● 第1章 明日までに改善して！と言われる重圧

1-1 エンジニアが感じる「開発生産性」という言葉の重圧

突然の重圧

午前7時、湊太郎のスマートフォンが振動した。Slackの通知音が今日も一日の始まりを告げている。

ベッドから起き上がりキッチンへ向かう。窓際の観葉植物に水をやりながらスマートフォンのSlackアプリを開く。パキラの葉はまだ青々としている。

#general チャンネルにはすでに数件のメッセージが届いている。夜勤の同僚からの報告、今日の会議のリマインダー、そして昨日の夜中に飛んできた緊急対応のメッセージだ。

「またバグ対応か…」

湊は今日のタスクを頭の中で整理する。昨日から引き続きのバグ修正、新機能の開発、そして来週のリリースに向けた準備。手帳に書き出したタスクリストを見るとすでに今日だけで10件以上の項目が並んでいる。

部長からの呼び出し

オフィスに到着しMacBookを開いてVSCodeを起動する。昨日の作業を続ける準備をしていると突然Slackの通知音が鳴った。

田中部長 が **#general** でメンションしてきました

> *湊君、ちょっと話があるから会議室A-1に来て*

湊は緊張しながら会議室A-1に向かった。

「開発生産性が低い」という重い言葉

田中部長はため息をついた。エンジニア出身で管理職歴8年、現場と経営陣の板挟みは慣れたものだが、今日は特に疲れているように見える。

「湊君、率直に聞くけど、うちの開発って遅くないか？」

「遅い…ですか？」

「うん。他社と比べて開発生産性が低いんじゃないかと思ってさ。僕もエンジニアだったからわかるんだけど、経営陣からは『数字を出せ』と言われるんだ。なんとかしてくれないか？」

田中部長の言葉に湊は困惑した。経営陣から数字を求められているというが、そもそも開発生産性をどう数値化すればいいのか。エンジニアと経営層の間で、開発生産性という言葉の意味が共有されていないのではないか。

湊の頭の中は疑問でいっぱいになった。

開発生産性って何？どうやって測るの？何をどう改善すればいいの？経営陣が求めている「数字」と、エンジニアが考える「生産性」は同じものなのか？

「えーっと…具体的にはどのような指標で測られているんでしょうか？」

「それが問題なんだよ。数字で示せないから経営陣からも『もっと効率化できないのか』って言われるんだ。湊君なら何か良いアイデアがあるんじゃないかと思って」

湊は返答に困った。確かに最近開発が思うように進まないことは感じていた。でもそれを「生産性が低い」と言われても何をどう改善すればいいのかわからない。

「申し訳ありません。すぐに改善策を考えてみます」

「頼むよ。来月の経営会議で報告しないといけないから何か具体的な数値で示せるものを準備してくれ」

困惑が伝染する

会議室を出て湊は自分の席に戻った。同僚の山田健一が心配そうに湊を見ている。5名のチームメンバーが座るエリアに戻ると他のメンバーも視線を向けてきた。

リーダーになって3ヶ月。こういう話をチームに伝えるのはまだ慣れない。

「どうした？部長に呼ばれたって聞いたけど」

山田の声に他のメンバーも作業の手を止めた。

12年のエンジニア経験を持つテックリードの山田は、いつも冷静だ。湊は深呼吸してチーム全員に向き直った。

「みんなちょっと時間もらえる？部長から開発生産性向上について相談を受けたんだ」

チームの若手エンジニア、佐藤が不安そうに聞いた。声が少し震えている。

「開発生産性向上...ですか？また何か問題があったんですか？」

湊は首を横に振った。

「問題というより、経営陣から開発部全体の生産性について指摘があったらしい。で、部長が僕に何か改善策を考えてほしいって」

「山田さん、開発生産性って何だと思います？」

山田が腕を組みながら言った。

「うーん、コード量？リリース頻度？正直、『生産性』の定義は時代とともに変わってきた。昔はコード量だったが、今は...よくわからないな」

山田は少し間を置いて続けた。

「コード量で測れば、無駄なコードを書けば数字は上がる。リリース頻度で測れば、小さな変更を細かくリリースすれば数字は上がる。でも、それって本当に生産性が上がったことになるのか？他社と比較しても意味がないと思うんだよね。自分たちの過去と比べて改善しているかが重要なんじゃないか」

山田も困惑している様子だった。

「僕も同じことを考えていました。確かに最近開発が思うように進まない気がするんですがそれをどう数値化すればいいのか...」

佐藤がディスプレイから目を離して言った。

「でも湊さん、うちのSaaSって顧客からの機能追加要望が多いじゃないですか。勤怠管理モジュールだけでも先月3件、経費精算モジュールも4件。それ全部対応しながら新機能も作ってって、正直手一杯ですよ」

佐藤が続けて言った。

「僕、プロジェクト管理モジュールの改修やってるんですけど仕様がコロコロ変わって...。これ以上スピード上げろって言われても」

湊はチームメンバーの疲れた表情を見て胸が痛んだ。リーダーとしてこの状況をどう改善すればいいのか。
リーダーとして自分で決めなければ。また一人で抱え込もうとしている。

PMからの緊急案件

その時Slackの通知音が再び鳴った。今度は飛鳥さくらPMからのメッセージだった。営業出身で3年前にPMに転身した飛鳥は、いつもこのペースだ。

飛鳥さくら が **#project-urgent** でメンションしてきました

> *緊急案件が入りました！来月末リリース必須で相談したいです。*
>
> *詳細は後で共有しますが、とりあえず全員で対応をお願いします。*
> *この案件は売上に直結する重要な機能なので、絶対に遅れるわけにはいきません！*

湊の焦りが最高潮に達した。

今でも手一杯なのに…でも生産性を上げろと言われたばかりだし…

湊は飛鳥PMのメッセージを見て、さらに混乱した。営業出身のPMは「売上に直結する重要な機能」と言い、経営陣は「数字を出せ」と言う。一方で、エンジニアの自分たちはコード品質や技術的負債を気にしている。それぞれが「開発生産性」という言葉に込める意味が違うのではないか。レイヤーごとに価値基準が異なることを、湊はこの瞬間に感じ始めていた。

湊の決断

チームメンバー全員が湊を見ている。リーダーとしての判断を求められている。

山田が冷静に言った。

「とりあえず飛鳥さんと話をして現実的なスケジュールを組むしかないんじゃないか？まず詳細を聞かないと判断できない。湊、焦る気持ちはわかるけど、一つずつ整理していこう」

佐藤が心配そうに言った。手がキーボードの上で止まったまま、何度もため息をつく。

「でも湊さん、今のプロジェクトもあるし本当に来月末なんて間に合うんですか？経費精算モジュールのバグ修正もまだ残ってるんですよね。優先順位どうしますか？」

湊は深呼吸した。チーム全体が不安を抱えている。リーダーとして冷静に判断しなければ。

「わかった。まず飛鳥さんと要件を詰めて見積もりを出そう。それからチームで作業分担を相談する。今抱えてるタスクの優先順位も含めて明日の朝イチでミーティングしましょう」

山田が頷いた。

「それがいいな。慌てて動いても余計混乱するだけだ」

湊は頷いたが内心では不安が募るばかりだった。生産性向上という漠然とした課題と具体的な緊急案件。どちらも重要だがどう両立すればいいのかわからない。

湊は深く息を吸い今日のタスクリストを見直した。しかし新たに追加された「生産性向上の検討」という項目がリストの一番上に赤い文字で書かれているのを見て改めてプレッシャーを感じた。

解説：「開発生産性」という言葉が持つ構造的問題

ストーリーで描かれる「重圧」

なぜ「開発生産性向上」が重圧になるのか。それは**何が問題かが関係者で共有されていない**ときによく現れる構造的な課題とつながっています。

- **突然の要求による混乱:** 経営層から「開発生産性を上げろ」と言われるが、具体的に何をどう改善すればいいのかわからない。そもそも「開発生産性」とは何なのか、その定義すら共有されていない

- **定義の不在:**　「開発生産性」という言葉の意味が関係者間で共有されていない。エンジニアが見ている指標（リードタイム、デプロイ頻度）と経営層が求めているもの（計画信頼性、予測可能性）が根本的に異なる
- **多様な価値基準の衝突:** ステークホルダーごとに「生産性」の意味が異なる。経営層は売上・コスト削減を、PMはユーザー価値を、エンジニアはコード品質を重視し、レイヤーごとに価値基準が異なることによる対立が生じている
- **測定方法の不在:** 「数字を出せ」と言われるが、どう数値化すればいいのかわからない。コード行数やリリース頻度では本質を捉えられず、適切な測定方法が見つからない

この重圧の背景には、**「開発生産性」という言葉自体が関係者間で共有されていない構造的な問題**があります。本質を理解する前に、まず「何が問題なのか」を関係者で揃えるところでつまずいているのです。

開発生産性の本質的な定義

湊が直面した状況は多くのエンジニアが経験する典型的な問題です。

「開発生産性」という言葉が重圧になる根本的な理由は、開発生産性が何を指すかが関係者間で共有されていないことにある。

開発生産性の正体は、開発組織の速度や件数ではなく、「事業責任者やPdMが『この組織の言葉を信じて意思決定できるか』という信頼性（Predictability / Reliability）である。または、事業責任者やPdMから開発組織への満足度である。

つまり開発生産性の本質は「開発組織が『いつまでに何ができるか』と語った約束が、どれだけ信頼できるか」なのです。

測っているものが違う

開発組織と事業責任者・PdMでは、測っている指標が根本的に異なります。

開発組織が見ているもの

- リードタイム、デプロイ頻度、変更失敗率、MTTR
- これらは開発組織の内部状態・流動性・安定性を測る指標

事業責任者・PdMが見ているもの

- 計画信頼性・予測可能性
- 欲しいタイミングで欲しいものが出るか
- 計画は信じて立ててよいか
- 意思決定の前提として使えるか

Four Keysなどの指標は開発組織の内部安定性を測りますが、事業計画の信頼性は直接測りません。これが、Four Keysが改善しているにもかかわらず、事業責任者・PdMには「良くなった実感」がないギャップの原因です。

なぜ「開発生産性」は重圧になるのか

従来の測定方法では、開発生産性の本質を捉えられません。

- **コード行数**: 書けば書くほど生産的？でも品質は？
- **リリース頻度**: 速ければ速いほど良い？でもバグは？
- **ストーリーポイント**: チームごとに基準が違う
- **付加価値が提供できた数**: どうやって計測するの？

さらに、ステークホルダーごとに「生産性」の意味が異なります。

- **経営層**: 売上・コスト削減・市場投入速度
- **PM**: ユーザー価値・機能完成度・スケジュール遵守
- **エンジニア**: コード品質・技術的負債・持続可能性

この認識のズレが、湊が感じた「何をすればいいのかわからない」という重圧の正体です。

本質的な問題

遅れること自体が問題なのではない。「なぜ遅れたのか」「次はどう変わるのか」を説明できないことが問題です。これは能力不足ではなく、構造的に説明可能な情報が欠けている状態なのです。

この状況で最初にすべきこと

まずは「開発生産性」という言葉の定義を関係者間で合意することから始めましょう。

- 何のために生産性を向上させるのか？
- どの指標で測定するのか？
- 短期的成果と長期的健全性のバランスをどう取るのか？
- 「なぜ遅れたのか」「次はどう変わるのか」を説明できる情報があるか？

1-2 スピード優先がもたらす負のスパイラル

急がば回れの逆転

スピードを優先した結果、品質を犠牲にする流れが始まっていた。

仕様も決まらないまま始まった開発

午後2時、急遽開催された要件定義ミーティング。会議室には湊のチーム5名全員と飛鳥PMが集まっていた。

飛鳥PMが資料を配りながら早口で説明を始めた。営業出身らしい勢いで話が進む。

「今回の機能はユーザーの行動分析機能です。具体的には...」

山田が資料を見ながら眉をひそめた。

「飛鳥さん、この分析機能って勤怠管理モジュールとの連携が必要ですよね？既存のデータベース構造だと結構大がかりな改修になりそうですが」

飛鳥は少し焦った様子で答えた。営業時代の癖で、スピード重視の判断が出る。

「その辺りは後で詳細を詰めるので、とりあえず大枠で進めてください。競合他社が似た機能を出す前にリリースしないと。営業時代の経験から言うと、市場を取られたら取り返すのは大変なんです」

佐藤が心配そうに聞いた。

「後で詰めるって、いつ頃までに仕様が固まる予定ですか？来月末リリースですよね？」

「遅くとも2週間後には...」

佐藤が小声で呟いた。

「2週間後に仕様確定して、実装とテストで2週間...かなりギリギリですね」

飛鳥の説明を聞きながら湊は違和感を覚えていた。詳細な仕様がまだ決まっていないのに来月末リリースというのは現実的ではないのではないか？

「飛鳥さん、すみません。具体的な仕様についてもう少し詳しく教えていただけますか？特にどのデータをどう分析するのか、そこが決まらないと設計できないです」

「あー、詳細は後で決めるからとりあえず大枠で進めて。とにかく早くリリースすることが重要だから」

湊の違和感はさらに強くなった。しかし「生産性向上」を求められたばかりの湊は反論する勇気が出なかった。リーダーとしてチームを守るべきなのに。リーダー3ヶ月目の自分には、まだPMに強く意見する自信がない。

「わかりました。大枠で進めさせていただきます」

ミーティング後、チームで打ち合わせをした。

山田が厳しい表情で言った。

「湊さん、これ本当に大丈夫ですか？仕様が曖昧すぎますよ」

「わかっています。でも部長からは生産性向上って言われてますし、断れませんでした...」

佐藤が不安そうに聞いた。

「どこから手をつければいいんですか？要件定義段階なら修正コストは1倍ですが、実装段階だと5-10倍になりますよね」

山田が頷いた。

「その通りだ。過去のプロジェクトでも、仕様変更1件で平均3日の手戻りが発生した」

湊は決断した。

「とりあえずダミーデータで動く画面だけ先に作りましょう。仕様が決まってからロジックを実装します。佐藤君と他のメンバーでUI作成、山田さんと僕でデータベース設計の案を複数パターン用意しておきます」

山田が頷いた。

「それしかないですね。でも後で大幅な手戻りになる可能性が高いですよ」

「それでも動かないよりはマシです。まず何か形にしないと」

仕様変更が止まらない

湊はすぐに開発を開始した。しかし仕様が曖昧なまま進めるのは予想以上に困難だった。どのようなデータを取得すべきかどのような分析結果を表示すべきか判断に迷う場面が続出する。

そして案の定、仕様変更が次々と発生した。

飛鳥さくら が **#project-urgent** でメッセージを送信

> *やっぱりここはこうしたい*
>
> *ユーザーの滞在時間も分析に含めてください*

飛鳥さくら が **#project-urgent** でメッセージを送信

> *この機能も追加で*
>
> *リアルタイムでの分析結果表示も必要です*

Slackの通知音が鳴り止まない。湊は作業を中断して次々と送られてくる仕様変更に対応する必要があった。

「また仕様が変わった...」

湊は今日作ったコードを見直し大幅な修正が必要になることを悟った。せっかく書いたコードの多くが無駄になってしまう。

夜10時のオフィス。まだ全員が残業中だった。湊のチーム5名全員がそれぞれのデスクで作業を続けている。

佐藤が疲れた声で言った。

「湊さん、経費精算モジュールとの連携部分、また仕様変わったんですけど...UIの配置も、さっきのSlackで変更指示が来ました」

佐藤が続けた。

「今月の残業時間、もう50時間超えてます…」

山田も疲れた表情で言った。

「僕も45時間。このペースじゃ持たない。新機能追加に3日→今は1週間かかるし、バグ修正のたびに別の場所が壊れる」

他のメンバーも頭を抱えている。

深夜1時過ぎ、湊は帰宅した。玄関を開けるとリビングのテーブルに昨日のビール缶が2本残っている。冷蔵庫から新しいビールを取り出し一気に飲み干した。窓際のパキラは少し葉が垂れ始めている。水をやる気力も残っていない。

翌朝のオフィス。湊が出社すると山田がすでに席についていた。

「また仕様が変わった…今日作ったコードを全部書き直しか…」

湊が呟くと山田が振り返った。

「技術的負債も溜まっていく一方だな…」

山田の表情は疲れ切っていた。

「新機能を追加するたびに既存のコードが複雑になっていく。スピードを優先して単体テストすら疎かになっているから、変更の影響範囲がわからない。一箇所を修正したら、どこまで影響するか予測できない。だから、予想外の場所でエラーが出る」

その時、QA部の高橋美咲が開発チームのエリアにやってきた。別フロアのQA部から歩いてきたようだ。26歳、QA経験3年の高橋は、細かいところまで気を配る性格で知られている。

「湊さん、ちょっといいですか？」

高橋の顔には明らかな疲労の色が浮かんでいた。

「正直、テスト時間が全然足りないんです。このままじゃまずいですよ。私、バグのある製品をユーザーに届けたくないんです」

高橋が続けた。

「仕様変更が週に3-5回で、テストケースも毎回見直さないといけない。でもリリース日は変わらないから検証時間がどんどん削られていく」

湊は申し訳なさそうな表情になった。

「すみません、仕様変更が多くて...」

山田が横から言った。

「QA部にも相当な負担かけてるな。高橋さんたちも他のプロジェクトあるだろうに」

高橋は首を横に振った。

「湊さんのせいじゃないよ。でもこのままだと品質が心配...正直、十分なテストができる自信がない」

湊は深く頭を下げた。

「本当に申し訳ないです」

山田がコーヒーを淹れながら言った。

「みんな疲れてるな。でも部長からは生産性向上を求められてるし飛鳥さんからはスピードを求められてる。どうすればいいんだろう」

山田は少し間を置いて続けた。

「このペースで開発期間が延びれば、開発コストも増える。でも部長はその財務への影響を理解しているのかな？エンジニアの僕らには、開発が遅れることが財務にどう影響するのか説明できない。経営層とエンジニアの間で、お金の話ができる共通言語がないんだよ」

湊は山田の言葉を聞いて、改めて問題の本質を考えた。開発が遅れること自体が問題なのではない。なぜ遅れたのか、次はどう変わるのかを説明できないことが問題なのではないか。エンジニアの見積を信じて事業計画を作るが、毎回遅れる。なぜ遅れるのかは説明されない。リカバリーとして人件費を追加投入するが、コストだけが増える。この構造的な問題が、湊のチームを苦しめているのではないか。

深夜2時、湊はコンビニ弁当を食べながらVSCodeの画面を見つめていた。今日だけで仕様変更が5回もあった。そのたびに書いたコードを修正しテストケースを見直しドキュメントを更新する必要があった。

帰宅するとテーブルには今日のビール缶が3本追加されていた。昨日の2本と合わせて5本。空き缶の山が自分の疲弊を物語っている。

これが本当に生産性向上なのだろうか？

湊は疑問に思った。確かにコードを書くスピードは速くなったかもしれない。でもその分品質を犠牲にしているのではないか？そしてチーム全体が疲弊している。

リリース後の悪夢

リリース予定日の朝、湊は早めにオフィスに到着した。最終テストを実行しリリースの準備を整えようとしていた。

しかしテスト結果を見て湊の顔が青ざめた。

「バグが15件も…クリティカルなバグが3件、その他にも多数」

高橋が報告した。

「月に5-10件のバグがリリース後に発見される状況が続いています。今回も同様の傾向です」

高橋が慌てて湊のところにやってきた。

「湊さん、大変です。クリティカルなバグが3件、その他にも多数のバグが見つかりました」

湊はテスト結果を確認した。確かにユーザー登録機能でエラーが発生するバグ、データ分析結果が正しく表示されないバグ、そして画面が固まるバグなど深刻な問題が複数あった。

その時飛鳥が慌ててやってきた。

「今日リリースしないと間に合わない！なんとかして！」

飛鳥の表情は必死だった。

「でもこのバグは修正に時間がかかりそうです...」

「致命的なバグだけ修正して他は後回しにできない？」

湊は迷った。確かに一部のバグは致命的ではないかもしれない。でもユーザーに影響を与える可能性があるバグをそのままリリースするのは責任として重い。

「このバグは致命的じゃないから...リリースしよう」

湊は苦渋の決断を下した。部長からのプレッシャー、飛鳥からの要請、そしてチーム全体の疲労を考えると完璧を求めている余裕はなかった。

リリースは予定通り実行された。湊は祈るような気持ちでユーザーからの反応を待った。

リリース後30分、サポートチームから緊急連絡が入った。

「ユーザーからエラー報告が殺到しています！リリース後30分で、すでに20件以上のエラー報告が入っています。ログインできない、データが表示されない、画面が固まるなどの報告が多数入っています！」

湊の心臓が止まりそうになった。リリース後30分で20件以上のエラー報告。通常なら、リリース後1週間で数件程度のバグ報告があるだけだった。でも、今回は30分で20件。これは異常な数値だった。

サポートチームのリーダーが続けた。

「特に、ログイン機能のエラーが深刻です。ユーザーの30%がログインできていない状況です。データ分析機能も、結果が正しく表示されないという報告が10件以上入っています。画面が固まるという報告も5件以上あります」

湊は画面を見つめた。エラーログには、次々と新しいエラーが記録されていく。1分ごとに、新しいエラーが発生している。

これが生産性向上の結果なのか…？

湊は急いで障害対応を開始した。しかし修正したコードが他の部分に影響を与え新たなバグを生み出している可能性もあった。チーム全体がパニック状態になった。

「どうしよう…」

湊は絶望的な気持ちになった。生産性を上げようとして結果的に品質を下げユーザーに迷惑をかけてしまった。これでは本末転倒だった。

解説：スピード重視の技術的背景と問題点

ストーリーで描かれる「重圧」

問題は見えているが悪化し続けているときに重くのしかかるのが、「スピード優先」という重圧です。湊のチームが経験したそれも、同じ構造から生じています。

- **「とにかく早く」という圧力:**　PMや経営層から「競合に後れを取る前にリリースしろ」という圧力がかかり、仕様が曖昧なまま開発を進めざるを得ない。要件定義の時間を十分に確保できない
- **仕様変更の連鎖反応:** 詳細が決まらないまま開発を開始するため、仕様変更が週に3-5回発生し、手戻りが続く。要件定義段階なら修正コストは1倍だが、実装段階では5-10倍になる

- **負のスパイラルの形成:** 急いだ実装 → バグ発生 → 修正工数増加 → さらなるスピード重視という悪循環が生まれている。短期的にスピードが上がっても、長期的には品質の低下により開発速度が大幅に低下する
- **技術的負債の蓄積が始まる:** 急いだ実装によりコードが複雑化し始めている。新機能追加に3日かかっていたのが1週間かかるようになり、バグ修正のたびに別の場所が壊れる状態になっている
- **チーム疲弊の兆候:** 残業が常態化（月40-60時間）し、判断力とモチベーションが低下し始めている。しかしまだ「このままでは持続不可能だ」という実感には至っていない

この重圧の背景には、**スピード重視がもたらす「負のスパイラル」が形成され、問題が深化していく構造**があります。本質（なぜ問題が起きるのか）を理解する前に、悪化が続いているのです。

本質的な問題

湊のチームが経験した問題は多くの開発チームが直面する典型的な「負のスパイラル」です。この問題の背景には、スピード重視がもたらす構造的な問題があります。

前節で触れたように、遅れること自体が問題なのではなく、なぜ遅れたのか・次はどう変わるのかを説明できないことが問題です。エンジニアの見積を信じて事業計画を作るが毎回遅れ、理由は説明されず、人件費の追加投入でコストだけが増える。この構造が多くの開発チームを苦しめています。

なぜスピード重視が負のスパイラルを生むのか

なぜスピード重視が問題を生むのか

スピード重視の開発で起こる典型的な問題

- **設計の軽視:** 十分な設計時間を取らずに実装を開始
- **テストの省略:** 品質保証の時間が削減される

- **技術的負債の蓄積:** 急いだ実装がコードの複雑化を招く
- **チームの疲弊:** 長時間労働が判断力とモチベーションを低下させる

これらが連鎖することで、仕様変更 → 急いだ実装 → バグ発生 → 修正工数増加 → さらなるスピード重視という「負のスパイラル」が形成されます。

「品質とスピードはトレードオフ」と言われるが、短期的にスピードが上がっても、長期的には品質の低下により開発速度が大幅に低下する。

開発組織と事業責任者・PdMの視点の違い

開発組織が見ているもの（リードタイム、デプロイ頻度、変更失敗率、MTTR）と、事業責任者・PdMが見ているもの（計画信頼性・予測可能性）は異なります。Four Keysなどの指標は開発組織の内部安定性を測りますが、事業計画の信頼性は直接測りません。これが、Four Keysが改善しているにもかかわらず、事業責任者・PdMには「良くなった実感」がないギャップの原因です。

技術的負債の予兆検知

技術的負債の蓄積は、以下の兆候で早期に発見できます。

- 仕様変更が週に何回発生しているか？
- バグ修正に開発時間の何%を費やしているか？
- チームの残業時間は適切な範囲か？
- **コードの複雑度の増加:** 循環的複雑度が10を超えるコードが増える
- **ビルド時間の延長:** 5分だったビルドが10分、20分と延びていく
- **バグ発生率の増加:** 新機能追加のたびにバグが増える
- **開発速度の低下:** 新機能追加に3日かかっていたのが1週間、2週間と延びる

詳細な改善手法については、後述のワークシートで具体的に見ていきます

1-3 「このままで無理だ」という気づき

転換点への気づき

リリース後の混乱のなかで、湊はようやく根本を問い直し始めていた。

徹夜の対応、限界を超えたチーム

リリース翌日の朝、湊のチーム全員が徹夜で障害対応を行っていた。ユーザーからの報告は相変わらず多く、チーム全体が対応に追われている。

山田が疲れた声で言った。

「湊さん、データベースの接続エラーはこっちで対応しています。でも根本原因がまだ特定できていません。エラーログを見ると、接続タイムアウトが頻発しているんですが、なぜタイムアウトが起きているのか、まだわからないんです」

佐藤がパニックになっている。佐藤の手は震えている。

「ログイン画面のエラー、原因わかりません…。コードを見直しても、どこがおかしいのかわからない。急いで書いたコードだから、どこに問題があるのか特定するのに時間がかかって…」

佐藤の声は震えていた。湊は佐藤の様子を見て、心配になる。

山田が助け船を出した。

「佐藤君、そっちは僕が見ます。君は経費精算モジュールのバグ対応に集中して。無理しすぎないで」

山田は佐藤の肩を軽く叩いた。でも、佐藤はその手を振り払った。

「大丈夫です。自分でなんとかします」

佐藤の声には、疲労と焦りが混じっていた。

その時、QA部の高橋が開発エリアに駆け込んできた。高橋の顔は青ざめている。

「湊さん、また新しいエラーが報告されました。今度は勤怠管理モジュールです。ユーザーから『勤怠データが表示されない』という報告が10件以上入っています」

高橋が疲れた表情で湊に報告した。

「ありがとうございます。すぐに確認します」

湊はログを確認しエラーの原因を特定しようとしていた。しかし急いで修正したコードが複雑になりすぎてどこに問題があるのか特定するのに時間がかかった。

山田が湊の隣に来て小声で言った。

「湊さん、チームのみんなもう限界ですよ。佐藤なんか昨日から一睡もしてません。高橋さんも、徹夜でバグ報告を整理しています。このままでは、チームが崩壊してしまいます」

湊は周りを見渡した。確かにチーム全員の顔に疲労の色が濃い。佐藤は画面を見つめながら、何度も頭を抱えている。高橋は報告書を書きながら、時々涙を拭いている。山田も、コーヒーを何杯も飲んでいるが、疲労は隠せない。リーダーとして何かしなければ。

「わかりました。優先順位をつけましょう。致命的なバグから順に対応します。それ以外は一旦ペンディングにして、みんな順番に仮眠を取りましょう」

部長からの追及

午後3時、田中部長から呼び出しがあった。

「湊君、昨日のトラブルはどういうことだ？」

田中部長の表情は厳しかった。経営陣からの追及を受けたのだろう。管理職としての板挟みの苦しさが、その表情から伝わってくる。

「申し訳ありません。バグの修正が不十分でした」

「生産性を上げろと言ったのに、これじゃ逆効果じゃないか。ユーザーからのクレームも来ているし、サポートチームも対応に追われている」

田中部長が続けた。

「リリース後のバグ修正は開発時の30-100倍のコストがかかるって聞いたが...技術的負債の蓄積で開発速度が年間20-40%低下しているというデータもある」

湊は返答に困った。確かに生産性向上を求められて結果的に大きな問題を引き起こしてしまった。

「部長、申し訳ありません。でもスピードを重視しすぎて品質を犠牲にしてしまいました」

「品質も大事だがスピードも大事だ。そのバランスをどう取るかが問題なんだ」

田中部長はため息をついた。

「とりあえず今回の件の報告書を作成してくれ。経営陣への説明も必要になる。それと、湊君、改善策を考えてみてくれないか？でも、その改善策がどれだけ効果があるのか、投資対効果がわからないと経営陣には説明できないんだ」

湊は返答に困った。改善策を考えることはできる。でも、その改善策がどれだけの効果をもたらすのか、具体的なROIを説明することはできない。エンジニアの自分には、開発生産性向上の投資対効果を財務的な言葉で説明する方法がわからない。

「はい、承知いたしました。改善策を考えてみます」

湊は答えたが、内心では不安だった。改善策を提案しても、「ROIがわからない」と却下されるのではないか。エンジニアと経営層の間で、お金の話ができる共通言語がない。田中部長も、バグ修正のコストについて数字は知っているが、それ

が財務にどう影響するのか、減価償却費や税前利益への影響を理解しているようには見えない。

湊は深々と頭を下げた。

山田からの温かい言葉

部長室を出ると山田が待っていた。

「どうだった？」

「報告書を書くことになった。それと…チームのみんなには本当に申し訳ないことをした」

山田は湊の肩を叩いた。

「湊さん、あなたのせいじゃないですよ。でも、このままじゃチームが持たないのは確かです」

一人の夜、枯れかけたパキラ

夜10時、オフィスには湊一人が残っていた。他のメンバーは疲労困憊で帰宅し湊は報告書の作成を続けていた。佐藤は最後まで残ろうとしたが湊が無理やり帰らせた。リーダーとしてこれ以上メンバーを巻き込むわけにはいかない。また一人で抱え込もうとしている。リーダー3ヶ月目の自分には、まだチームに頼る勇気が足りない。

深夜0時過ぎ、湊は帰宅した。玄関を開けるとテーブルには空き缶が10本近く積み上がっている。リビングに目をやると窓際のパキラの葉が茶色く枯れかけていた。水をやったのはいつだっただろう。先週？先々週？

「ああ…」

湊は枯れかけたパキラを見つめながら自分自身の状態を重ね合わせた。チームのみんなも同じように疲弊している。このままでは本当に持続不可能だ。

しかし報告書を書いているうちに湊は根本的な疑問を抱き始めた。

そもそも開発生産性って何なんだろう？

湊は手を止めて改めて考えてみた。

- コードを書くスピードが速いこと？
- リリース頻度が高いこと？
- バグが少ないこと？
- ユーザーに価値を提供できること？

技術的負債の蓄積による開発速度低下（年間20-40%低下）を目の当たりにし、湊は根本的な問題に気づき始めていた。新機能追加に3日かかっていたのが1週間、2週間と延びていく。バグ修正のたびに別の場所が壊れる。このままでは開発が止まってしまうのではないか。

湊は深く考え込んだ。ビジネス側は「売上に直結する重要な機能」と言い、経営層は「数字を出せ」と言う。一方で、エンジニアの自分たちはコード品質や技術的負債を気にしている。それぞれが「開発生産性」という言葉に込める意味が違うのではないか。レイヤーごとに価値基準が異なることを、湊はこの瞬間に強く感じていた。

でも、もっと根本的な問題があるのではないか？

湊は改めて考えてみた。開発が遅れること自体が問題なのではない。なぜ遅れたのか、次はどう変わるのかを説明できないことが問題なのではないか。エンジニアの見積を信じて事業計画を作るが、毎回遅れる。なぜ遅れるのかは説明されない。リカバリーとして人件費を追加投入するが、コストだけが増える。この構造的な問題が、湊のチームを苦しめているのではないか。

開発生産性の本質は、開発組織が「いつまでに何ができるか」と語った約束が、どれだけ信頼できるかということなのではないか？

湊はノートに書き出した。開発組織が見ているもの（リードタイム、デプロイ頻度、変更失敗率、MTTR）と、事業責任者・PdMが見ているもの（計画信頼性・予測可能性）は異なる。Four Keysなどの指標は開発組織の内部安定性を測るが、事業計画の信頼性は直接測らない。これが、Four Keysが改善しているにもかかわらず、事業責任者・PdMには「良くなった実感」がないギャップの原因なのではないか。

今回の経験を通じて湊は一つのことに気づいた。

スピードだけを追求しても結果的に品質が下がりユーザーに迷惑をかけてしまう。それでは本当の意味での生産性向上にはならない。ビジネス要求とエンジニアリングの優先度のずれをどう埋めればいいのか。

湊はノートに書き出した。

今回の失敗から学んだこと

1. 仕様が曖昧なまま開発を進めることの危険性
2. スピードと品質のバランスの重要性
3. チーム全体の疲労が品質に与える影響
4. 技術的負債の蓄積が将来の開発速度に与える影響

でもどうすればスピードと品質を両立できるんだろう？

湊は深く考え込んだ。部長からは生産性向上を求められている。でも今回のような失敗を繰り返すわけにはいかない。

このままでは持続不可能だ。何か根本的な解決策が必要だ。

湊は決意を新たにした。明日から開発生産性についてもっと深く学んでみよう。そしてチーム全体で話し合い本当の意味での生産性向上とは何かを探ってみよう。

一人で抱え込むのではなくみんなで考えてみる必要がある。

28歳、経験5年、リーダー3ヶ月目。まだ若いリーダーだが、今回の失敗から学ぶことは多い。山田の12年の経験、高橋の品質への責任感、チーム全員の力を借りれば、きっと道は開ける。

湊は最後にノートに一行書き加えた。

明日やること

- 開発生産性について調べる
- 山田さんに相談する
- チーム全体で話し合う機会を作る
- 佐藤や他のメンバーの意見も聞く
- QA部の高橋さんにも協力を求める

湊はオフィスを出た。夜風が頬を撫でる中湊は改めて決意した。

今度こそ本当の意味での生産性向上を実現してみせる。チームのみんなを守るために。

チームへの呼びかけ

翌朝、チームメンバー全員にSlackでメッセージを送った。

> *おはようございます。昨日は本当にお疲れ様でした。*
>
> *今日の午後、チームミーティングを開きたいと思います。*
> *今回の件を踏まえて、これからの開発の進め方について*
> *みんなで話し合いたいです。*
>
> *一人ひとりの意見を聞かせてください。*

山田からすぐに返信が来た。

> *了解。必要なことだと思う。*

佐藤や他のメンバーからも賛同のリアクションが返ってきた。

湊は少し希望を感じた。チーム全体で考えれば何か道が開けるかもしれない。

解説：持続可能な開発への転換点

ストーリーで描かれる「重圧」

「このままで無理だ」と感じたことがあるなら、それは**どう変えるかを考え始めるとき**に現れる構造的な課題の表れかもしれません。湊が経験した重圧もそこから生じています。

- **リリース後の惨状:**　リリース後30分で20件以上のエラー報告が殺到し、チーム全員が徹夜で障害対応を行う。リリース後のバグ修正コストは開発時の30-100倍になり、本来の開発が完全に停止している
- **チームの限界点:** 長時間労働の常態化により、判断力とモチベーションが著しく低下している。技術的負債の蓄積により開発速度が50%低下し、このままでは開発が止まってしまうという危機感が生まれている
- **改善への障壁:** 改善提案をしても「ROIがわからない」と却下される。エンジニアと経営層の間で、お金の話ができる共通言語がない。効果測定方法が確立されていないため、説得力のある説明ができない
- **説明責任の重圧:** 「なぜ遅れたのか」「次はどう変わるのか」を説明できない状態が常態化している。開発期間の延長による開発コストの増加、減価償却費の増加による税前利益の圧迫が、財務に直接影響を与えていることを理解していても、それを説明する方法がない
- **転換点への気づき:** 「そもそも開発生産性って何なんだろう」という根本的な疑問が生まれ、問題の本質を理解する必要性を感じ始めている。一人で抱え込むのではなく、チーム全体で考えてみる必要があることに気づき始めてい

る

この重圧の背景には、**持続不可能な開発がもたらす構造的な問題が顕在化し、「どう変えるべきか」を考える転換点に到達している**という状況があります。認識と深化を経て、ようやく転換を考え始めているのです。

本質的な問題

湊が経験した失敗とその後の気づきは、多くの開発チームが直面する転換点である。持続不可能な開発がもたらす構造的な問題が背景にある。

本質的な問題は前の解説で述べた通り、遅れを説明できないことと、見積を信じて計画するが毎回遅れ、人件費追加でコストだけが増える構造である。

開発生産性の本質的な定義

開発生産性の本質については前で述べた通り、約束の信頼性（いつまでに何ができるかがどれだけ信頼できるか）にある。

持続不可能性の兆候を見極める

湊のチームで見られた持続不可能性の兆候

- **長時間労働の常態化**: 夜10時以降の残業が日常化
- **チーム全体の疲弊**: 疲労による判断力の低下とモチベーション低下
- **技術的負債の顕在化**: コードの複雑化により開発速度が低下
- **ユーザーへの影響**: エラー報告が殺到し、ビジネスに直接影響
- **説明責任の欠如**: 「なぜ遅れたのか」「次はどう変わるのか」を説明できない
- **指標と実感のギャップ**: Four Keysなどの指標が改善しているにもかかわらず、事業責任者・PdMには「良くなった実感」がない

- 開発期間の延長による開発コストの増加
- 減価償却費の増加による税前利益の圧迫
- 市場投入遅れによる機会損失

ROIの重要性

- ROIが明確な改善提案は、承認率が80%以上
- ROIが不明確な改善提案は、承認率が20%以下
- 効果測定方法が確立されていない組織では、改善提案が却下されることが多い

内部品質投資の再定義

技術的負債の返済は、単なる「負債の返済」ではなく、「将来の見積精度向上とリードタイム分散低減への投資」として位置づける必要があります。これにより、内部品質投資が事業価値に直接つながることが明確になります。

手法1 開発生産性の危険な落とし穴チェックリスト

湊の経験を通じて明らかになった、開発生産性向上の際に陥りがちな落とし穴をチェックリストとして整理しました。これらの項目に当てはまるものがないか確認してみましょう。

チェック項目

1. 要件定義・仕様管理に関する落とし穴

- [ ] 仕様が曖昧なまま開発を開始している
- [ ] 仕様変更が頻繁に発生し、開発計画が狂っている
- [ ] 仕様変更の影響範囲を十分に検討していない
- [ ] ステークホルダー間で仕様の認識が一致していない
- [ ] 仕様書が古くなり、実際の実装と乖離している

2. スケジュール・工数管理に関する落とし穴

- [ ] テスト工数を十分に見積もっていない
- [ ] バグ修正の工数を見積もりに含めていない
- [ ] 技術的負債の返済時間を確保していない
- [ ] 緊急対応のためのバッファ時間がない
- [ ] チームメンバーの疲労度を考慮していない

3. 品質管理に関する落とし穴

- [ ] スピードを優先して品質を犠牲にしている
- [ ] テストケースが不十分で、バグを見逃している
- [ ] コードレビューの時間が確保できていない
- [ ] 技術的負債が蓄積し続けている
- [ ] リリース後の障害対応に追われている

4. チーム運営に関する落とし穴

- [ ] チームメンバーが疲弊している
- [ ] 残業が常態化している
- [ ] メンバー間のコミュニケーションが不足している
- [ ] 責任の所在が不明確で、問題が放置されている

- [ ] チーム全体のモチベーションが低下している

5. 指標・評価に関する落とし穴

- [ ] コード行数やリリース頻度だけで生産性を測っている
- [ ] 短期的な指標に偏重している
- [ ] 品質やユーザー満足度を考慮していない
- [ ] 技術的負債の影響を数値化していない
- [ ] チームの成長や学習時間を評価に含めていない

チェック結果の見方

0〜5個: 比較的良好な状態です。現在の取り組みを継続しつつ、改善の余地がある項目に取り組んでみてください。

6〜10個: 注意が必要な状態です。特に深刻な項目から優先的に対策を検討することをお勧めします。

11個以上: 危険な状態です。早急にチーム全体で問題を共有し、根本的な改善に取り組む必要があります。

改善のヒント

1. **仕様管理の改善**: 要件定義の時間を十分に確保しステークホルダー間での合意形成を図る
2. **工数見積もりの精度向上**: 過去の実績データを活用しテストやバグ修正の工数も含める
3. **品質の可視化**: バグ発生率、テストカバレッジ、コードの複雑度などの指標を定期的に確認する
4. **チームの健康管理**: 残業時間の上限設定、定期的な振り返り、メンバーの声を聞く機会の確保

5. **多面的な評価**: スピードだけでなく品質、ユーザー満足度、チームの成長も含めた総合的な評価

次のステップ

　このチェックリストで問題が発見された場合、次の章では湊がこれらの問題にどう取り組んだかを学べます。一人で抱え込まずチーム全体で問題を共有し段階的に改善していく。

章のまとめ

　本章では湊太郎が「開発生産性向上」という重圧に直面しスピードを重視した結果、品質を犠牲にしてしまった経験を描きました。

学んだポイント

1. **開発生産性の誤解**: スピードを上げることだけが生産性向上ではない
2. **スピードと品質のバランス**: どちらか一方を犠牲にすると長期的には開発効率が下がる
3. **チーム全体への影響**: 個人の判断がチーム全体の疲弊や品質低下につながる
4. **根本的な問題**: 表面的な改善ではなく根本から問題を解決する必要がある

次章への展望

　湊は今回の失敗を通じて開発生産性について根本的に考え直す必要性を感じました。次の章では湊が「そもそも開発生産性とは何か」という疑問に向き合い、多角的な視点から生産性を理解していく過程を描きます。

一人で抱え込まずチーム全体で問題を共有し段階的に改善していく。次の章でその過程を追う。

参考文献

学術論文

- Forsgren, N., Humble, J., & Kim, G. (2018). “Accelerate: The Science of Lean Software and DevOps”. IT Revolution.
- Forsgren, N., Storey, M. A., Maddila, C., Zimmermann, T., Houck, B., & Butler, J. (2021). “The SPACE of Developer Productivity: There’s more to it than you think”. Communications of the ACM, 64(1), 62-69.

業界調査レポート

- DORA (DevOps Research and Assessment). (2023). “Accelerate State of DevOps Report”. https://dora.dev/
- McKinsey & Company. (2023). “Measuring Developer Productivity: A Practical Guide”. https://www.mckinsey.com/
- Stack Overflow. (2023). “Developer Survey 2023”. https://survey.stackoverflow.co/
- Standish Group. (2023). “CHAOS Report: Project Success and Failure Rates”. https://www.standishgroup.com/
- IBM Systems. (2023). “Software Development Cost Analysis: Bug Fix Costs by Phase”. IBM Research.

企業事例・ケーススタディ

- Stripe. (2023). “The Developer Coefficient: A $300B Opportunity”.

https://stripe.com/

- GitHub. (2023). “State of the Octoverse”. https://octoverse.github.com/
- Google re:Work. (2023). “The Five Keys to a Successful Google Team”. https://rework.withgoogle.com/
- Gallup. (2023). “State of the Global Workplace Report”. https://www.gallup.com/