

モノリスの認知負荷に立ち向かう

コードの所有者という思想と現実

自己紹介

- 前田 和樹 (kzk_maeda)
- atama plus株式会社 VPoE / 技術フェロー
 - データや機械学習、生成AIに関する開発
- AWS Community Builder / AWS Startup Community Core Member



アジェンダ

- | 事業成長に伴い直面した開発課題
- | 「コードを所有する」という考え方と取り組み
- | 「コードを所有する」状態と組織柔軟性の課題
- | 所有者運用と組織柔軟性の最大公約数
- | まとめ

アジェンダ

事業成長に伴い直面した開発課題

「コードを所有する」という考え方と取り組み

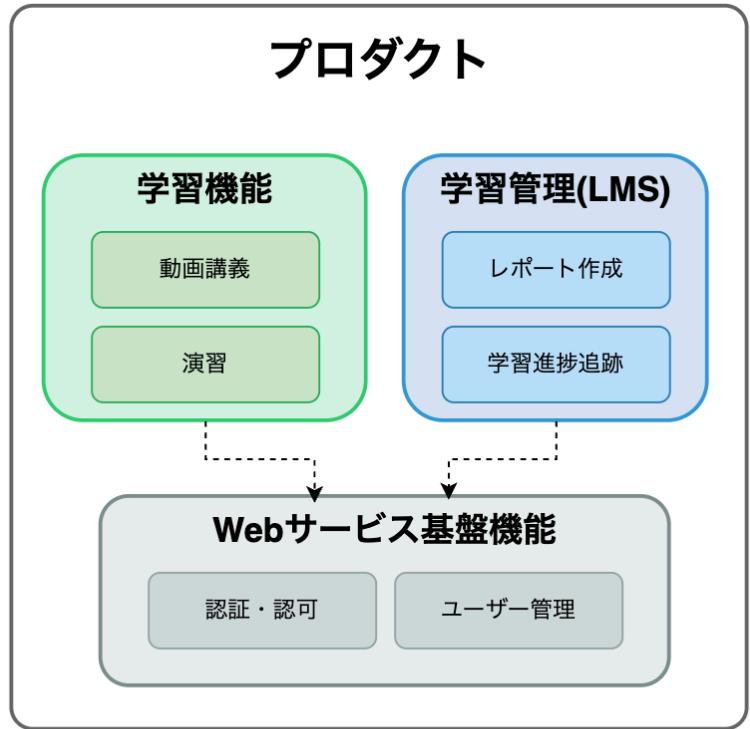
「コードを所有する」状態と組織柔軟性の課題

所有者運用と組織柔軟性の最大公約数

まとめ

背景：スタートアップの成長

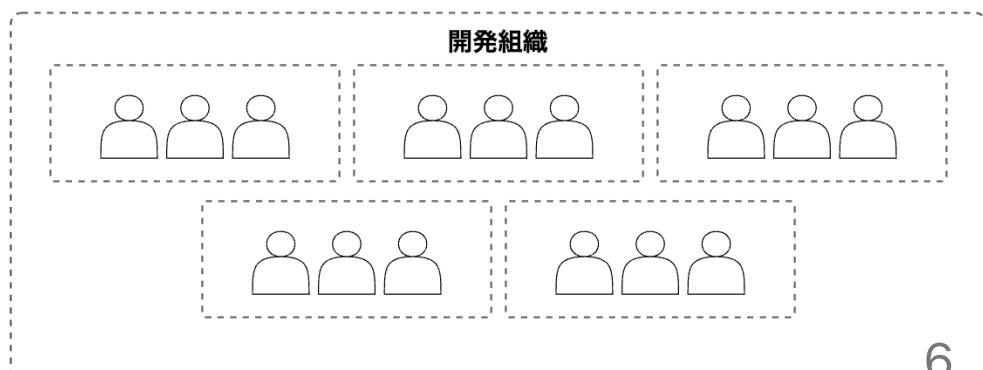
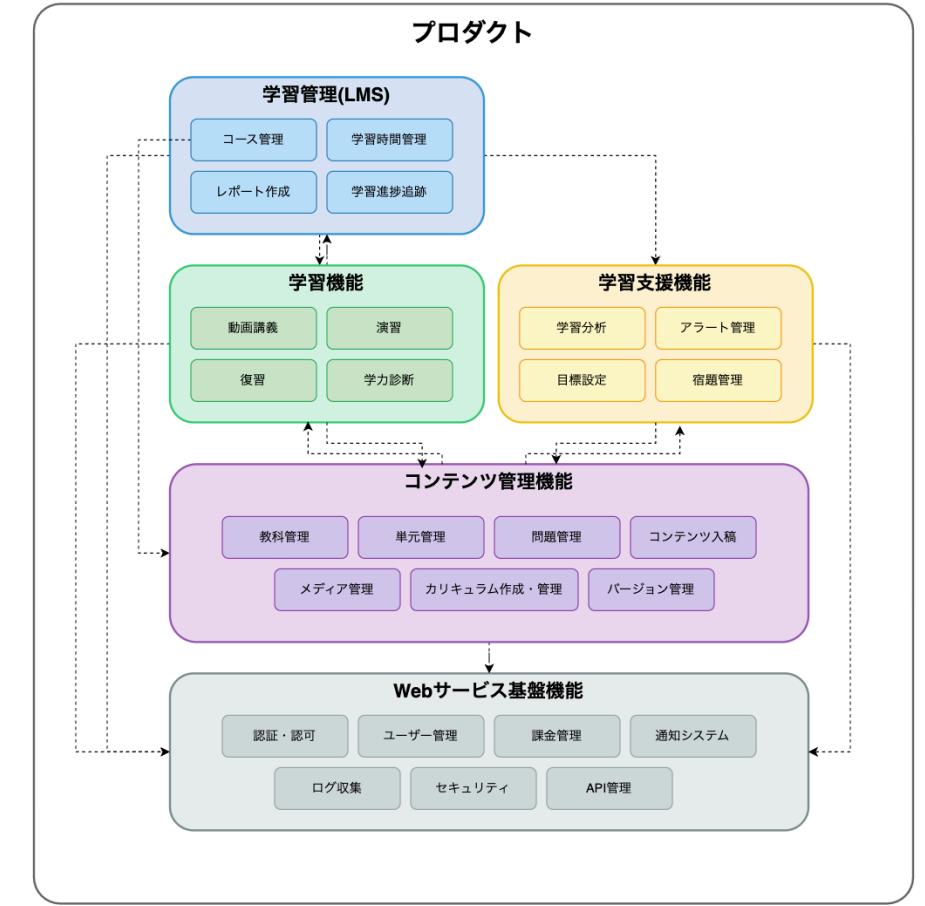
- ・シンプルなプロダクト・小さなチームからスタート



モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

背景：スタートアップの成長

- 事業成長に伴い、機能・エンジニアが増加
- 一方、コードベースはモノリシックアーキテクチャを維持
 - 単純なモノリスではなく、モジュラーモノリスを志向した成長
 - 分割するコストと時間の制約
 - ビジネス成長スピードの優先

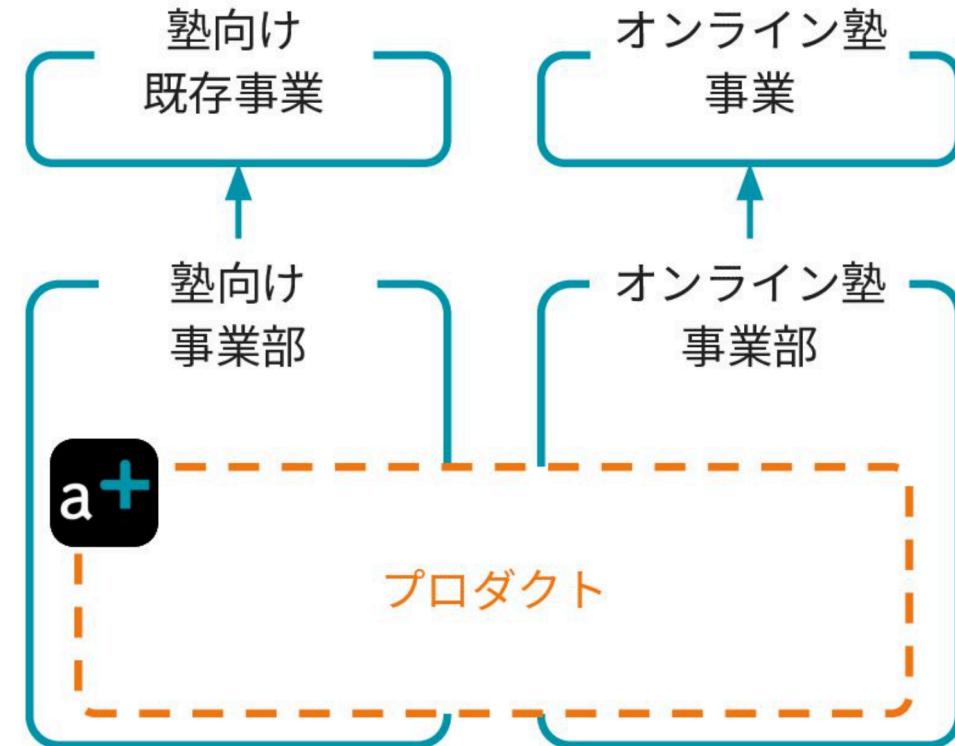


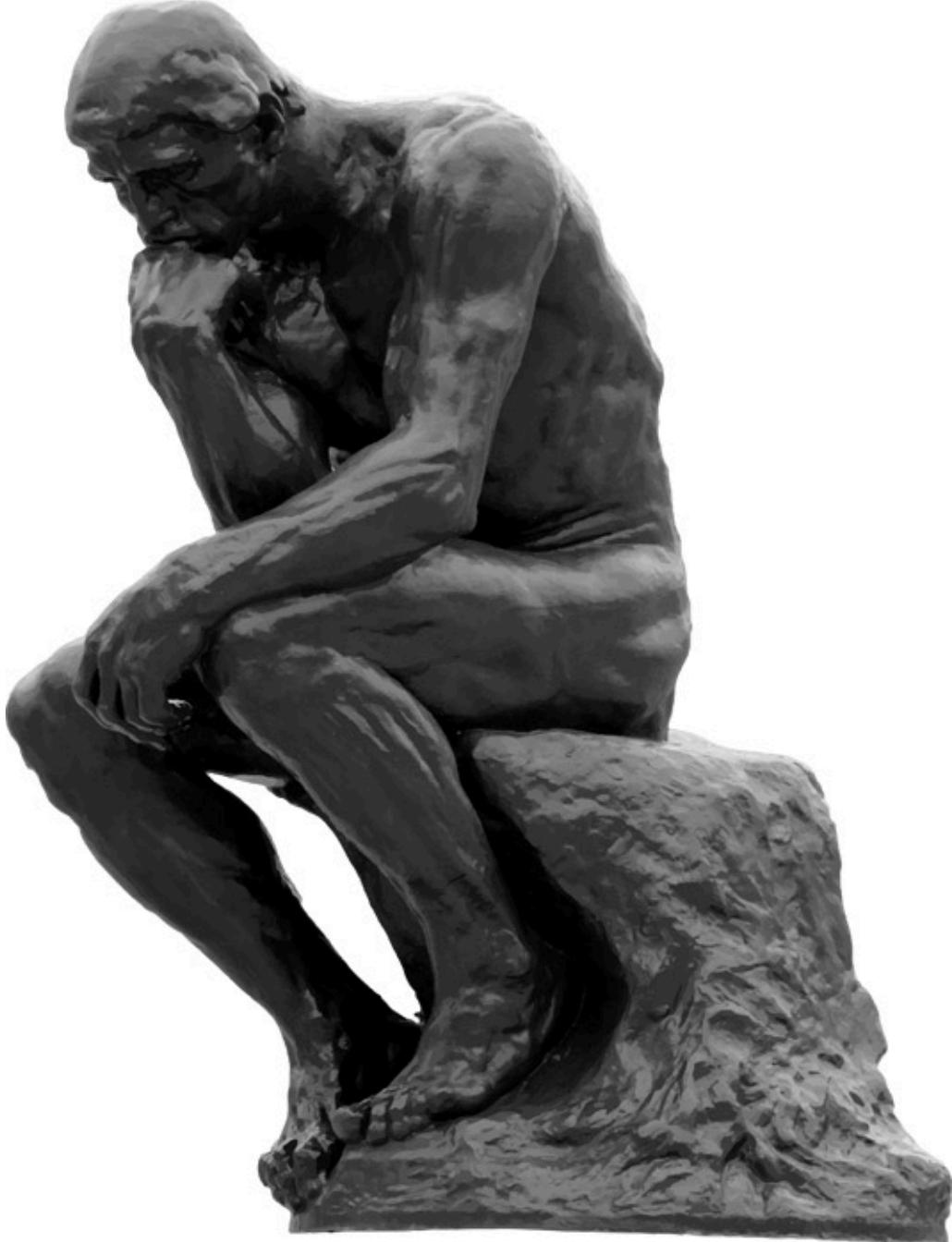
モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

2022年：事業の多角展

開

- 事業の多角化戦略
 - 同一のコードベースで複数の顧客・事業への展開
- 短期での事業立ち上げを最小の痛みで実現する必要性





直面した課題

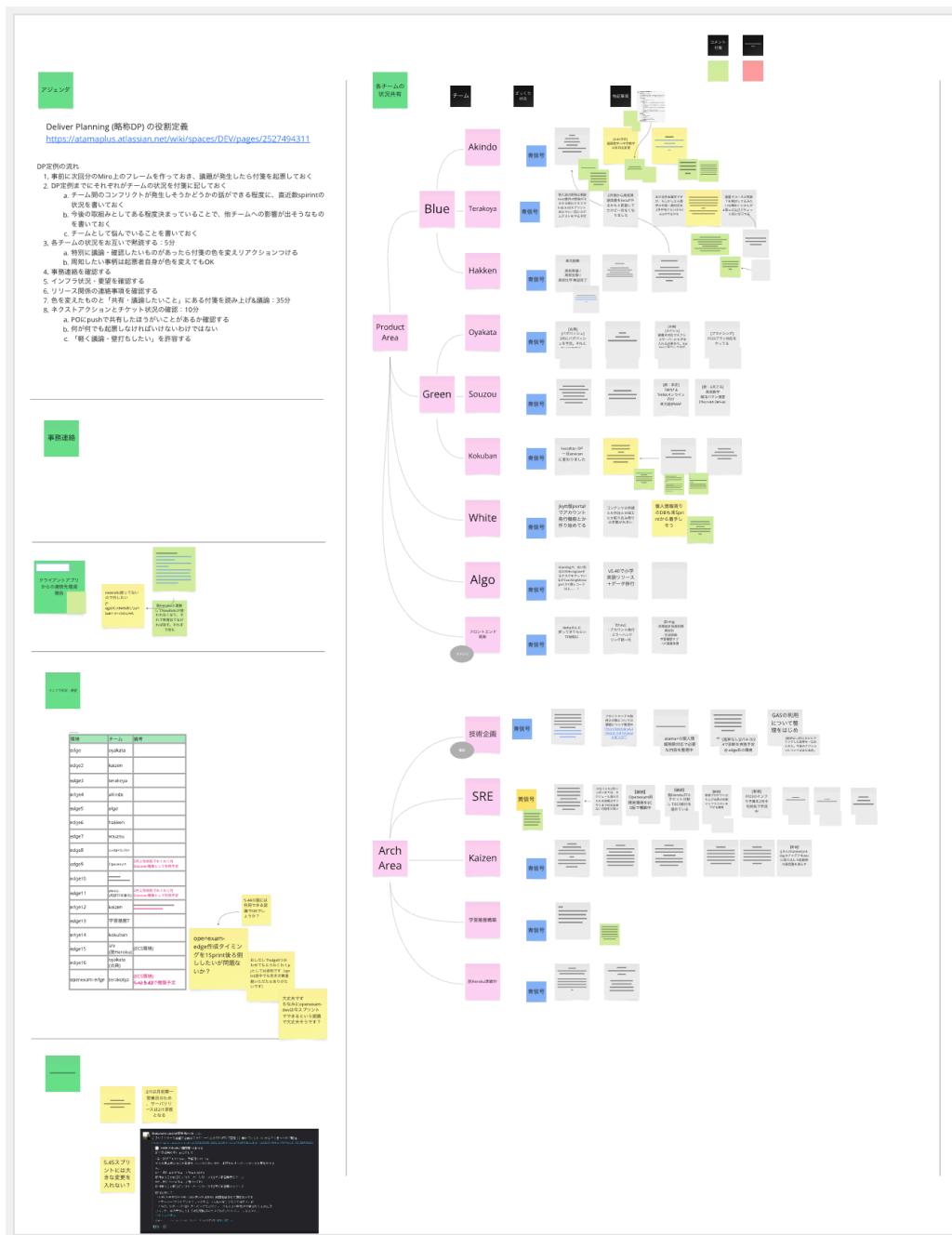
- 認知負荷の増大
 - 「全部を全員で見切れない」
- 開発コンフリクトの頻発
 - 新事業：高速な価値検証
 - 既存事業：安定した価値提供
- チーム間調整コストの爆発的増加
 - 新事業での変更が既存事業では不要
 - 顧客コミュニケーションコスト

モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

対話での回避

- 開発負荷増大に対して、チーム間で開発についてsyncする場を定期的に設けて対応
 - 一定の効果は得られたものの、中長期的な知識の醸成やコンフリクト防止の課題は依然残る

モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実



アジェンダ

- | 事業成長に伴い直面した開発課題
- | 「コードを所有する」という考え方と取り組み
- | 「コードを所有する」状態と組織柔軟性の課題
- | 所有者運用と組織柔軟性の最大公約数
- | まとめ

「コードの所有」とは

- コードの各部分に明確な「所有者」を設定する
- 所有者はそのコードについての最終責任を持つ
- Googleなどの大規模組織でも実践されている手法
 - 「Code Ownership」の考え方
 - ファイルレベルでのオーナーシップ
 - レビューやメンテナンスの責任の明確化

モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

Architecture and Hardware Contributed articles

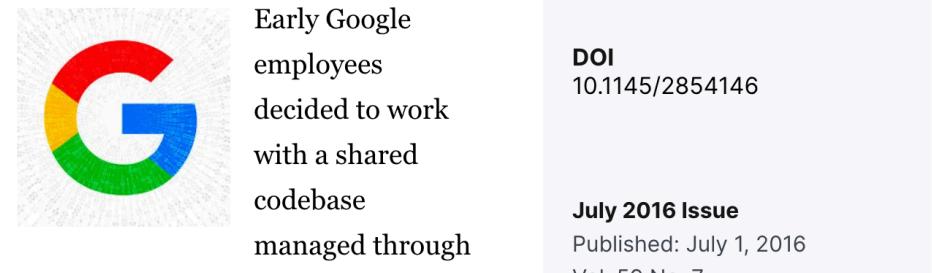
Why Google Stores Billions of Lines of Code in a Single Repository

Google's monolithic repository provides a common source of truth for tens of thousands of developers around the world.

By [Rachel Potvin](#) and [Josh Levenberg](#)

Posted Jul 1 2016

Share Print Join the Discussion View in the ACM Digital Library ↗



コード所有者モデル

コードの各部分に明確な「所有者」を設定する開発ルール

```
src/
└── module_a/
    ├── feature_1/          # Product Team A
    └── feature_2/          # Product Team B
└── module_b/
    └── submodule/          # Product Team C
└── shared/               # Platform Team
```

- 所有の単位はmoduleだったり、コードファイル単位だったり
- 実態に合わせて設計する必要がある

モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

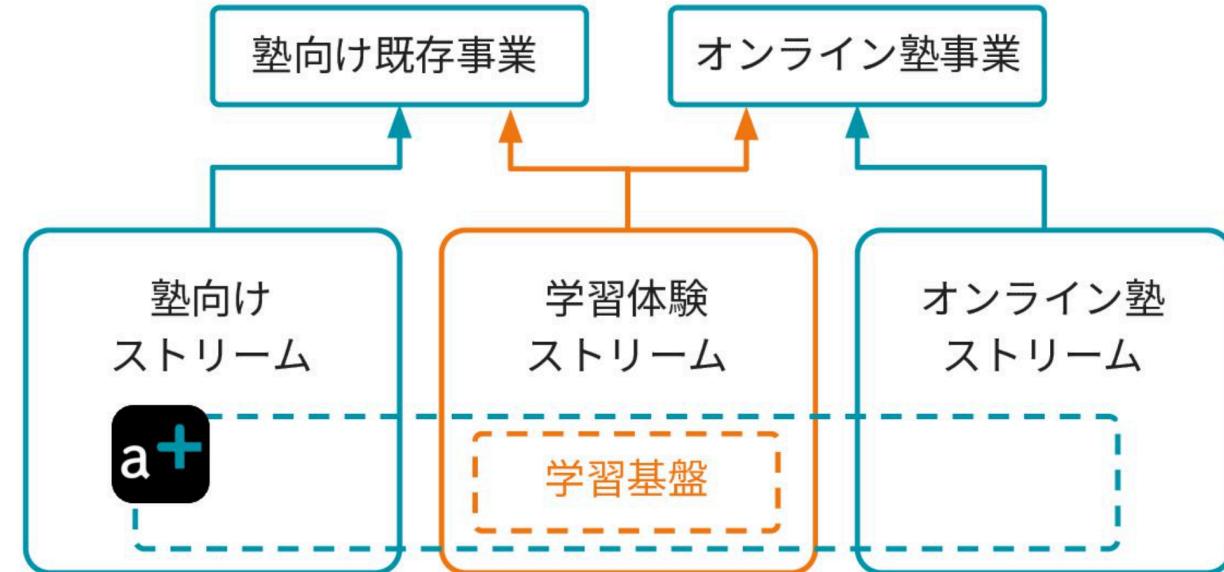
所有者モデルの設

計

チームをストリームアラインドに

区切る

- 事業・サービス単位での機能責任
- ドメイン知識の集約
- 関連するコードの所有
 - ドメイン関連コードの明確な割り当て
 - チーム間の責任境界の設定



導入プロセス

1. コードベース分析と所有エリアのマッピング
2. クロスエリア開発のためのガイドライン策定
 - 事前相談ルール
 - レビュー依頼フロー
3. シームレスな所有者の管理・可視化
 - メタデータファイルの配置
 - エディタ拡張の開発

導入プロセス

コードベース分析と所有エリアのマッピング

- ・ プロダクトの機能/module/APIの粒度で、どのエリアが所有するものかを整理
- ・ 所有関係・依存関係を一覧にして可視化

機能	所有者	1の状況	2の状況	3の状況	4の状況	備考	親機能	親機能の所有者	Port	生徒	診断	学習	Port	生徒	CO	HOI
2.学習体験	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	○	▼	▼
2.学習体験	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	○	▼	▼
2.学習体験	所有	▼	▼	▼	▼	コマンドのみ。	▼	▼	▼	▼	▼	▼	▼	▼	▼	▼
3.塾SaaS	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	▼	▼	▼
3.塾SaaS	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	▼	▼	▼
3.塾SaaS	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	▼	▼	▼
3.塾SaaS	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	○	▼	▼
1.オンライン	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	▼	▼	▼
1.オンライン	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	▼	▼	▼
1.オンライン	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	▼	▼	▼
3.塾SaaS	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	○	▼	▼
2.学習体験	所有	▼	▼	▼	▼		▼	▼	▼	▼	▼	○	▼	○	▼	▼
SST（作戦会議サポート）	1.オンライン	▼	▼	▼	▼		▼	▼	▼	○	▼	▼	▼	▼	▼	▼
SST（作戦会議サポート）	1.オンライン	▼	▼	▼	▼		▼	▼	▼	○	▼	▼	▼	▼	▼	▼

モノリスの認知負荷に立ち向かう二つの所有者という思想と現実

導入プロセス

クロスエリア開発のためのガイドライン策定

- ・ 所有をまたがる変更を他チームから行いたい場合、所有チームに相談する
- ・ 何をトリガーに所有者に相談するかのガイドラインなど制定

トリーク SaaS所有コードへの変更相談 WORKFLOW 10:43 AM
相談 @dev_deliver_塾saas

依頼者

@

対応期日

2025/03/24 (今sprint中)

モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実
相談内容

atama+熟エリアでportal v2を消したい関係で、質問詳細画面（q-and-a-detail）のミニマム移行を行いました

導入プロセス

コード所有の可視化

- エディタでコードファイルを開くと
所有エリアが表示される拡張の提供

モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

The screenshot shows a code editor interface with the following details:

- Title Bar:** atamaplus [拡張機能開発ホスト]
- Sidebar:** Shows files: __init__.py, constants.py, .mod, and others.
- Code Area:** Displays Python code, with a tooltip: "※ I を押して、GitHub Copilot Chat に何らかの操作を依頼するかを開始して閉じます。"
- Bottom Status Bar:** Includes tabs for 問題 (Issues), テスト結果 (Test Results), ターミナル (Terminal), 出力 (Output), and a date/time indicator (10:46).
 - Terminal pane shows command history:
 - :10:46
 - source
 - mv atamaplus/.module.toml .
 - mv atamaplus/.module.toml .
- Owner Information:** The status bar also displays "Owner: Plattform" (highlighted with an orange box).

導入プロセス

コード所有の可視化

- 敬虔なVimmerが作成した拡張も



A screenshot of a terminal window titled "Alacritty". The window shows a single line of Python code:

```
1 from pathlib import Path
```

The code imports the Path class from the pathlib module. The terminal window has a light gray background and a dark gray header bar.

モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

初期の成果

- 認知負荷の軽減
 - 自分のチームが所有するコードの変更に対する安全性が向上
 - 所有していないコードに関しても「誰に聞けばいいか」の明確化
- 品質維持活動の効率化
 - トリアージしたバグのアサインを機械的に実施
 - 特にDevOpsの中で、検知したエラーの担当アサインに迷わなくなった

アジェンダ

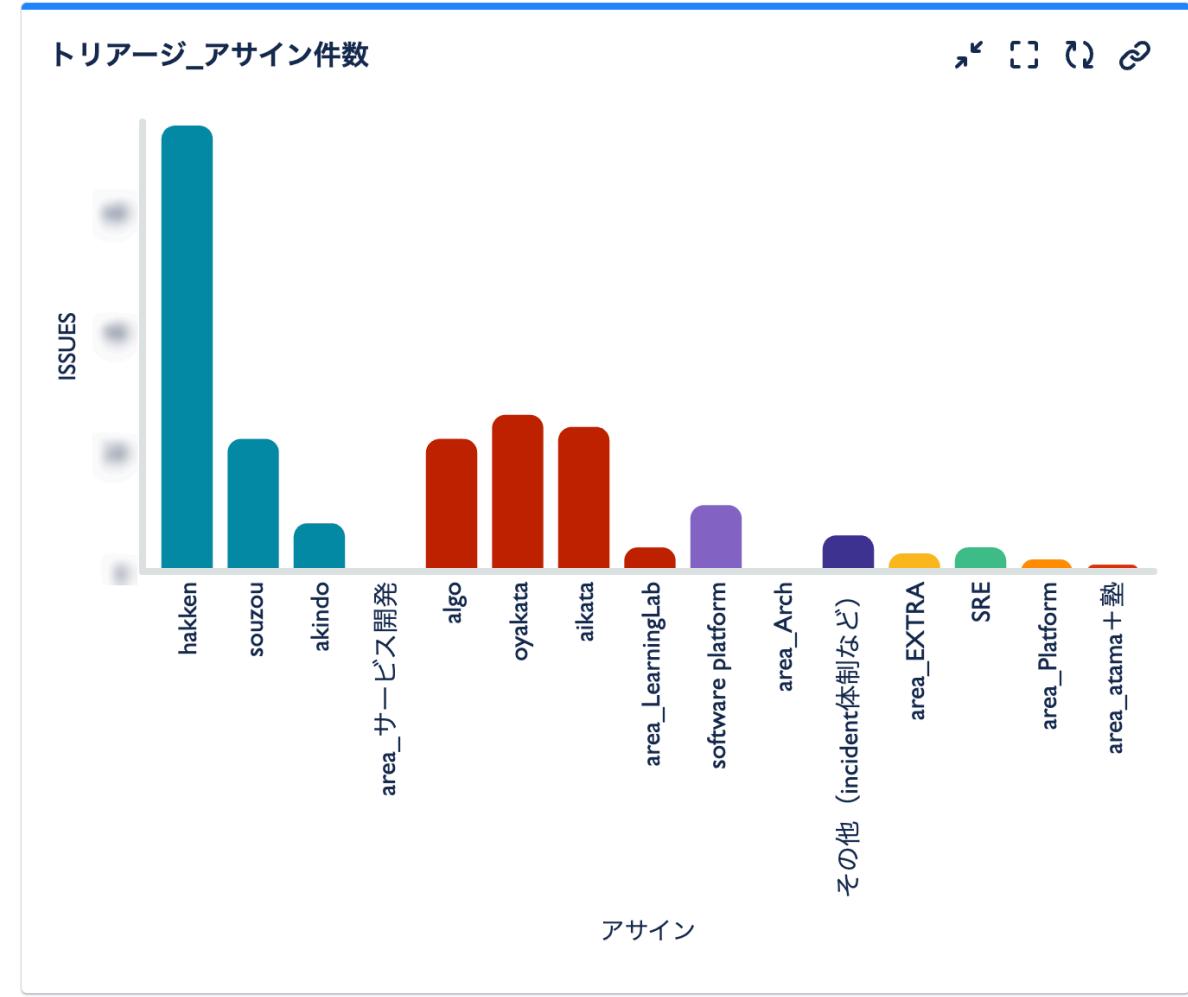
- | 事業成長に伴い直面した開発課題
- | 「コードを所有する」という考え方と取り組み
- | **「コードを所有する」状態と組織柔軟性の課題**
- | 所有者運用と組織柔軟性の最大公約数
- | まとめ

直面した課題

- 負荷の偏り
 - 活発な開発領域 vs. レガシー領域
 - メンテナンス負荷の不均衡
- 技術的負債の責任
 - 「継承した負債」に対する抵抗感
- クロスカッティングな変更の複雑化

モノリスの認知負荷に立ち向かう。コードの所有者という思想と現実

- チームを跨る変更の調整コスト

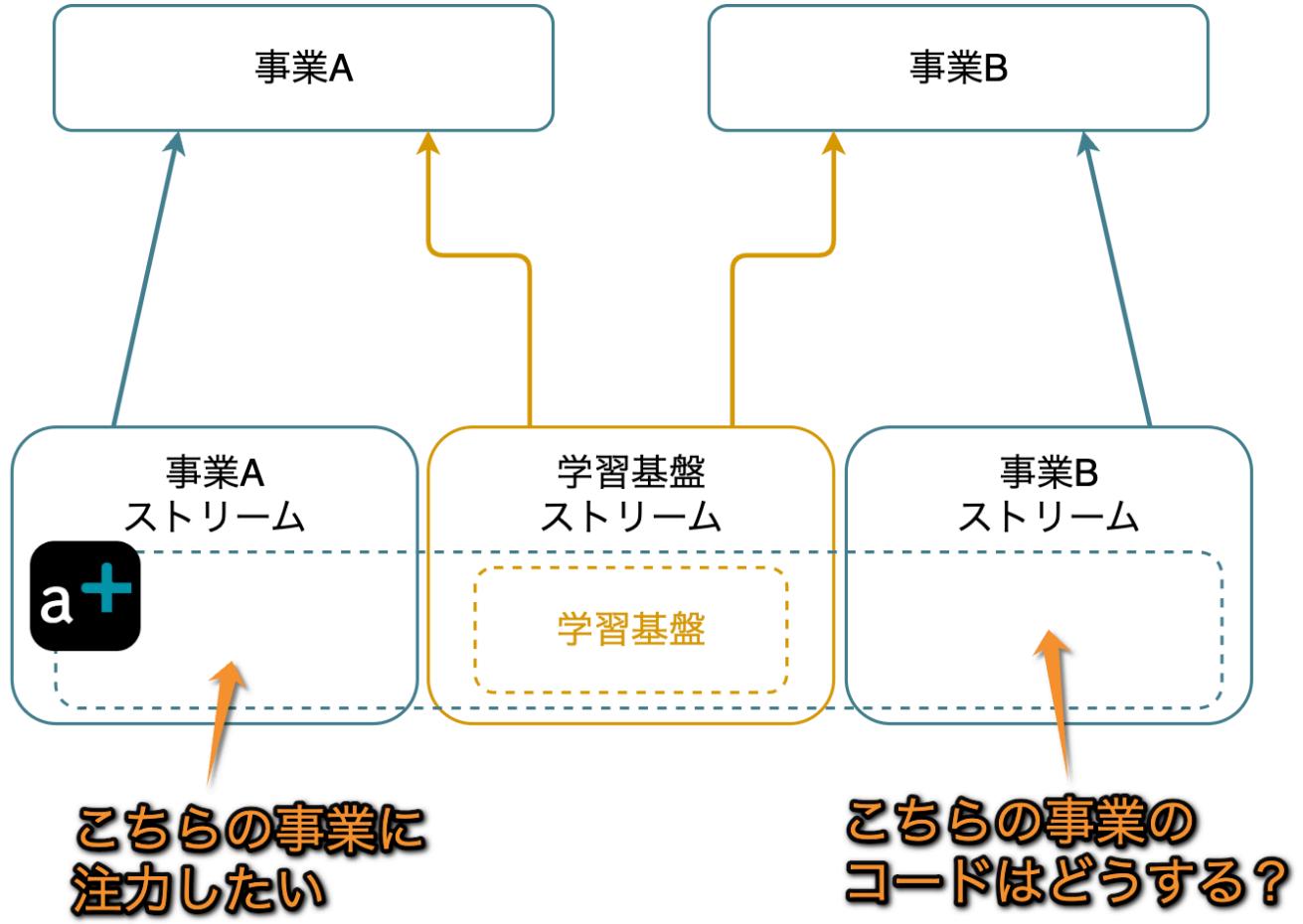


組織変更とのコンフリクト

- 事業戦略変更による組織再編
 - 事業戦略に合わせ、チーム統合・分割、新規事業立ち上げなどが必要になる
- 所有者移管の難しさ
 - 組織変更と合わせて、コードの知識移転を行うコストが着いて回る
 - 組織変更の影響で、事実上所有者が不在となるコードが現れる

課題の事例

- ・事業Aに注力し、事業Bから人をアサインしたいという組織変更
- ・事業Bが所有しているコードには、事業Aに依存されているものも多数
- ・人が少なくなるが、依存されるコードが多く存在するケースに、どう立ち向かえばいいのか？？



モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

余談:マイクロサービス化?

- この手の話でよく上がる解決策だが・・
- マイクロサービスは必ずしも答えではない
 - 分割コストの高さ
 - 運用複雑性の増大
 - 組織規模とのバランス
- モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実
 - サービス所有の問題は残る

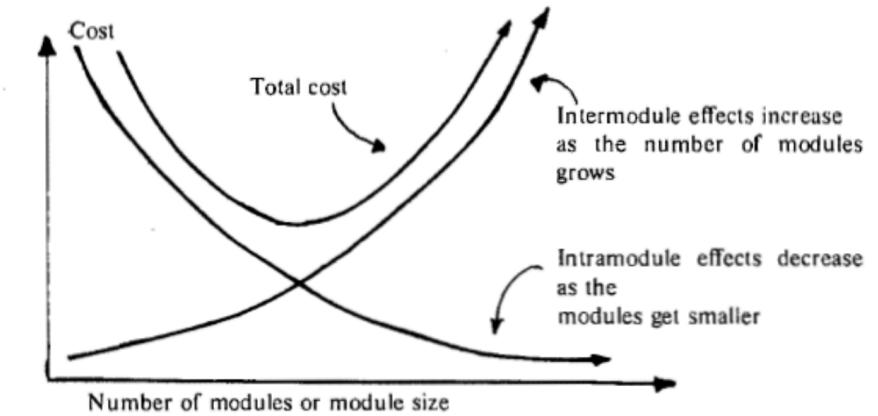


Figure 5.3. Opposing influences of intramodule errors and intermodule errors.

CLOUD NATIVE, MICROSERVICES, ARCHITECTURE

Why I'm No Longer Talking to Architects About Microservices



Ian Miell

March 11, 2025
10 minutes Read

アジェンダ

- | 事業成長に伴い直面した開発課題
- | 「コードを所有する」という考え方と取り組み
- | 「コードを所有する」状態と組織柔軟性の課題
- | **所有者運用と組織柔軟性の最大公約数**
- | まとめ

以降は現在挑戦中の新しい所有運用についてです

新しいアプローチ： 領地と領主

- ・従来の静的な所有権から動的な管理責任へ
 - ・**領地 (Territory)**
 - 事業ドメインに基づくコード領域
 - 機能的なまとめを優先
 - ・**領主 (Lord)**
 - 変更管理の責任を持つチームまたは個人
- モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

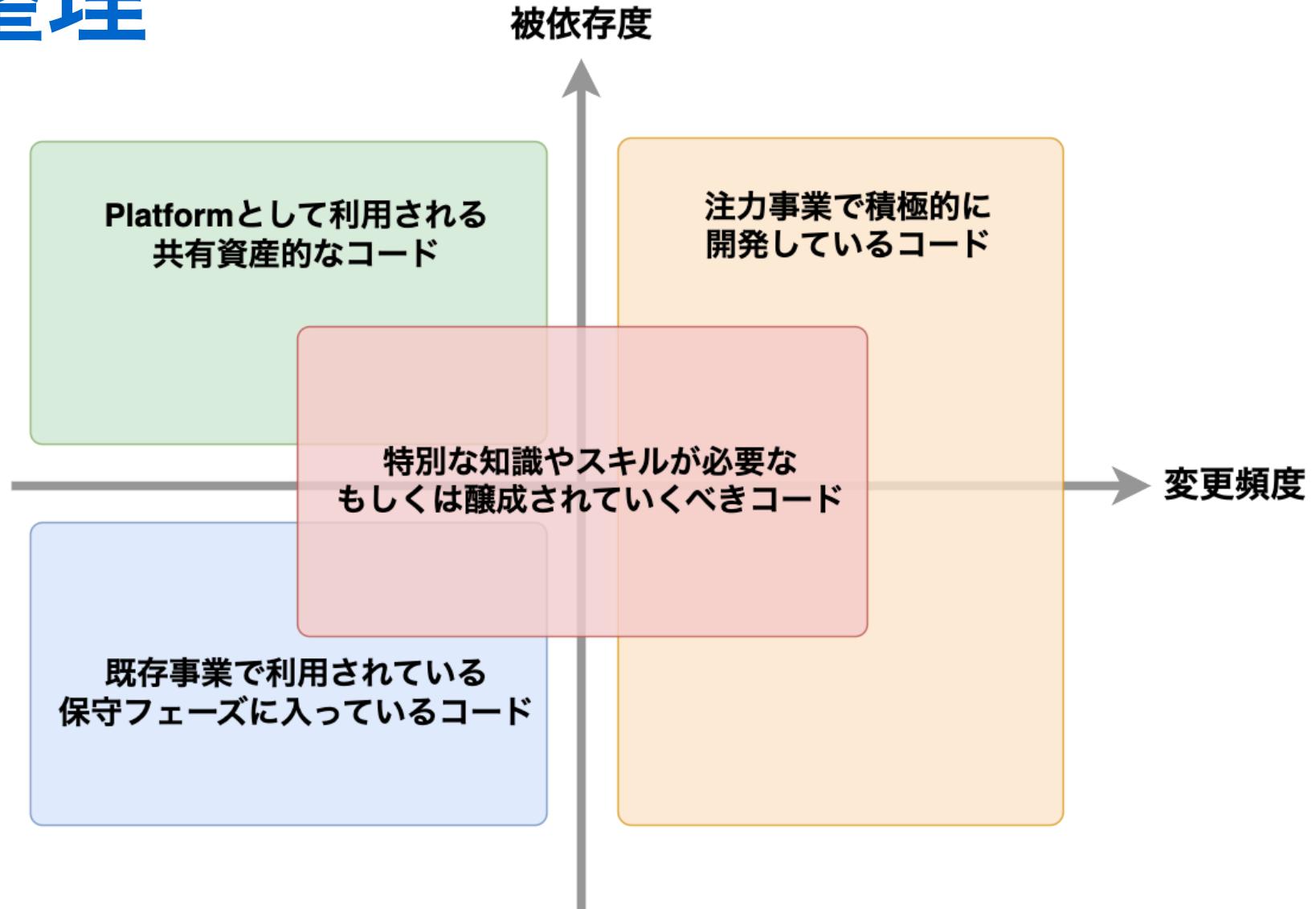


領主の整理

コードを事業ドメインでなく、被依存度と変更頻度、複雑度の軸で整理

1. 被依存度が低く、変更頻度も低い
2. 被依存度が高く、変更頻度は低い
3. 変更頻度が高い
4. 複雑性が高い

領主の整理



モノリスの認知負荷に立ち向かう、コードの所有者という思想と現実

領主の整理

被依存度が低く、変更頻度も低い

- 保守Phaseに入った機能、コード
- 積極的に体制はつけないが、知識承継と運用保守のガイドは決める

被依存度が高く、変更頻度は低い

- Platform的に複数事業で利用されるコード
- 所有コストを下げる目的としたモノリス化認知負荷に立ち向かう、コードの所有者という思想と現実

変更頻度が高い

- 注力事業で積極的に開発しているコード
- 事業チームのエンジニアで所有する

複雑性が高い

- Complicated Subsystemの要素が強いコード
- 特殊な知識を要するケースが多いため、

「領地と領主」モデルで狙う利点

- 組織変更への適応性
 - 領主の交代が容易
 - 事業変化に合わせた柔軟な責任移管
- 明確な責任分担
 - コードの「住所=事業ドメインに紐づく領地」は変わらない
 - 管理責任=領主だけを移行
- オーナーシップの促進
 - 単なる「担当」から「責任ある管理」へ
 - 一時的な領主としての改善インセンティブにも寄与させたい

アジェンダ

- | 事業成長に伴い直面した開発課題
- | 「コードを所有する」という考え方と取り組み
- | 「コードを所有する」状態と組織柔軟性の課題
- | 所有者運用と組織柔軟性の最大公約数
- | **まとめ**

まとめ

1. 事業成長の中で認知負荷の軽減が重要課題に
2. コード所有権モデルで成果を得るも、組織変更との衝突に直面
3. 領域の状態に応じた領主の割り当てで柔軟性を確保
4. 領地(Territory)と領主(Lord)の分離がモノリス運用の新しい形
5. 変更頻度や被依存度に基づいた動的な所有構造へ

学びと今後の展望

- コード所有は思想だけでなく運用の問題
 - 単にルールを決めるだけでは不十分
 - 組織変化に対応できる柔軟な仕組みが必要
- マルチ事業展開への対応
 - 事業ごとの変更速度とリスク許容度の違いを調整
 - 共通部分と変動部分の境界線設計
- 技術と組織の共進化
 - コードの構造と所有者モデルを同時に進化させる
 - 「領地と領主」という概念を文化として定着させる

