプロと効率の考え方

モブプロと聞くと、全員が同期的に作業をする必要があり、
全員がバラバラにタスクに取り掛かるより
非効率に感じることがあります

ある意味正しい一方で、釈明の余地があります

目指すのはフロー効率の最大化

モブプログラミングで着目しているのは **フロー効率** です。
リソース効率 に着目すると、その点では効率は悪くなります

- **フロー効率**: 仕事の流れを最大限に良くする
- **リソース効率**: リソースを最大限に利用する

なぜフロー効率を重視するのか

市場に価値を届けるスピードに見通しを効かせ、より速く、より臨機応変にプロダクトを育てて行くため

フロー効率を高めるには

- 意思決定待ちによる待ち時間が減らす
- 製造中の製品の問題点の発見を早くする
- 予期せぬ問題や変更に強くする
- ドメイン知識が均一化し、人員の入れ替わりにも強くする

おまけ 1: リソース効率を求めるのはダメなの？

- そこまで白黒つける必要もないし、**適度なバランスが大事**
- リソース効率を最大化するメリットのあるプロジェクトもある
  - 予算や期間が限られている
  - プロジェクト規模が小さい
    - PJ メンバーやステークホルダーが少ない
  - 途中での大きな変更や問題が起きにくいことが予期される

おまけ 2: フロー効率重視と言われても、やっぱり納得しきれない

- リソース効率は個の効率にも直結するのでエンジニアでも体感し易いし、納得感も得やすい
  - 👍: コードを書く時間が多く取れた、PR が多く出せた
  - 👎: 自分のタスクの手を止めて、他の人の相談に乗ってた
- フロー効率はプロジェクトや組織の効率に焦点が当たるため、PM や PO の視点に立たないと体感しにくいカモ 🍆🍆🍆🍆🍆🍆🍆🍆🍆
  - 👍: 予定していたストーリーをリリースできた
  - 👎: 仕掛りのストーリーは多いが、リリースできたものは少なかった

モブプロによる恩恵

- **メンバーのスキルアップ**
- **ドメイン知識の均一化**
  - この領域はこの人が得意。よく知っている。がなくなり、モブのみんなが自信を持って実装等について理解している状態になる
- **レビューが不要になる**
  - 一緒に作業し、懸念点を解消していくので、PullRequest のレビューは基本不要 🎉
- **朝会等での進捗確認の時間が短くなる**
- **リリース後製品の欠陥修正（品質向上）**

モブプロには向かないこと

- **全く知らない言語**のモブに参加して、その言語を学ぶ
- 方針等は決まっており、ただやるだけの**単純作業**
  - データ入力、軽微なリファクタリング
  - この場合はモブしない、やめるというよりは、次のストーリーについてタイピスト以外が議論を進めておくというのもよい

ここまでのまとめ

- モブプログラミングはフロー効率を重視している
- チームが同期的にストーリーに取り組むことで、コーディング自体のスピードが遅くなったり、並列で複数のタスクに取り組むことはできなくなるが、メリットも十分ある
- これらメリットを享受したい場合に、**モブプログラミングおすすめです** と言える！

ここでアンケート 📝

モブプロのメリット・向かないところを話してきましたが、モブプロはみなさんのプロジェクトにとって恩恵を得られそうですか？

- そう思う！
  - A: 🙌🙌🙌🙌🙌
- 今はなんともいえないな！
  - A: 🙌🙌🙌🙌🙌
- モブよりフィットするやり方をすでに取り入れているよ！
  - A: 🙌

どうやってモブプログラミングを始めるの？

一歩踏み出しにくいモブプログラミング

- 一見して非効率っぽく見えるモブプログラミング
  - 上司や客先に咎められないだろうか
- チーム内の知識やスキルの差があるとやりにくそうなモブプログラミング
  - 技術力ある人の足を引っ張らないだろうか
  - コードを書く時間が減り、教えてばっかりになってしまわないだろうか

...

なんか「やろう!」って言いにくい!

実験的に始めて見る

- 先程挙げったメリットを実際に感じられるか、**実験** と位置づけて始めてみよう
- モブプロに参加する人は、**参加したい人だけ**にしよう
  - 3~5 人位を目安とし、参加者はなるべく固定
- 1,2 回の実施ではなく、数週間ほど実施しよう
  - 1 回のモブごとに振り返りをしよう
- うまくいってもいかなくても、参加しなかった人たちにも**感想を共有しよう**

モブプログラミングの主な流れ

1. モブプログラミングで取り組む内容を決める
2. タイピストを決める
3. タイピストは 10 分ごとに交代する
4. 1 時間〜1 時間半くらいで休憩を挟む
5. トータルでも 3 時間でモブプログラミングを終える

取り組む内容を決める

- モブプログラミングでどんな内容に取り組むか、参加者の中で認識を合わせておく
- 抽象的な話よりも、より具体的なスコープで内容を決めたほうが参加者全体での話がまとまりやすい
  - 👍: モデルに XX というメソッドを追加する
  - 👎: YYY というユーザーストーリーを進める

おまけ: 認識を合わせやすいタスク分解

- ユーザーストーリーを実現するためのタスクを予め参加者同士で分解しておくと認識が合いやすい
  - YYY というユーザーストーリーを進めるには...
    1. テーブルを作る
    2. モデルを実装する
    3. ...
- 認識を合わせるという観点では、モブプログラミングを **TDD** で進めて行くのもオススメ

タイピストの役割

- その他のモブが言ったことを理解する人
  - 理解が追いつかなければ説明を求めること
- その他のモブの言ったことをタイピングする人
- 勝手にひっそりとコーディングしてはいけない
- もし 10 分間、実装方針が固まらず手が動かなかった場合は次の 10 分も継続する

その他のモブの役割

- 実装方針を議論し、実際に進む方向性を決める
- タイピストに指示を出し、コードを書いてもらう人たち
- タイピストがうまく理解できていなかったり、手が動いてなければ、より詳細に指示を出すようにする

おまけ: タイピスト苦手・疲れる問題

- 人にコーディングしているのを見られるのが苦手だ。という場合にはそれをチームに明らかにしよう
  - そして、無理にタイピストにならなくても大丈夫
- タイピストをしていると、なんか書かないと。。！かっこよく書かないと！となりがちだが、落ち着こう
  - 基本的にはその他のモブが指示を出してくれるし、困ったら聞けば良い

全員が手詰まりしたとき

- 全員が頭を抱え、実装が進まず、調査が必要になる場合があります
- 一旦時間を決めて、散り散りに調査を行おう
  - ドキュメントを読んだり、デバッガーで自分の思うようにデバッグしてみるなど
- 決めた時間にしっかり戻ってくるようにすること
  - わかったこと、わからなかったことをモブに共有する

休憩をきちんと取ろう

- 1 時間〜1 時間半に一度は必ず休憩を取ろう
- タイピストもその他のモブも集中が続き疲れるけど、ハイになって疲れに気が付きにくい
- モブの最中に途中で抜けるときには、声を掛け、いつ頃戻るかを伝えておこう
  - 途中で抜けたときには、自分が抜けていたときにどういう進捗だったかを聞くのはやめよう

振り返りをする

- モブプログラミングが一区切りついたら、振り返りをしよう
  - よかったこと・改善すべきことをドキュメントにまとめて置くと、より自分たちの手に馴染んだモブプログラミングが行えるようになる

おまけ 1: モブの中での初心者メンバーの振る舞い

- 積極的にタイピストをやろう
  - 製品が作られている様を眺めるより、実際に手を動かしたほうが理解が深まりやすい
- わからないところは奥せず質問しよう
  - 自分の質問と同じ疑問を持っているメンバーの助けになったり、回答をするメンバーの理解を深めることができる
- リアルタイムで質問しにくいときは、質問を書き溜めておこう
- 何らかの恐怖心があるなら、打ち明けておこう

おまけ 2: モブの中でのエキスパートメンバーの振る舞い

- タイピストはあまりやらないようにしよう
  - その他のモブの指示が最低限で、どんどん作業が進んでしまうと、周りのメンバーがついて来れなくなる可能性がある
    - でも、周りのメンバーはどんどんコードが書かれていくので、わかったような気になってしまう
    - タイピスト交代のときに支障をきたしてしまう
- 知識の溝を感じたら積極的に埋めにいこう

モブプロを継続していくには

- モブプロをする時間を決めよう・声を出そう
- 振り返りが大事

最後に

今回はモブプログラミングのメリットや
進め方について話をしてきましたが

...

チームの雰囲気が何より大事

ここに尽きます

チームの雰囲気が何より大事

- 周囲に忖度せず、成果に対してその良し悪しを評価できる雰囲気
- 知識やスキルの差があっても、認めあえる雰囲気
- 個ではなく、チームとして以下に組織に貢献するぞという雰囲気

こういったものがモブプログラミングをより良く、楽しく、
効率よく進めていくコツになります

おまけ 1: ポジティブな発言を心がけよう

「うわぁぁぁぁ！」「げっ！？」「はぁ。。。」

と言ったような発信は、不用意にタイピストを不安にさせてしまうことがある
決してタイピストに向かっての発信ではなかったにしても、
タイピストがそう思うかはタイピストとの関係次第

ため息をつきたくなっても、一旦落ち着こう
（関係性によっては全然アリなこともあります）

おまけ 2: ポジティブだけじゃやっていけないときもある

納期が差し迫っているときにモブの中に作業ペースの遅い人がいて、苛立ってしまうときもある

そんなときは、今このときだけ例外的にペースを速めることを明言しておこう
静かにストレスを貯めることも無いし、嫌な圧を感じる必要もない

ここで最後のアンケート 📝

自分のプロジェクトでもモブプロ使いたいですか？

- 使ってるよ！
  - A: 🙌🙌
- 使いたい！
  - A: 🙌🙌🙌🙌🙌🙌🙌🙌🙌
- 今はいいかな！
  - A: 🙌

さぁ、はじめてみよう！

（終わり）

参考資料

モブプログラミング・ベストプラクティス ソフトウェアの品質と生産性をチームで高める