











あなたのコンセプトをテス トするためにあなたの 実行 可能な製品を達成してくだ さい。





#### [MVP]

MVPの**周辺**は**あなたの製品を販売することを可能にしながら**減らされなければなりません\*\*。

早期導入者に賭けて最大限のフィー ドバックを得る

あなたのMVPは、プロダクションに 導入され、使用可能です。







失敗速い学習高速です。





#### [FAIL-FAST]

**すぐに**解決策(数週間)を経験し、 ユーザーからフィードバック\*\*を収 集し、あなたの間違いから学ぶ。

すべてを変えることを恐れないでく ださい\*\*。

あなたが失敗することを忘れないで ください!









\*\* K \*\* eep \*\* I \*\* t \*\* S \*\* imple **&** \*\* S \*\* tupid.

シンプルになると複雑にな るのはなぜですか?





#### [キッス]

オーバーエンジニアリングを避けてください。 "ペーパー"モデルや Googleフォームがコンセプトをテ ストするのに十分であれば、さらに 進まないでください。

シンプルに! 技術的にも機能的にも。







lessを指定し、\*\* more \*\*を 展開します。





#### [ 牛産性]

あなたの仕様を裸の必需品に限定し、\*\*どのように"\*\*に 焦点を合わせます。

製品は、\*\*ほとんどの場合、自己文 書化されたものでなければなりませ ん。

ドキュメンテーションは、コードと同じ方法でバージョン管理する必要があります。







\*\* SaaS \*\*ソリューションを 体系的に学習します。





#### [SAAS]

\*\* SaaS ソリューションは持続可能 でコスト効率に優れています\*\*。

場合によっては、\*\* SaaS は MVP の <u>実装をスピードアップすること</u>がで きます。

経済的ビジョンを、総費用(\*\* TCO : T \*\* otal \*\* C \*\* ost \*\* \*\* O \*\* wnership) だけでなく、ライセン スコストの面で。







コアビジネスは、新しいサ ービスやアプリケーション の構築を妨げるものではあ りません。





#### [ビジネスの中心]

コアビジネスの進化と配信のペース は、それを消費するサービスの俊敏 性\*\*と互換性がなければなりませ h.

コアビジネスは**サービス**を公開する 必要があります。

コアビジネスは\*\* Event-Driven \*\*の <u> 原則を採用しなければならず、イベ</u> ントの形で管理アクションを報告し ます。







# [継続的な導入]

## プロダクションでの展開は 非イベントです。





## [継続的な導入]

**継続的な展開**を活用して、**ビジネス**の要件に\*\* **プロダクション**を適応さ せます。

環境全体の配備は、生産まで**自動**お よび頻繁です。







永続的なベータアプローチにより、開発プロセスにユーザーを巻き込むことができます。



#### [ PERPETUAL BETA ]

**ユーザが開発に参加する**永久ベータ の原則を自由に使用してください \*\*

永久ベータという用語は、ジャストインタイムで開発されたアプリケーションを指します。**常に変化しています**。不完全な製品ではありません。













## [知覚]



ユーザが感じる経験は基本 的なものです。

人間工学は交渉可能ではあ りません。



#### [知覚]

\*\* UXデザイナーの仕事を怠っては いけません\*\*、それはアプリケーション開発の基本です。

あなたのユーザーの**フィードバック** を統合しなさい、これは本質的で す。









## [パフォーマンス]

内部と外部の両方の用途に 強力なインターフェースを 使用してください。





#### [パフォーマンス]

インターフェイスは**効率**に対応して います。

インターフェイスの**パフォーマンス** は時間を節約し、**ユーザーの満足度** を高めます。したがって、**は欲求不 満を救います**。









## [モバイルファースト]

## モバイルファースト戦略を 採用してください。





#### ユーザー体験

### [ モバイルファースト]

モバイルデバイスは、**市場**の最も重要な\*\*部分です。

Thinking Mobileは**必須**を考えています。

**レスポンシブルデザイン**は標準であり、貯蓄源(\*\* MVP \*\*)です。







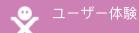


## [OMNI-CANAL]

オムニチャンネルが標準で す。







#### [OMNI-CANAL]

オムニチャンネルのアプローチは、 ユーザーに**統一されたエクスペリエ ンス**(例:Netflix)を提供します。

異なる**チャンネル**は**同期とコヒーレ ント**です(バッチ処理とは異なりま す)。

すべてのアクター(クライアント、 アドバイザー)は同じ情報にアクセ スします。





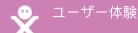




## [自己データ]

\*\* \*\* ユーザーは、データと そのコースの所有者です。





#### [自己データ]

個人データはいつでもコント ロールにしておいてください。

ユーザーにトレーサビリティと制御 をリアルタイムで許可することで**信** 頼を確立します。

**サブシステム**は同じ要件を満たさなければなりません。





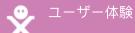




### [CRM / SFA]



顧客との関係は、柔軟で統一されたイベント主導のCRM / SFAによって統一され、文脈化される必要があります。



#### [CRM / SFA]

顧客関係とセールスフォースの両方のリーダーシップを管理する\*\* CRM を選択してください(SFA: S \*\* ales \*\* F \*\* orce \*\* A \*\* utomation)。

\*\* CRM **は新しい機会に**開かれている\*\*必要があります。

\*\* CRM **は**プラットフォームの**イベントドリブン**ロジックに適合する管理アクションに対応する**イベント**を生成します。







## [ビッグデータ]

ビッグデータプラットフォームではを一元化し、ユーザーデータを処理してあなたの旅に最適に対応できます\*\*。





#### [ビッグデータ]

マイフグループ、パートナーおよび ベンダーデータを経路ロジックに集中します。

"データの準備"と処理\*\* **データを統 合することができます**。

**ビッグデータチームはフィーチャーチームと協力して**データ\*\*ガバナンスを確保します。









## [ワークステーション]

ワークステーションは、\*\* との最新のチャンネル\*\*を 使用するように適応されて います。







#### [ワークステーション]

統一された経験のために\*\* IDフェデレーション\*\*を採用してください。

**ポータル**では**概要**を提供できますが、アプリケーションを置き換えるものではありません。

ワークステーションは、**拡張エンタ** ープライズ内で開くためにモバイ ル、マルチチャネルおよび標準であ る必要があります。









## [協力者]

あなたのアソシェーツが自 宅で最新のアプリケーショ ンをUXで使用していること を忘れないでください。





## [協力者]

**すべてのユーザーを「顧客」**として 扱う\*\*: インターネットユーザー、 マネージャー、運用、開発者など 内部使用管理アプリケーション用に

内部使用官理アプリケーション用に 実装する\*\*UXの努力\*\*を過小評価し ないでください。







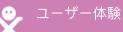


## [すべての測定]



測定可能なものはすべてな ければなりません。

措置なしでは、すべてが意 見のみです。



## [すべての測定]

アプリケーションの**開発中の**メトリックを考えてみてください。 **ログ** には**ビジネスだけでなく、技術的な** ディメンションが必要です。

**パフォーマンス指標**を無視しないでください。これらは基本的なものです。

フィーチャーチームは**操作**を提供します: **アプリケーションを**使用可能にする責任があります。







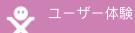


## [A/Bテスト]

\*\* A / Bテストは、フィード バック\*\*を決定することに よって時間を節約します。







### [A/Bテスト]

2つのソリューションの間で任意に 決定するのではなく、"A/Bテストを設定することを躊躇しないでく ださい\*\*。

このパターンは、**同一のアプリケーションの** 2つの異なるバージョン**を 提示し**、ユーザ活動の**客観的尺度**に 基づいてそれらの1つを選択することからなる。









## [変質や劣化]



障害が発生した場合のサービス中断ではなく、劣化を 考慮してください。





## [変質や劣化]

サブシステムのうちの1つの**失敗**では、**サービスの劣化**バージョンは**中断ではなく**最初に考慮されなければなりません\*\*。

サーキットブレーカを使用すると、 \*\* システム全体に影響とスプレッド を回避するために**故障を分離できま** す。













チームは、<mark>製品</mark>またはサー ビスの周りに編成されてい ます。





## <u>[フィーチャーチ</u>ーム]

チームはコヒーレントな機能セットを中心に構成された\*\* Feature Teams であり、このセットに必要なすべてのスキル\*\*で構成されています。

たとえば、Business Expert + Web Developer + Java Developer + Architect + DBA + Operationalなど です。

**責任**は集団です。機能チームはこの 責任のために必要な力を持っていま す。







# [ 2-PIZZA TEAM ]

フィーチャーチーム\*\*のサ イズを制限する(5人から12 人まで)。





### [2-PIZZA TEAM]

フィーチャーチームのサイズを制限 する: \*\*5~12人\*\*。

5歳未満では、彼女は外部の出来事にはあまりにも敏感で、創造性に欠けています。 12を超えると、生産性が失われます。

「\*\* 2ピザチーム」という用語\*\*は、 フィーチャーチームのサイズが2ピ ザを摂取できる人数を超えてはなら ないことを示しています。









# **やり方を知っている**や\*\*誰 がやりたいのか多才な人に ベットしてください。





#### HUMAN

## [アーティザンソフトウェア

最も重要なのは、開発**の**文化、**スケ** ーラビリティ、適応性です。

ソフトウェア職人とフルスタック開発者を募集することで、彼らはノウハウと全体的なビジョンを通じて真の付加価値をもたらします。

それにもかかわらず、モバイル開発 者は通常、**専門開発者**です。







魅力的 最高のを募集する。





## [募集]

モビリティ、家庭作業、CYOD (Choose Your Own Device)。

実験に時間を費やし、**作業時間に**する。





組織はスリープエンジンで なければなりません

前日は仕事の一部です。





### [EVE]

組織は**継続教育**または**ビジネス大学** のようなシステムをセットアップすることによって**デイケア**エンジンでなければなりません。

\*\*コーディングDojos \*、\* Brown
Bag Lunchs 、External \*\*
Conferencesなどの他のより多くの **非公式な**方法と自由に組み合わせて
ください。









## [ CO-CONSTRUCTION ]

# コンバージェンスの目標に 賭けて、取引の間の障壁を 取り除く。





### [CO-CONSTRUCTION]

トレード間の障壁を解消するため に、一般的な製品を共通の場所でグ ループ化するだけでは不十分です。

**アジャイルアプローチ**は、目的のコンバージェンスを確実にするためにこれらの障壁を排除します<sup>\*\*</sup>。

これらのプラクティスは成功への鍵 の不可欠な部分です。組織は保証人 です。







\*\* DevOps \*\*プラクティスでは、壁がBuildとRunの間に入ることができます。



### [DEVOPS]

共通の目標に向けて\*\* Dev **と** Ops **を統合するには**、 DevOps \*\*を採用してください。

取引は変わりません! DevOpsは、同じ人がDevとOpsのタスクを実行することを意味しません。 開発者と運用は\*\* \*\* スキルからを恩恵を受けるために共同作業を要求され、共感\*\*を改善する必要があります。











フィーチャーチームによっ て厳しいタスク**が実行され** ます。

オートメーションは続いて います。



### [痛み]

伝統的な組織では、チーム間の理解の欠如は、通常、距離やコミュニケーションの欠如\*\*に関連しています。

フィーチャーチームの**メンバーは**す べてのタスクに対して**共同責任と固い**です。

**痛み**は、**継続的改善**の重要な要素です。







サービスセンターは集団コ ミットメントとは調和しま せん。





### [CDS]

フィーチャーチームは、**コラボレー ションと集団エンゲージメント**に大 きく依存する原則に基づいて構築さ れています。

サービスセンターは、企業によるIT の合理化と統合に向かっており、これは集合的コミットメントのこの概 念に反しています。







組織は独断的ではなく、検 証\*\*の役割を担っていま す。





### [検証]

組織がツールと用途で検証の役割を 保持していることを確認してください。特に遺産に影響を与えるツール (例:ソースコードの管理)。

\*\* の機能チームは、その選択肢をサ ポートするを意味します。

独断的ではない 実験を励ますこと を忘れないでください\*\*。









フィーチャーチームはコミ **ュニケーション**を期待し、 **経験とスキル**を共有するこ とが期待されます。





## [横断]

\*\* Feature Teams \*\*の間に障壁を作りません。

フィーチャーチームが互いにコミュ ニケーションを取り、スキルと経験 を共有するために必要な**組織とアジ リティ**を設定します。

\*\* Spotify (部族、章および組合) における横断の組織は、雄弁な例で す。















\*\*すべての用途のためのAPI

\*\*: 社内、顧客およびパー トナー、一般。





### [ API FOR ALL ]

\*\*公開API \*\*を使用して、組織を新しい用途や新しい顧客に開放します。

**商用**パートナーシップ、顧客\*\* **プロ バイダー**では、APIは標準的な交換 フォーマットです。

\*\* API **は、組織の内部使用**にも使用 されることを意図しています。











APIを使用するには、シンフ ルと高速を使用する必要が あります。





#### [セルフサービス]

APIの使用は可能な限り単純でなければなりません。 開発者の経験を考えてください。

必要性の妥当性を検証する最善の解 決策は\*\* APIを素早くテストするこ とです\*\*: 数分で十分です!

プラットフォームはAPIを簡単にテ ストするための**グラフィカルインタ フェース**を提供する必要がありま す。







APIの使用は、<mark>制御</mark>および<mark>制</mark> 御でなければなりません。





#### [API管理]

**クォータ、スロットル、認証、ロギング**を管理するAPI管理ソリューションを実装します。

**監視、フィルタリング、レポート**を 管理するためのメトリックを収集し ます。





#### 相互運用性



プラットフォームに組み込まれた<mark>外部システムおよびサービスの要件</mark>を設定します。





#### [要件]

内部システムと同じ要件を満たすために**外部システム**を必要とします。

外部システムは**イベント**を公開し、 **技術**モニタリングを許可する必要が あります。

外部システムデータを統合する必要 がある場合、**合計**同期は**可能でなければなりません**。









# [多人数テナント]

アーキテクチャは<mark>マルチテ</mark> ナントと考える必要があり ます。





#### [多人数テナント]

ホワイトマークがベースで考慮され ていない場合でも、マルチテナント アーキテクチャを設定します。あな たの最初のアプリケーションは、最 初の保持です。

システムの**多機能インスタンシエー ション**を最初から考えてください。







システムは**ネイティブに設** 定可能でなければなりません。





#### [SETTING]

言語、通貨、ビジネスルール、セキュリティプロファイルは設定が簡単でなければなりません。

**ハイパージェネリック**に注意してください、それはしばしば役に立たず、**コストソース**です。

**セットアップ**はスケーラブルで、必要に応じて高速にする必要があります。











機能の反転を使用して柔軟 性の高いシステムを作成し ます。





#### [フィーチャー・フリッピン グ]

機能の反転は、アプリを機能のセットとしてデザインすることです……………使用可能にすることができます。

マルチテナントアプリケーションでは、フィーチャーを反転させるとサ ポーターをカスタマイズできます。

**簡単なA / Bテスト**の特徴を反転し ます。









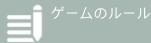


技術的な選択肢は、フィーチャーチームによって\*\*\*\*\*\*

\*\* \*\* \*\*は想定されています。







### [技術的選択]

フィーチャーチームは、責任ある行 動を行い、排他的に影響を与える選 択肢と組織に影響を与える選択肢を 特定する必要があります。

フィーチャーチームの範囲を超える 選択肢**(ライセンス、まれ**なプログ ラミング言語など)は、組織または ピアコンバージェンスプロセスによ **って**検証されなければなりません\*\*







## 正しい使い方のための正し いツール\*\*は節約の源で す。



#### [良い使用]

すべての人に課された悪いツールは リスクです。良いツールの誤用は非 常に有害な結果\*\*を持つことができ ます。たとえば、あまり使用されて いないアジャイルメソッドは危険で す。

**ツール**は**質問**する必要があります。

\*\* Excel **はしばしば合理的な選択肢** ですが、**すべてを行うための** <u>\*\*(CRM</u>、ERP、Datamart、...)





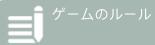


特権コアビジネスのための ビルド

その他の場合は、購入を考慮してください。







#### [BUILD VS.購入]

ツールが組織の機能を差別化する機能を備えているほど、構築される方が多くなります。コアビジネスは特異性を可能にする必要があり、は迅速かつ頻繁に適応しなければなりません。いくつかのソフトウェアパッケージは、このニーズに合わせて適応されることがあります。

\*\* **その他の場合**: SaaS、オープンソ ース、ビルド、オーナーは**ケースバ イケース**で学びます。



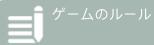




オープンソースを最大限に 活用。

代わりの選択肢をサポート する必要があります。





#### 「オープンソース」

プロプライエタリなソリューション は、必要に応じてメンテナンスを再 開できる必要がある組織のリスクで す。

オープンソースの代替手段を持たな いプロプライエタリなツールはほと んどありません。

組織はオープンソースコミュニティーのメリットを提供し、は貢献することができます。



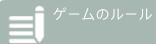






スタンドアロンおよび弱く 結合したサービスを開発す る。





弱いカップリングは標準でなければ なりません。

各マイクロサービスは明確に定義さ **れたインターフェース**を持っていま す。

このインターフェースはマイクロサ ービス間のリンクを決定します。

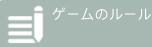
ドメイン駆動設計では、特に有界コ ンテキストでこの問題を予期するこ とができます。







各サービスには独自の\*\*デ ータストレージシステムが あります。



#### [DATA]

データストアは、単一のマイクロサービスとのみ結合されることを意図しています。

あるマイクロサービスから別のマイクロサービスへの**データへのアクセス**は、そのインターフェース**を介して排他的に**行われます。

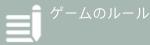
この設計はプラットフォーム全体の時間の経過と共に一貫性を意味します。 UXを含むすべてのレベルで認識されなければなりません。





各マイクロサービスは、合理的な機能的境界を持っていなければなりません\*\*
「頭に合っている」。





#### [SCOPE]

マイクロサービスは**合理的な数の機能**を提供します。

成長が始まると、マイクロサービス を切断するのをためらってくださ い。

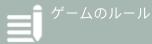
リーズナブルなサイズのサービスは、**必要に応じて**書き換え\*\*を穏やかに検討することを可能にします。







\*\* Reactive Manifesto \*\*は、 反応性のあるアーキテクチャの設計に向かう道を開き ます。



#### [ 応答性]

レスポンシブなプログラミングは、 データの流れと変化の伝播に焦点を 当てています。これは、より伝統的 なアプローチ "**反復子**"に反するパタ ーン "\*\* Observer \*\*"に基づいていま す。

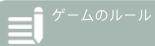
リアクティブマニフェストは基本的 な軸を設定します: **可用性**とスピー ド、**反発力**、柔軟性、弾力性、メッ セージオリエンテーション\*\*。







非同期プロセスはデカップ リングを、スケーラビリティはパフォーマンスを優先 します。



#### [ ASYNC-FIRST ]

アプリケーション間の交換は**非同期** でなければなりません。

非同期交換は自然に**弱い**カップリング、**分離、フロー制御(背圧**)を可 能にします。

**同期通信**は、アクションがそれを必要とするときのみ\*\*考慮されるべきです。









情報システムは、イベント を指向していなければなり ません。





\*\* イベント駆動型機能プロセスは自然に 非同期に実装されています。 イベントオリエンテーションは、\*\* C と Q \*\* uery \*\* R の可能性 S \*\*

egregation (\*\* CQRS \*\* ) 2\*\* Event

Sourcing \*\*です。



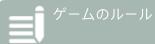






特権シンプルで堅牢で強力 なメッセージブローカーを "スマートパイプ"に特権を 与えます。





#### \_\_\_\_\_ [メッセージブローカー]

\*\* ESB **は**限界**を示しました**:スケーラブルなメンテナンス**は**技術**と**組織**の観点の両方から**重要\*\*です。

\*\* **カフカ**のようなブローカー**メッセ ージは、シンプルな、耐久、弾力の ある**ソリューションを提供します。

スマートエンドポイントとシンプル パイプは規模で動作するアーキテク チャです。インターネットです。







# システムの完全な同期は、

\*\* 設計されるとすぐに考え られるべきです。





イベントフローによって2つのシステム間の同期が保証されている場合、これらのシステムの合計再同期は設計時に計画する必要があります。

自動\*\* \*\* **同期監査**(例: サンプル 別)は、**測定**および**可能な**同期エラ ー**の検出**を可能にします。







サービスの構成は集中、発 見はディレクトリによって 保証されています。



マイクロサービスの構成はすべての 環境のための集中です。

中央ディレクトリはマイクロサービ スの動的検索を保証します。

\*\* **グローバル**スケーラビリティ**はこ の**ディレクトリ\*\*に依存します。

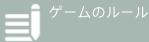






フィーチャーチームはサン ドボックス環境を提供しま す。





#### [サンドトレイ]

フィーチャーチームは**サンドボック** ス環境(現在のバージョンと今後の リリース)を維持して、他の**チーム を拡大することができます**。

**いくつかの非名義**のケースでは、機 能は開発環境で無効になることがあ ります。





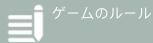




あなたのシステムはクラッ シュするでしょう!

それが耐性を持つように設 計する。





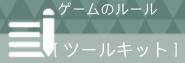
### [失敗のための設計]

あなたの**システムは失敗します**、それは避けられません。このために設計されていなければなりません(\*\* Design For Failure \*\*)。

**ハードウェア**(ネットワーク、ディスクなど)、**アプリケーション**(複数のアプリケーションのインスタンス)、**地理的なゾーン**プロバイダ \*\* (例: AWS + OVH) 。







\*\* toolkits \*\*を提供し、厳格 な枠組みを課してはいけま せん。



# **重**りゲームのルール

#### [ツールキット]

**テクニカルコンポーネントの住宅および横断への注意!** それらは制限的で、費用がかかり、維持するのが難しい。

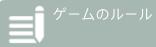
**アクセラレータ、ツールキット、テクニカルスタック プール、無料**フィーチャーチーム、独断的なアプローチを避ける。







パブリック、プライベート またはハイブリッドのクラ ウド(\*\* laaS または PaaS \*\*)は、制作の標準です。



[雲]

\*\* PaaS サービスは優先、簡単\*\*、および規模をすばやく調整できます。

\*\* laaS **サービスでは**、柔軟性\*\*を高める必要があるケースに対処できますが、より多くの運用作業が必要になります。

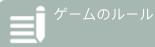
プライベートクラウドは従来の仮想 化環境ではなく、**コモディティハ**ー **ドウェア**に依存しています。







フィーチャーチームはイン フラストラクチャを管理し ていませんが、組織によっ て提供され管理されていま す。



#### [インフラ]

インフラストラクチャの問題は\*\*
Feature Teams **にありません。イン**フラストラクチャは機能×サービス **によって**提供され**維持**されなければ
なりません。







コンテナは異機種ツーリングに必要な柔軟性を提供します。



#### [ CONTAINERS ]

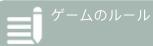
コンテナは**異質なツール**を**同種のコンテキスト**で可能にするためにフィーチャーチームが必要とする**柔軟性**を提供します。







容器の使用は、技術環境の 問題を克服することを可能 にする。



#### [ENVIRONMENTS]

コンテナ(例: \*\* Docker) は、環 境の違いを \*\*解放することを可能に します。

配備プロセスは環境に対して**不可知** 論的でなければなりません。

データベースなどの一部のコンポー ネントはコンテナに配置しないでく ださい。彼らの展開はまだ自動化さ れています。







対策はすべてに中央および アクセス\*\*する必要があり ます。



指標は、異なるレベルの細かさを持つすべての人に**アクセス可能**です。 関連するチームフィーチャーの詳細 ビュー、組織の他のメンバーの集 計。

指標へのアクセスはユニットデータ へのアクセスを意味するものではな く、機密性を維持するように制御さ れなければなりません。

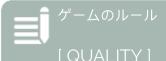
すべての環境が影響を受けます。





ソフトウェア品質は重要な 要素です。





**コードレビュー**は**体系的**です。 **継 続的改善**の一環として、フィーチャーチームのメンバーまたは組織の他のメンバーが実施します。

**あなたが監査されているのではなく、あなたのコード:** "あなたはあなたのコードではありません!"

**光度測定**は部分的に自動化できます が、\*\* "新しい目" **に勝るものはあり** ません。

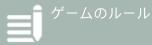






自動テストは継続的な導入 のための交渉可能な前提条 件です。





## <u>[</u>自動テスト]

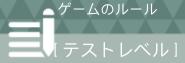
自動テスト 時間の経過とともに製品の品質\*\*を保証します。

継続的な導入には**前提条件**があります。\*\* **変更**および**頻繁な展開**が可能 です。

プロダクションの公開は逸話イベントになります!







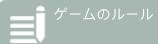
## すべてのレベルでのテス

ト: ユニット、統合、機能性、反発力、パフォーマン

ス







#### [テストレベル]

**統合**および機能テストは最も重要で す\*\*保証\*\*効果的な運用\*\*

**ユニット**テストは**開発**に適しています。

**パフォーマンス**は時間の経過ととも にパフォーマンスを測定します\*\*。

弾力性テストは失敗を予測するのに 役立ちます。







# カバーはテスト品質の主な 客観的指標です。



テストによる**コードカバレッジ**は、 コード品質の**良い**メトリックです。 これは、**必要な条件です**しかし、十 **分ではありません**、コードの良質を 保証することなく、**悪い**テスト戦略

の適用範囲が高くなります。







セキュリティはプロセスで す。問題に応じて対処すべ きではありません。



**セキュリティ専門家**はフィーチャーチームに直接統合することができます\*\* **必要に応じて**。

セキュリティ専門家は、監査、認 識、フォワードのために組織内で利 用可能です。

