基調講演

チームが一つになるための継続的インテグレーション

長沢 智治
アトラシアン株式会社
シニア エバンジェリスト @tnagasawa

長沢 智治

1996
インテック
ソフトウェアエンジニア

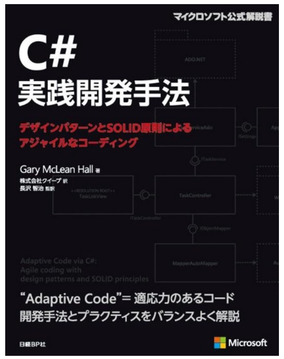
2000
Rational Software
プロセス改善コンサルタント

2003
日本アイ・ビー・エム
プロセス改善コンサルタント

2005
Borland Software
プロセス改善コンサルタント
ソリューション アーキテクト

2007
Microsoft
エバンジェリスト
プロダクト マネージャ

2014
Atlassian
シニア エバンジェリスト



監訳 / 共著書 多数

プレゼンテーション協力

認定スクラムマスター

新刊

今すぐ実践！
カンバンによるアジャイルプロジェクトマネジメント

エリック・ブレックナー （著）, 長沢 智治 （監訳）, クイープ （訳）

発売予定日：2016/06/03
出版社：　日経ＢＰ社
ISBN：　978-4-8222-9871-5

本書籍ないでも継続的インテグレーションは必須のプラクティスとして紹介されています。

Agenda

継続的インテグレーション（CI）

CI とチーム

CI と仕組み（ツール）

Agenda

継続的インテグレーション（CI）

CI とチーム

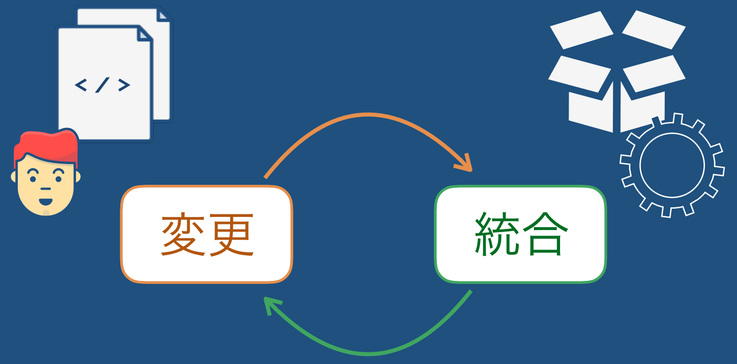
CI と仕組み（ツール）

継続的インテグレーション（CI）

継続的インテグレーション

Continuous Integration

コードを変更したら都度、統合してビルド、検証をして、早期に問題を検知するとともに、
リリースに備えることが大切だというプラクティスです。



開発者がソフトウェアに加えた
変更を取り込んで、
ソフトウェア全体として統合する作業を
途切れることなく続けていく取り組み

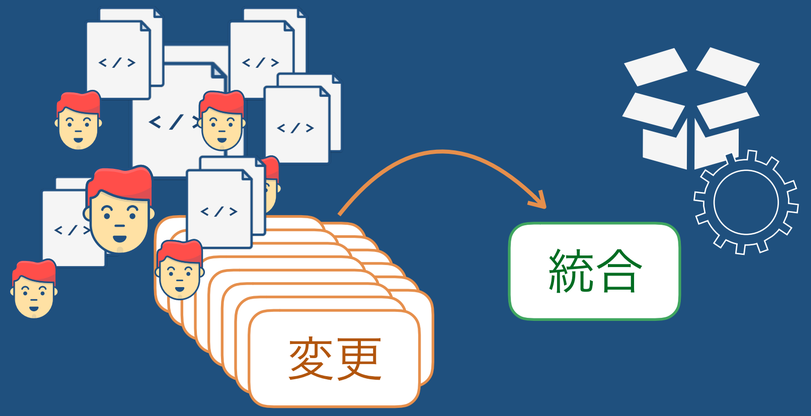
出典:『アジャイルサムライ』 (オーム社) 第15章 継続的インテグレーション: リリースに備える

ビッグバン インテグレーション

Big Bang Integration

各工程（要件定義、設計・分析、実装、検証など）での成果が正しく、統合後に後戻りがない前提ならば、ビッグバン リリースが適切なはずです。

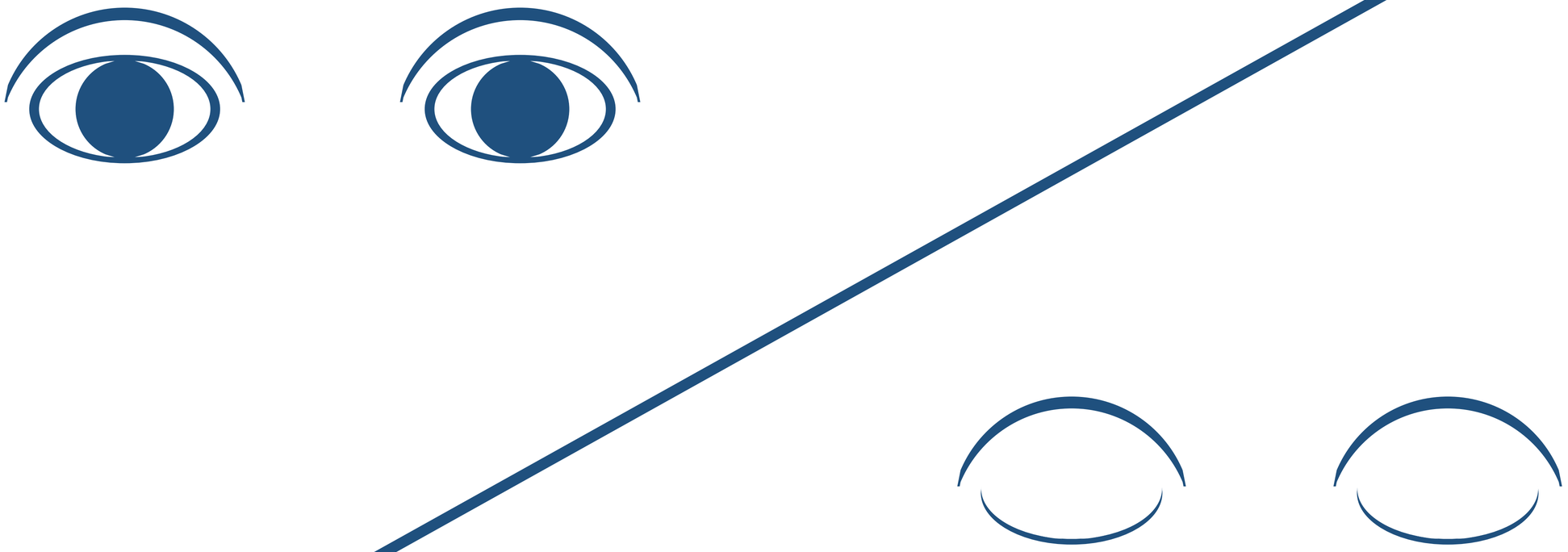
後戻りがおこなるならば、それらに備える必要があり、それはおそらく別のプラクティスが有効であると言えます。

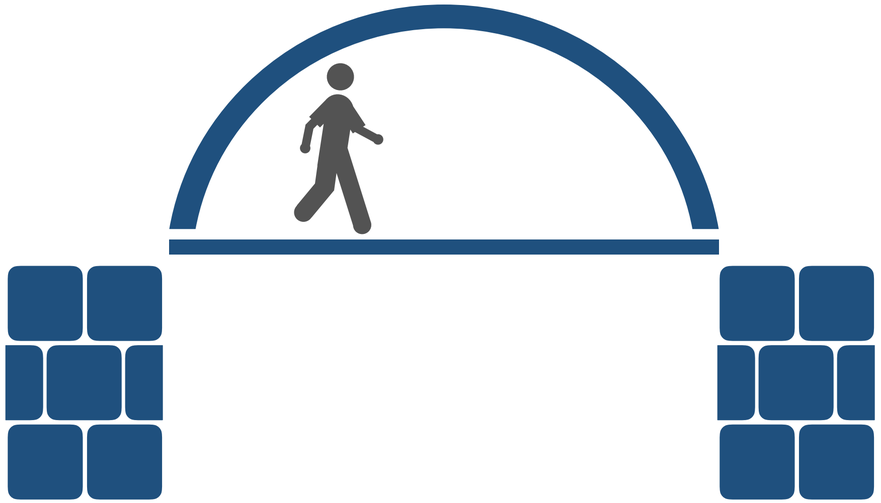


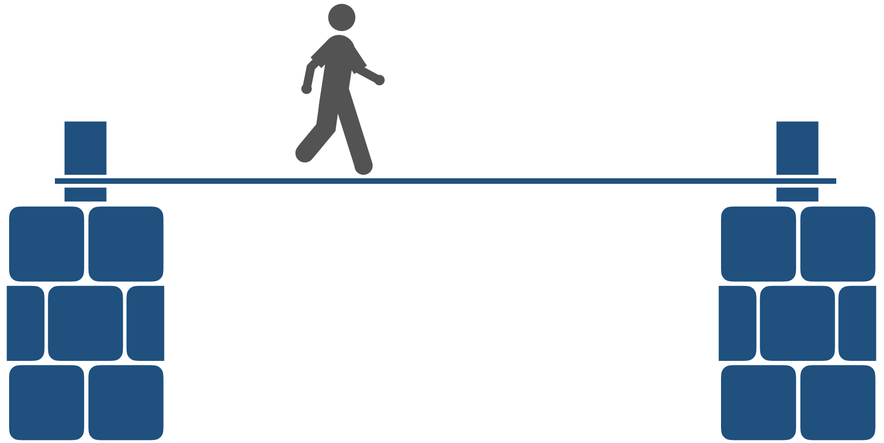
開発者がソフトウェアに加えた
変更を**溜め込んで**、
ソフトウェア全体として統合する作業を
各工程の成果が正しいこと前提に
最後の工程で行う

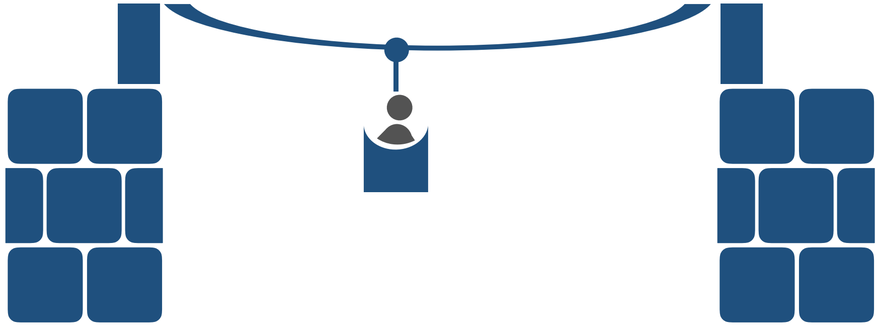
エクササイズ

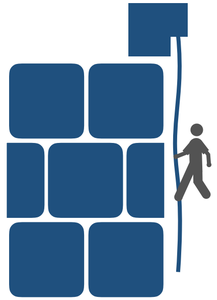
エクササイズの詳細説明は、省略します。
「向こう岸に渡りたい」という要求があった場合に、あなたは、どのようなソリューションを思い描きますか？それを一回だけ見せて、要求に答えることが果たしてできるでしょうか？

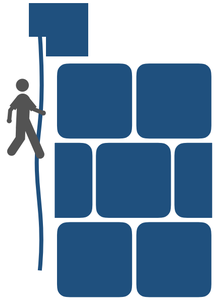












捉えかたはそれぞれです。

また、ニーズを正確に捉えることも難しくなってきています。依頼主が正確にニーズを表現することも難しくなってきています。また、ニーズは、環境要因により変化していきます。

ローケーション

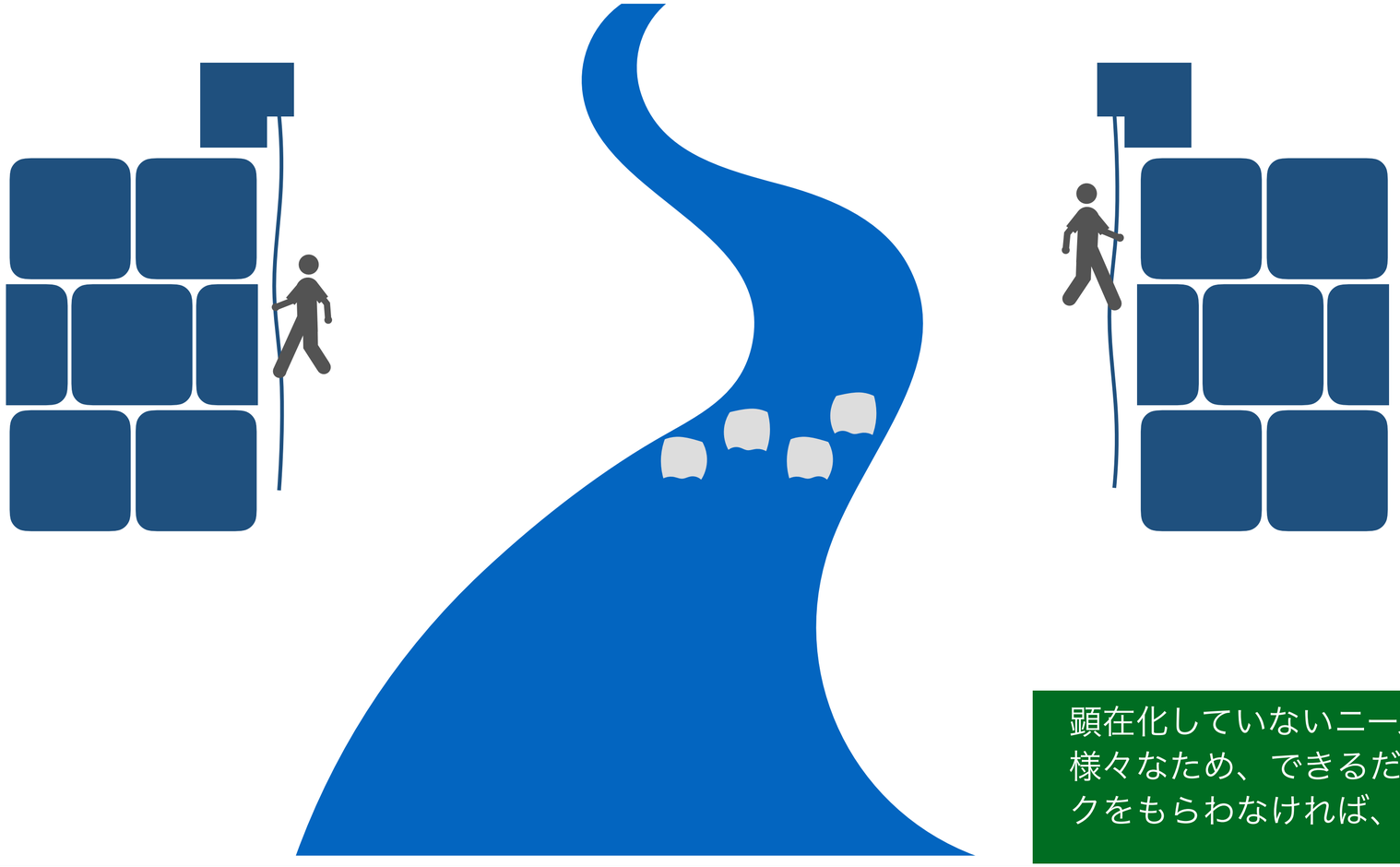
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？

顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

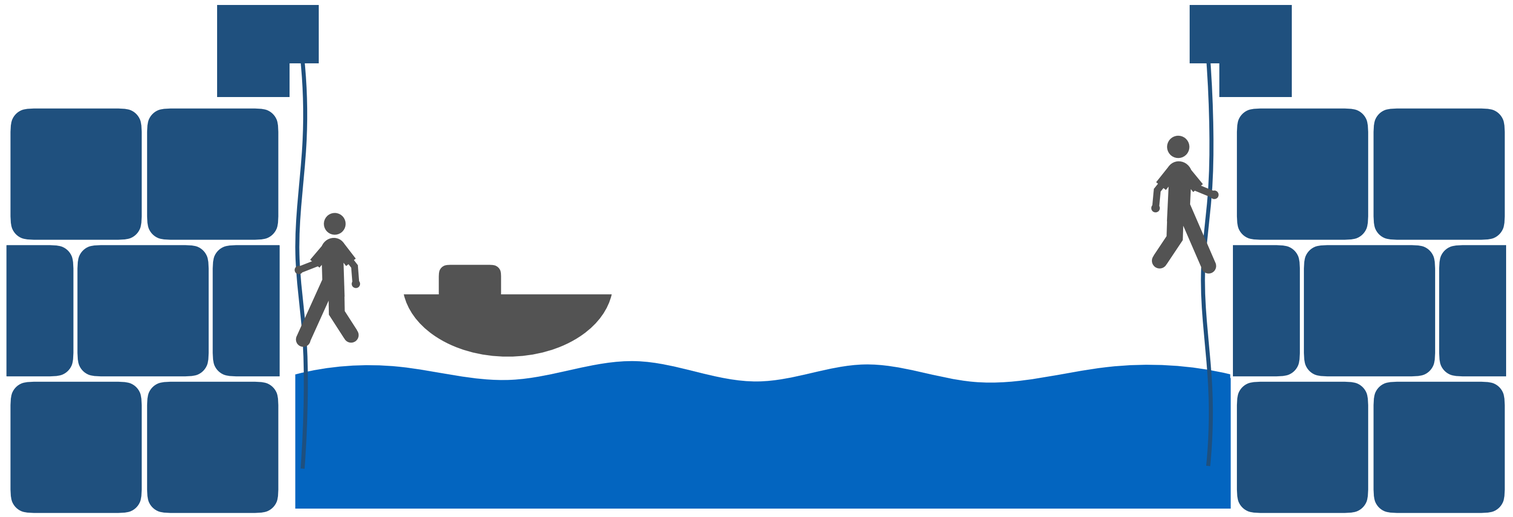
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

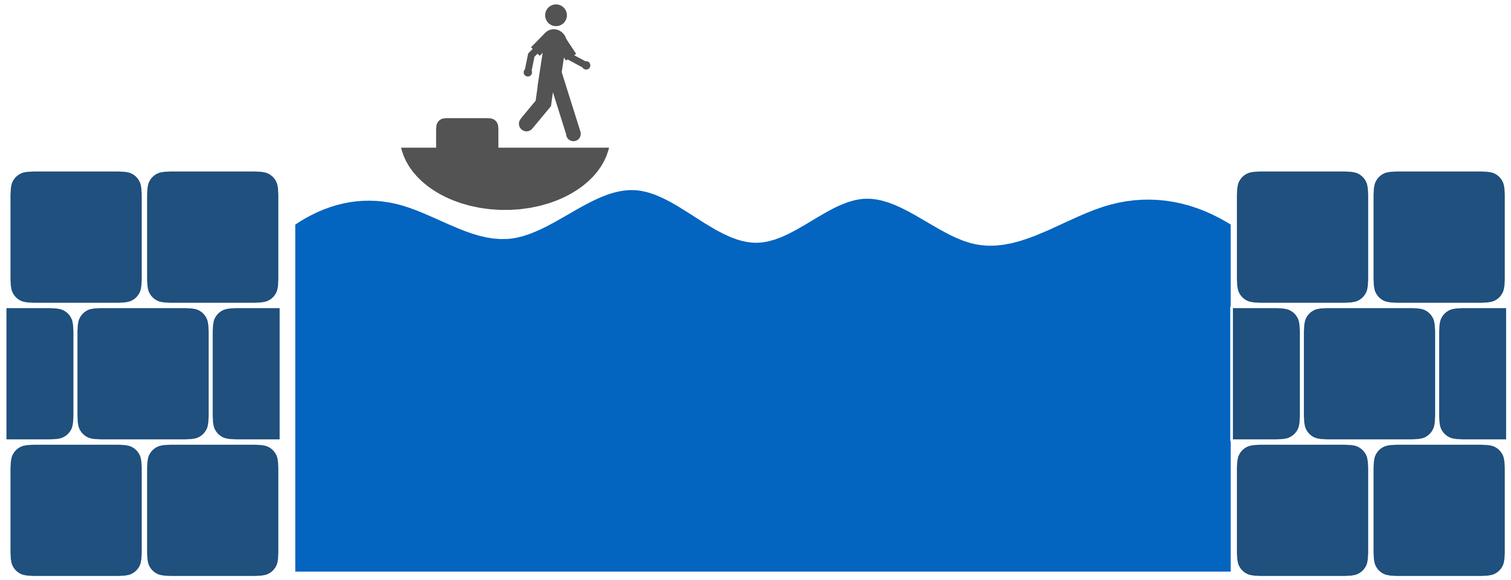
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

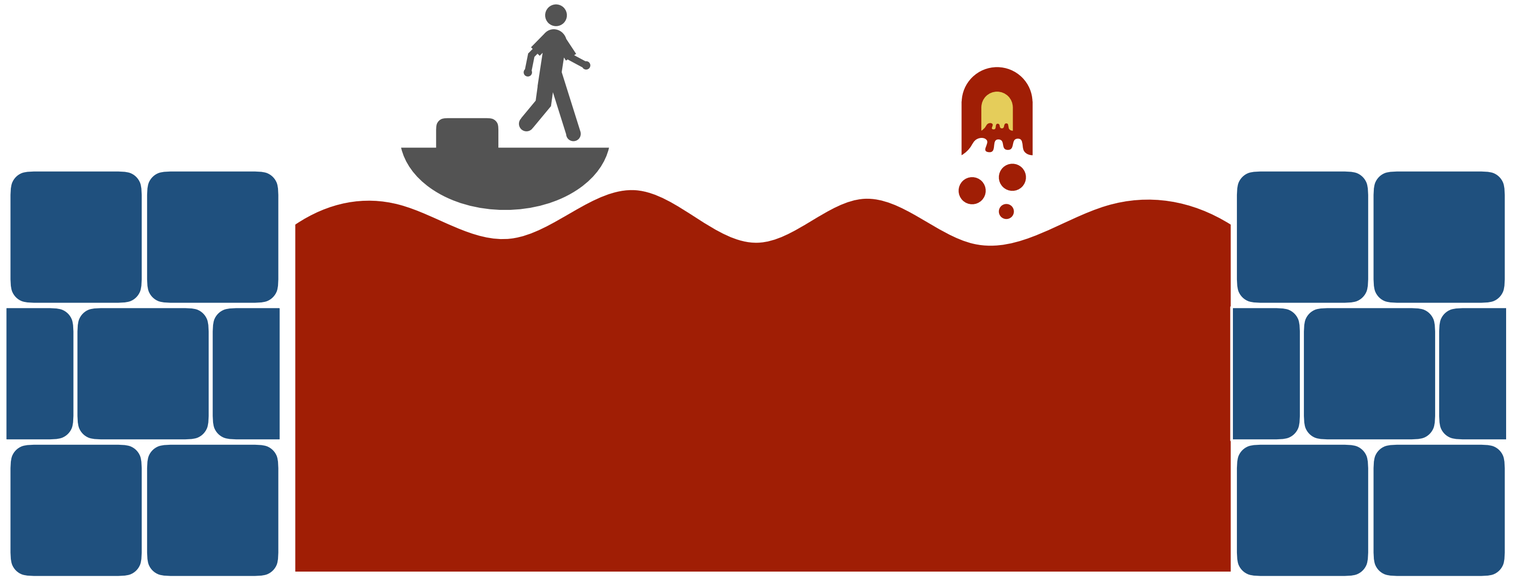
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

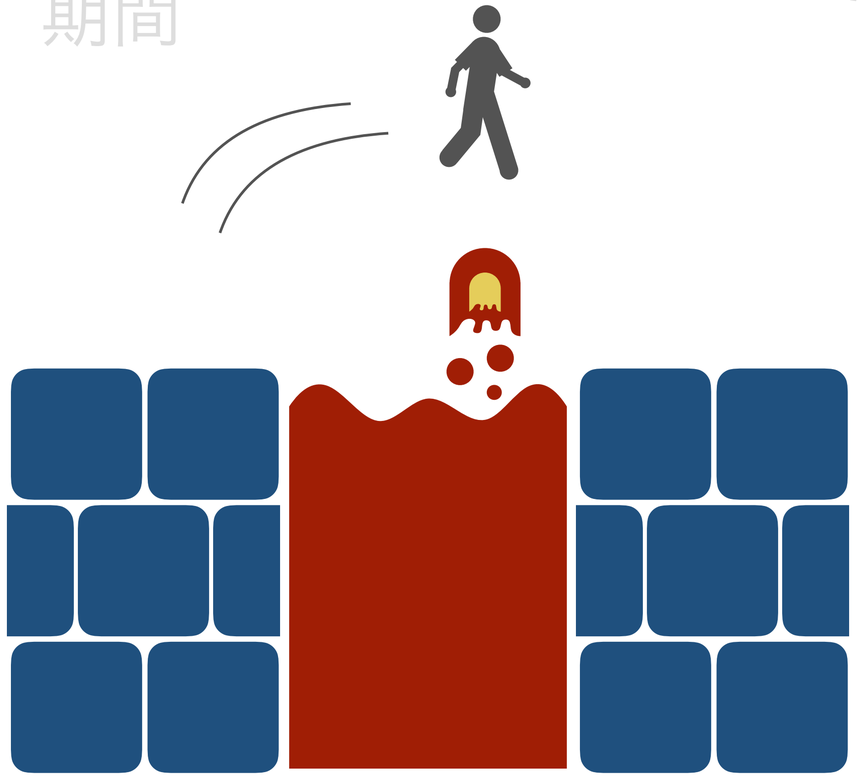
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

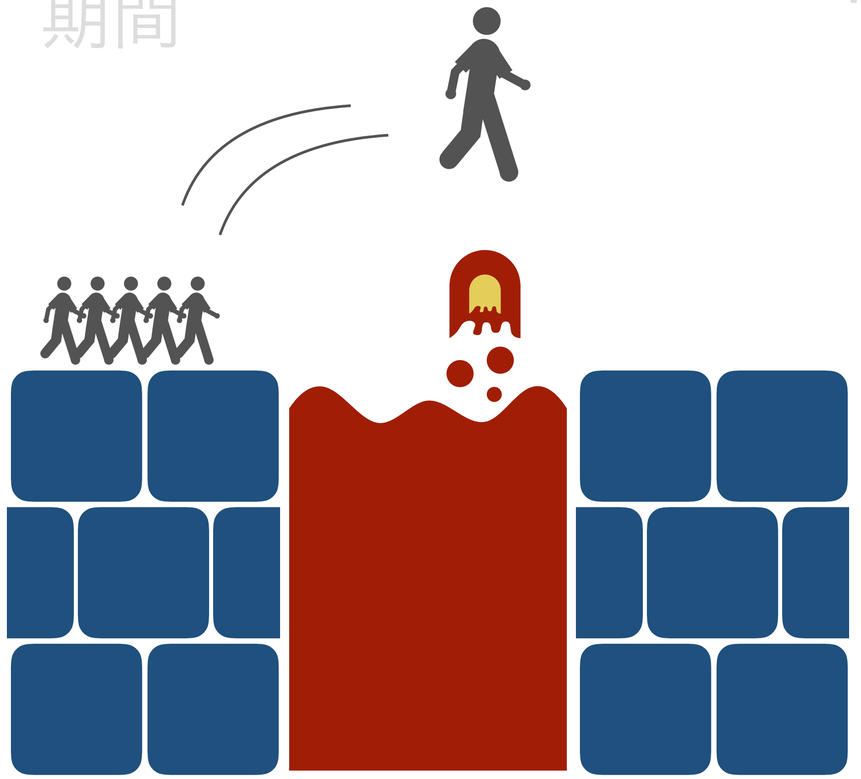
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

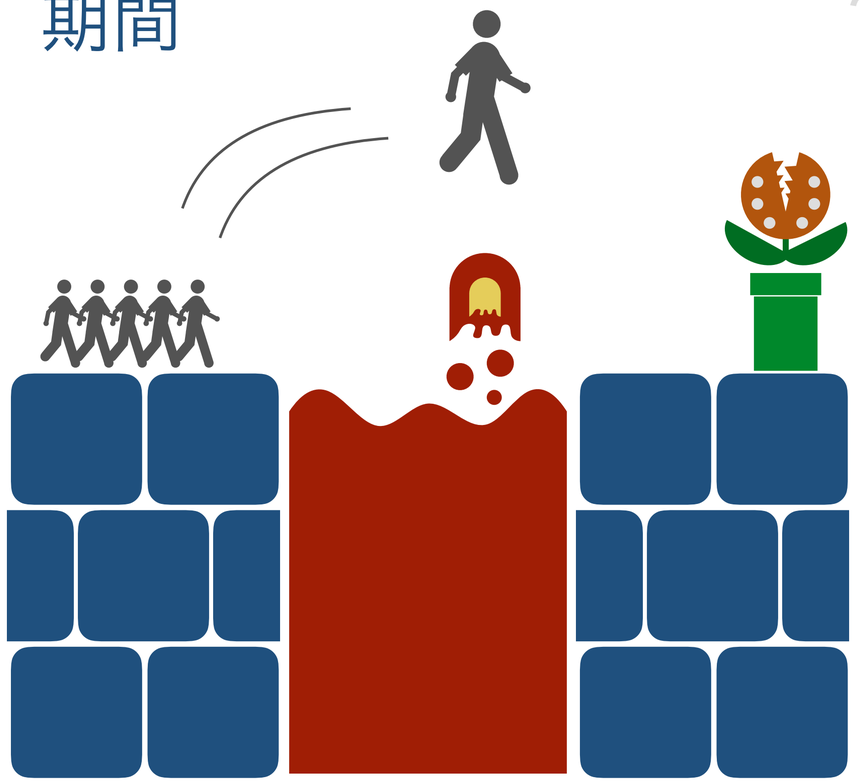
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

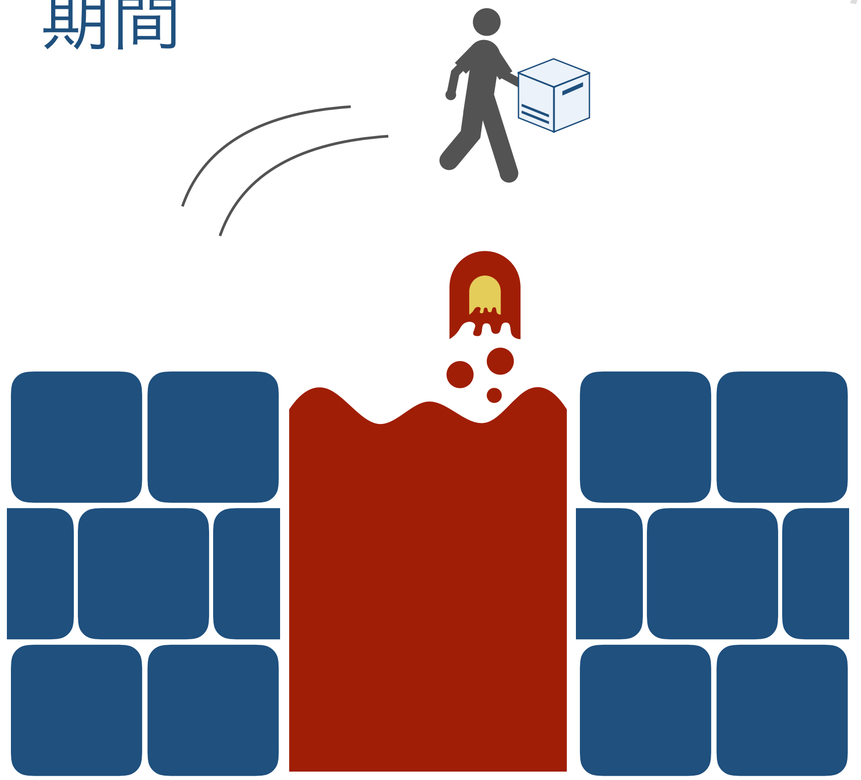
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

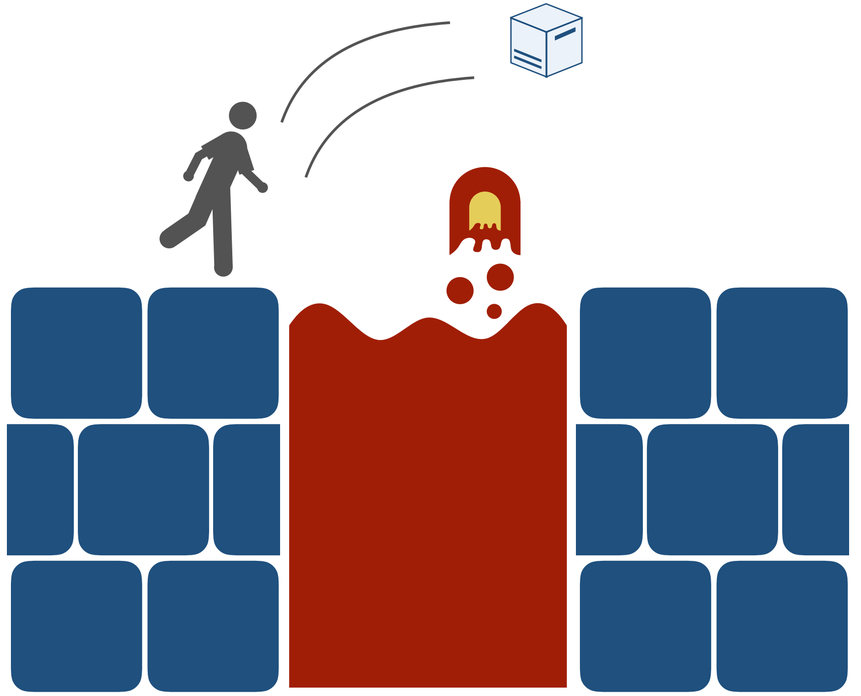
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

ローケーション

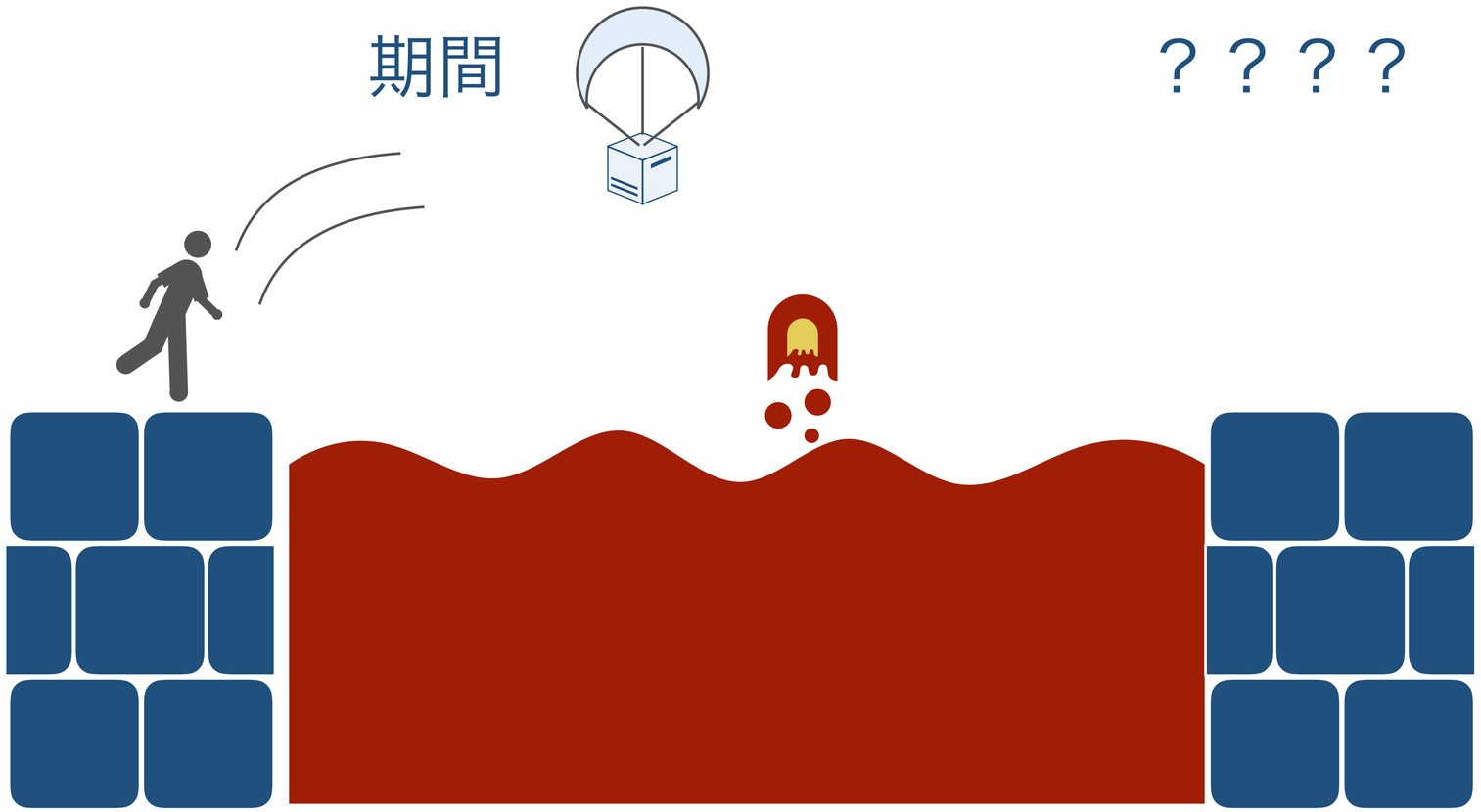
渡る回数

渡るモノ (モノ・ヒト)

分量

期間

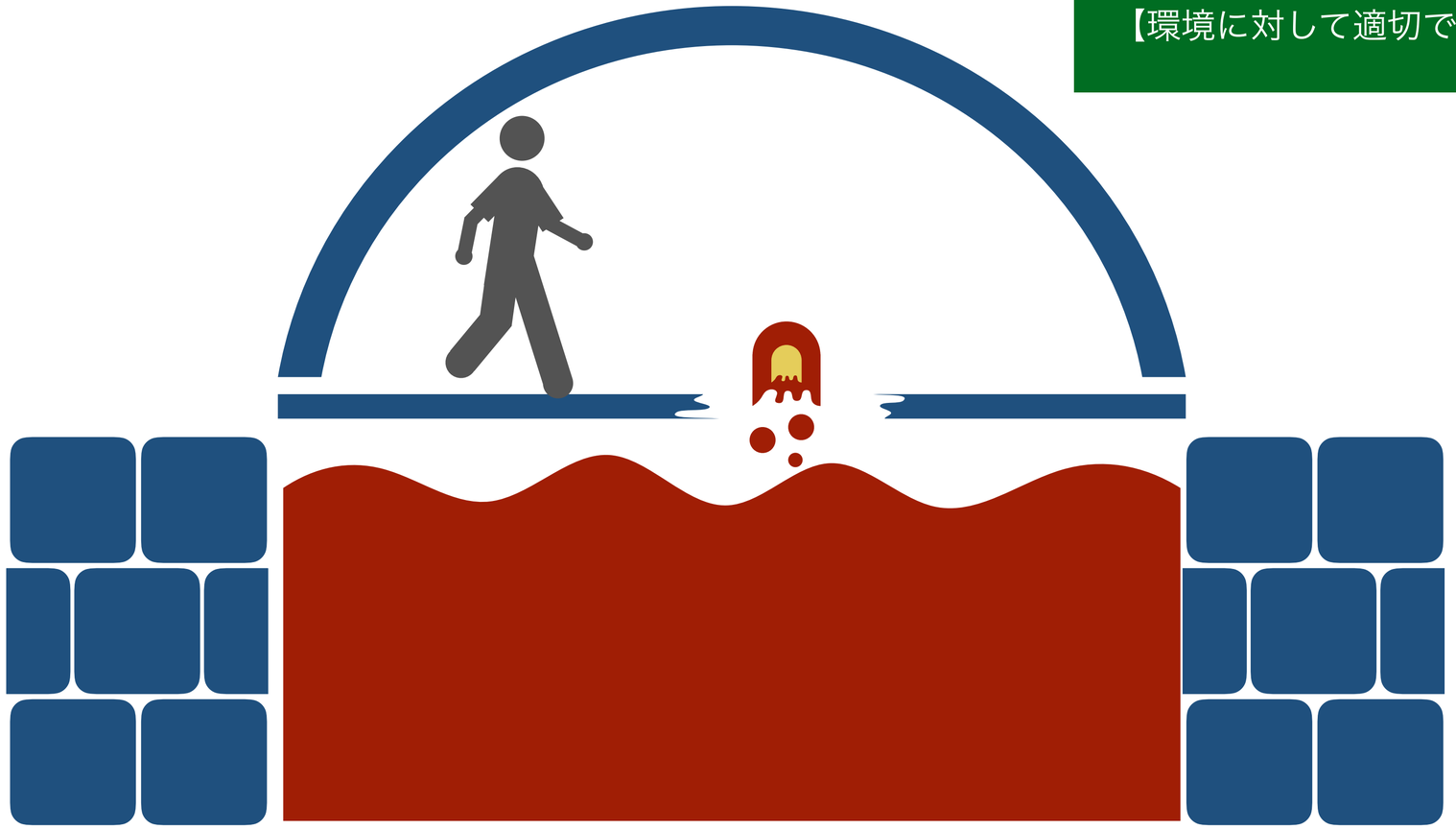
？？？？



顕在化していないニーズ、表現しきれないニーズなど様々なため、できるだけ現物やモデルでフィードバックをもらわなければ、解に近づけません。

仮に依頼主からフィードバックを得る機会や、現地に配置する機会が終盤だとすると、対応が遅れてしまうことが出てきてしまいます。

【環境に対して適切でない】



ビッグバン インテグレーション

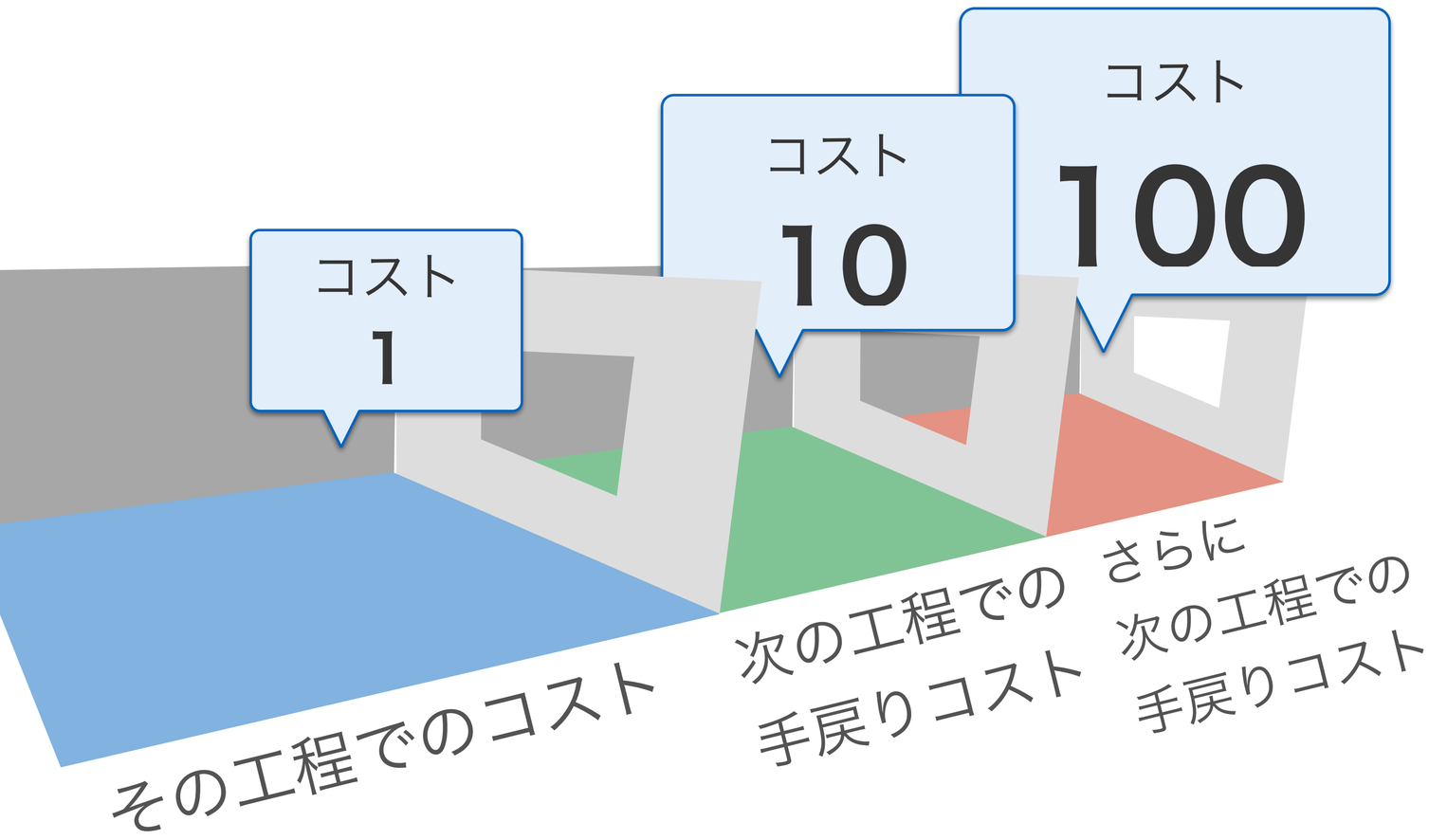
仮に依頼主からフィードバックを得る機会や、現地に配置する機会が終盤だとすると、対応が遅れてしまうことが出てきてしまいます。

【環境に対して過度な対応】

ビッグバン インテグレーション

1 : 10 : 100

継続的インテグレーション



その工程で発見して修正したコストを1とすると、次の工程に持ち越してしまった場合は、問題の分析や人の割り当て、変更の統合に追加コストがかかるため、10 倍くらいのコストが必要と言われています。さらにその次の工程まで持ち越してしまったら（コスト増に対するプラクティスもあることは確かですが）

変更時に頻繁にコードを統合し、BVT することで、手戻りコストを最小限に抑える

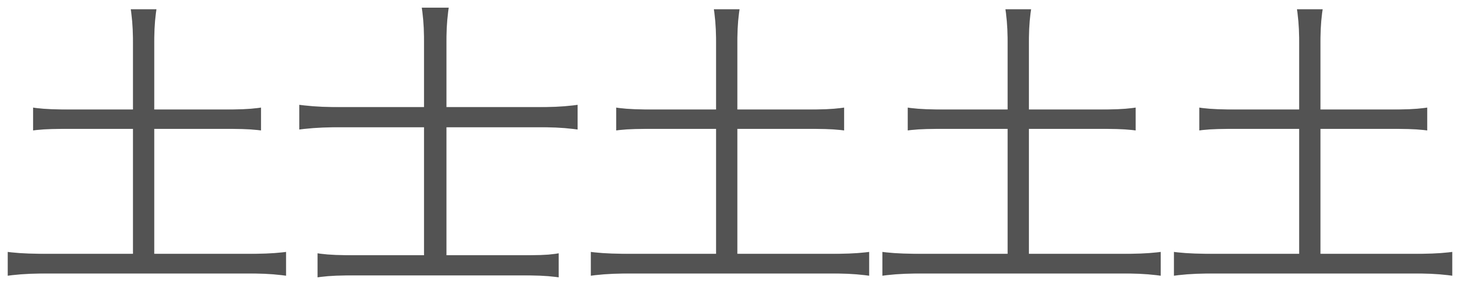
エクササイズ

まず3つほど、その工程で対処できた場合をエクササイズでやってみます。

その後に、次やその次の工程に持ち越してしまった場合、最初の３つのエクササイズがどれくらい難易度を上がっているのかを体験してみてください。

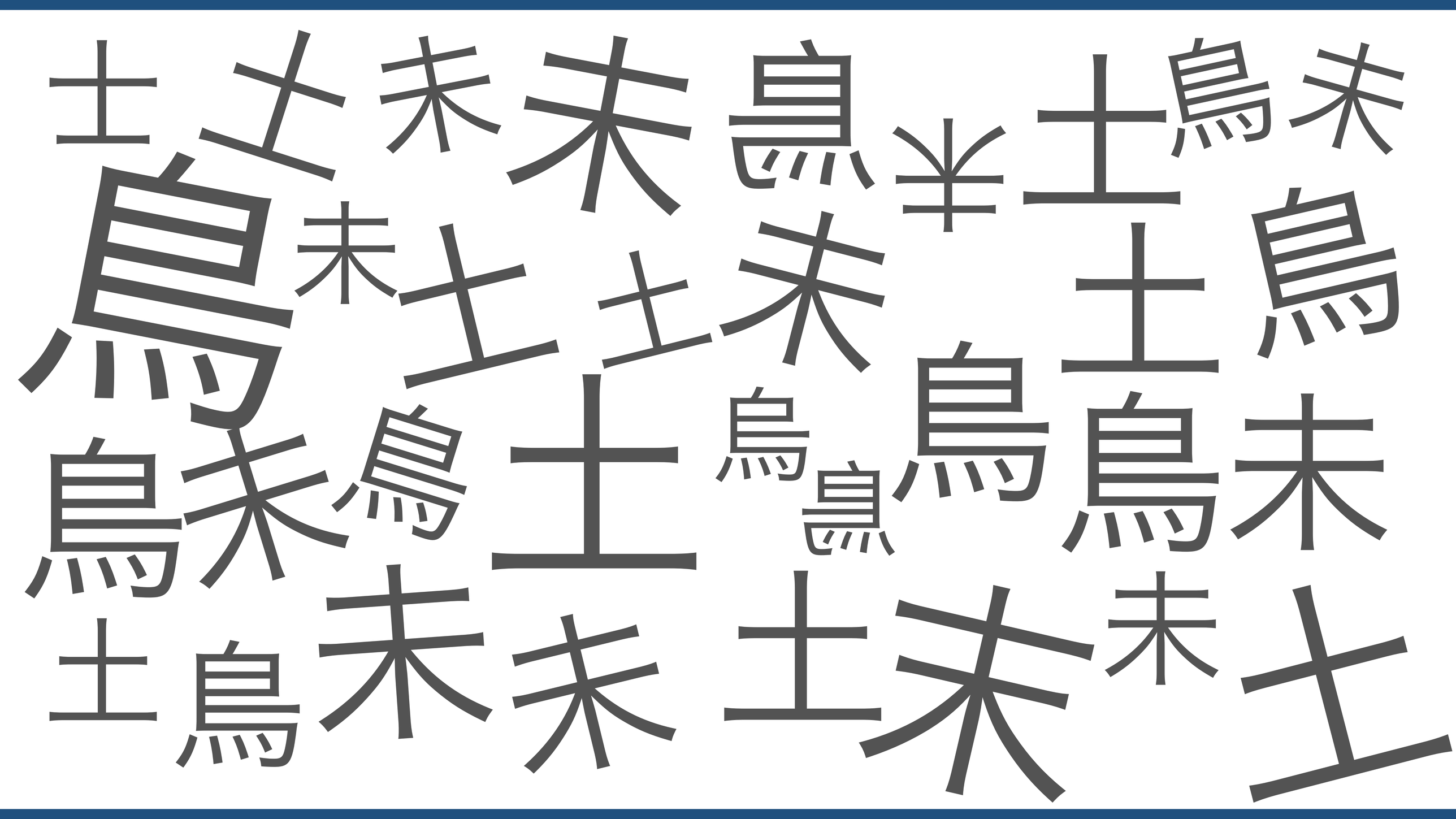
（答えは記載していません）

鳥鳥鳥鳥鳥
鳥鳥鳥鳥鳥
鳥鳥鳥鳥鳥




未
未
未
未
未

土土未未鳥鳥鳥未
鳥未土土未鳥土未
鳥未鳥土鳥未鳥未
土鳥未未土土未土

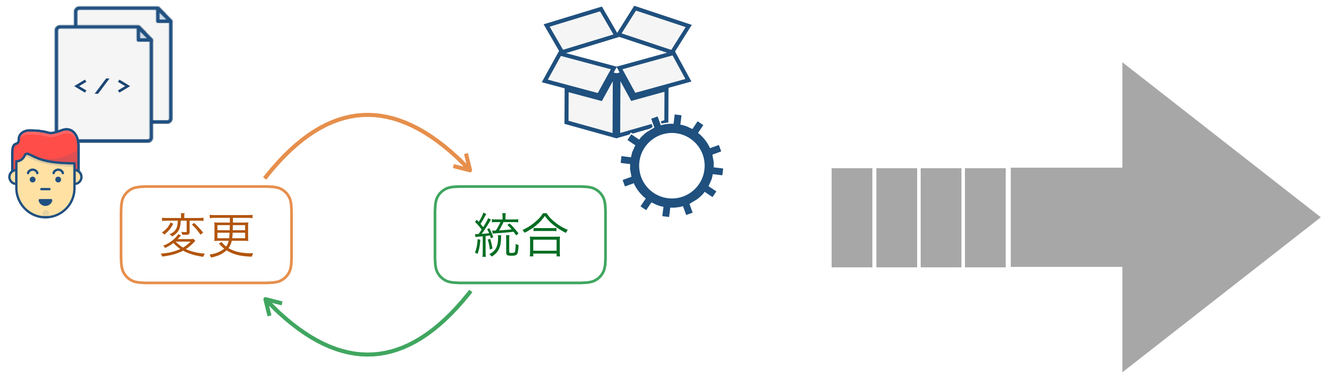

土
土
未
未
鳥
土
鳥
未
鳥
未
土
土
未
土
鳥
鳥
未
鳥
土
鳥
鳥
鳥
未
土
鳥
未
未
土
未
未
土

継続的インテグレーションの利点

CI の利点

①

変更時に頻繁にコードを統合し、
BVT することで、
手戻りコストを最小限に抑える

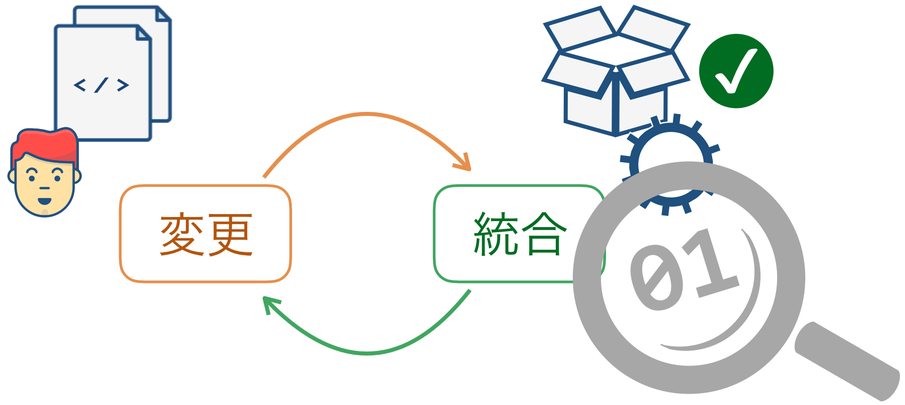


1 : 10 : 100

CI の利点

②

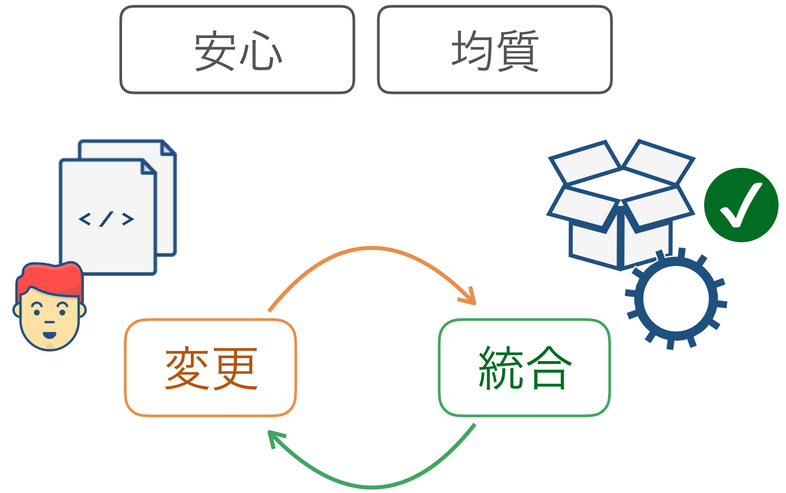
変更時に頻繁にコードを統合し、
BVT することで、
品質を維持・把握できる



CI の利点

③

ルーチンワークや属人的な手順を
形式知化し、
自動化や簡素なワークフロー
にする

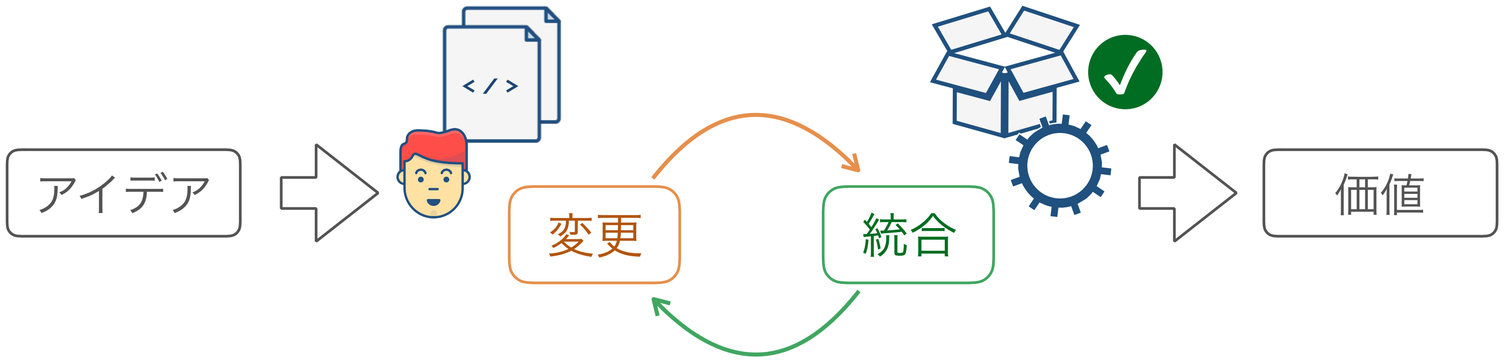


暗黙知 → 形式知

CI の利点

④

「動くソフトウェア」という
共通ゴールで結束でき、
チームと関係者の価値観や
認識を得やすくする



CI の利点

まとめ

1. コストと手間の軽減
2. 品質とその制御
3. リズムと自動化
4. 信頼と結束

継続的インテグレーションの前提

小さな単位での作業

変更の作業の単位は小さめで、制御可能なものに揃えられるとよいです。

また、複数の作業を一括でコミットなどを行わず、意味のある変更作業の単位でコミットする（トランザクション）のも大切になります。

コードの共同所有 (コード リポジトリ)

(自動化) ツールの積極的な活用

アーキテクチャの見直し (テスト可能性, 独立性)

変更を統合し、ビルド、検証し、フィードバックを得るには、テスト可能なコードにすることや、SOLID 原則、アーキテクチャの見直しも必要となります。

CI の結果に対して、早い対応（変更への適応力）が求められるわけですし、CI してゴールではありません。

コンセンサス

関係者とのコンセンサスは重要です。

開発者だけで CI を実践することも大切ですが、意図とメリットを関係者と共有し、理解と協力体制を得ることで CI のプラクティスをチーム内外で効果的に実践できるようになります。

継続的インテグレーションとアジャイル（考察)

eXtreme Programing におけるプラクティス

特定プロセスに依らないお作法

- バックエンド プロセス
- 自動化ツールの進化
- ビルド成果物の重要性

CI とビルド成果物 (動くソフトウェア)

継続的インテグレーションは、「脈拍」である

リズム

ペース

確認

Agenda

継続的インテグレーション（CI）

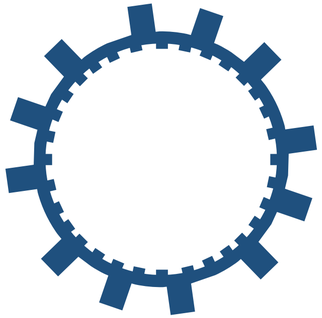
CI とチーム

CI と仕組み（ツール）

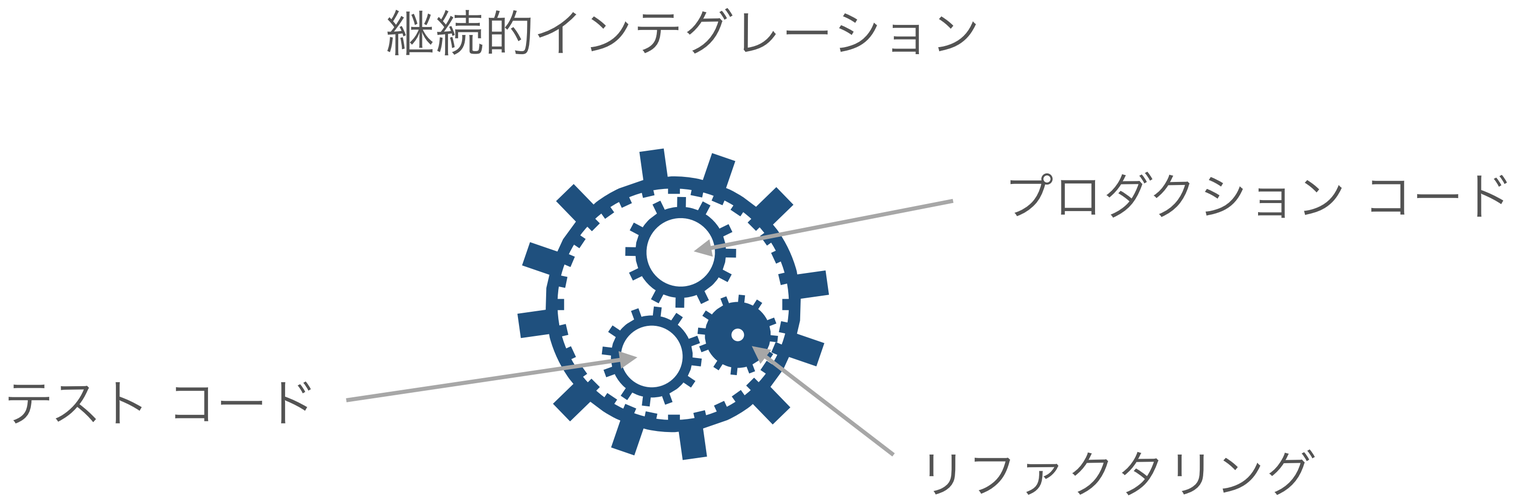
CI とチーム

CI の恩恵は開発チームだけのものか

継続的インテグレーション



CI の恩恵は開発チームだけのものか

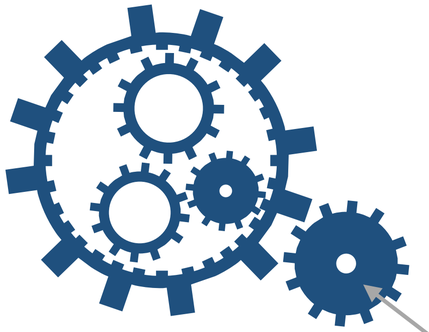


開発者のための CI

- ✓ 自分の作業への安心と確信
- ✓ 開発者同士の協調のペースメーカー
- ✓ 手間・凡ミスの回避

CI の恩恵は開発チームだけのものか

継続的インテグレーション

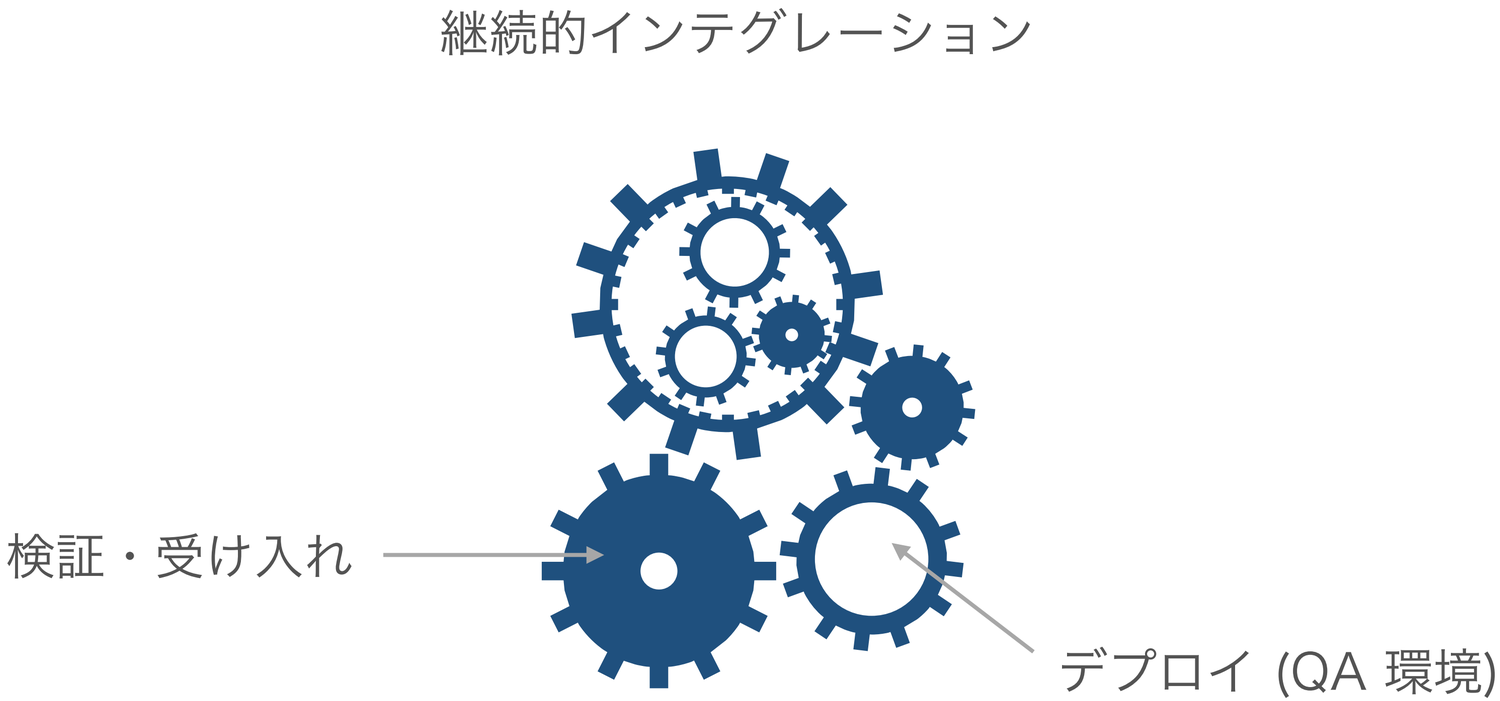


バックログ (要件) / タスク

開発チームのための CI

- ✓ プロジェクトをコントロール
- ✓ 計画と成果の確認 ＝ 現実的な進捗
- ✓ 情報収集、レポートの軽減

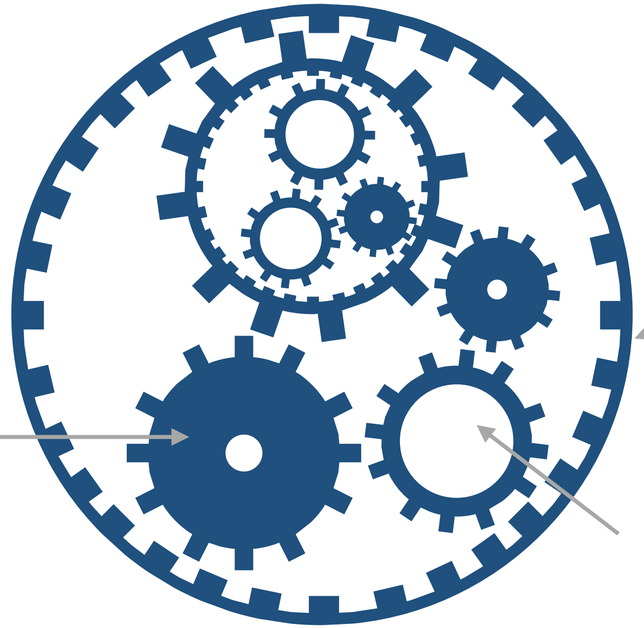
CI の恩恵は開発チームだけのものか



QAチームのための CI

- ✓ 明確なテスト対象
- ✓ 計画に沿った検証と結果報告
- ✓ 開発チームとの不毛なやりとりの軽減 (バグ ピンポン)

CI の恩恵は開発チームだけのものか



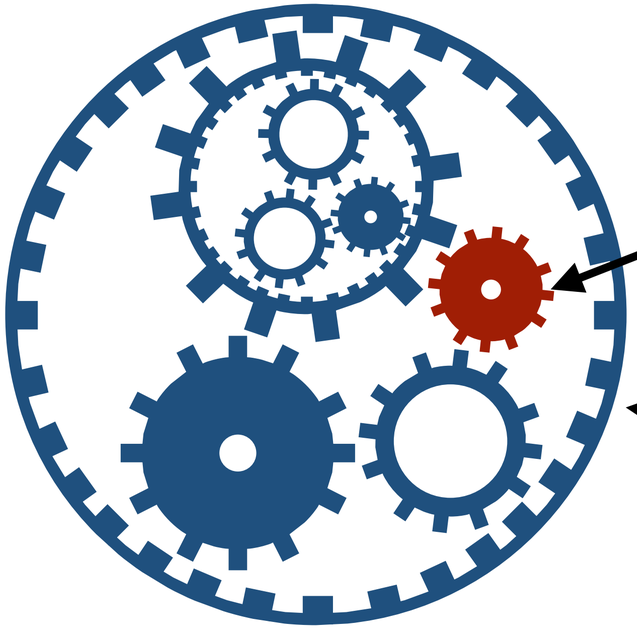
プロダクトのための CI

- ✓ プロダクトをコントロール
- ✓ 計画と成果の確認 ＝ プロダクトの状態
- ✓ 情報収集、レポートの軽減

継続的インテグレーションは万能か

ビルドの識別とトレーサビリティがカギを握る

継続的インテグレーション



誰が見てもわかる情報は何か

バックログ (要件)、タスク

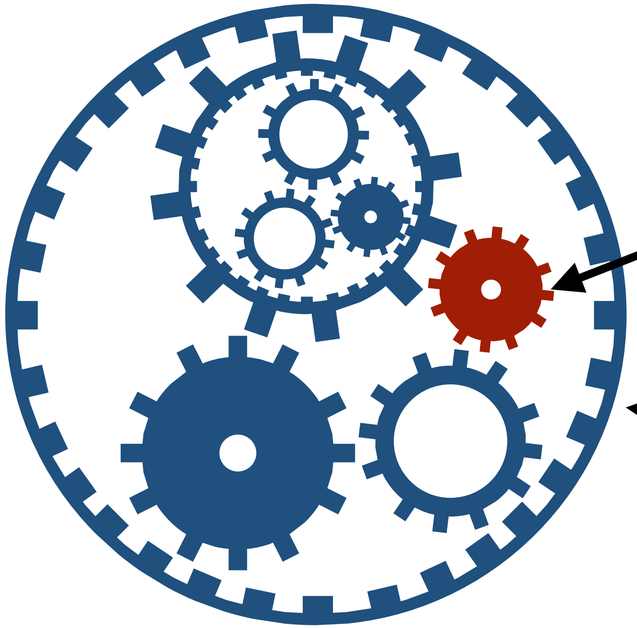
自然言語、箇条書き、共通の関心ごと

動くソフトウェア (各関係者向けデプロイ環境)

試せる、体感できる環境、ビルドに触れる機会

ビルドの識別とトレーサビリティがカギを握る

継続的インテグレーション



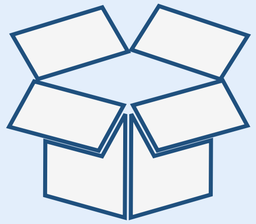
誰が見てもわかる情報は何か

バックログ (要件)、タスク

自然言語、箇条書き、共通の関心ごと

動くソフトウェア (各関係者向けデプロイ環境)

試せる、体感できる環境、ビルドに触れる機会

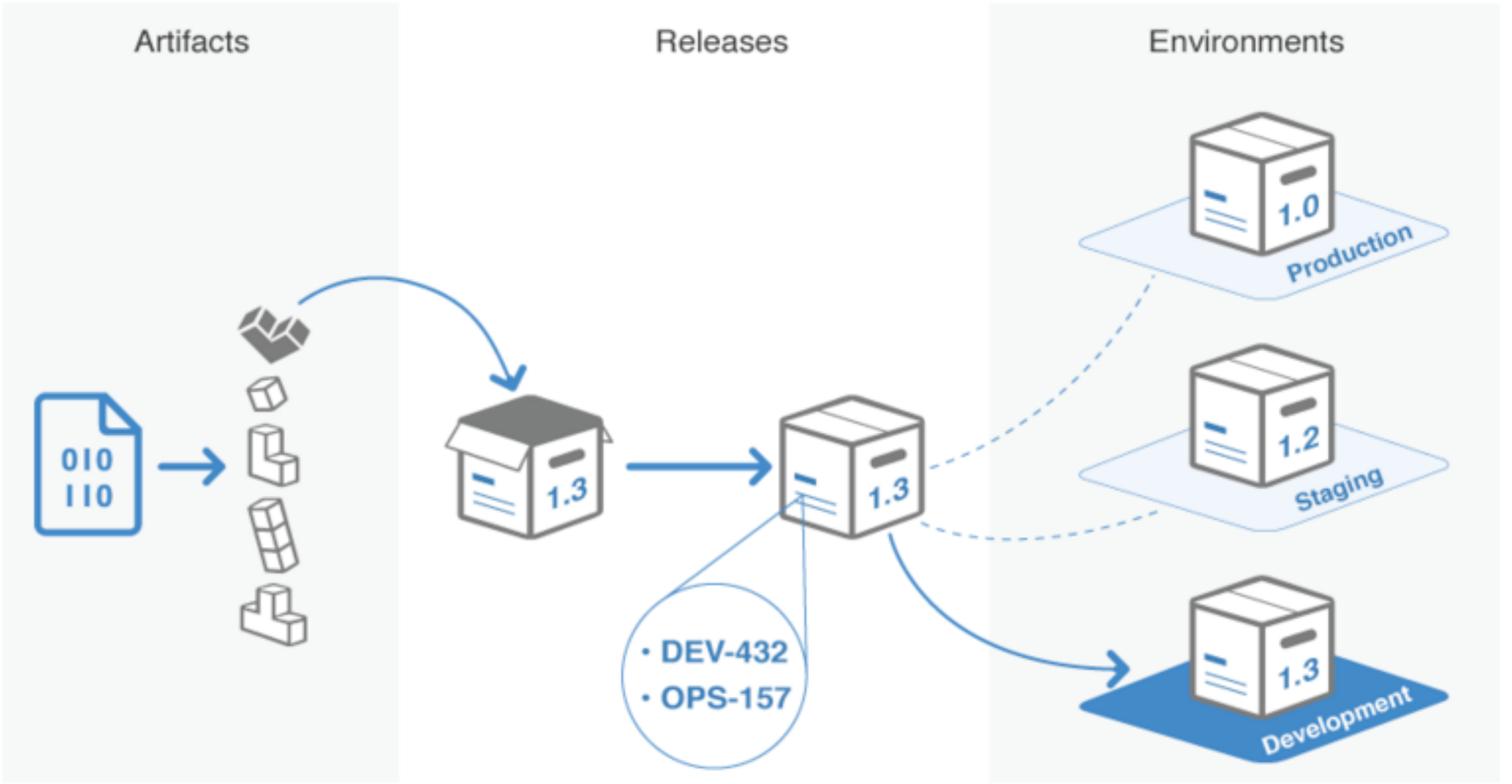


ビルド成果物の識別

- ✓ 新機能, 拡張機能, バグ改修
- ✓ コードベース ｜ 変更コード
- ✓ ビルドの品質指標とステータス

ビルドの識別とトレーサビリティがカギを握る

Consider the following diagram:



https://ja.confluence.atlassian.com/display/BAMBOO051/Deployment+projects

Agenda

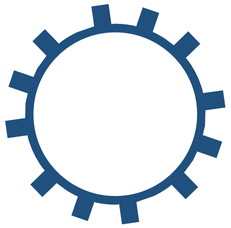
継続的インテグレーション（CI）

CI とチーム

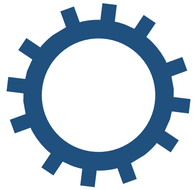
CI と仕組み（ツール）

CI と仕組み（ツール）

継続的インテグレーションを支える仕組み (ツール)



自動ビルドのツール

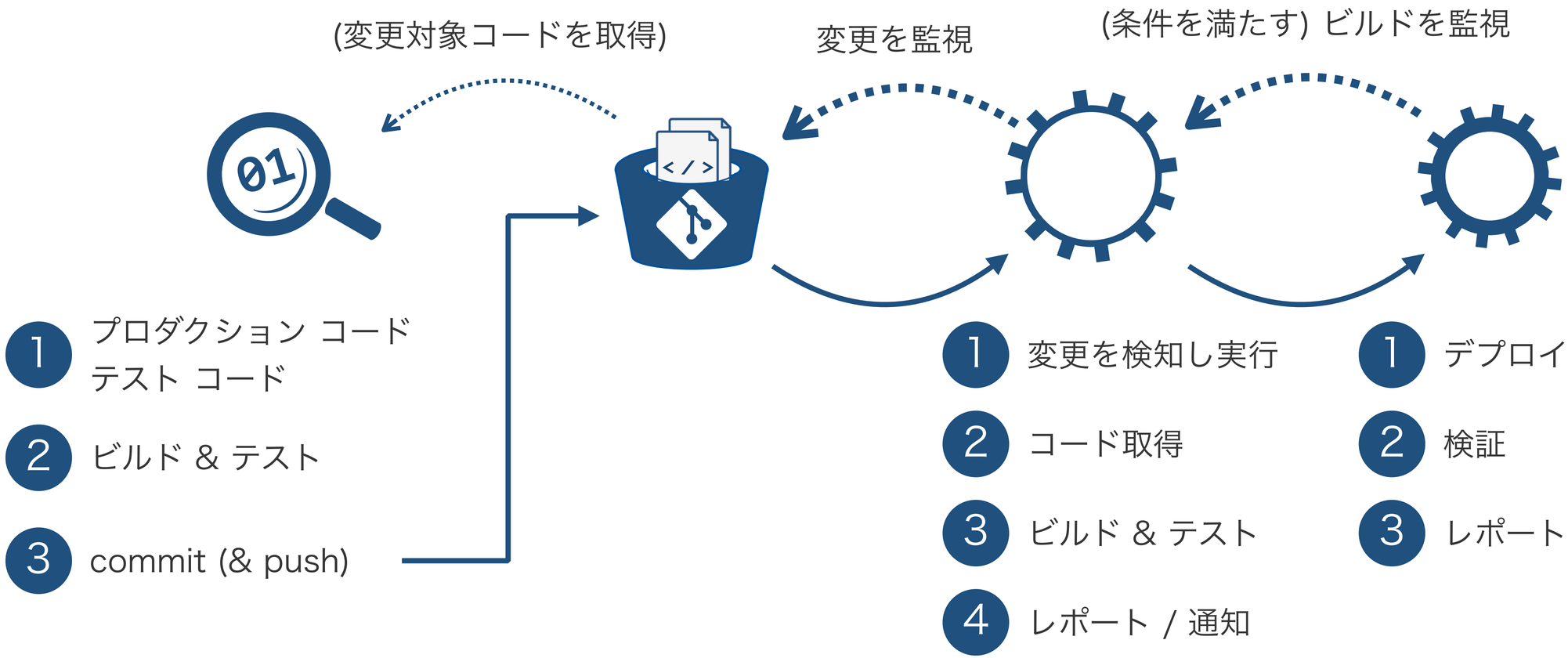


自動デプロイのツール

バージョン管理ツール

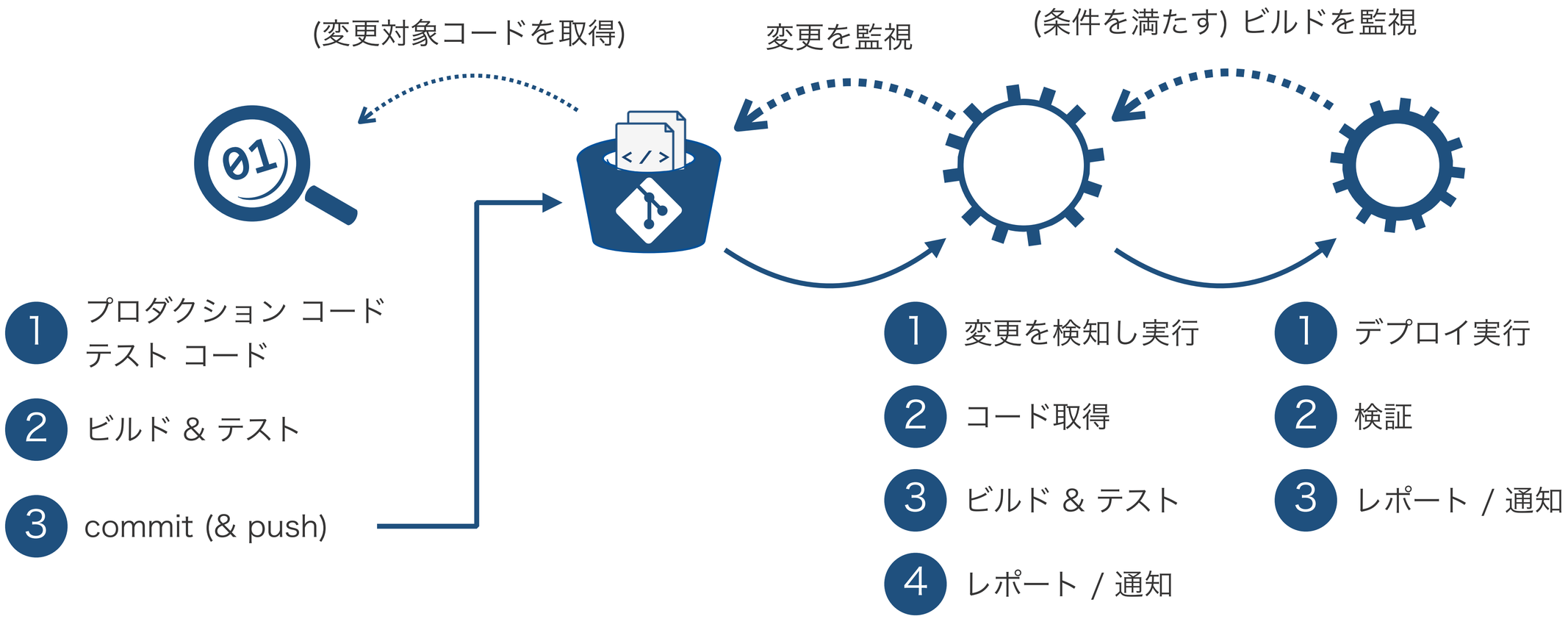
テスト フレームワーク

継続的インテグレーションを支える仕組み (ツール)



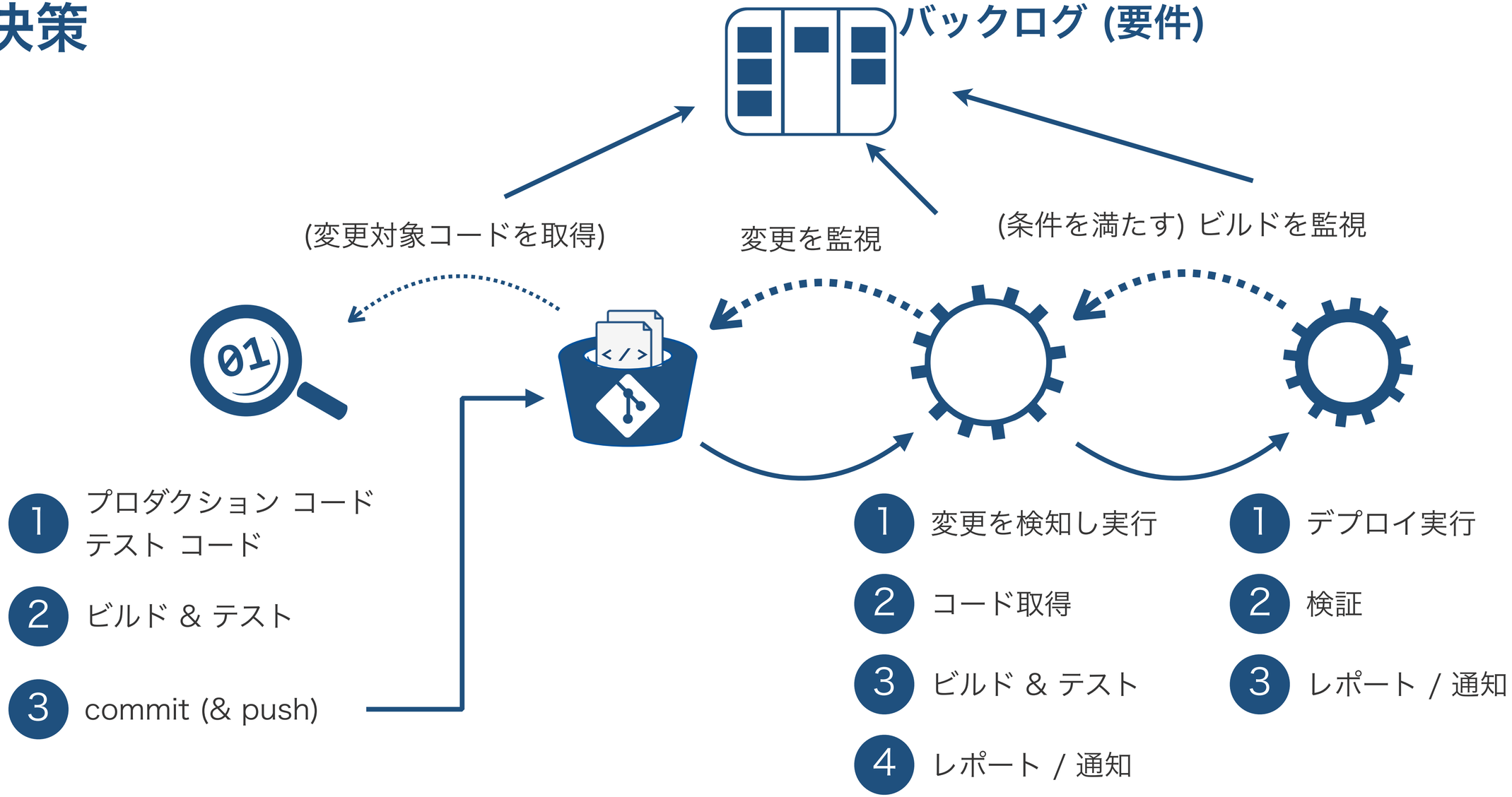
継続的インテグレーションを支える仕組み (ツール)

課題｜作業の省力化と自動化、その恩恵は受けられるようになるが
全体俯瞰するのは難しい。開発者以外には把握が難しい。



継続的インテグレーションを支える仕組み (ツール)

解決策

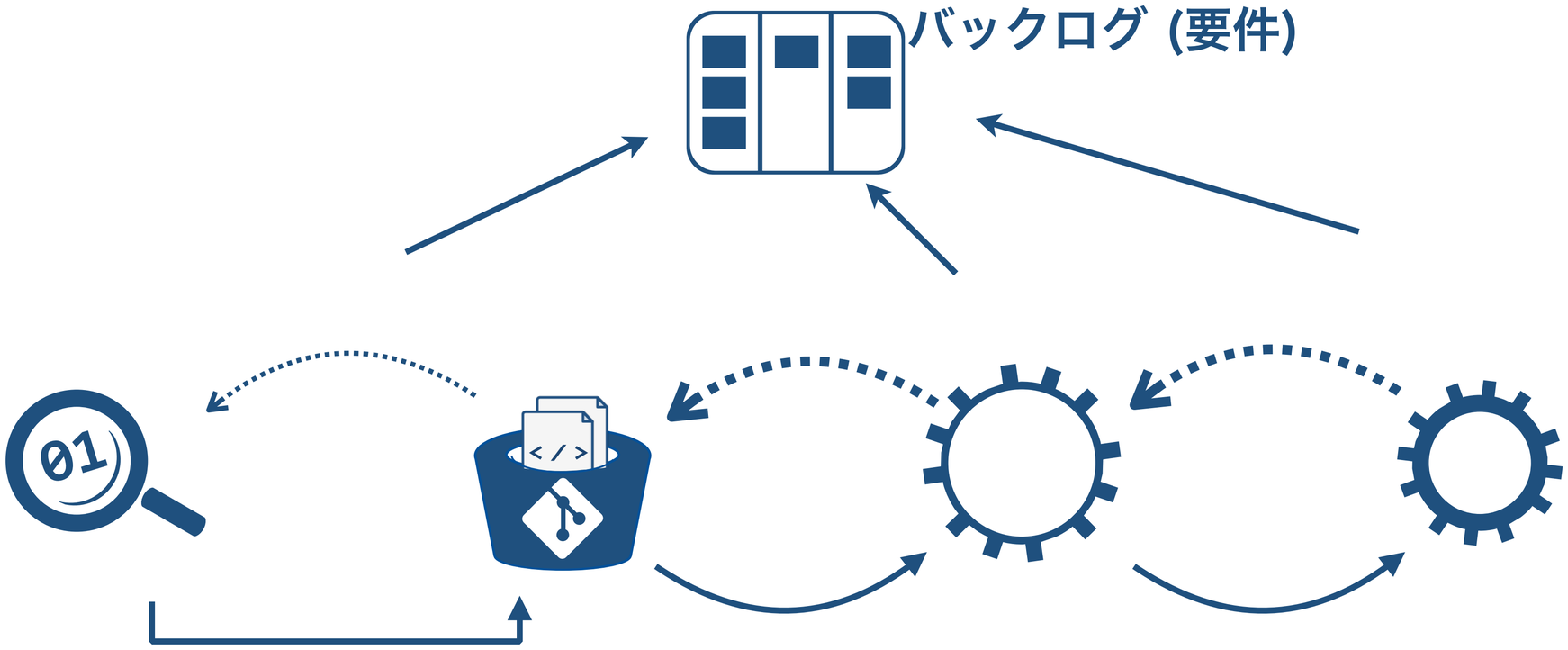


リリースに備えるとは

何がリリースされるのか

それは適切なのか

改善ポイントはないのか



継続的インテグレーションの段階

CI の 3 段階

安心

安心のための CI

石橋を叩いて渡る

開発者のコード変更時に常に実行されるビルド

手戻りコストを最小限にとどめる｜コード変更のストレスを軽減する

(テストはあったほうがいいが、なくても始めるべき、醸成させる)

CI の 3 段階

改善

改善のための CI

早期フィードバックの恩恵

ビルドやデプロイのプロセスの見直しや、課題や知見の洗い出し

開発フローや、規約、ルール、ツール活用の見直しの機会

QAチームや運用チームからのフィードバックと協力の機会

CI の 3 段階

信頼

信頼のための CI

価値のパイプライン化

バリューストリームの確立によるコンセンサス

コードだけでなく、ソフトウェアの変更の恐れを取り除く

QAチームや運用チームとの共同所有と一体化の機会

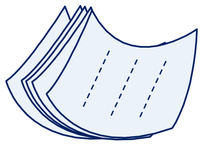
継続的インテグレーションの悪手

アンチパターン

- CI システムに余りマシンを充てる
- 成熟するまでスタートしない
- バリューストリームを描けない

ここに挙げたのは一例にすぎません

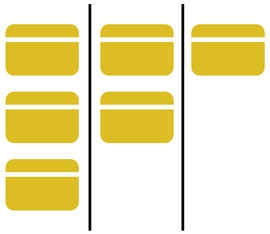
企画　計画　開発　ビルド　デプロイ

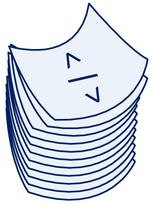


企画書

バックログ

タスクボード





コード w/ DVCS　ビルド w/ CI

ステージング

プロダクション

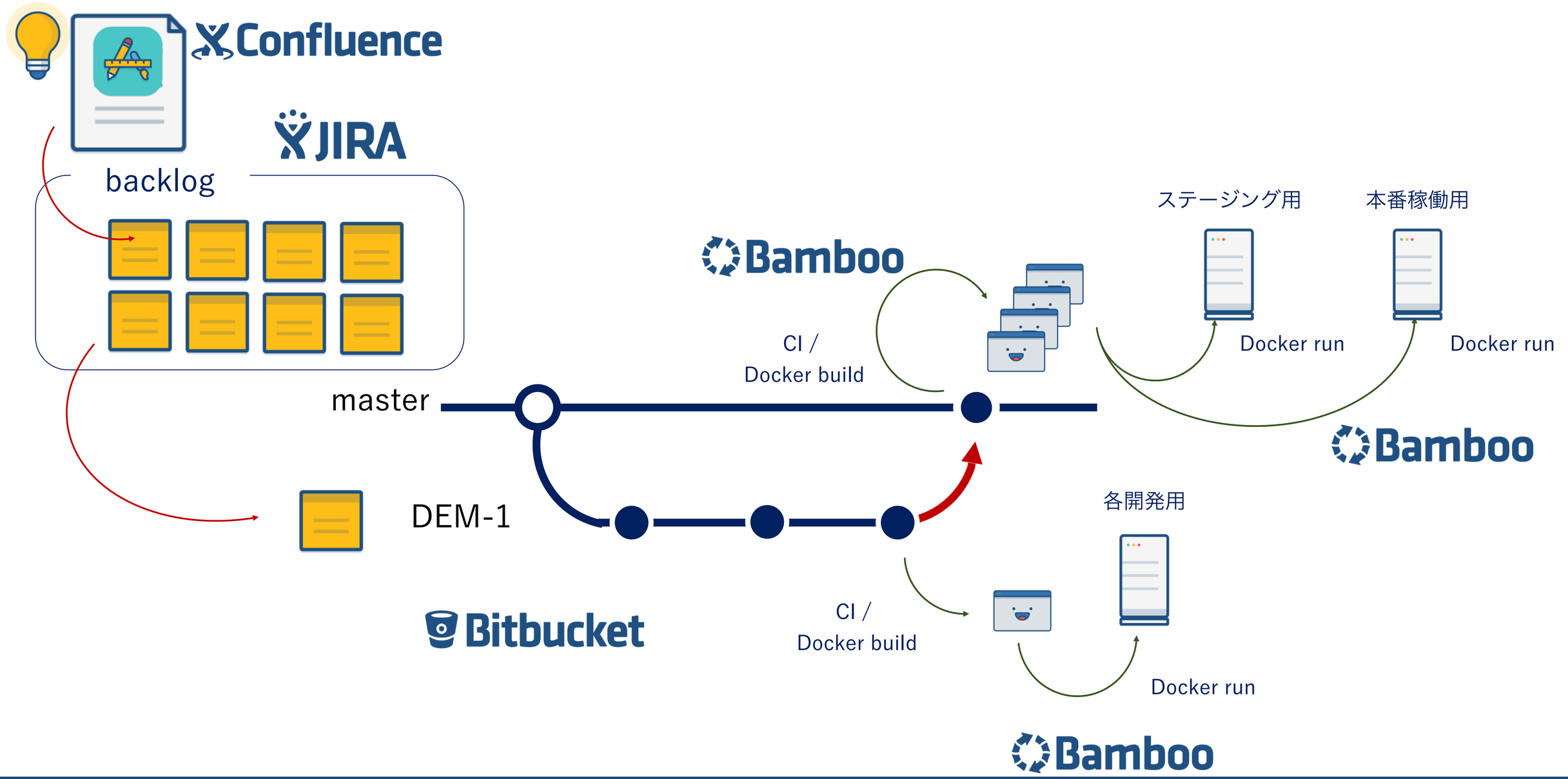
バリューストリーム

※ ユーザーに提供するまでの流れ

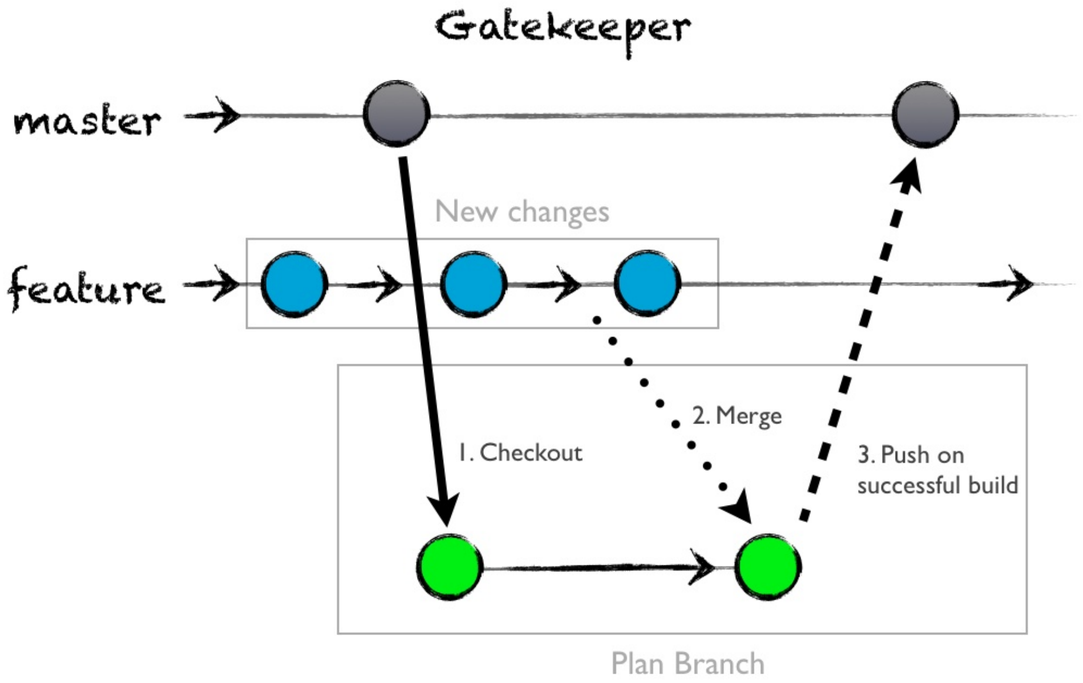
デモンストレーションで見る CI の必要性

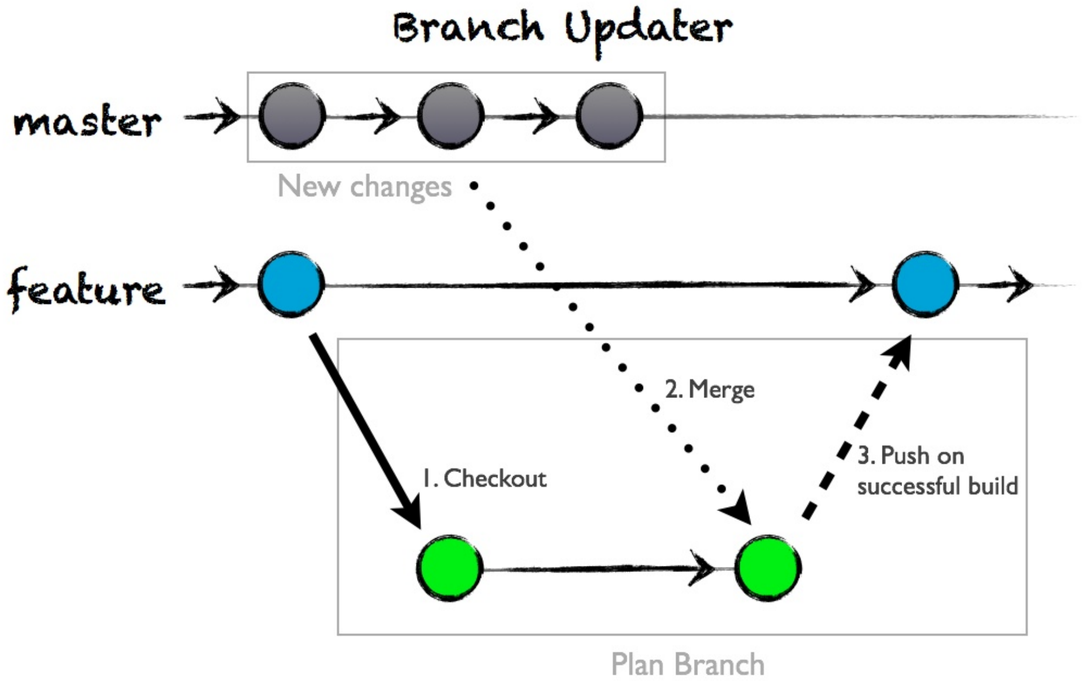
バックログからデプロイの流れの例

フィーチャーブランチ、プルリクエスト、Docker コンテナを活用している例



フィーチャーブランチとマージ戦略





https://ja.confluence.atlassian.com/bamboo/using-plan-branches-289276872.html

まとめ

継続的インテグレーションとは

リリースに備え、チームが一つになって行う取り組み

「CI ＝ 自動化｜ツール｜アジャイル」とは限らない

チームが戦略的に取り組むべきプラクティス

だが、比較的取り組みやすいところからはじめられる

Thank you!

TOMOHARU NAGASAWA • SENIOR EVANGELIST • ATLASSIAN • @TNAGASAWA