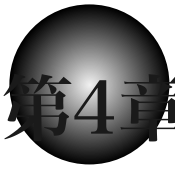

第4章 重圧から生み出される小手先の指標

4-1 小手先の指標に依存せずに問題を特定する手法

ワークショップの準備

金曜日の午後、大会議室に湊は一人で準備をしていた。ホワイトボードに付箋を貼る準備をしながら、心臓の鼓動が早まっているのを感じた。

先週、一人でデータ分析をしていた湊は、山田からの後押しを受けて、チーム全体で問題を共有するワークショップを企画した。でも、本当にみんな来てくれるだろうか。忙しい中、時間を割いてもらえるだろうか。もし誰も本音を言わなかったら、このワークショップは意味のないものになってしまう。

湊は付箋を数えながら考えていた。リーダーとして、チームを正しい方向に導く責任がある。でも、その方法がわからない。一人でデータ分析をしていた時は、数字を見つめていれば良かった。でも、今は違う。人を動かし、本音を引き出し、共通の課題を見つけなければならない。

「準備手伝うよ」

背後から声がかかった。振り返ると、山田が立っていた。

「山田さん、ありがとうございます」

「湊さんが一人で抱え込まないで、チーム全体で考えようって決めたのは正解だと思うよ。僕も昔、同じようなことを経験したことがあるんだ」

山田はホワイトボードにマーカーを並べながら言った。

「でも、最初はみんな遠慮がちかもしれない。特に、別組織の飛鳥や高橋さんは、開発チームの内情を直接言いにくいかもしれない。別組織だから、遠慮してしまうんだよね」

「そうですね...どうやって本音を引き出せばいいんでしょうか」

「まずは、湊さんが率先して具体的な痛みを話すことだね。リーダーが本音を言えば、他の人も話しやすくなる。それから、誰の発言も否定しないこと。『それは違う』とか『それは問題じゃない』とか言わない。どんな小さな痛みでも、まずは受け止めることが大事だよ」

山田はマーカーを置き、湊の方を向いた。

「あと、飛鳥と高橋さんには、最初に『別組織だけど、今日は開発チームの一員として意見を聞かせてほしい』って伝えるといいかもしれない。そうすれば、遠慮せずに話してくれるはずだ」

「なるほど...ありがとうございます」

湊は山田のアドバイスを頭の中で整理していた。リーダーとして率先して本音を言う。誰の発言も否定しない。別組織のメンバーにも配慮する。これらを意識すれば、きっと本音を引き出せるはずだ。

時計を見ると、開始時刻まであと10分。

メンバーが集まる

徐々にメンバーが集まってきた。

最初に到着したのは佐藤だった。佐藤は、少し緊張した様子で会議室に入ってきた。

「湊さん、お疲れ様です。何か手伝えることありますか？」

「ありがとう。でも大丈夫だよ。座って待っていて」

続いて、他のチームメンバー2名も到着した。それぞれが席に座り、スマートフォンをいじったり、ノートを見つめたりしている。まだ緊張した空気が漂っている。

「お疲れ様です」

飛鳥が会議室に入ってきた。カバンを置き、湊の方を向いた。

「湊君、今日はどんな内容ですか？事前に資料は送ってもらってるけど、実際に何をするのか楽しみにしてるわ」

「ありがとうございます。今日は、みんなで日々感じている『痛み』を共有して、一緒に改善策を考えたいと思っています」

「それはいいわね。プロダクト部としても、開発チームの課題を理解したいから」

最後に、高橋が到着した。

「すみません、少し遅れました。テストの最終確認をしていて...」

「大丈夫です。まだ開始時刻まで時間がありますから」

5名のチームメンバー全員、飛鳥、高橋。全員が揃った。

湊は深呼吸をして、全員に向き直った。

「今日はお忙しい中、ありがとうございます。先週、技術的負債の影響を可視化したんですが、それだけでは不十分だと思って。みんなで日々感じている『痛み』を共有して、一緒に改善策を考えたいんです」

湊は山田のアドバイスを思い出しながら続けた。

「飛鳥さん、高橋さん、今日は別組織から来ていただいていますが、今日は開発チームの一員として、遠慮せずに意見を聞かせてください。開発チームだけでは見えない視点を、ぜひ教えてほしいんです」

ぎこちない開始

湊が説明を終えると、会議室に重い沈黙が訪れた。最初の5分間、誰もが遠慮がちに付箋を見つめている。スマートフォンをいじる人、ノートに何か書き込む人、ただただ空気を見つめる人。それぞれが何かを言おうとしているが、誰も最初の一歩を踏み出せない。

湊は山田のアドバイスを思い出していた。リーダーが率先して本音を言えば、他の人も話しやすくなる。でも、何を話せばいいのか。湊自身も、まだ整理できていない痛みがたくさんある。

「まず、日々の開発で感じている『痛み』を付箋に書いてみませんか？どんな小さなことでも構いません。例えば、『今日も残業になった』とか、『また仕様が変わった』とか、そんなことでもいいんです」

湊の言葉に、メンバーたちは遠慮がちに書き始めた。最初は当たり障りのない内容ばかりだった。

佐藤が小さく声を出して読んだ。

「時間が足りない」

別のメンバーが続けた。

「仕様変更が多い」

「残業が多い」

付箋がホワイトボードに貼られていく。でも、まだ表面的な内容ばかりだ。湊は内心で焦っていた。このままでは、本当の問題が見えてこない。

山田の勇気ある発言

その時、山田が立ち上がった。ホワイトボードの前に歩み寄り、マーカーを手に取った。

「じゃあ、僕から具体的な話をしますね。技術的負債で新機能が作りにくい。同じ規模の機能開発に、以前は3日かかっていたのが、今は1週間かかる。コードが複雑になりすぎて、どこを触っても影響範囲が広がる。テストを追加するのも大変だし、リファクタリングする時間もない」

山田は付箋に書きながら続けた。ペンの音だけが会議室に響く。

「特に問題なのは、スピードを優先してテストを書かないことだ。テストがないと、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」

「それから、ドメイン駆動設計を採用しているんだけど、イベントストーミングでのモデリングができていない。ドメインモデルが曖昧なまま開発が進んで、積み重なった技術的負債と膨れ上がった設計の乱雑さが問題になっている」

「それから、バグ修正のたびに別の場所が壊れる。モグラ叩きみたいな状態が続いている。先週も、経費精算のバグを修正したら、勤怠管理の画面が表示されなくなった。なんでそんなことが起きるのか、自分でもわからない。これが、僕が感じている痛みです」

山田が付箋をホワイトボードに貼った瞬間、会議室の空気が少し変わった。佐藤が小さく「わかる…」と呟いた。別のメンバーも深く頷いている。山田の具体的な発言が、他のメンバーにも勇気を与えたようだ。少しずつ、本音を出し始めた。

本音の爆発

山田の発言の後、再び沈黙が訪れた。でも、今度は違う種類の沈黙だった。誰かが何かを言おうとしている、そんな緊張感が漂っている。

高橋が小さく息を吸い込んだ。両手を膝の上に置き、背筋を伸ばした。そして、勇気を振り絞るように、はっきりと言った。

「正直、テスト時間が全然足りません。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」

高橋の声は少し震えていたが、その言葉には強い意志が込められていた。

「先月のリリースでも、最終テストでバグが3件見つかったんです。でも、リリース日が決まっているから、そのままリリースしてしまった。結果として、リリース後にユーザーからエラー報告が殺到して、徹夜で対応することになりました。QA部として、品質に責任を持っているんですが、テスト時間がスケジュールに入らないことが多いんです。開発チームには悪いと思いつつも、もっと時間をかけてほしいって思ってしまう」

高橋の言葉に、エンジニアのメンバーたちも頷いた。佐藤が「すみません...」と小さく謝った。別のメンバーも「確かにテスト時間を削ってしまって...」と反省の表情を見せた。誰もが同じ経験をしていた。でも、それを口に出すことができなかった。

高橋の言葉がきっかけで、堰を切ったように本音が溢れ出した。

「コードレビューする時間がない。急いでコードを書いて、レビューを依頼するけど、レビューする側も忙しくて時間が取れない。結局、レビューなしでマージしてしまうこともある。それで後からバグが見つかって...」

佐藤が小さく呟いた。

「見積もりがいつも外れる理由がわからない。失敗パターン『とりあえず作ってから考える』に陥ってる。最初は『3日でできる』って言ったのに、実際には1週間かかってしまう。その理由を説明するのが難しい」

「プロジェクト管理モジュールも同じ。データベースの構造が複雑で、新しい機能を追加するのが大変。同じようなコードが3箇所に散らばっていて、全部修正しないと動かない。でも、全部修正するのは時間がかかりすぎる」

別のメンバーが続けた。

「勤怠管理モジュールのコード、前の担当者が書いたんだけど、なぜこの設計にしたのかわからない。その人に聞かないと理解できない。でも、その人はもう別のプロジェクトに移ってしまって...」

「ライブラリのバージョンも古いままだ。セキュリティパッチを適用したいけど、バージョンアップすると動かなくなる可能性がある。でも、このまま放置するのもリスクだし...」

PMの飛鳥が手を挙げた。

「プロダクト部としても、悩みがあるんです。ユーザーや経営層から機能追加の要望が来る。それを開発チームに伝えると、『技術的に難しい』とか『時間がかかる』とか言われる。でも、ビジネス的には早く出したい。そのバランスを取るのが難しいんです」

付箋がどんどん増えていく。ホワイトボードに貼られた付箋は、すでに30枚を超えていた。そして、まだ書き足りない様子のメンバーもいる。

湊は付箋を見つめながら、改めて問題の深刻さを実感していた。一人で抱え込んでいた問題が、実はチーム全体の課題だった。エンジニアだけの問題ではなく、QA部、プロダクト部、それぞれが同じような悩みを抱えていた。

「みんな、同じようなことで悩んでたんですね...」

湊は小さく呟いた。その言葉に、全員が深く頷いた。

痛みのグルーピング

付箋が30枚を超えたところで、湊は次のステップを提案した。

「じゃあ、これらの痛みを整理してみましょう。似た内容をグループに分けてみませんか？そうすれば、問題の全体像が見えてくると思います」

湊の提案で、メンバーたちは立ち上がり、ホワイトボードの前に集まった。それぞれが付箋を見つめながら、どこに分類すべきか考えている。

「この『仕様変更が多い』と、この『要件定義が曖昧』は、同じグループじゃない？」

佐藤が指差しながら言った。

「そうだね。それから、『見積もりが外れる』も、要件定義の問題かもしれない」

別のメンバーが続けた。

「じゃあ、この辺りに『要件定義・仕様変更』っていうグループを作ろうか」

山田が付箋を動かしながら言った。

「『技術的負債で新機能が作りにくい』と、『コードが複雑すぎる』は、同じグループだね。それから、『テスト時間が足りない』も、品質の問題だから、ここに入るかもしれない」

「でも、『テスト時間が足りない』は、時間の問題でもあるよね」

高橋が疑問を投げかけた。

「確かに。でも、根本的には品質保証の問題だと思う。時間が足りないのは結果であって、原因は別にあるかもしれない」

山田と高橋が議論を始めた。他のメンバーも、それぞれの意見を出し合いながら、付箋を動かしていく。

議論が深まる

「『残業が多い』と『時間が足りない』は、同じグループかな？」

「それと、『コードレビューする時間がない』も、時間の問題だよね」

「じゃあ、『時間・リソース不足』っていうグループを作ろう」

議論が進むにつれて、自然に5つのカテゴリが浮かび上がってきた。

「要件定義・仕様変更」

「技術的負債・品質」

「時間・リソース不足」

「コミュニケーション」

「評価・指標のズレ」

各カテゴリに分類された付箋を見て、メンバーたちは改めて問題の構造を理解していた。

飛鳥がホワイトボードを見つめながら言った。

「こうして見ると、構造的な問題が多いですね。個人的な努力では解決できない、組織レベルの課題だと思います。例えば、『要件定義・仕様変更』の問題は、PMだけの問題でも、エンジニアだけの問題でもない。プロセス全体の問題です」

「そうですね。『技術的負債・品質』も、エンジニアだけの問題ではなく、QA部やプロダクト部も関わってくる問題です」

高橋が続けた。

「『時間・リソース不足』も、個人の努力では解決できない。組織全体で時間の使い方を見直す必要があると思います」

湊は5つのカテゴリを見つめながら、改めて問題の深刻さを実感していた。これらは、すべて関連し合っている。一つの問題が、別の問題を引き起こしている。だからこそ、チーム全体で解決していく必要があるのだと。

形骸化した指標の発見

5つのカテゴリが整理された後、湊は「評価・指標のズレ」のグループを詳しく見てみることを提案した。

「このグループには、いくつか興味深い付箋があります。詳しく見てみませんか？」

メンバーたちは「評価・指標のズレ」のグループに集まった。そこには、5枚の付箋が貼られていた。

「ストーリーポイントが形骸化してる。見積もりを水増しする人もいるし、簡単なタスクばかり選ぶ人もいる」

佐藤がその付箋を指差しながら言った。

「これ、僕が書いたんです。先月のスプリント計画で、同じようなタスクばかり選んで、ストーリーポイントを稼いでいる人がいました。難しいタスクは誰も選ばない。それで、本当に必要な機能が後回しになってしまう」

「コード行数で評価されても...短く書くことが価値なのに、行数を増やすインセンティブが働く」

別のメンバーが続けた。

「以前、コード行数で評価されていた時期があったんです。それで、無駄なコードを書いて行数を増やす人が出てきました。リファクタリングしてコードを短くしたら、評価が下がってしまった。それ以来、リファクタリングする人が減りました」

「リリース頻度だけ見られても品質が...小さな変更を細かくリリースすれば数字は上がるけど、本当に価値があるのかわからない」

飛鳥が頷きながら言った。

「プロダクト部としても、リリース頻度を重視されることがあります。でも、小さな変更を細かくリリースしても、ユーザーには価値が伝わらない。むしろ、大きな機能を一度にリリースした方が、ユーザーには価値があることもあるんです」

湊はこれらの発言を聞きながら、数ヶ月前、田中部長から「開発生産性を上げてくれ」と言われた時のことを思い出していた。その時は、「どうやって数値化すればいいのか」と悩んでいた。でも、今は違う。数値化することが目的ではなく、本質的な価値を創出することが目的なのだと理解できた。

「これらの指標って、本質的な問題解決につながってないですよね」

湊の言葉に、全員が深く頷いた。

「そうそう、まさにそれ！」

「指標を上げることが目的になっちゃってる。本当に価値のあることをするよりも、数字を上げることの方が評価される」

「本当に必要なこと（リファクタリングとか、技術調査とか）が評価されない。だから、誰もやらなくなる」

山田が冷静に言った。

「グットハートの法則って知ってる？チャールズ・グッドハートさんが提唱した法則で、『指標が目標になると、その指標は機能しなくなる』というものだ。指標を上げるための行動が起きる。でも、それは本質的な改善とは違うんだよね」

「例えば、ストーリーポイントを上げるために、簡単なタスクばかり選ぶ。コード行数を増やすために、無駄なコードを書く。リリース頻度を上げるために、小さな変更を細かくリリースする。これらは、すべて指標を上げるための行動であって、本質的な改善ではない」

「キャベルの法則も同じ。状態が数値を作るのであって、数値が状態を作るわけではない。数値を高めると強い組織になるのか、強い組織の状態が数値を高めるのか、その違いを理解しないと、指標が形骸化してしまう」

湊は深く頷いた。数ヶ月前、「開発生産性をどう数値化すればいいのか」と悩んでいた自分を思い出していた。その時は、数値化することが目的だと思っていた。でも、今は違う。数値化することは手段であって、目的ではない。本質的な価値を創出することが目的なのだと理解できた。

「じゃあ、私たちは本当に何を大切にしたいんでしょうか？」

湊の問いかけに、会議室が再び静かになった。でも、今度は前向きな静けさだった。答えを探そうとする、そんな空気が漂っている。

解説：なぜ指標が形骸化するのか

ストーリーで描かれる「重圧」

「重圧から生み出される小手先の指標」が重圧になるのは、**課題が言語化される前に**数値目標だけが先行してしまうためです。湊のチームが直面したのも、この構造から生じています。

- **ワークショップのぎこちなさ** 最初の5分間は重い沈黙。付箋に「痛み」を書くと言われても遠慮がちで、当たり障りのない内容から始まる。チーム内のコミュニケーションの壁を感じている
- **形骸化した指標への気づき**　「ストーリーポイントが形骸化してる」「コード行数で評価されても…」「リリース頻度だけ見られても品質が…」とメンバーから声が上がる。評価・指標のズレが一つのカテゴリとして浮かび上がる
- **本質からの乖離**　これらの指標が本質的な問題解決につながっていないという共通認識が生まれる。指標を上げることが目的化し、本当に必要な改善が評価されないもどかしさ
- **重圧の正体**　重圧から生み出される「小手先の指標」に依存している現状。数

値目標が先行し、開発活動の複雑さを単一の数値では表現できていない

この重圧の背景には、**指標が目標化し本質的な価値創出から乖離する構造**があります。形骸化した指標を特定し、本質的な問題に目を向ける必要性に直面しているのです。

なぜ指標が形骸化するのか

湊のチームが発見した問題は、多くの開発組織が直面する典型的な課題です。

指標が形骸化する主な原因は以下の通りです

- **グットハートの法則**　指標が目標になると、指標を上げるための行動が起きる。本質的な改善ではなく、数字を上げることが目的化する
- **キャベルの法則**　状態が数値を作るのであって、数値が状態を作るわけではない。「数値を高めると強い組織になるのか、強い組織の状態が数値を高めるのか」を理解しないと、指標が形骸化する
- **指標の限界**　単一の数値では複雑な開発活動を表現できない。コード行数、コミット数、リリース頻度など、それぞれに限界がある
- **ゲーミフィケーション**　指標を達成することが目的化し、本質的な価値創出を見失う

小手先の指標の問題点

形骸化した指標は、以下の問題を引き起こします

- **ストーリーポイントの形骸化**　見積もりを水増しする、簡単なタスクばかり選ぶ、難しい課題に誰も取り組まなくなる
- **コード行数での評価**　冗長なコードを書くインセンティブが働く。短く書くことが価値なのに、行数を増やすことが評価される
- **リリース頻度での評価**　品質を犠牲にして小さな変更を細かくリリースする。本当に価値のある機能ではなく、数字を上げるためのリリースになる

- **本質的な問題解決を妨げる**　本当に必要な改善（リファクタリング、技術調査、ドキュメント作成）が評価されず、誰も取り組まなくなる

問題を特定するための手法

形骸化した指標を発見し、本質的な問題を特定するためには、以下の手法が有効です

- **指標がゲーム化されていないかチェック**　指標を上げるための行動が起きていないか、本質的な改善と乖離していないかを確認
- **指標と実際の価値創出の因果関係を確認**　指標が上がっても、実際の価値創出（ユーザー満足度、ビジネス成果）につながっているかを検証
- **多面的な指標の組み合わせ**　単一指標に依存せず、複数の指標を組み合わせて評価する
- **定期的な指標の見直し**　四半期ごとに指標の妥当性をレビューし、必要に応じて変更する

詳細な手法については、章末の「手法4 脱・形骸化指標リスト」を参照してください。

DORAの科学的アプローチとFour Keys

形骸化した指標の問題を解決するために、業界では科学的なアプローチが確立されています。DORA（DevOps Research and Assessment）は、Google Cloudが2014年から継続的に発行している調査レポートで、ソフトウェア開発のパフォーマンスを科学的に測定する手法を提供しています。

科学的アプローチとしてのDORA

DORAの成果をまとめた書籍『リーンとDevOpsの科学（Accelerate: The Science of Lean Software and DevOps）』が示すように、その根幹には「科学」があります。これは、曖昧な現象を厳密な手法で分析するアプローチを指します。

DORA以前、パフォーマンス測定はしばしばバグ曲線のような直感的な指標に依存していました。これに対しDORAは、コンセプトの定義、測定変数の特定、そして統計的分析という科学的リサーチ手法を導入しました。

この科学的アプローチから導き出された成果が、以下の4つの指標、通称「Four Keys」です。

Four Keys（4つの指標）

1. **デプロイの頻度（Deployment Frequency）** ソフトウェアを本番環境にリリースする頻度。組織の「速度」を示す指標です。
2. **変更のリードタイム（Lead Time for Changes）** コードのコミットから本番デプロイまでの時間。同じく「速度」を示す指標です。
3. **変更失敗率（Change Failure Rate）** 本番デプロイが原因で障害を引き起こす割合。「安定性」を示す指標です。
4. **サービス復元時間（Time to Restore Service）** 本番障害からサービスを復旧させるまでにかかる時間。同じく「安定性」を示す指標です。

速度と安定性の両立

Four Keysは、開発チームのパフォーマンスを客観的に測定し、改善の方向性を議論するための「共通言語」となりました。そしてDORAレポートは、「**速度と安定性はトレードオフではない**」という、業界の常識を覆す事実を統計的に証明しました。高パフォーマンスな組織は、速度と安定性の両方を高いレベルで実現していたのです。

DORA Core Modelとの相関

DORAの価値は指標の提示に留まらず、継続的デリバリーやバージョン管理といった具体的なプラクティス群を定義した **DORA Core Model（DORAコアモデル）** の獲得が、Four Keysのスコア向上と統計的に強い相関関係にあることを示

しました。これにより、組織は単に数値を眺めるだけでなく、「どの能力を身につければパフォーマンスが向上するのか」という具体的な改善アクションへと繋げることが可能になりました。

Four Keysは、形骸化した指標とは対照的に、本質的な価値創出につながる指標として機能します。コード行数やコミット数といった表面的な指標ではなく、実際のビジネス価値（速度と安定性）を測定することで、チームの健全性と成果を同時に評価できるのです。

> ***出典*** *DORA（DevOps Research and Assessment）レポート、『Accelerate: The Science of Lean Software and DevOps』（Gene Kim, Jez Humble, Nicole Forsgren著）*

4-2 それぞれの立場から見た「痛み」を共有し理解する対話

痛みの共有と相互理解

形骸化した指標について議論した後、湊は各職種の痛みをもっと詳しく聞いてみることにした。

「それぞれの立場で、どんな痛みを感じているか、もう少し詳しく聞かせてください。エンジニア、QA、PM、それぞれの視点で、問題をどう見ているか知りたいんです」

ワークショップが進むにつれて、各職種の痛みが明確になってきた。

高橋が深く息を吸い込んでから、話し始めた。

「バグが多いと結局後で工数が10倍かかる。要件定義段階が1倍、実装段階で10倍、リリース後は30-100倍。品質も生産性のうちだと思うんですけどね。テスト時間を削ってリリースを早めても、結局後でバグ対応に時間がかかって、トータルでは時間がかかってしまう」

高橋の言葉に、エンジニアのメンバーたちも頷いた。誰もが同じ経験をしていた。

山田が少し考えてから、口を開いた。

「高橋さんの言う通り、品質も生産性のうちだと思います。ただ、品質と言っても多面的な側面があります。この前見たレポートの中でISO/IEC　25010という国際規格があり、ソフトウェア品質を8つの特性に分類できるらしいです。機能適合性、性能効率性、信頼性、保守性など、それぞれが重要な側面です。今のチームの課題を整理する際も、このような多面的な視点で品質を捉える必要があるかもしれませんね」

山田の言葉に、高橋も深く頷いた。

「プロダクト部として、ユーザー価値を届けたいんです。でも、開発速度が遅いと感じることもあって...技術的負債の重要性は理解したいんですが、ビジネス的なプレッシャーもあって」

飛鳥が頷きながら続けた。

「実は私も悩んでる。短期的な売上は大事だけど、長期的な持続可能性も大事。そのバランスが難しいんです。経営陣からは『もっと早く機能を出せ』と言われる。でも、開発チームからは『技術的負債を返済する時間が必要』と言われる。どちらも正しいと思うんですが、どうバランスを取ればいいのかわからない」

「それから、ユーザーからの要望も多いんです。『この機能を追加してほしい』『あの機能を改善してほしい』。それらを全部開発チームに伝えると、『優先順位をつけてほしい』と言われる。でも、ビジネス的には全部重要に見える。その優先順位をつけるのが難しいんです」

飛鳥の言葉に、山田が反応した。

「技術的な視点から言うと、技術的負債を無視して新機能を追加し続けると、開発速度はどんどん低下する。過去のデータを見ると、技術的負債の蓄積で開発速度が年間20-40%低下している。今は3日でできる機能も、1年後には1週間かかるようになるかもしれない」

「でも、それをどうビジネス側に説明すればいいのか。技術用語で説明しても理解してもらえない。『技術的負債』って言葉自体が、ビジネス側には伝わりにくい。『コードが複雑になっている』と言っても、それがどうビジネスに影響するのか、説明するのが難しい」

山田の言葉に、飛鳥が頷いた。

「確かに技術的な話をされても、私には理解できないことが多いです。でも、それがビジネスにどう影響するのか、もっとわかりやすく説明してもらえれば、理解できると思います」

湊はメンバーの話を聞きながら、数週間前、各職種にヒアリングした時のことを思い出していた。それぞれの立場で「生産性」の定義が違う。でも、それは対立の原因ではなく、相互理解の出発点なのだと気づいた。

各職種の痛みが交差する

「みんな、それぞれの立場で正しいことを言ってるんですよね。エンジニアは技術的品質を重視する。QAは品質保証を重視する。PMはユーザー価値を重視する。でも、それがバラバラだから問題が起きる」

湊は少し間を置いてから続けた。

「でも、今日のワークショップで気づいたことがあります。それぞれの立場は違うけど、感じている痛みは共通している。『時間が足りない』『品質を保ちたい』『ユーザーに価値を届けたい』。これらは、すべての職種が感じていることです」

「じゃあ、どうすればいいんでしょうか？」

佐藤が不安そうに聞いた。

「まずは、お互いの痛みを理解することからだと思います。エンジニアの痛み、QAの痛み、PMの痛み。それぞれを理解して、共通の課題を見つける。そして、その共通の課題を、チーム全体で解決していく」

湊はホワイトボードの5つのカテゴリを指差しながら言った。

「これが、私たちの共通の課題です。『要件定義・仕様変更』『技術的負債・品質』『時間・リソース不足』『コミュニケーション』『評価・指標のズレ』。これらは、エンジニアだけの問題でも、QAだけの問題でも、PMだけの問題でもない。チーム全体の問題です。だからこそ、チーム全体で解決していく必要があると思います」

湊は各カテゴリを順番に指差しながら説明した。

「『要件定義・仕様変更』の問題は、PMだけの問題でも、エンジニアだけの問題でもない。プロセス全体の問題です。要件定義の段階で、PM、エンジニア、QAが一緒にレビューすれば、仕様変更を減らせるかもしれません」

「『技術的負債・品質』の問題も、エンジニアだけの問題ではありません。技術的負債が蓄積すると、開発速度が低下し、結果としてユーザーに価値を届けるのが遅くなる。それは、PMにとっても、QAにとっても、問題です」

「『時間・リソース不足』も、個人の努力では解決できません。組織全体で時間の使い方を見直す必要があります。コードレビューの時間を確保する、テスト時間を確保する、リファクタリングの時間を確保する。これらは、すべて組織レベルの意思決定が必要です」

「『コミュニケーション』の問題も、チーム全体の問題です。エンジニアとPM、エンジニアとQA、それぞれの間で情報が共有されていない。だから、認識のずれが起きる。定期的な対話の場を作ることで、改善できるかもしれません」

「そして、『評価・指標のズレ』の問題。これは、私たちが今日発見した問題です。形骸化した指標に依存せず、本質的な価値を創出することを目指す」

湊の説明に、メンバーたちは真剣に聞き入っていた。それぞれが、自分の問題だと思っていたことが、実はチーム全体の問題だったのだと理解していた。

「じゃあ、これからどうすればいいんでしょうか？」

佐藤が質問した。

「まずは、これらの課題を一つずつ解決していくことだと思います。でも、全部を一度に解決するのは難しい。優先順位をつけて、小さなステップから始める。そして、定期的に振り返って、改善していく」

湊はメンバーたちを見回しながら続けた。

「今日は、痛みを共有して、共通の課題を見つけることができました。これが第一歩です。次は、この課題をどう解決していくか、一緒に考えていきましょう」

全員が頷いた。一人で抱え込んでいた問題が、チーム全体の課題として共有された。そして、それをチーム全体で解決していく決意が、全員の間に生まれていた。

解説：なぜ痛みの共有をするのか

ストーリーで描かれる「重圧」

痛みが表に出る前に重くのしかかるのが、「痛みを共有するまでに越える壁」です。湊のチームが経験した重圧も、同じ構造から生じています。

- **本音が出るまでの壁**　高橋の「テスト時間が全然足りません」という一言をきっかけに、山田や佐藤らも本音を話し始める。それまで表に出ていなかった痛みが付箋で可視化され、一気に増えていく
- **各職種の痛みの可視化**　技術的負債で身動きが取れない、見積もりが外れる理

由がわからない、コードレビューの時間がない、モジュール間の複雑さ、属人化した設計。要件定義・仕様変更、技術的負債・品質、時間・リソース不足、コミュニケーション、評価・指標のズレの5カテゴリに整理される

- **「みんな同じことで悩んでた」**　バラバラに見えた問題が構造的な課題として共有される。一人で抱え込んでいた問題がチーム全体の課題になり、対立ではなく協働の土台ができる
- **痛みの共有がもたらす転換**　本音を言えない環境から、痛みを出し合うことで問題の全体像が見え、共通の課題を発見する段階に進んでいる

この重圧の背景には、**情報の非対称性や心理的安全性の欠如で痛みが表面化せず、共有することで初めて共通課題と協働の基盤が生まれる構造**があります。痛みの共有という第一歩を踏み出した段階にあるのです。

なぜ痛みの共有をするのか

湊のチームが経験したように、各職種の痛みを共有することは、チーム全体での問題解決の第一歩です。

痛みを共有することで、以下の効果が得られます

- **問題の全体像が見える**　各職種が個別に感じている問題が、実は構造的な課題であることがわかる
- **相互理解が深まる**　エンジニア、QA、PMそれぞれの立場での苦労を理解できる
- **共通の課題を発見**　バラバラに見える問題が、実は同じ根本原因から生まれていることがわかる
- **協働の基盤ができる**　お互いの痛みを理解することで、対立ではなく協働の関係を築ける

各職種の視点の違い

各職種は、以下のような視点で開発生産性を見ています

- **エンジニア** 技術的負債、コードの品質、開発速度、持続可能性
- **QA** テスト時間、品質保証、バグの少なさ、検証の十分性
- **PM** ユーザー価値、市場対応速度、機能リリース、ビジネス成果
- **管理職** 売上貢献、コスト効率、KPI達成、経営層への報告

それぞれが正しい視点を持っていますが、視点が異なることで対立が生まれることがあります。

対話による相互理解の重要性

対話を通じて相互理解を深めることで、以下の効果が得られます

- **共通の課題を発見** 各職種の痛みを整理することで、共通の課題が見えてくる
- **協働の基盤を構築**　お互いの立場を理解することで、協働して問題を解決できる
- **バランスの取れた判断**　各職種の視点を統合することで、短期的成果と長期的健全性のバランスを取れる
- **チーム全体での問題解決**　一人で抱え込むのではなく、チーム全体で問題を共有し、解決に取り組める

詳細な手法については、章末の「手法4 脱・形骸化指標リスト」を参照してください。

ソフトウェア品質特性（ISO/IEC 25010:2011）による品質の多面的理解

ワークショップで山田が言及したISO/IEC　25010は、ソフトウェア品質を体系的に理解するための国際規格です。品質を単一の指標で測るのではなく、8つの品質特性に分類することで、多面的に評価できます。

ISO/IEC 25010:2011の8つの品質特性

1. **機能適合性（Functional Suitability）**

- 機能の完全性、正確性、適切性
- ユーザーが求める機能が正しく実装されているか
- （例）要件定義で定義された機能が全て実装されているか

2. **性能効率性（Performance Efficiency）**
    - 時間特性、リソース利用効率性、容量性
    - システムが効率的に動作するか
    - （例）レスポンスタイム、メモリ使用量、スループット
3. **互換性（Compatibility）**
    - 共存性、相互運用性
    - 他のシステムや環境と共存・連携できるか
    - （例）異なるOSやブラウザでの動作、APIとの連携
4. **使用性（Usability）**
    - 認識しやすさ、学習しやすさ、運用しやすさ、ユーザーエラー防止、ユーザーインターフェースの美的品質、アクセシビリティ
    - ユーザーが使いやすいか
    - （例）UIの直感性、操作の習得しやすさ、エラーメッセージの分かりやすさ
5. **信頼性（Reliability）**
    - 成熟性、可用性、障害許容性、回復性
    - システムが安定して動作するか
    - （例）ダウンタイムの少なさ、障害発生時の自動復旧
6. **セキュリティ（Security）**

- 機密性、完全性、否認防止性、責任追跡性、真正性
- セキュリティリスクに対処できるか
- （例）データの暗号化、アクセス制御、ログの記録

7. **保守性（Maintainability）**
    - モジュール性、再利用性、解析しやすさ、修正しやすさ、テストしやすさ
    - システムを維持・改善しやすいか
    - （例）コードの可読性、テストカバレッジ、ドキュメントの充実度
8. **移植性（Portability）**
    - 適応性、インストール性、置換性
    - 異なる環境に移行しやすいか
    - （例）クラウド移行の容易さ、コンテナ化の対応

品質特性を活用した課題整理

ワークショップで発見された「技術的負債・品質」カテゴリの課題を、ISO/IEC 25010の品質特性で整理すると、以下のようになります。

- **保守性の課題**　コードの複雑度が高く、修正しにくい。テストが不十分で、テストしにくい
- **信頼性の課題** バグが多く、障害が発生しやすい。障害発生時の回復が遅い
- **性能効率性の課題** ビルド時間が長い、レスポンスが遅い
- **セキュリティの課題**　ライブラリのバージョンが古く、セキュリティパッチが適用されていない

このように品質特性で整理することで、課題を体系的に把握し、優先順位をつけやすくなります。

品質特性と開発生産性の関係

各品質特性は、開発生産性に直接影響します。

- **保守性が低い** コードの修正に時間がかかり、開発速度が低下する
- **信頼性が低い** バグ対応に時間が取られ、新機能開発が後回しになる
- **性能効率性が低い**　ビルドやテストの実行時間が長く、開発サイクルが遅くなる
- **セキュリティが低い**　セキュリティ対応に時間がかかり、開発リソースが分散する

つまり品質特性への投資は、短期的には開発速度を低下させるように見えますが、長期的には開発生産性の向上につながります。

品質特性を活用した改善計画

品質特性を活用することで、以下のような改善計画を立てられます。

1. **現状の把握** 各品質特性の現状を評価し、課題を特定する
2. **優先順位の決定** 開発生産性への影響が大きい品質特性から優先的に改善する
3. **改善目標の設定** 各品質特性について、具体的な改善目標を設定する
4. **効果測定** 改善後の品質特性を測定し、開発生産性への影響を評価する

このように、ISO/IEC　25010の品質特性を活用することで、品質を多面的に評価し、開発生産性の向上につなげることができます。

4-3 チーム全体で合意できる本質的な価値の発見

本質的な価値の模索

痛みを共有し、共通の課題を発見した後、湊は次のステップを提案した。

「じゃあ、私たちは本当は何を大切にしたいんだろう？形骸化した指標ではなく、本質的な価値について考えてみませんか？」

ブレインストーミングの熱気

ブレインストーミングが始まった。

「ユーザーに価値を届けること。機能の数ではなく、ユーザーの問題を解決すること」

飛鳥が最初に言った。

「持続可能な開発速度。短期的なスピードではなく、長期的に維持できる速度」

山田が続けた。

「チーム全員が成長できる環境。学習機会、心理的安全性、相互支援」

高橋が加わった。

「品質を保ちながらスピードも出す。トレードオフではなく、両立を目指す」

佐藤も小さく言った。

「コードレビューする時間がある。急がず、丁寧に開発できる環境」

他のメンバーも次々と意見を出した。

「技術的負債を適切に管理する。後回しにせず、計画的に返済する」

「要件定義に十分な時間をかける。仕様変更を最小限に抑える」

「コミュニケーションが活発。情報が共有され、認識のずれが起きない」

山田が少し考えてから、別の観点を追加した。

「事業への貢献が実証されているシステムこそ、ドメイン駆動設計で取り組む価値が最も高い。このシステムは、すでにユーザーに価値を提供している。だからこそ、長期的な持続可能性を確保するために、ドメインモデルを明確にし、イベントストーミングで設計を整理する必要がある。積み重なった技術的負債と膨れ上がった設計の乱雑さを刷新することで、将来の開発速度を大幅に向上させられる」

ホワイトボードには、本質的な価値が並んでいた。

湊はそれらを見つめながら言った。

「これが私たちの『本当の生産性』なのかもしれませんね。数値ではなく、価値創出と持続可能性。それが、私たちが目指すべきものだと思います」

アクションプランの策定

本質的な価値が明確になった後、湊は次のステップを提案した。

「じゃあ、これらの価値を実現するために、具体的に何をすればいいでしょうか？各カテゴリに対する改善アクションを考えてみませんか？」

メンバーたちは、5つのカテゴリを見ながら改善策を考え始めた。

「要件定義・仕様変更」について、飛鳥が提案した。

「事前レビュー会を導入してはどうでしょうか？PM、エンジニア、QAの3者で、実装前に要件をレビューする。Three Amigosの手法です」

「技術的負債・品質」について、山田が提案した。

「週に1回、リファクタリングの時間を確保する。20%ルールです。開発時間の20%を技術的負債の返済に充てる」

「品質保証の仕組み作り」について、高橋が提案した。

「自動テストを導入する。CI/CDパイプラインで自動テストを実行して、品質を担保する」

「チーム内コミュニケーション改善」について、湊が提案した。

「朝会を導入する。毎朝15分、チーム全員で進捗を共有する。デイリースタンドアップです」

4つの重点アクションが決まった。

1. 要件定義プロセスの改善（飛鳥・プロダクト部担当）
2. 技術的負債返済時間の確保（テックリードの山田担当）
3. 品質保証の仕組み作り（QA部の高橋担当）
4. チーム内コミュニケーション改善（湊リーダー担当）

それぞれ2週間後に進捗報告をすることになった。

「よし、やってみよう！」

全員が前向きに頷いた。一人で抱え込んでいた問題が、チーム全体の課題として共有され、解決に向けて動き始めた。

アクションプランの実行と成果

2週間後、フォローアップミーティングが開催された。各担当者から、試行錯誤を経て得られた成果が報告された。

飛鳥が最初に報告した。

「要件定義の事前レビュー会を導入しました。最初はエンジニアとQAのスケジュールが合わなくて大変でしたが、続けるうちにコツがつかめてきました。仕様変更が週に3-5回だったのが、週に1-2回に減りました」

続けて、山田が続けた。

「週1回、リファクタリングの時間を確保しました。毎週金曜日の午後2時間。最初の週は緊急のバグ対応が入って時間が取れませんでしたが、2週目からは守れるようになりました。まだ大きな成果は出ていませんが、継続的に改善していく基盤はできました」

高橋も報告した。

「自動テストの導入を開始しました。CI/CDパイプラインで自動テストを実行するようにしました。最初はテストが失敗してデプロイが止まってしまいましたが、エンジニアのメンバーが協力して修正できました。まだカバレッジは低いですが、徐々に増やしていく予定です」

湊が最後に報告した。

「朝会を導入しました。毎朝15分、チーム全員で進捗を共有します。最初は時間がかかっていましたが、タイマーを使い、『昨日やったこと、今日やること、困っていること』の3つに絞ることで、15分で終わるようになりました。情報の透明性が高まり、認識のずれが減りました」

小さいながらも確実な改善を実感していた。

「少しずつだけど、変わってきてますね」

湊はメンバーたちを見ながら言った。

「一人で抱え込んでいた時は、何も変えられなかった。でも、チーム全体で考えれば、きっと道は開ける」

山田が微笑みながら言った。

「湊さん、君が種を蒔いた木が、少しずつ育ってきてるね」

解説：本質的な価値とは何か

ストーリーで描かれる「重圧」

「本当に何を大切にしたいか」と問い詰められた経験があるなら、それは**本質的な価値に目を向け始めた**段階で現れる構造的な課題の表れかもしれません。湊のチームもそこに立っています。

- **「本当に何を大切にしたいか」**　痛みのグルーピングと形骸化した指標への気づきの先に、チームで本質的な価値を模索する問いが立つ。ユーザーに価値を届けること、持続可能な開発速度、チーム全員が成長できる環境、品質を保ちながらスピードも出す
- **数値では測れない価値**　ストーリーポイントやコード行数では表せない「本当の生産性」を言語化する。事業に貢献しているシステムだからこそドメインモデルを明確にし、技術的負債と設計の乱雑さを刷新する必要性が共有される
- **アクションへの落とし込み** 4つの重点アクション（要件定義プロセスの改善、技術的負債返済時間の確保、品質保証の仕組み作り、チーム内コミュニケーション改善）を担当者と期限で決め、全員が「よし、やってみよう」と前向きになる
- **最初の成果**　2週間後のフォローアップで、事前レビュー会の導入、週1回のリファクタリング時間確保、自動テスト導入の開始、朝会での情報共有改善など、小さくとも確かな改善が報告される

この重圧の背景には、**形骸化した指標に依存せず、数値では測れない本質的な価値をチームで合意し、アクションプランに落とすことで継続的改善が動き始める構造**があります。転換を経て、協働による改善の第一歩を踏み出しているのです。

本質的な価値とは何か

湊のチームが発見した本質的な価値は、多くの開発チームが目指すべき目標です。

本質的な価値とは、数値では測れない、しかし開発チームが目指すべき価値です

- **ユーザーに価値を届けること**　機能の数ではなく、ユーザーの問題を解決すること。ユーザー満足度、ビジネス成果につながる価値
- **持続可能な開発速度**　短期的なスピードではなく、長期的に維持できる速度。技術的負債を適切に管理し、チームの健全性を保つ
- **チーム全員が成長できる環境**　学習機会、心理的安全性、相互支援。チームメンバーが成長することで、長期的な生産性が向上する
- **品質を保ちながらスピードも出す**　トレードオフではなく、両立を目指す。自動テスト、CI/CD、コードレビューなど、品質を担保する仕組みを整える

チーム全体での合意形成の重要性

チーム全体で本質的な価値に合意することで、以下の効果が得られます

- **共通の目標を持つ** チームの方向性が明確になり、個々の判断が統一される
- **各職種の視点を統合**　　エンジニア、QA、PMそれぞれの視点を統合することで、バランスの取れた判断ができる
- **協働の基盤を構築**　共通の価値観を持つことで、対立ではなく協働の関係を築ける
- **継続的な改善**　アクションプランを実行し、定期的に振り返ることで、継続的な改善が可能になる

アクションプランの実行と継続的改善

アクションプランを実行し、継続的に改善するためには、以下のポイントを押さえます

- **小さなステップから始める** 大きな変化ではなく、小さな改善から始める。2週間、1ヶ月という短いサイクルで進捗を確認する

- **定期的な振り返り**　四半期ごとに、アクションプランの効果を振り返り、必要に応じて調整する
- **成果の可視化**　小さな成果でも可視化し、チーム全体で共有する。モチベーションの維持につながる
- **継続的な改善**　一度の改善で終わらず、継続的に改善を続ける。開発生産性の向上は、継続的な取り組みによって実現される

詳細な手法については、章末の「手法4 脱・形骸化指標リスト」を参照してください。

手法4 脱・形骸化指標リスト

形骸化した指標のチェックリスト

以下の項目をチェックしてください。該当する項目が多いほど、指標が形骸化している可能性が高いです。

指標のゲーム化

- [ ] 指標を上げるための行動が起きている（例）コードを細かく分割してコミット数を増やす
- [ ] 簡単なタスクばかり選ぶ人がいる
- [ ] 見積もりを水増しする人がいる
- [ ] 本当に必要な仕事（リファクタリング、技術調査）が評価されない

指標と実際の価値創出の乖離

- [ ] 指標が上がっても、ユーザー満足度が上がらない

- [ ] 指標が上がっても、ビジネス成果につながらない
- [ ] 指標が上がっても、チームのモチベーションが上がらない
- [ ] 指標が上がっても、品質が向上しない

単一指標への依存

- [ ] 1つの指標（例えばコード行数、コミット数、リリース頻度）だけで評価している
- [ ] 指標の妥当性を定期的に見直していない
- [ ] 指標の意味をチーム全体で理解していない
- [ ] 指標が改善のツールではなく、評価のためだけに使われている

判定

- 0-2個該当 良好な状態。指標が適切に機能している
- 3-5個該当 注意が必要。指標の見直しを検討する
- 6-8個該当 危険な状態。指標が形骸化している可能性が高い
- 9-12個該当 深刻な状態。指標が逆効果になっている

本質的な価値を測る指標の選び方

形骸化した指標ではなく、本質的な価値を測る指標を選ぶためには、以下のポイントを考慮してください。

アウトカム指標を重視する

- **アウトプットではなく、アウトカムを測定**　コード行数、コミット数ではなく、ユーザー満足度、ビジネス成果を測定する
- **複数の指標を組み合わせる**　単一指標に依存せず、複数の指標を組み合わせて評価する

- **定性的な評価も併用**　数値では測れない価値（チームの健全性、心理的安全性）も評価する

DORA Metricsの活用

DORA Metrics（Four Keys）は、形骸化した指標とは対照的に、本質的な価値創出につながる指標として機能します。以下の4つの指標を組み合わせることで、バランスの取れた評価が可能になります。

Four Keysの詳細

1. **デプロイ頻度（Deployment Frequency）**
    - **測定方法** 本番環境へのリリース回数を期間で割る（例）週1回、月1回
    - **意味** 組織がどれだけ迅速に価値を届けられるかを示す
    - **注意点** 頻度だけを追うのではなく、価値のある変更かどうかを確認する
2. **変更のリードタイム（Lead Time for Changes）**
    - **測定方法** コードコミットから本番デプロイまでの時間を測定
    - **意味** アイデアから価値提供までの時間を示す
    - **注意点** リードタイムが短いほど、市場への対応速度が上がる
3. **変更失敗率（Change Failure Rate）**
    - **測定方法** 本番デプロイ後に障害が発生した回数を全デプロイ回数で割る
    - **意味** デリバリーの品質を示す
    - **注意点** 失敗率が低いほど、安定したデリバリーができている
4. **サービス復元時間（Time to Restore Service）**
    - **測定方法** 本番障害の発生から復旧までの時間を測定
    - **意味** 障害発生時の対応能力を示す

- **注意点** 復旧時間が短いほど、ユーザーへの影響を最小化できる

DORA Metricsの活用方法

- **チーム単位で測定** 個人評価ではなく、チーム全体の成果として測定する
- **定期的なレビュー** 四半期ごとに指標を確認し、改善の方向性を議論する
- **DORA Core Modelとの連動** 継続的デリバリー、バージョン管理などのプラクティスと連動させて改善する
- **ビジネス指標との関連付け** DORA Metricsの改善が、顧客満足度や売上などのビジネス指標にどう影響するかを追跡する

これらの指標を組み合わせることで、速度と安定性の両立を実現し、本質的な価値創出につながる評価が可能になります。

> ***出典*** *DORA（DevOps Research and Assessment）レポート、『Accelerate: The Science of Lean Software and DevOps』（Gene Kim, Jez Humble, Nicole Forsgren著）*

チーム単位で測定する

- **個人評価には使わない** 指標を個人の評価に使うと、ゲーム化が起きやすい
- **チーム単位で測定** チーム全体の成果として測定し、改善のツールとして活用する
- **定期的に見直す** 四半期ごとに指標の妥当性をレビューし、必要に応じて変更する

指標の定期見直しプロセス

指標を形骸化させないためには、定期的な見直しが必要です。

四半期ごとのレビュー

1. **指標の妥当性を確認** 指標が本質的な価値創出につながっているかを確認
2. **ゲーム化のチェック** 指標を上げるための行動が起きていないかを確認
3. **指標の変更・追加**　必要に応じて指標を変更したり、新しい指標を追加したりする
4. **チーム全体での合意** 指標の変更について、チーム全体で合意を形成する

指標の見直しチェックリスト

- [ ] 指標が本質的な価値創出につながっているか
- [ ] 指標がゲーム化されていないか
- [ ] 指標と実際の価値創出の因果関係が明確か
- [ ] チーム全体が指標の意味を理解しているか
- [ ] 指標が改善のツールとして機能しているか

　これらのチェックリストを定期的に確認することで、指標の形骸化を防ぐことができます。

章のまとめ

　本章では、湊のチームがワークショップを通じて、形骸化した指標の問題に気づき、本質的な価値を発見する過程を描きました。

主な学び

1. **指標の形骸化**　グットハートの法則、キャベルの法則により、指標が目標になると本質を見失う
2. **痛みの共有** 各職種の痛みを共有することで、共通の課題を発見できる

3. **本質的な価値** 数値ではなく、価値創出と持続可能性が、本当の生産性である
4. **チーム全体での問題解決**　一人で抱え込むのではなく、チーム全体で問題を共有し、解決に取り組む

次章への橋渡し

　本章でチーム全体での問題解決を描いた。次の章では、AIとの協働による持続可能な開発へと進む。