



市場投入までの時間





市場投入までの時間





市場投入までの時間



[ MVP ]



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。





市場投入までの時間

[ 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。





市場投入までの時間



[ FAIL-FAST ]



失敗速い学習高速です。





市場投入までの時間

[ 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!







## 市場投入までの時間



[キッス]



\*\* K \*\* eep \*\* I \*\* t \*\* S \*\*  
impleと\*\* S \*\* tupid。

シンプルになると複雑になるのはなぜですか？





市場投入までの時間

〔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。





市場投入までの時間



〔生産性〕

lessを指定し、\*\* more \*\*を  
展開します。





## 市場投入までの時間

### 〔生産性〕

あなたの仕様を裸の必需品に限定し、\*\*どのように"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。





市場投入までの時間



[ SAAS ]

\*\* SaaS \*\*ソリューションを  
体系的に学習します。







市場投入までの時間

[ 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 の実装をスピードアップすることができます。

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。





市場投入までの時間



「ビジネスの中心」

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。





市場投入までの時間

## 「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ん。

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 \*\*の原則を採用しなければならず、イベントの形で管理アクションを報告します。





市場投入までの時間



〔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。





市場投入までの時間

〔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。







市場投入までの時間



[ PERPETUAL BETA ]

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。





市場投入までの時間

[ 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。







# ユーザー体験





# ユーザー体験







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。



## ユーザー体験

### [ 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。







ユーザー体験



「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。





ユーザー体験

〔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。





ユーザー体験



「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

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。





ユーザー体験



[ OMNI-CANAL ]

オムニチャンネルが標準で  
す。







## ユーザー体験

### [ 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。





## ユーザー体験



### 「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。





## ユーザー体験

### 〔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。





ユーザー体験



[ 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。





## ユーザー体験

[ 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：Sales Force Automation）。

\*\* 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。







ユーザー体験



「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\*\*。





ユーザー体験

## 〔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。





## ユーザー体験



### 「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。





## ユーザー体験

### 〔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」

あなたのアソシエ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。





ユーザー体験

〔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実装する\*\* UXの努力\*\*を過小評価しないでください。







ユーザー体験



[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。





## ユーザー体験

### 〔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。





## ユーザー体験



### [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。





## ユーザー体験

### [ 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。





ユーザー体験



「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。







ユーザー体験

〔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。







HUMAN





HUMAN





HUMAN



[フィーチャーチーム]



チームは、製品またはサービスの周りに編成されています。





HUMAN

## 【フィーチャーチーム】

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。







HUMAN



[ 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）。





HUMAN

## [ 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。





HUMAN



[アーティザンソフトウェア]



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、スケールビリティ、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。





HUMAN



[ 募集 ]

魅力的 最高のを募集する。





HUMAN

〔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。







HUMAN



[ EVE ]

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。





HUMAN

[ 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。





HUMAN



[ CO-CONSTRUCTION ]

コンバージョンの目標に賭けて、取引の間の障壁を取り除く。





HUMAN

## [ 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\*\*。

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。





HUMAN



[ DEVOPS ]

\*\* 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。







HUMAN

[ 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。





HUMAN



[痛み]



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。





HUMAN

[ 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。





HUMAN



[ CDS ]

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。





HUMAN

[ 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。







HUMAN



[ 検証 ]

組織は独断的ではなく、検証\*\*の役割を担っています。





HUMAN

〔 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\*\*。





HUMAN



[ 横断 ]



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。





HUMAN

[ 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。







相互運用性





相互運用性







相互運用性



[ API FOR ALL ]



\*\*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。





相互運用性

[ 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。**





相互運用性



「セルフサービス」



APIを使用するには、**シンプル**と**高速**を使用する必要があります。





相互運用性

## 「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。





相互運用性



[ API管理 ]

APIの使用は、**制御**および**制御**でなければなりません。







相互運用性

## 〔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



〔要件〕

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





## 相互運用性

### 〔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。





相互運用性



「多人数テナント」

アーキテクチャはマルチテナントと考える必要があります。





## 相互運用性

### 〔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。







相互運用性



[ SETTING ]

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。





## 相互運用性

### [ 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。





## 相互運用性



[フィーチャー・フリッピング]



**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。









# ゲームのルール





# ゲームのルール









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。



## ゲームのルール

### 〔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。



## ゲームのルール

### 〔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

\*\* (CRM、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。







## ゲームのルール

### [ 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。

代替りの選択肢をサポート  
する必要があります。



## ゲームのルール

### 〔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のメリットを提供し、は貢献することができます。







【マイクロサービス】



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。



## ゲームのルール

### 「マイクロサービス」

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。



## ゲームのルール

### [ 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」。



## ゲームのルール

### [ 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。





## ゲームのルール

### 〔応答性〕

レスポンスなプログラミングは、データの流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。



## ゲームのルール

### [ 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。



ゲームのルール

[ EVENTS ]

\*\* 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 \*\*) と\*\* Event Sourcing \*\*です。







## 【メッセージブローカー】



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。







## ゲームのルール

### 「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、シンプルで、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。



## ゲームのルール

### [ TIMING ]

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。



ゲームのルール

〔一元化〕

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。







ゲームのルール



【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を提供します。





## ゲームのルール

### 〔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。







【失敗のための設計】



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。





## ゲームのルール

### 「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。



## 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 (\*\* IaaS または PaaS  
\*\*) は、制作の標準です。



## ゲームのルール

[ 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。

**\*\* IaaS 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。



ゲームのルール

【インフラ】

インフラストラクチャの問題は\*\*  
Feature Teams ではありません。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。





## ゲームのルール

### [ 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。



## ゲームのルール

### [ 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。



## ゲームのルール

### [ METRIC ]

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。







ゲームのルール



[ QUALITY ]

ソフトウェア品質は重要な  
要素です。





## ゲームのルール

### [ QUALITY ]

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"に勝るものではありません。







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。



ゲームのルール

〔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！





ゲームのルール



【テストレベル】

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





## ゲームのルール

### 〔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。





## ゲームのルール

### [ COVER ]

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**ですが、**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。



## ゲームのルール

### [ SAFETY ]

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。



