アジャイル開発の生産性とは

~リードタイムのメトリクスを 3 階層に分けたときに見えた生産性の指標~

What is agile development productivity?

- Productivity indicators seen when lead time metrics are divided into three layers-

合同会社 DMM.com 石垣 雅人
DMM.com LLC. Ishigaki Masato

Abstract About Agile productivity.

1. はじめに

近年，様々な分野においてソフトウェアの重要性が増してきている.
エンタメから社会インフラまで, 今ではソフトウェア存在していない領域のほうが少ない方である. 当然ソフトウェアが存在するということは, そこには組織があり, 開発を行うエンジニアだけではなく, デザイナー, マーケター, 営業など様々なメンバーがいる. 最近では, 刻一刻と変化する市場の変化やユーザニーズに合わせるために，継続的にソフトウェアを変化させながらユーザーへデリバリーしていくアジャイル開発が主流となっている.

アジャイル開発を成功させるには, 限られた予算においてイテレーティブ（反復）で, レジリエンス（柔軟な）な開発プロセスが必要になる. あらゆる施策を実行する中においては, できるだけ速く, 品質の高い機能開発を行い, ユーザーフィードバックをもとにどれだけ柔軟に対応できるかがプロダクト開発の現場には求められるわけだが, そこでよくボトルネックとなるのは, リードタイムの問題である.「開発が遅くてリリー スができない」「他チームとのコミュニケーションの問題で手戻りが発生した」「意図したものが開発されていない」など課題は上げればきりがない. こうした問題に対してどこに問題の根源があるのか, 何にどのぐらい時間がかかっているのかを追求して対策をするべきだが，定量的にプロダクト開発のプロセスを可視化している現場は少なく, 詳細な洗い出しが困難な現場は多い. それらの問題に対して定量的ではなく定性的な判断になるケースが多く見られる.

本研究では、それらの問題に対してプロダクト開発全体の定量化, 可観測性の向上に挑んだ取り組みである. 一方、こうしたリードタイムに向き合うことは, リードタイムを短縮し開発プロセスを削減するだけではなく, 最終的には生産性を上げる取り組みにつながる. すなわち, アジャイル開発における生産性とは何かを考えるきっかけにもなる.

リードタイムの可視化について 3 つのレイヤーにわけて考える. 第 2 章では, プロダクト開発におけるリードタイムを 3 つのメトリクスにわけたときの可視化のアプローチ方法を述べる. 第 3 章では, 可視化した結果見えてきたリードタイムのミクロとマクロな情報をどう融合して全体最適化を行っていくか, 役割によって見るべきメトリクスの変化について考察していく.

合同会社 DMM.com 石垣 雅人 部長
DMM.com LLC. Ishigaki Masato. Manager
東京都港区六本木三丁目 2 番 1 号住友不動産六本木グランドタワー24 階 ishigaki-masato@dmm.com

【キーワード】 アジャイル開発,ソフトウェア工学,リードタイム,生産性

2. 研究内容

プロダクト開発のリードタイムを可視化し削減しようと考えるときに開発メンバー,とりわけエンジニアだけが開発を早くすれば価値提供が早くなるとイメージする人は多い.しかし,いくら開発チームが改善しても,ユーザーへ価値を届けるスピードに変化がないことに違和感を覚えた.プロダクト開発の現場をよく観察してみると,アイディア（仮説）が生み出され、成果物としてリリースされるまでには様々なチーム内外のメンバーが参加しており,プロダクトの意思決定に関わっているすべてのメンバーがリードタイムを意識していかないとスピードという面では加速していかない.組織のセクショナリズム構造を理解し,それぞれにあったリードタイム指標を定め,可視化し,削減することがプロダクト開発全体のリードタイムを局所最適化ではなく,全体最適化につながると仮説付けた.

今回,その仮説を検証するために3階層にリードタイムを分けるアプローチをした.以下に概要を示す.

- Layer 1.ソースコードレベルでのリードタイムの可視化
  - Pull request, commit, comment から生産性の健全性を可視化
  - 想定としては主に開発者（エンジニア）がこの指標を見る
- Layer 2.開発フレームワークに合わせたリードタイムの可視化
  - Velocity, Cumulative flow, Control Char,累積フローにおけるチーム全体の生産性
  - 成果物を作るエンジニア、デザイナーが見るべき指標として想定
- Layer 3.プロダクト開発全体のリードタイムの可視化 -
  - 仮説が出てからリリースまでの全体リードタイムを網羅した部分の可視化
  - プロダクト開発に関わる全員が見るべき指標として想定

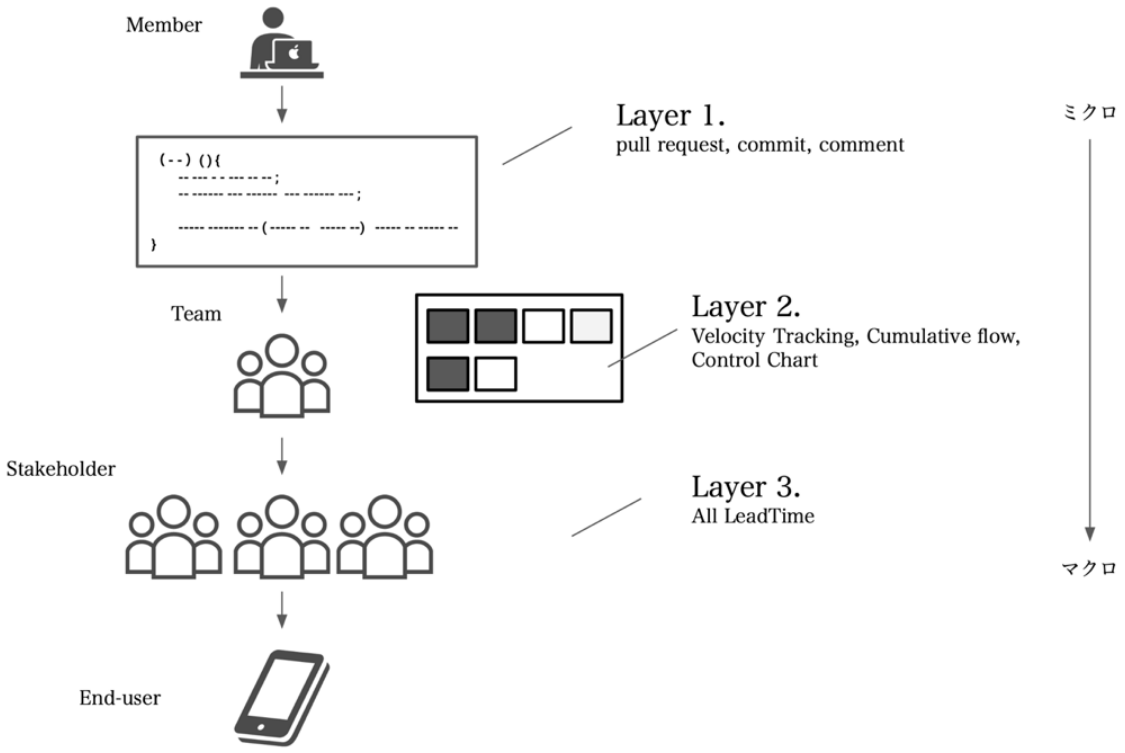


図 1

2.1 メトリクスの設計思想

プロダクト開発の流れを意識してみる.

Layer 1 はエンジニアが中心となる開発目線.開発作業とともに成果物を納品するために必要なコードレビューなどを通じてマージされる部分までを観測する.Layer 2 では,チームでの価値を出すことを意識する部分.エンジニアもそうだがデザイナーの作業なども含めた部分までを観測する.Layer 3 は,プロダクト機能を届けるにあたってチーム内外問わず全員を対象としたリードタイムを観測する.ミクロな視点とマクロな視点を Layer を通して連続的に可視化できるように設計している.それぞれ Layer が,どういったアプローチでメトリクスを表出化させていったかについてはこのあと述べていくが,計測にあたって工夫して点をはじめに 2 つ述べておく.

1 つ目は,できるだけ初動の集計するコストを下げることである.2 つ目は運用コストでメトリクスは一度可視化したら終わりではなく継続的に可視化していく必要がある.そのランニングコストに関しても積み重ねによっては大きな稼働時間がかかるため工夫した.

そのため,本研究では Layer 1 であればコード管理の「GitHub」から.Layer 2 はプロジェクト管理の「ZenHub」から.Layer3 は一部手動もあるが勤怠・工数管理ツールの「TeamSpirit」から集計した.

2.2 Layer 1.ソースコードレベルでのリードタイムの可視化

では,具体的に各 Layer のメトリクスについて紹介する.まずは,Layer1 についてである.この Layer はもっとも個々の活動に近くミクロな視点でのリードタイムの計測となる.そのため,今回はエンジニアに対象を絞った上でメトリクスの集計対象とした.メの例としては以下のようなものがある.

a. 特定期間中のプルリクスト作成数
b. 平均プルリクストマージ時間
c. プルリクエスト作成からレビューまでの平均時間
d. プルリクストへの平均コメント数

2.3 Layer 2.開発フレームワークに合わせたリードタイムの可視化

ここでいう開発フレームワークとはアジャイル開発を行っていることを前提にした,スクラムや XP といった枠済みのことである. ここではスクラムを導入したケースを述べていく.

スクラムは,イテレーション（スクラムでいうスプリント）といった一定の間隔があり,そのタイミングでチームとして振り返り,次のイテレーション期間で開発するべき成果物を確認する.ここにメトリクスを仕込み計測していく.具体的に使うものは,Cumulative flow, Control Chart の 2 つとした.それぞれの意味を以下で述べるが具体的なグラフのイメージは第 3 章で詳しく述べていく. これらの指標については,プロジェクト管理ツール（JIRA や ZenHub）を利用していればおおよそ自動で集計され提供されている事が多い.

項目	意味
Cumulative flow	累積フロー.時間ごとのレーンの移動時間
Control Chart	特定の時系列をローリング平均と標準偏差を使った完了時間の予測

2.4 Layer 3.プロダクト開発全体のリードタイムの可視化

最後にプロダクト開発全体を見る.ユーザーに価値を届けるには成果物を開発すれば終わりというプロセスは少ない.成果物の合意を取るためにミーティングを行ったり,スクラム開発を 1 週間で行っていれば, 週 1 日は次のスプリント計画や振り返りなどで開発する時間はなくなる.本来であれば開発以外にどのようなことをどのぐらいの時間をかけて行っているかを一覧で可視化することを目指す

が，本研究では初手として開発している時間と開発以外の時間を有効稼働率として計測して可視化することを目指した.有効稼働率とは文字通り有効に稼働できている時間のことだが,ここでは開発者が開発に当てられている時間のことを指す.計測方法については勤怠・工数管理ツール（TeamSpirit）を使い「今日 1 日,何にどのぐらいの工数をかけたか」をタスクコードをもとに勤怠と工数を紐付けて入力してもらっている.メトリクスの例を出すと以下のようになる.

a. 有効稼働
   (ア) 開発比率（%）
      ① 要件定義比率（%）
      ② 設計作業比率（%）
      ③ 開発作業比率（%）
      ④ テスト作業比率（%）
   (イ) 保守・運用比率（%）
   (ウ) プロジェクト管理比率（%）
b. 開発外比率（%）（非有効稼働）

基本的には,有効稼働率と非有効稼働率（開発していない時間）を見ていく.有効稼働率については,複数プロジェクトを行っている場合にはブレークダウンしていきプロジェクトごとに要件定義→設計→開発→テストといったプロセスを見ても良い.既存プロダクトを抱えているのであれば,チーム外からの依頼対応などを保守・運用比率として算出したり,Issue の作成や WBS を引く作業といったプロジェクト管理比率も出しても良い.非有効稼働率（開発していない時間）については,MTG や 1on1 といった開発とは関係ない作業のときに開発外比率として工数入力してもらう.

この集計方法のデメリットとしては他の Layer に比べて労力コスト（LOE）が若干高めになってしまう点にある.ただ,勤怠管理と連携しているため必然的に提出しなければいけないものの中に工数出しが含まれているため,できるだけ労力は避ける工夫をしている． この Layer 3 の取り組みについては本研究の一貫としての取り組みというよりかは組織的な部分で DMM.com LLC が実施していることである.

3. 結果と考察

では,Layer1〜3 のメトリクスを可視化したことでの効果について述べていく.

3.1 Layer 1 の結果と考察

まず,Layer 1 について.さまざまなメトリクスがある中で,例として「平均マージ時間」を集計した図 2 を示す

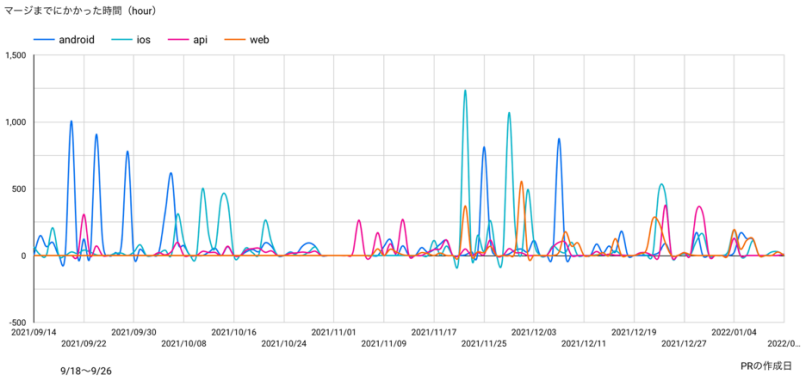


図 2

横軸で日付,縦軸でマージまでの時間(Hour)を示しており,デバイスや領域(Android,iOS,Web,API)ごとにグルーピングしている.これを見ただけでも改善のヒントは隠されている.

前提としてプラクティスとして開発がスピーディに行われるチームのプラクティスとしてプルリクのマージ時間が短いとされている.理由は,ブランチの生存時間が短いことでマージに伴うコンフリクトを起こりにくくしたり,開発単位を小さく保ち頻繁にPushすることでその都度CI/CDを実行することで,自動テストや脆弱性診断などのフィードバックを素早く得られる.また,DORA（DevOps Research and Assessment）が公開しているレポートでも,パイパフォーマンスチームの共通点としてブランチの短命化が挙げられている.こうしたブランチ戦略と照らし合わせてグラフを見ると,ビジュアライズからも理想の状態（正しいグラフの状態）は以下のようにわかる.

- 縦軸の高さが全体的に低いこと
- 縦軸の波が落ち着いていること

つまり,平均マージ時間だけを切り取ると基本的には波が落ち着いていることと,それが連続的に実施されていることが重要となる.図2で言えば,比較的「API」の波は安定しているが「iOS」や「Android」に関しては,所々波が荒ぶっているのがわかる.こういったところが改善ポイントとなり,比較的チーム方針を決めることですぐに実践できる部分である.「API」と「iOS」「Android」では、月平均のマージ時間が全体を通して524時間の差があった.こうした課題に対して静的解析ツールを導入してソースコードのレビュー時間を短くしたり,チームとして優先的にレビューを促進することで約100時間マイナスされた例もある.そうなるとユーザーの価値提供が100時間（約4日間）早くなるということである.仮にその機能が1日100万円の売上を上げるものであれば,400万円の収益増という風にも考えられる.

こういった形でさまざまなメトリクスをわかりやすいビジュアライズで示すことでチームメンバーにもわかりやすく伝えられ,すぐに改善アクションへと繋げられる.

3.2 Layer 2の結果と考察

続いてLayer 2について見ていく.ここではプロジェクト管理ツールを使っていることを前提にControl Chartによるローリング平均（Rolling average）による改善例を中心に見ていく.ローリング平均とは特定の時系列のサイクルタイムのことである.

開発のフローとしてSprintBacklog→ Doing → Review/QA → Closedといったパイプラインのレーン設計があったときに,どのぐらいのサイクルで回っているかを可視化できる.例えば,1週間のイテレーションで開発しているチームであれば,SprintBacklogに置かれたIssue郡は1週間かけてClosedまで行くのが正しいため、ローリング平均は7.0 daysとなるはずである.

また,全体のパイプラインを見るのはもちろん,Doing→Review/QAに行くまでのサイクルタイム,Review/QA→Closeに行くまでのサイクルタイムといった細かく見られことも重要となる.例として,あるチームの直近3ヶ月間を集計期間としたローリング平均を見ていく.

1. SprintBacklog→ Doing → Review/QA → Closedまでのサイクルタイム
   (ア)ローリング平均 ： 8.2days
   (イ)所感 ： チームは1週間のイテレーションを回しているので,8.2daysでサイクルしていることを考えると1週間を過ぎているので何かしら課題を抱えていることがわかる.

図 3

2. Doing → Review/QA までのサイクルタイム

(ア) ローリング平均 ： 2.5days

(イ) 所感 ： 作業を開始(Doing)してから 2.5days で実装が完了し多くはコードレビューへ依頼が行われているため開発速度自体に課題はなさそうに見える.

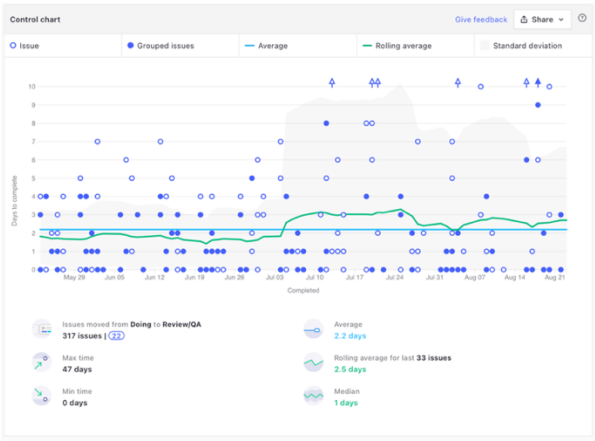


図 4

3. Review/QA → Closed までのサイクルタイム

(ア) ローリング平均 ： 3.6days

(イ) 所感 ： いわゆるレビュー中になってから Close するまでに 3.6days かかっている.チームの内情にもよるが,ここの改善幅は大きそうに見える.

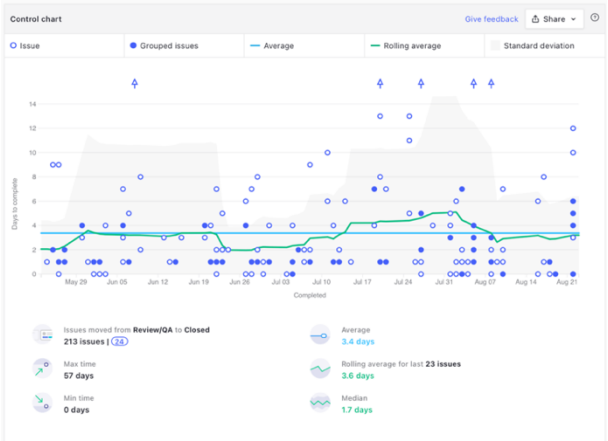


図 5

Review/QA のサイクルタイムに課題感がありそうだと分かれば,Cumulative flow（累積フロー）を見てみる.図が 6 同じ期間（直近 3 ヶ月）の Doing と Review/QA の累積フロー図だが,Doing の個数に対して,常時 Review/QA のレーンに多く Issue が残っているのがわかる.図 6 の例では,Doing の Issue が 4 つに対して Review/QA が 20 issues も積み上がっていることがわかる.ここは大きな改善ポイントになるだろう.

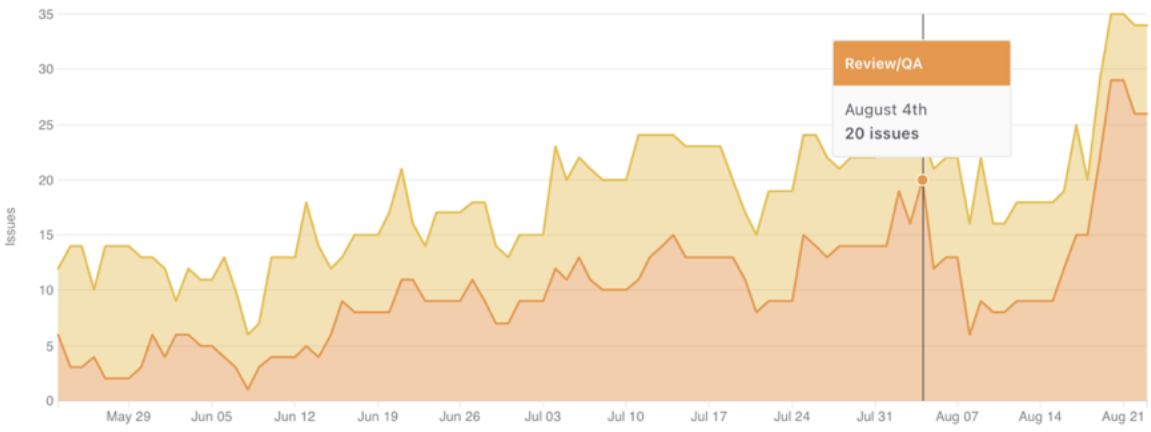
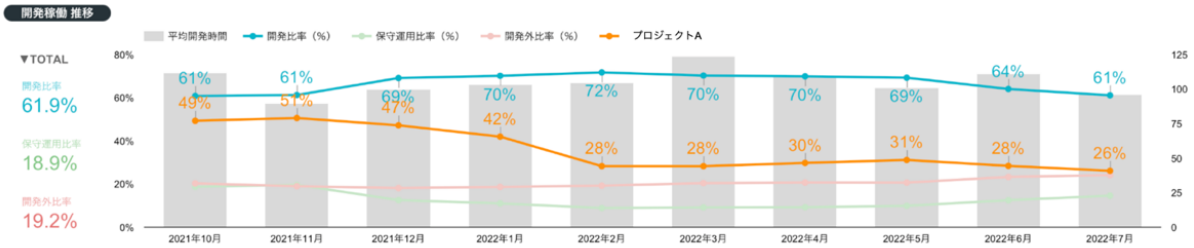


図 6

こうした形で,Control Chart や Cumulative flow を使ってフローを可視化することをお薦めしている.それに対しての改善方法や実態のすり合わせはチームごとに事情が違うが、可視化することでそこに関係してくるメンバー,例えば成果物をレビューするのがエンジニアだけではなくデザイナーや営業などがいるケースも多く,その全員に問題意識と改善幅を定量的に提示できることがメリットとして上げられる.

3.3 Layer 3 の結果と考察

最後に Layer 3 について見えてきた部分を述べる.有効稼働率をキーワードに勤怠管理ツールから算出したものについて,Google Data Studio でビジュアライズした例を紹介する



雇用区分別 開発比率

雇用区分	2021年10月	2021年11月	2021年12月	2022年1月	2022年2月	2022年3月	2022年4月	2022年5月	2022年6月	2022年7月
委託	61.28%	60.29%	70.31%	74.44%	80.63%	79.75%	81.57%	80.95%	76.6%	71.65%
社員	60.63%	61.96%	68.41%	67.16%	65.38%	64.2%	61.16%	60.69%	55.91%	53.34%
総計	**60.88%**	**61.31%**	**69.2%**	**70.22%**	**71.8%**	**70.34%**	**69.98%**	**69.35%**	**64.11%**	**61.18%**

	詳細区分	エンジニア	総計
プロジェクトA	開発	3,424.01	**3,424.01**
	設計	760.26	**760.26**
	プロジェクト管理	560.92	**560.92**
	テスト	80	**80**

図 7

このデータは,直近 10 ヶ月の有効稼働率を示したものだが,上段の部分を見ると,大きく分けて開発比率が 61%,開発街比率（非有効稼働率）は 19.2%となっている.そこまで悪い数値ではないように思えるが,中段の雇用区分別に見ると社員は 55%〜67%の間となっている.約 4 割近くは開発以外の作業をしているため,ここは改善できそうに思える.下段は特定プロジェクトの開発工程別の時間である.ここについては細かい数値というよりかはプロジェクトの特性における傾向が分かれば良い.例えば決済基盤といった冪等性が必要でセキュアな要件では設計に大きな時間（760h）がかかるといった具合である.これを次に同じようなプロジェクトが来たときの参考にもなる.

4. まとめ

まとめとして,それぞれの Layer の関係性と,当事者意識について考えてみる.Layer が深くなればなるほど抽象度が上がってくるわけだが,実は一本の線で繋がっているようにも見える.どうしても組織の中で役割が違ったり職種が違うとセクショナリズムで階層化していくため,たとえば営業は「開発プロセスは自分には関係ない」といった思考になりがちだが,営業先のクライアントの要求吸い上げ→要求定義→要件定義→開発→リリースと考えれば,点ではなく線で考えられ,開発チームへの要求の伝え方,伝えるタイミングひとつでプロダクト開発のリードタイムは変わってくる.当然逆も然りである.最後に本研究のこれからの課題については,Layer 3 の部分をより細かく柔軟にビジュアライズしていきたい.エンジニアやデザイナーといった成果物を作るにあたって近くにいるメンバーについてはコミュニケーションを蜜に取りながら改善はしやすい.一方,リードタイムの多くはそこから外れたステークホルダーとの合意や調整で大きな時間をかけていることが多い.そのため,本研究では有効稼働率といった側面でしか可視化していないが,実際に VSM（Value Stream Mapping）などのソリューションを使いながらプロダクト開発における仮説検証の開始と終わりをきちんと可視化していきたいと考えている.

参考文献

[1] https://docs.github.com/ja/rest

[2] https://www.zenhub.com

[3] ZenHub, "Use Control Charts to Review Issue Cycle/Lead Time",https://help.zenhub.com/support/solutions/articles/43000300345,2022

[4] https://www.teamspirit.com/ja-jp/

[5] https://datastudio.google.com/u/0/