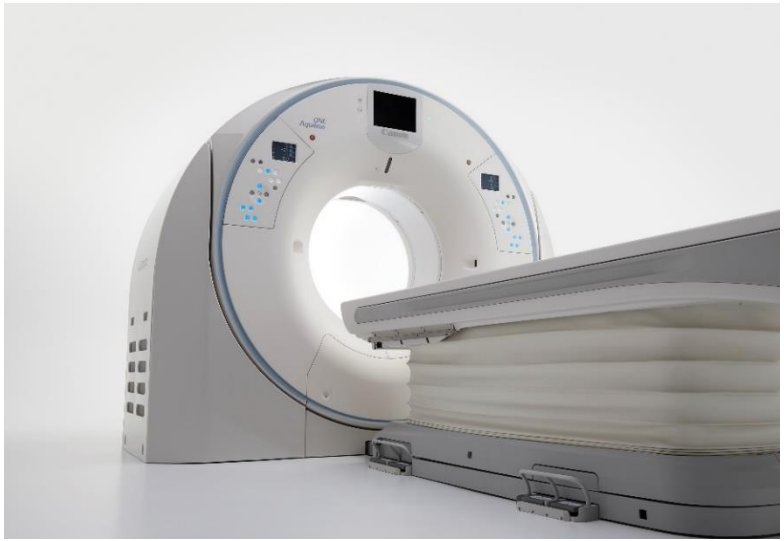
アジャイルと反復開発
～忍者式テスト20年の実践から～

キヤノンメディカルシステムズ株式会社
深谷美和
関将俊
September 2023

今日の話

- 私たちのチームはX線CT装置をXPで開発しています
- 反復開発に有効なプラクティス「忍者式テスト」を紹介し、20年以上にわたるアジャイル開発の豊富な経験から、反復開発をうまくやるためのヒントを伝えます



私たちについて

- 深谷美和
  - 医療機器（自社製品）のソフトウェア開発に従事
  - eXtremeなチームのテスター
  - 動かして試すのが好き

- 関将俊
  - 医療機器（自社製品）のソフトウェア開発に従事
  - eXtremeなチームのプログラマ
  - Ruby Core Committer

SS2023/SQiP2023

- SS2023
  - 忍者式テスト基礎編
    - 「テストからはじめよ」～忍者式テスト20年の実践から～
    - https://www.sea.jp/ss2023/programme.php

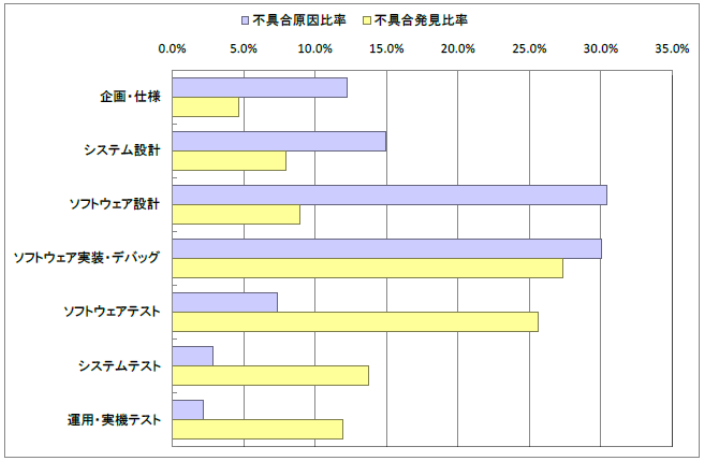
- SQiP2023
  - 忍者式テスト応用編（反復開発をうまくやるためのヒント）

今日の話

- 大規模ソフトウェア開発の難しさ
- 忍者式テスト
- 反復開発をうまくやるには
  - ストーリーを考えるときのヒント
  - バグの修正を後回しにしない
  - ペースの違いをどう考えるか
  - 規模と向き合う
- 忍者式テストの効果

大規模ソフトウェア開発の難しさ

- ソフトウェア開発
  - 企画・仕様、設計の問題を、その工程で見つけるのは難しい
  - 試さずに会議やレビューだけですべての問題を発見するのは難しい
  - 試してみたほうが効率が良い（ソフトウェアの特徴のひとつ：安く試せる）

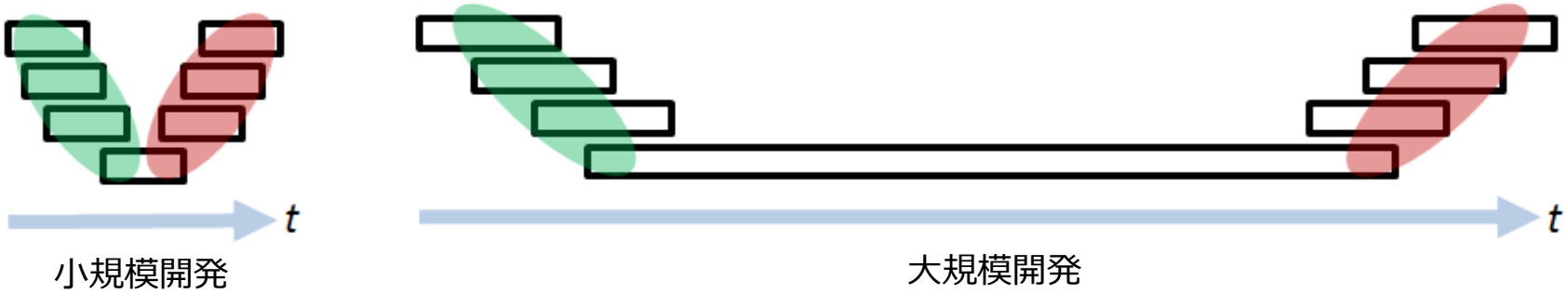


不具合の原因工程と不具合の発見工程

（独立行政法人情報処理推進機構 2012 年度「ソフトウェア産業の実態把握に関する調査」調査報告書 pp. 78 2013）

大規模ソフトウェア開発の難しさ

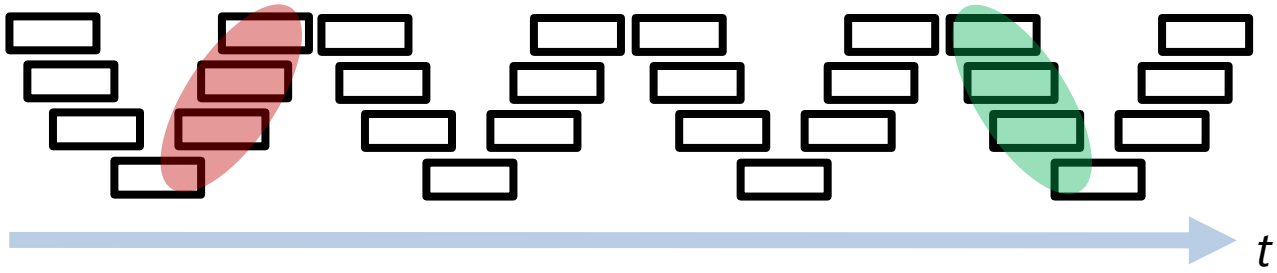
- 大規模開発は作り出す工程と確認する工程が離れてしまう
  - プロジェクトの最終段階で問題が見つかっても対策が間に合わない
  - 確認する工程でより良い仕様や設計を思いついても対応できない



小規模開発　　大規模開発

大規模ソフトウェア開発の難しさ

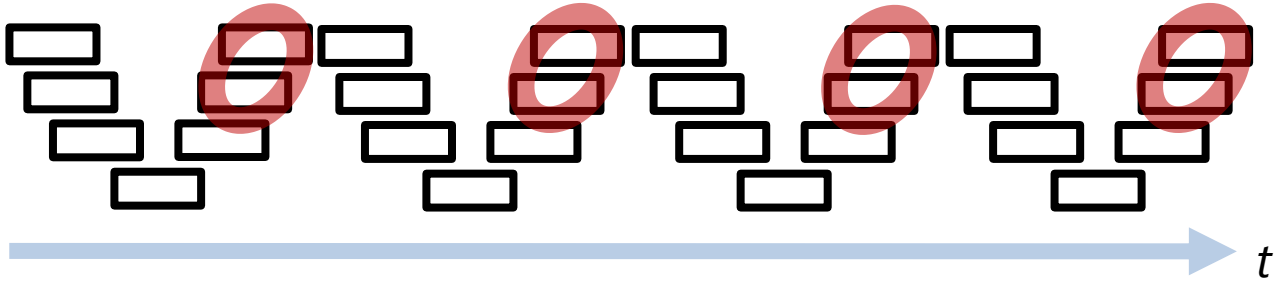
- 反復開発
  - 大きな要求を数日程度の小さな単位にわけて開発する
  - 確認する工程をより早く、ぎりぎりまで仕様や設計を調整できるようにする



  - 反復開発の利点
    - 一度に扱うスコープが小さいので細部にまで目が届きやすくなる
    - 次のV字に前回のV字で発見した問題や違和感、知見をフィードバックできる

大規模ソフトウェア開発の難しさ

- 頻繁にシステムテスト、運用・実機テストを実施しなくてはならない
  - 新しい反復により改定された仕様の確認
  - それまでに搭載されたものがすべて問題なく動作することの確認

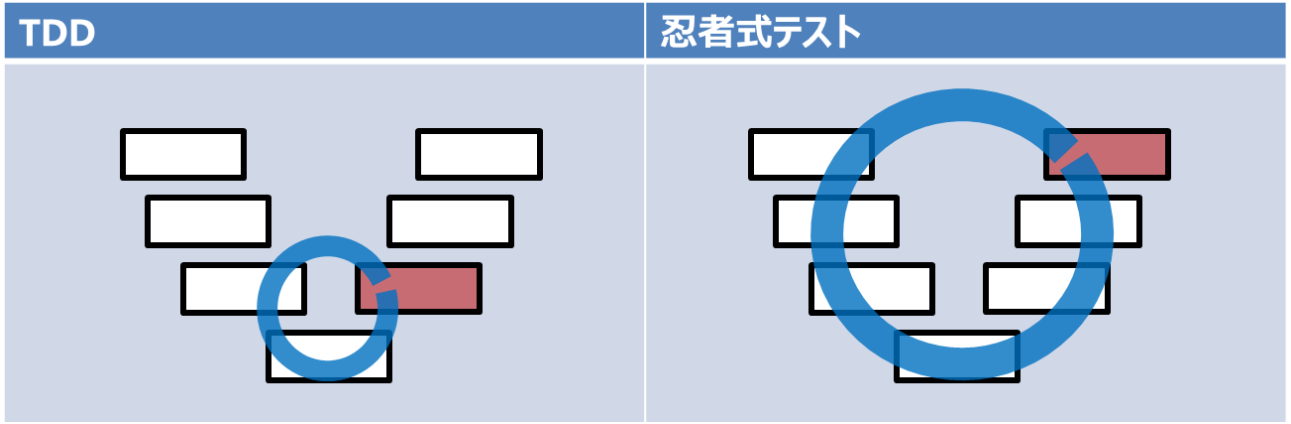


今日の話

- 大規模ソフトウェア開発の難しさ
- 忍者式テスト
- 反復開発をうまくやるには
  - ストーリーを考えるときのヒント
  - バグの修正を後回しにしない
  - ペースの違いをどう考えるか
  - 規模と向き合う
- 忍者式テストの効果

忍者式テスト

- テスト駆動開発（TDD）を受け入れテストのレベルで行うプラクティス
  - xUnitを使ったTDDのように、受け入れテストからはじめる開発
  - テストコードに導かれてプログラムを開発するように、受け入れテストに導かれてストーリーを実現する
  - ストーリーが増えるたびに、それまでに作ったすべてのストーリーの受け入れテストをやり直す

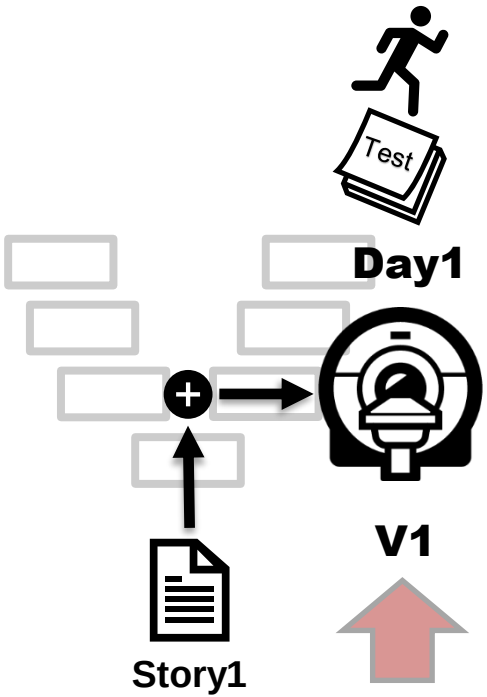


忍者式テスト

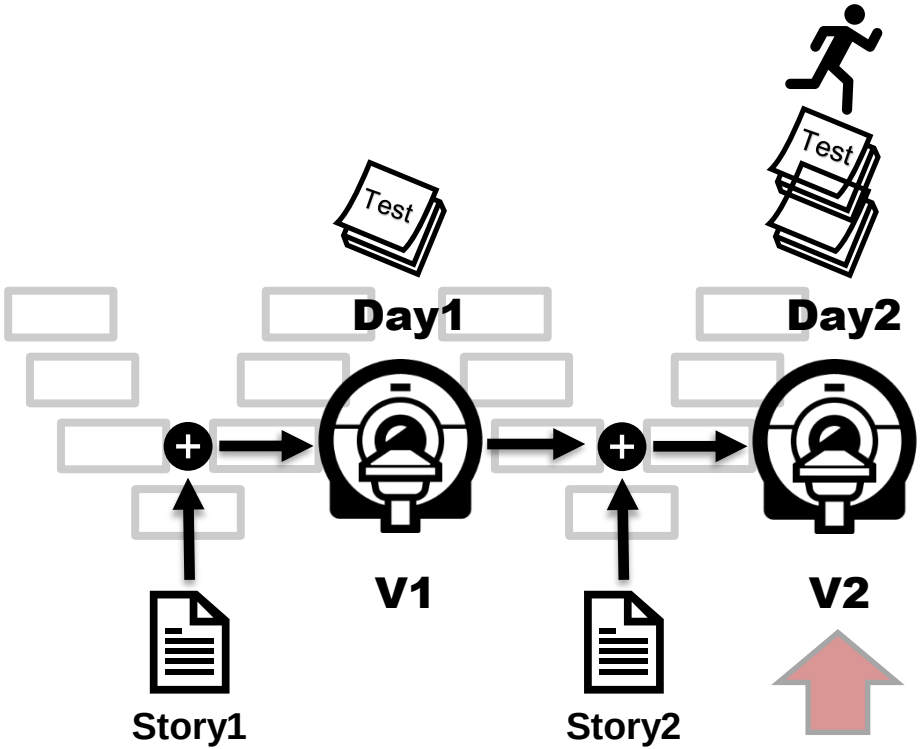
- 受け入れテストからはじめる
  - どう作るのか？ではなく、どう試すのか？から考える
    - どう試せば、ユーザーが困っていることが解決できたと分かるのか
    - どう試せば、意図したとおりに作れたと言えるのか
  - テストを起点として、ソフトウェア開発全体を考え、問い続ける
    - 本当によいシステムとなっているのか？

命名の由来：忍者が毎日成長する麻や竹の上を飛び越える修行に由来。毎日増えるテストケースの束を毎日全部やり直す様子が似ている

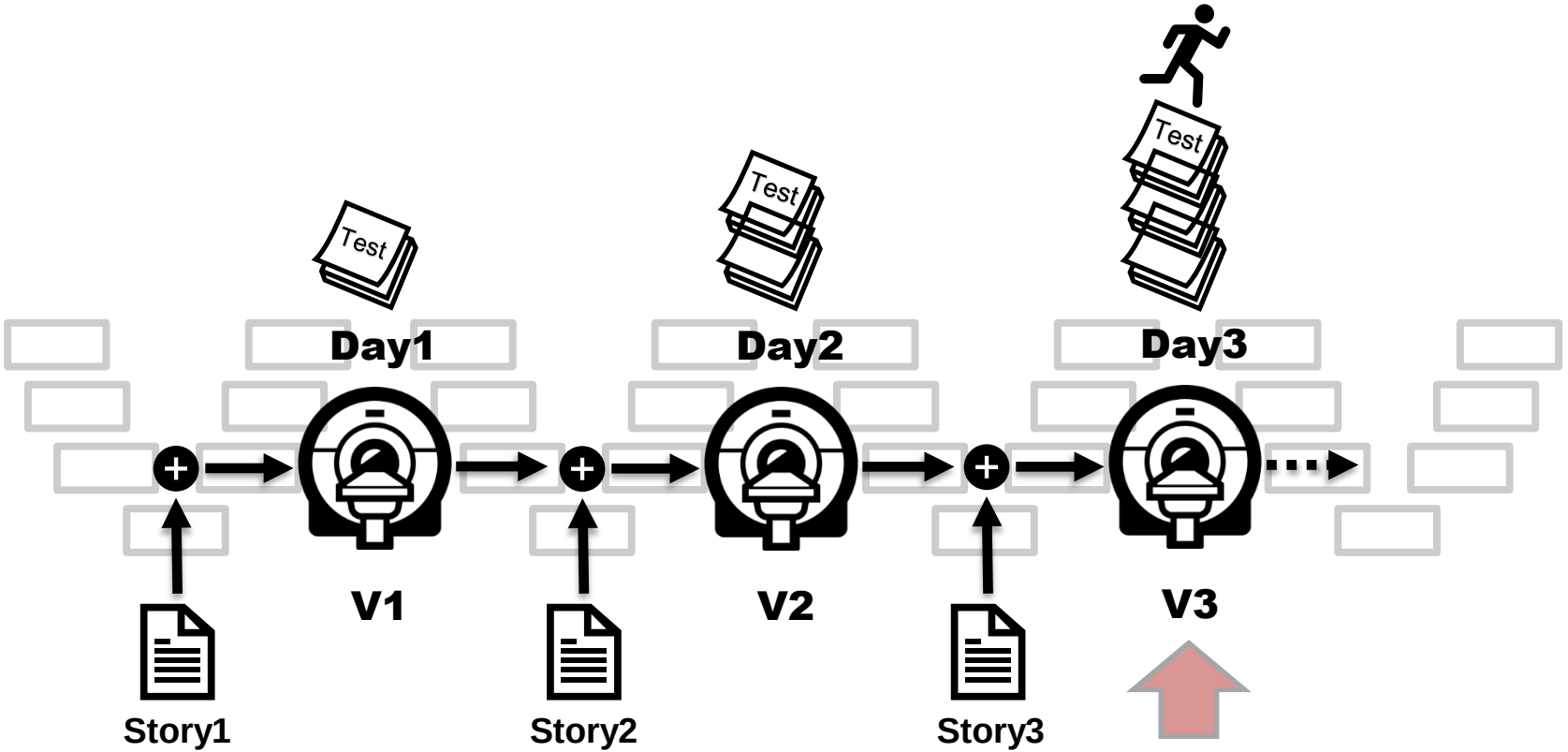
【図解】忍者式テスト



【図解】忍者式テスト



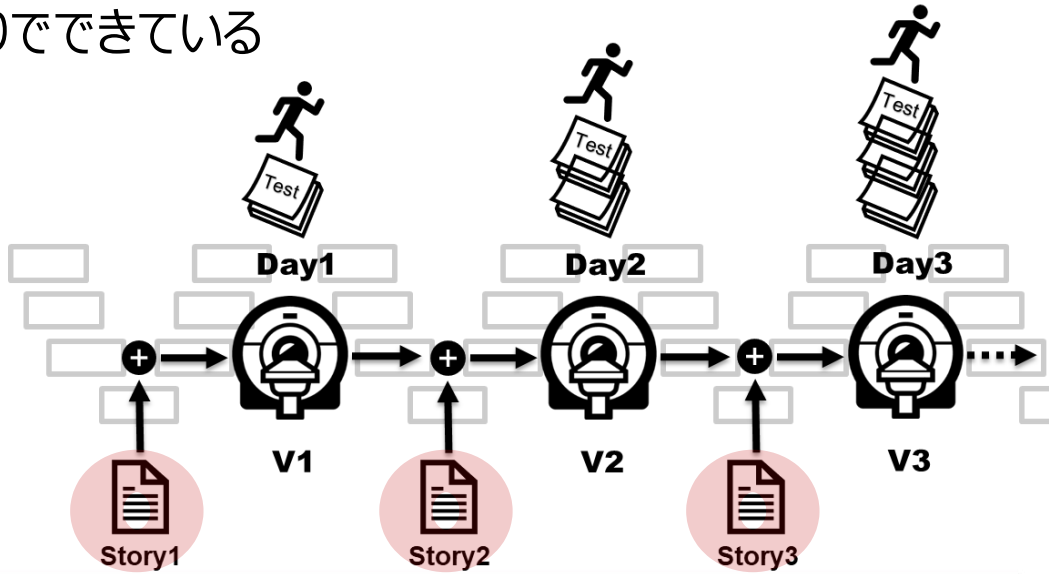
【図解】忍者式テスト



忍者式テスト

- ストーリーはXPなどのアジャイル開発で製品を変更する単位
  - 受け入れテストで確認する
  - ユーザーにとってシステムがどのように変化するか

- 製品はたくさんのストーリーの集まりでできている



忍者式テスト

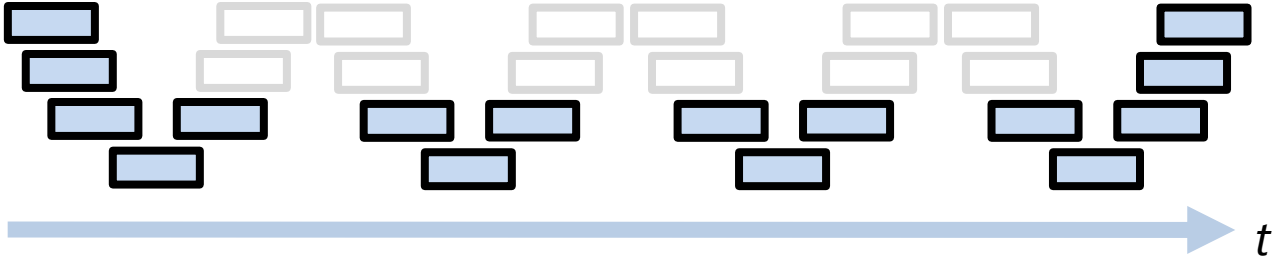
- ストーリーはチケットで管理している
  - 概要、要求の背景
  - テストケース
    - システムの具体的な変化と確認方法
  - 開発日記
    - 毎日の試行錯誤の様子、実験した内容や結果 など
    - 設計や実装の根拠がわかる
- ストーリーは数日で完成する大きさ
  - 大きな要求を適切な大きさのストーリーに変換するのは技術と訓練が必要
- 開発中に見つけたバグもストーリーと同様に扱う

今日の話

- 大規模ソフトウェア開発の難しさ
- 忍者式テスト
- 反復開発をうまくやるには
  - ストーリーを考えるときのヒント
  - バグの修正を後回しにしない
  - ペースの違いをどう考えるか
  - 規模と向き合う
- 忍者式テストの効果

アジャイル開発がうまくいかないケース

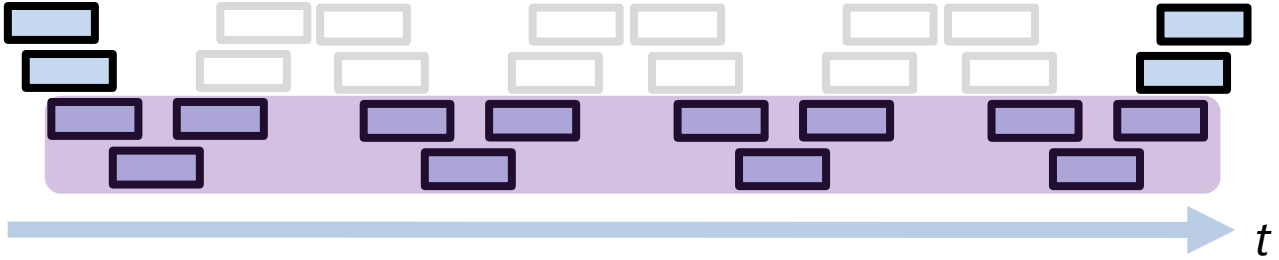
- 「スクラムはうまくできているのに、開発がうまくいかない」
  - V字の底あたり（ソフトウェア設計、実装・デバッグ）をずっとやっている
  - 複数イテレーションのあとに、テスト担当がまとめて受け入れテストする



アジャイル開発がうまくいかないケース

じわじわ開発

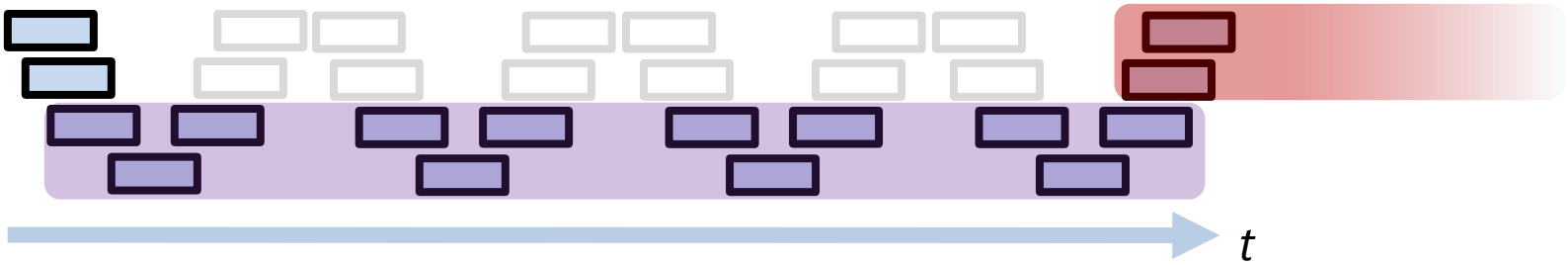
- 「スクラムはうまくできているのに、開発がうまくいかない」
  - V字の底あたり（ソフトウェア設計、実装・デバッグ）をずっとやっている
  - 複数イテレーションのあとに、テスト担当がまとめて受け入れテストする



アジャイル開発がうまくいかないケース

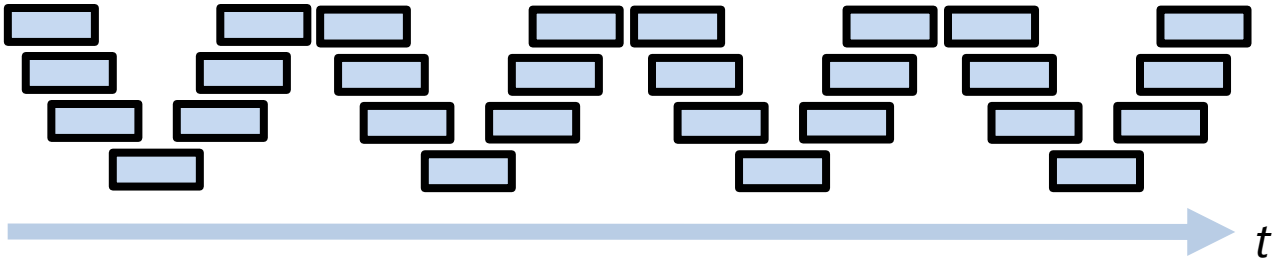
じわじわ開発

- 「スクラムはうまくできているのに、開発がうまくいかない」
  - V字の底あたり（ソフトウェア設計、実装・デバッグ）をずっとやっている
  - 複数イテレーションのあとに、テスト担当がまとめて受け入れテストする



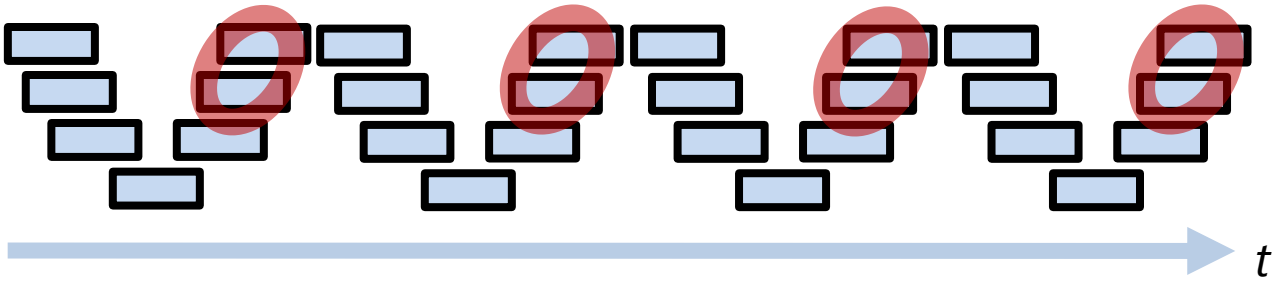
私たちのチーム

- 反復ごとにV字のすべてのステップを行う
- 全員が開発者
  - すべての工程を行うプログラマー
  - プログラミング以外のすべての工程を行うテスター



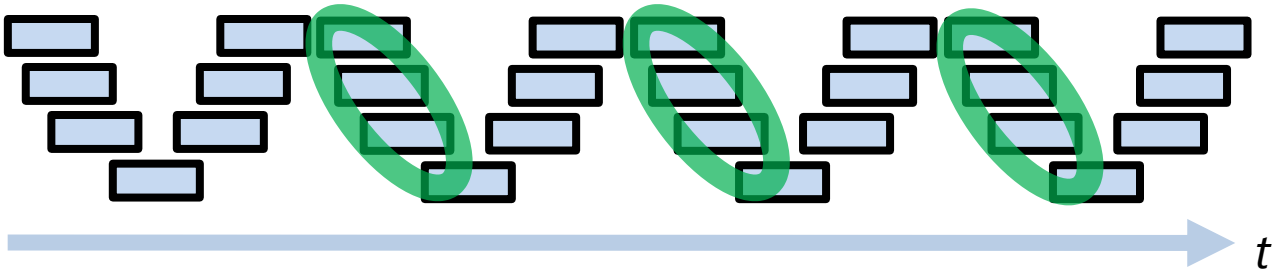
私たちのチーム

- よい製品になっているかどうかに興味がある
  - 短い周期でできばえを見たい → 反復ごとの受け入れテスト
  - 短い周期で製品の仕様や設計をチューニングしたい
  - ぎりぎりまで製品を磨きたい



私たちのチーム

- よい製品になっているかどうかに興味がある
  - 短い周期でできばえを見たい → 反復ごとの受け入れテスト
  - 短い周期で製品の仕様や設計をチューニングしたい
  - ぎりぎりまで製品を磨きたい

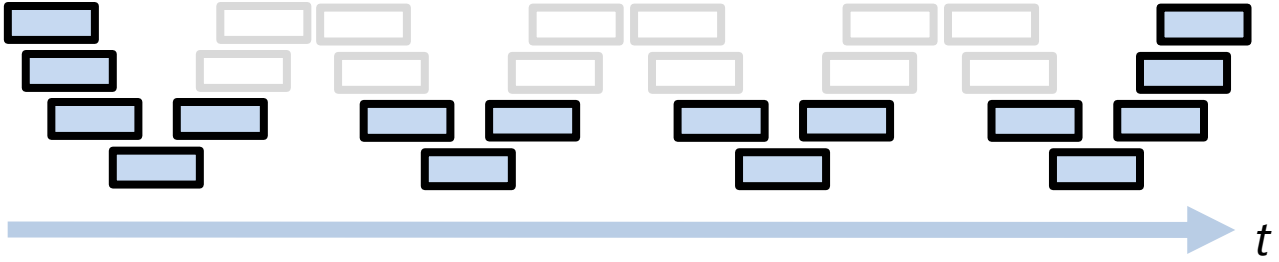


ストーリーを考えるときのヒント

じわじわ開発を誘う要因

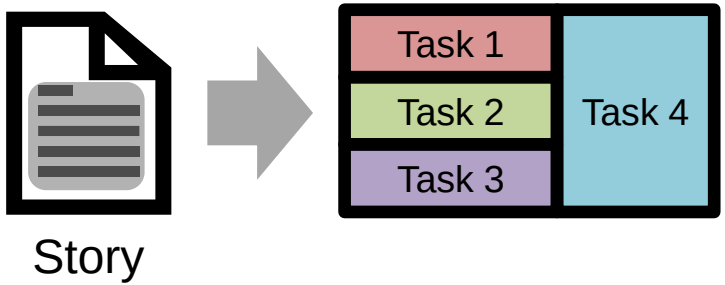
じわじわ開発

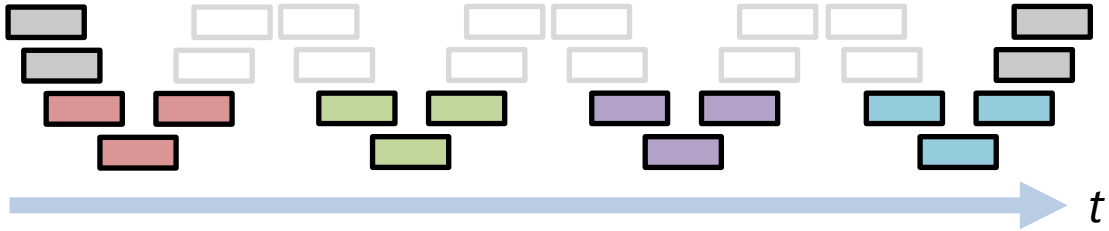
- 「スクラムはうまくできているのに、開発がうまくいかない」
  - V字の底あたり（ソフトウェア設計、実装・デバッグ）をずっとやっている
  - 複数イテレーションのあとに、テスト担当がまとめて受け入れテストする



ストーリーを入れ子のチケットにしている

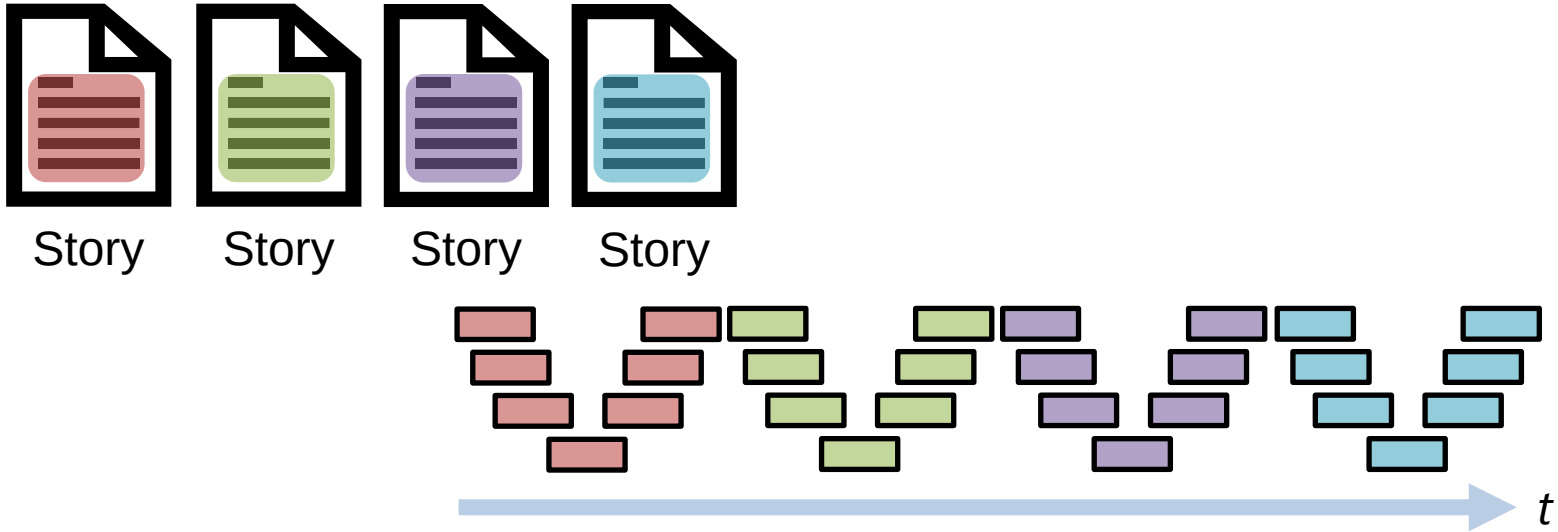
- ストーリーを複数のタスクで構成することが多いのでは？
  - すべてのタスクが揃わないとストーリーを試すことができない
- 結合を先延ばしにするスタイル





私たちのチーム

- ストーリーを入れ子にしない
  - ストーリーを複数のタスクに分けるのではなく、ストーリー自体を小さくする
  - 小さな変更であってもシステム全体でできばえを確かめる
- 絶えず結合するスタイル

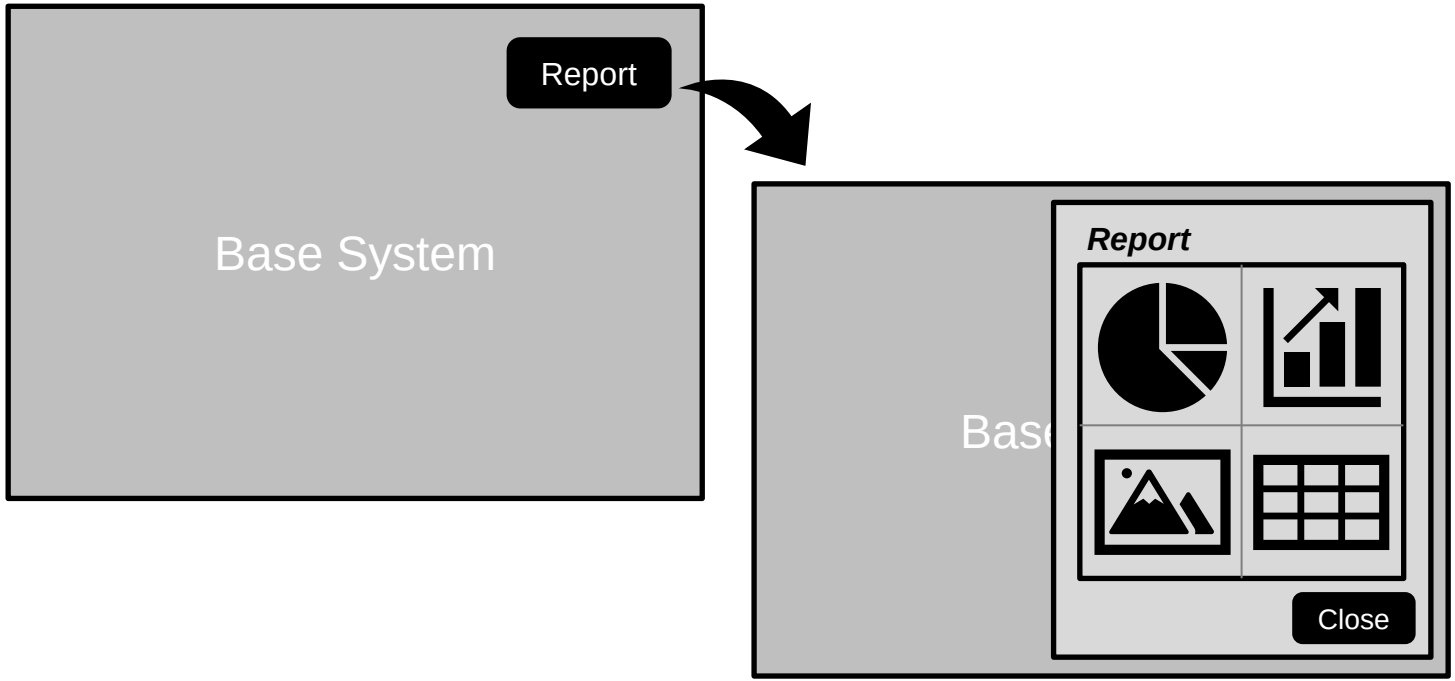


私たちのチーム

- ストーリーを入れ子にしない
  - ストーリーを複数のタスクに分けるのではなく、ストーリー自体を小さくする
  - 小さな変更であってもシステム全体でできばえを確かめる
- 絶えず結合するスタイル

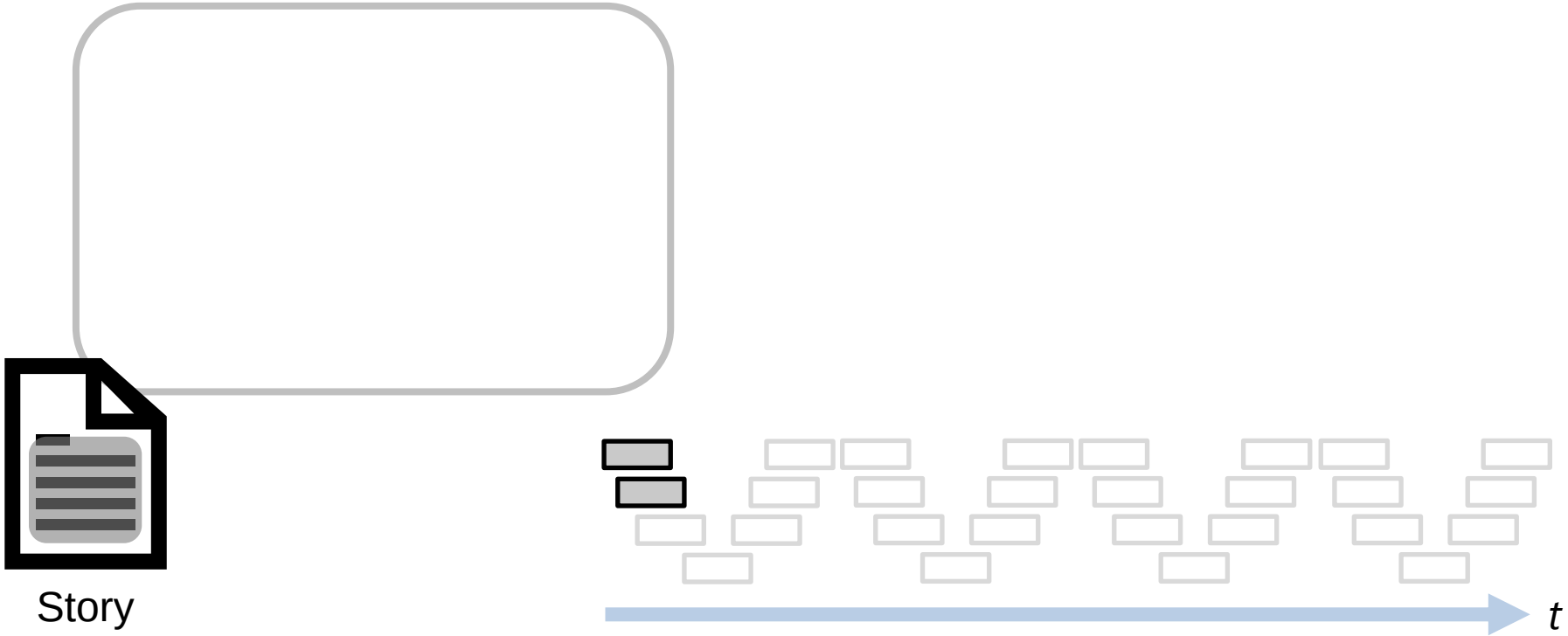
これができないと「じわじわ開発」に陥ってしまう

ケーススタディ：レポートを表示する



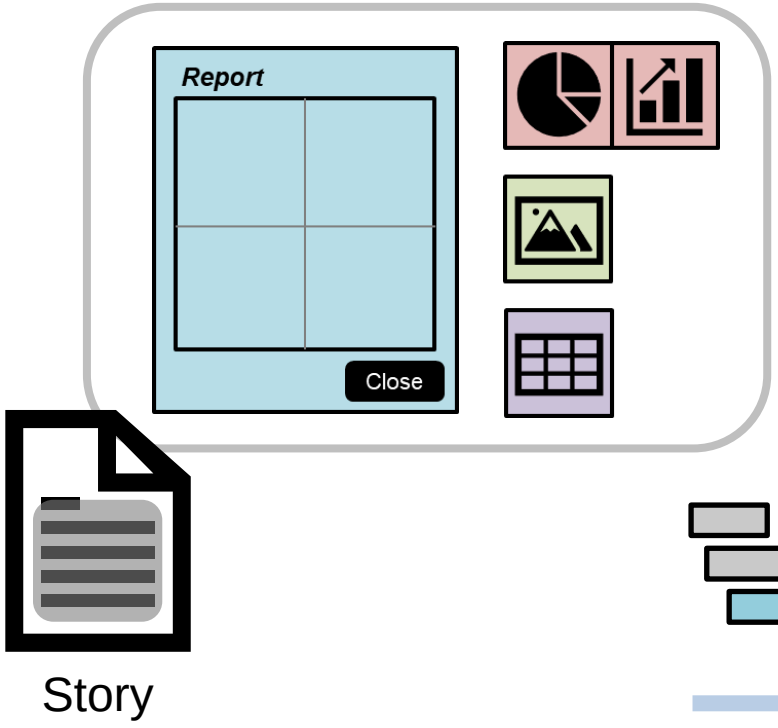
レポート機能を作ってからシステムに統合する作戦

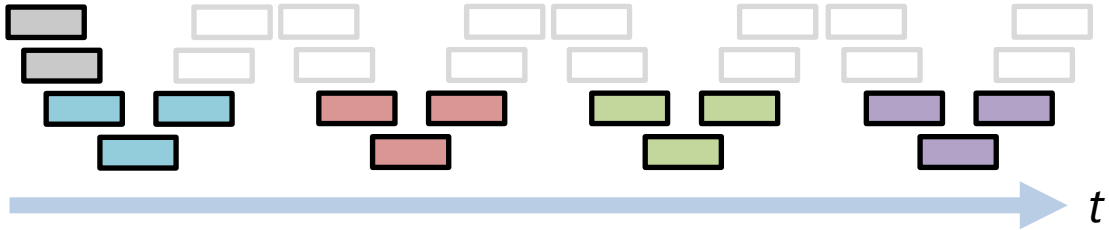
- ストーリーを複数のタスクで構成する



レポート機能を作ってからシステムに統合する作戦

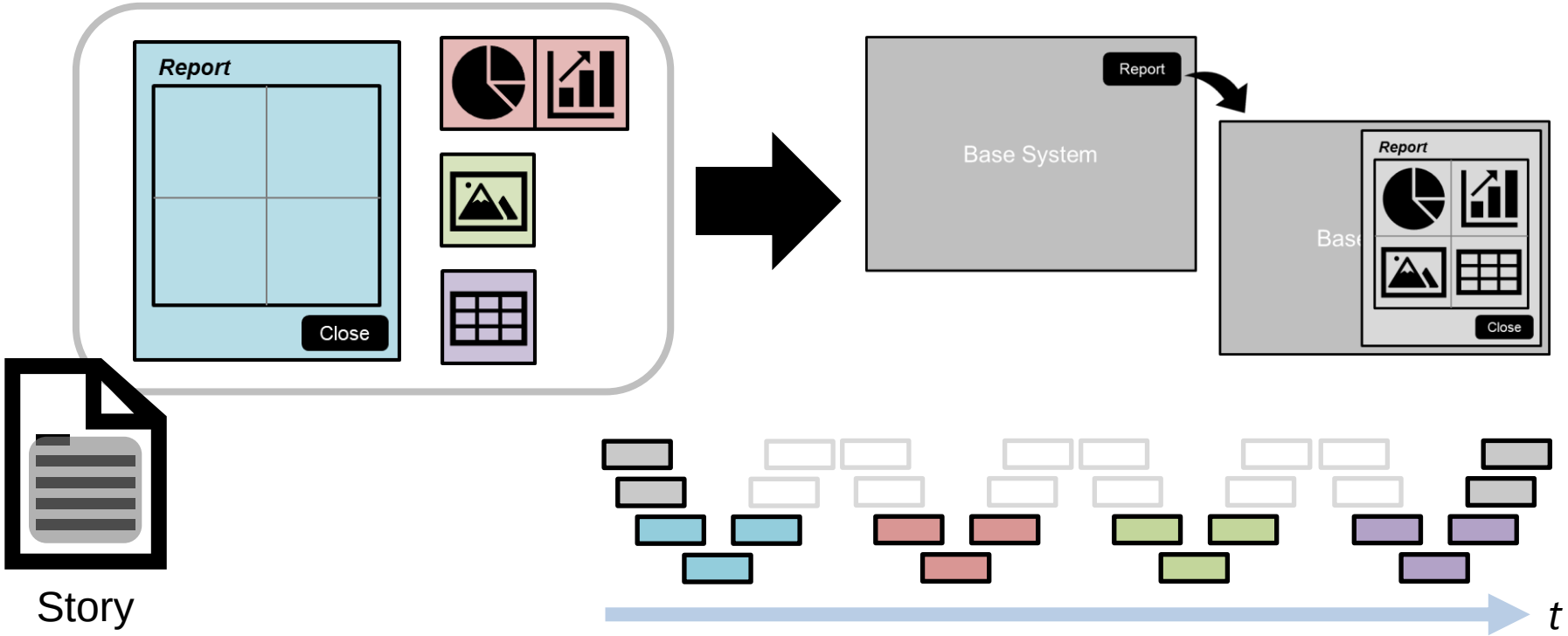
- 各タスクで部品を作る





レポート機能を作ってからシステムに統合する作戦

- 部品がすべて揃ったらシステムと結合してレポート機能を確認する

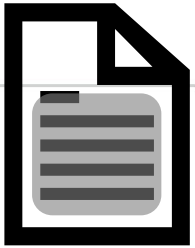


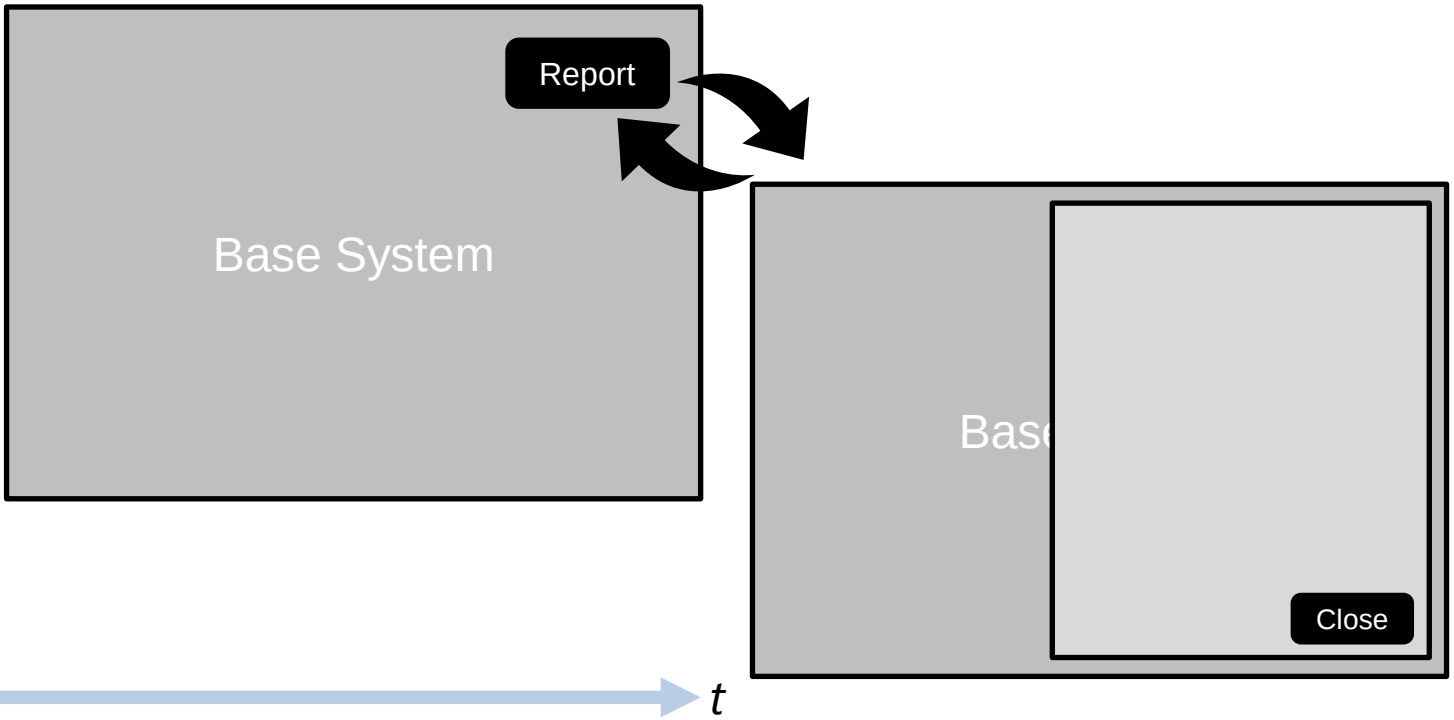
私たちのストーリーの作り方

- 「システムで試せる一番小さな変化はどこか」を考える

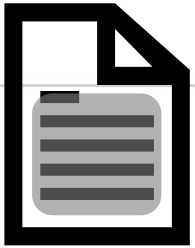
システムで試せる一番小さな変化はどこか

- 「既存のシステムにボタンをつけて中身のない画面を表示する」から開始





システムで試せる一番小さな変化はどこか



- 「既存のシステムにボタンをつけて中身のない画面を表示する」から開始

Report

- ✓ システム側が機能を拡張できるようになっているか
- ✓ どんなときに起動できるのか
- ✓ 終了したらどんな画面になっているべきか
- ✓ 起動に関わる設定ファイル(システム全体、機能別)
- ✓ 起動方法をどうするのか(メニュー、ボタン)
- ✓ 二重起動を許す(制御方法)／許さない
- ✓ ウィンドウのZオーダー、モーダル／モードレス
- ✓ 画面(ウィンドウ／ダイアログ)が動く／動かない
- ✓ スクリーンセーバーなどが動いたときの挙動
- ✓ キーボードショートカットの割り当て
- ✓ 他の機能との並行動作
- ✓ 機能の起動中にできることはなにか
- ✓ 起動中にシステムを終了したらどうなるのか
- ✓ システムを再起動したときどうなってほしいか

Close

t

小さな変化に価値があるのか

- 「既存のシステムにボタンをつけて中身のない画面を表示する」
  - 中身のない画面を表示してもシステムが壊れないことは価値である
  - 最初からシステムに統合した状態で開発ができるのは価値である
  - 懸念点を実際にシステムで確かめられたことは価値である

ストーリーを作るときのヒント

- ユーザーの視点で書かれたストーリーをそのまま扱わなくてもよい
  - ストーリーをシステムや実装や製品のドメインの視点で解釈しなおす
  - ユーザーに見えているものだけで考えると開発の実体に合ってないことがある

- どんなに小さなストーリーでも守ること
  - システム全体で試せるようにストーリーを作る
  - ストーリーのテーマの中での完璧を目指す
  - ストーリーを搭載してもシステムを破綻させない

システムで試せる一番小さな変化はどこか

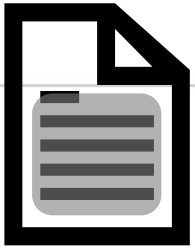
こんなシグナルがあがったら

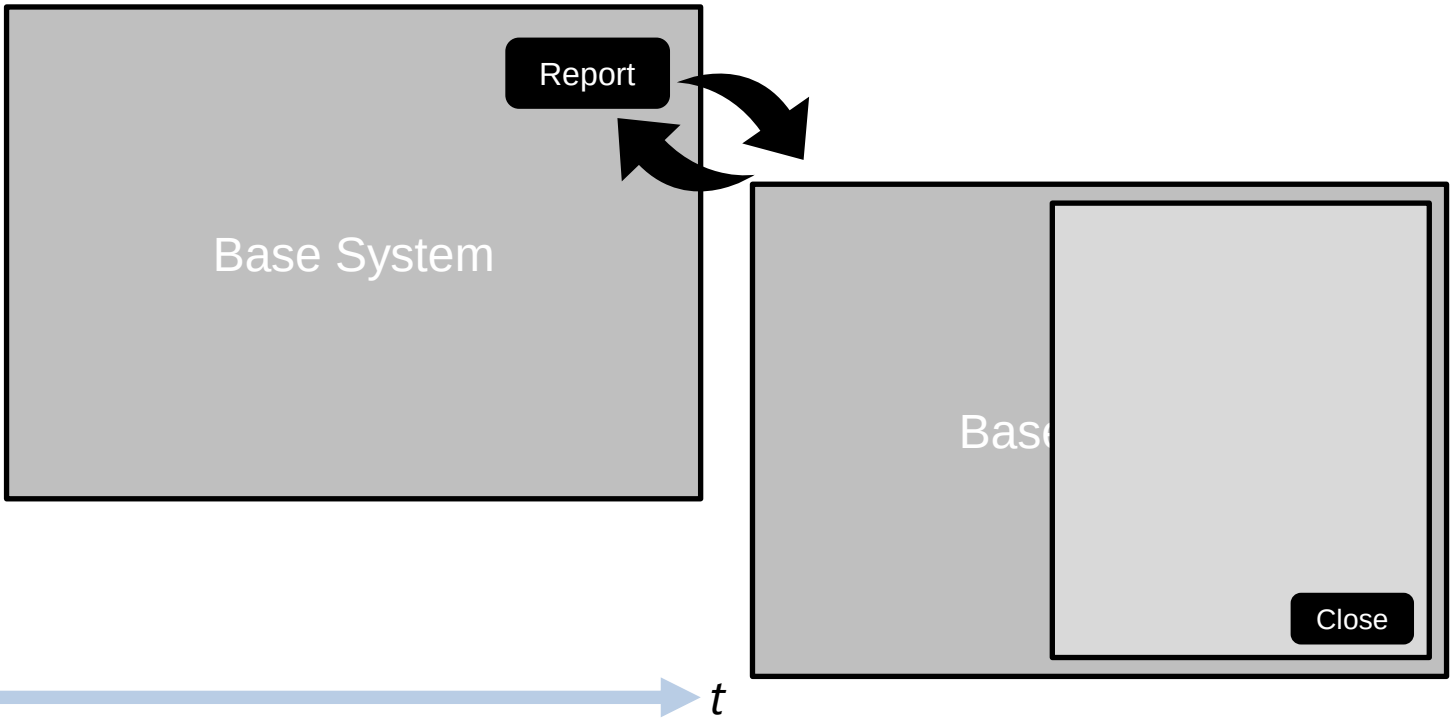
- 試し方が思いつかない
- ストーリーがなかなか完成しない
- 制約が多すぎて本来のストーリーのテーマが試せない

- ストーリーの分割や順序が適切でないかもしれない
- 計画の見直しやストーリーを再構築してみよう

システムで試せる一番小さな変化はどこか

- 既存のシステムにボタンをつけて中身のない画面を表示する

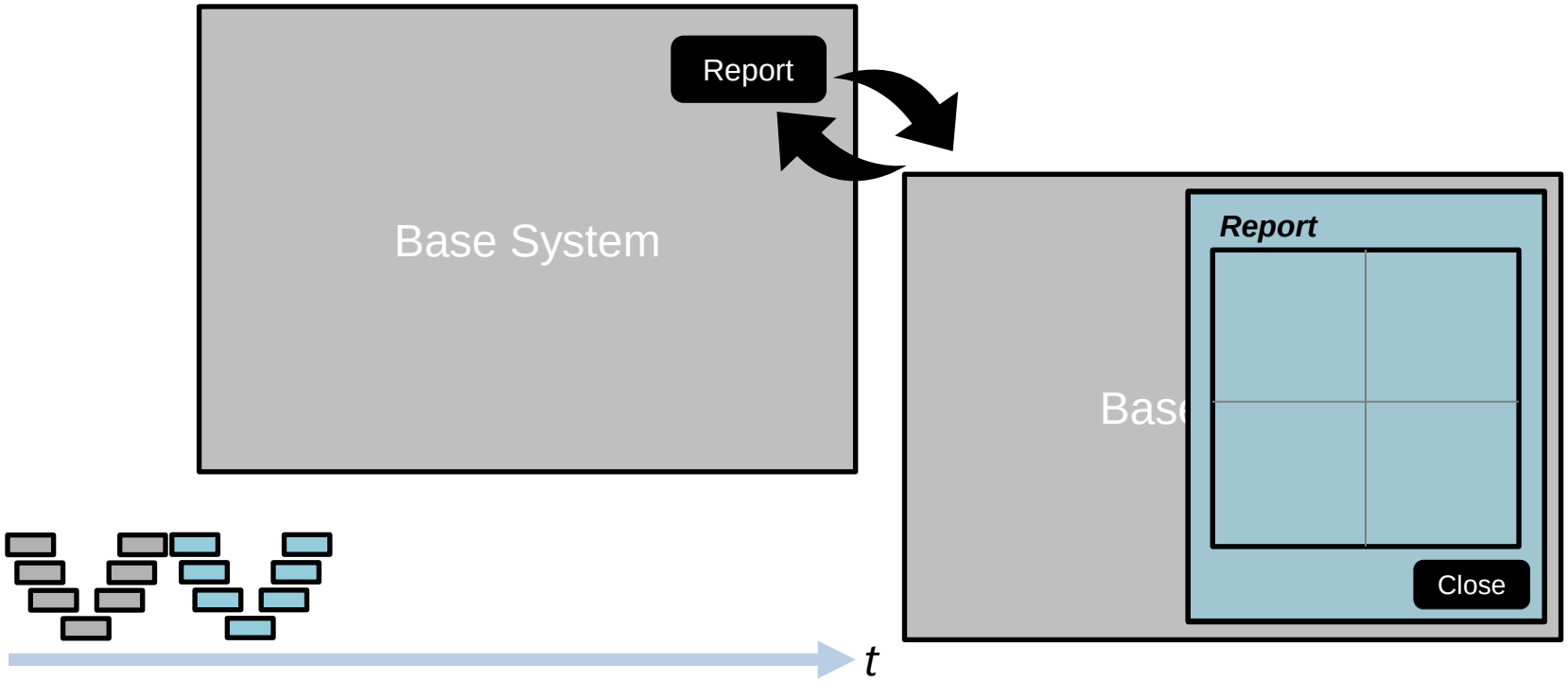




システムで試せる一番小さな変化はどこか

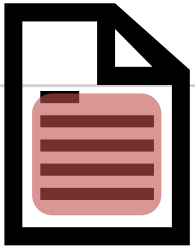
- レイアウトを表示する

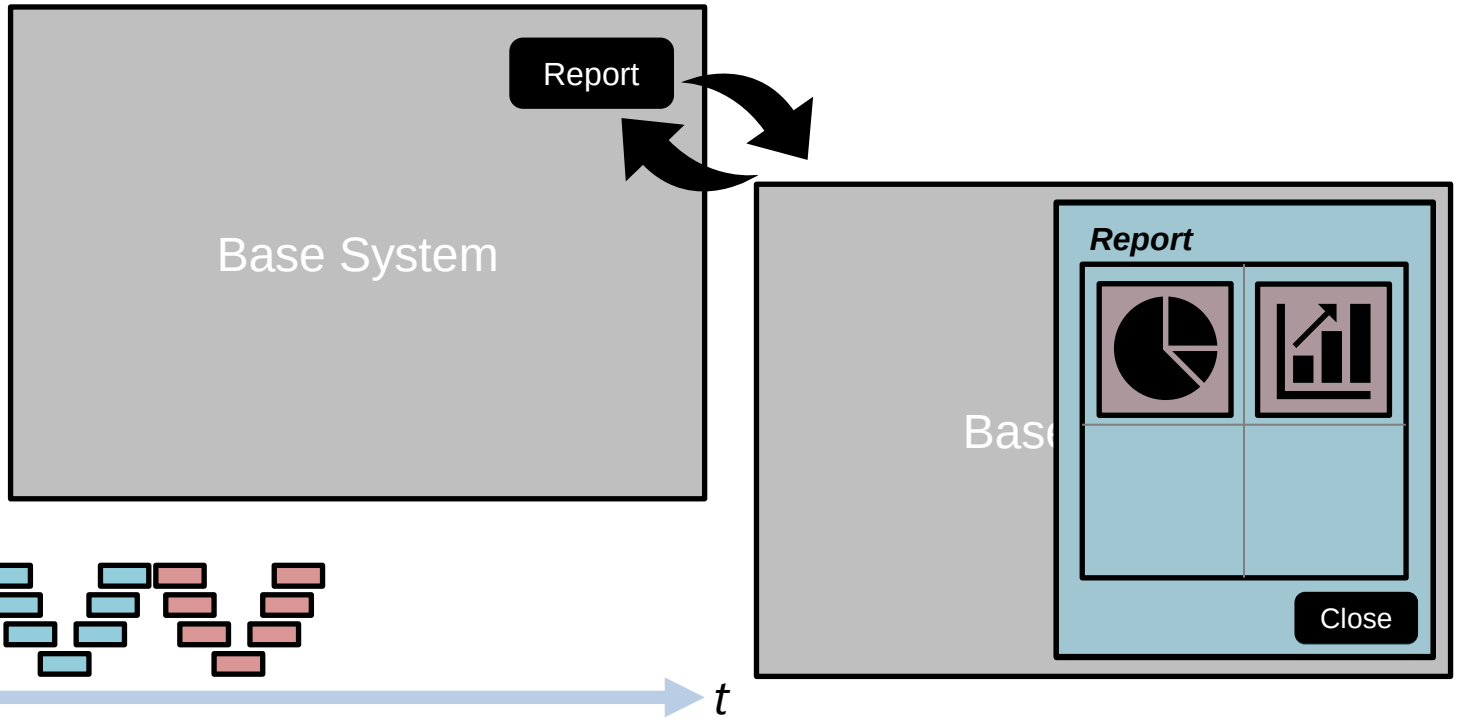
Story



システムで試せる一番小さな変化はどこか

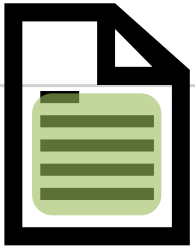
- グラフを表示する

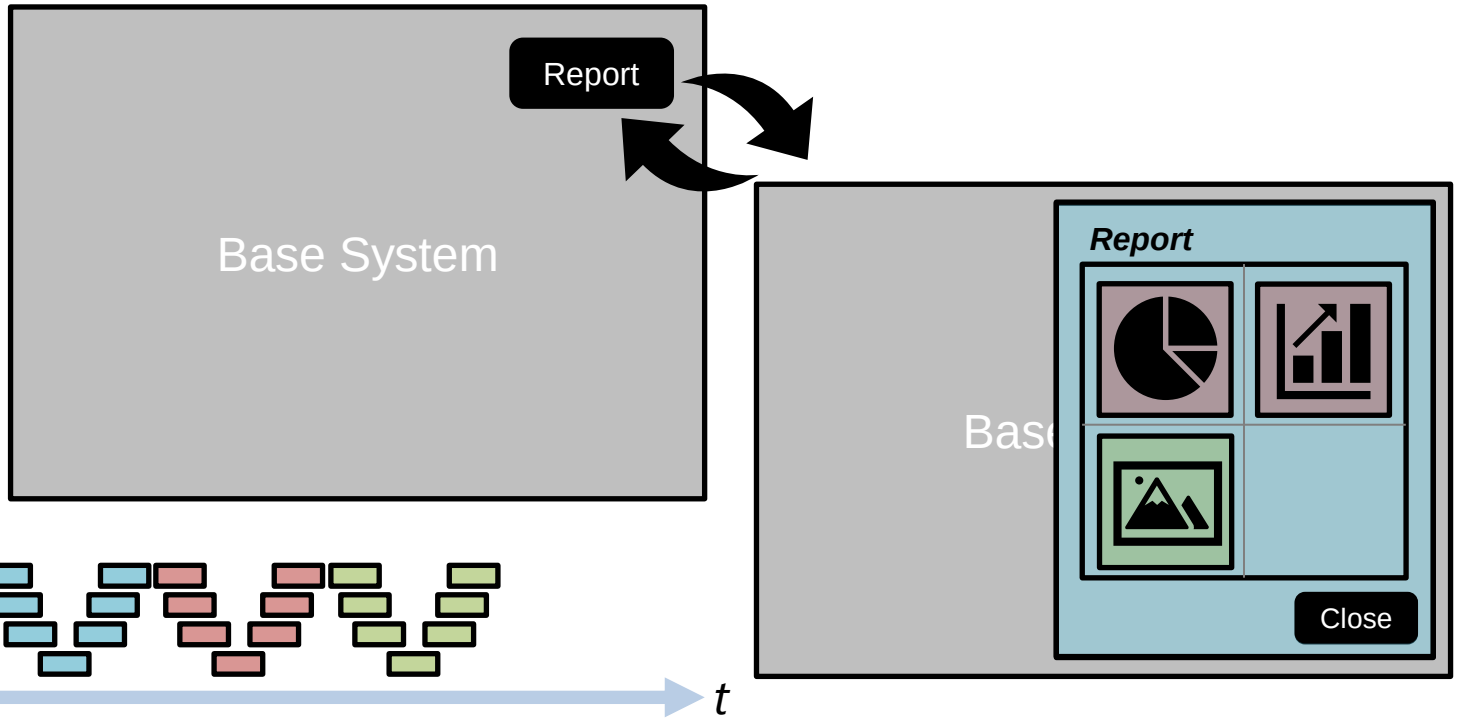




システムで試せる一番小さな変化はどこか

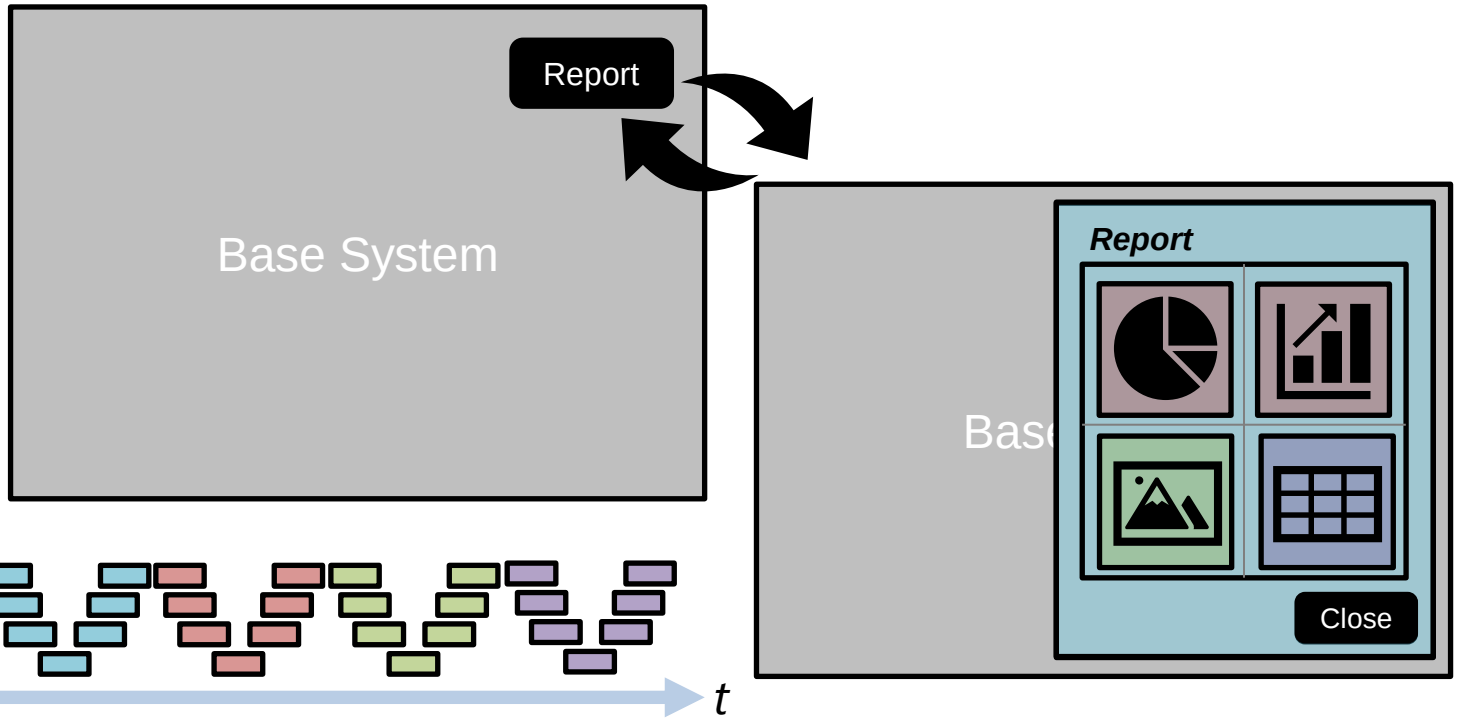
- 画像を表示する





システムで試せる一番小さな変化はどこか

- 計測値を表示する



システムと結合する難しさを後回しにしない

- 小さな変更でもシステムで結合して試す
- 結合は混乱を伴うが、すこしずつ結合することで混乱を小さくできる
- 難しくてもやる（難しいからやる）

今日の話

- 大規模ソフトウェア開発の難しさ
- 忍者式テスト
- 反復開発をうまくやるには
  - ストーリーを考えるときのヒント
  - バグの修正を後回しにしない
  - ペースの違いをどう考えるか
  - 規模と向き合う
- 忍者式テストの効果

バグの修正を後回しにしない

- バグを見つけたら原因がわかるまではストーリーの開発を止める
  - 見かけの現象だけで深刻さを判断しない
  - バグは問題の一部分が見えただけに過ぎないので本当の問題を探す

- 後回しにするとバグがある状態で開発することになり、速度が落ちる
  - バグを避けて開発／テストしなければならない
  - バグがどんどん増える状態になる

バグの修正を後回しにしない

- 調速装置
  - チームが一度に作れる量には限界があり、限界を超えると問題が増える
  - 問題解決を優先し、ストーリーの開発量を減らすと、徐々に問題の量が減っていき、再び、ストーリーの開発量を増やせる
    - 適切な開発速度に自然に調整される

- 心配容量は一定
  - いまある問題を解決しなければ、もっと高度な問題を見つけられない
  - チームで扱える心配の量は一定

今日の話

- 大規模ソフトウェア開発の難しさ
- 忍者式テスト
- 反復開発をうまくやるには
  - ストーリーを考えるときのヒント
  - バグの修正を後回しにしない
  - ペースの違いをどう考えるか
  - 規模と向き合う
- 忍者式テストの効果

ペースの違いをどう考えるか

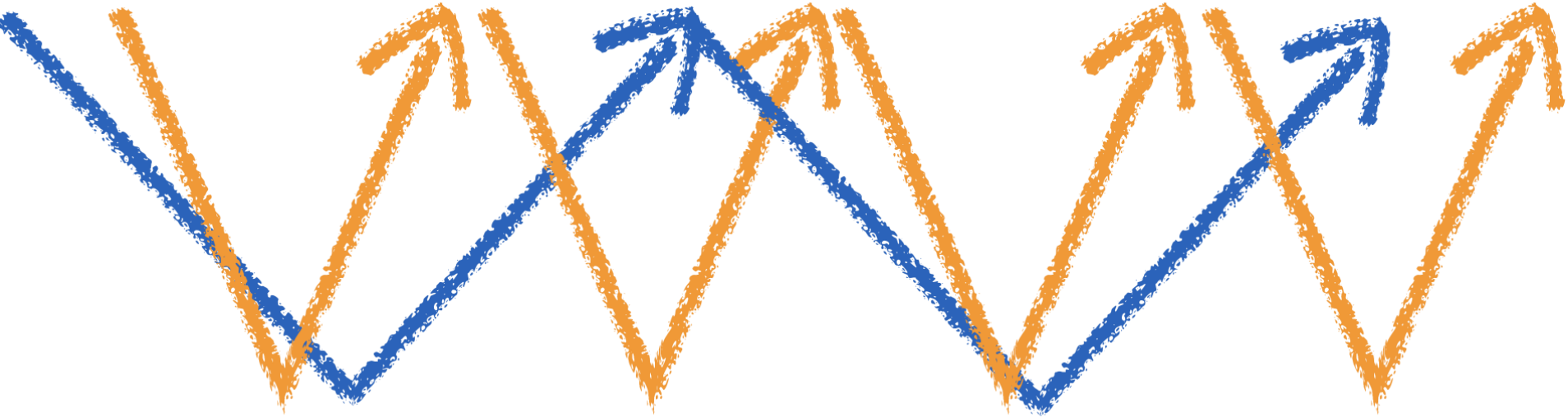
- 大きなシステムは全員同じペースではないかもしれない
- ペースの違いから衝突が起こりがち
  - サブシステム別チーム
  - ソフトウェアとハードウェア
  - 自分のチームととなりのチーム

人数が多いと全然あわない

- 私たちのチーム
  - 人数が多く、1イテレーション中に作るストーリーも多い場合、つまり、十分複雑な開発をしている場合、ストーリーの完了が揃わない
  - 無理にイテレーションの切れ目に合わせると待ちばかりになってしまう。ほどほどでいい

実はペースを混ぜても大丈夫

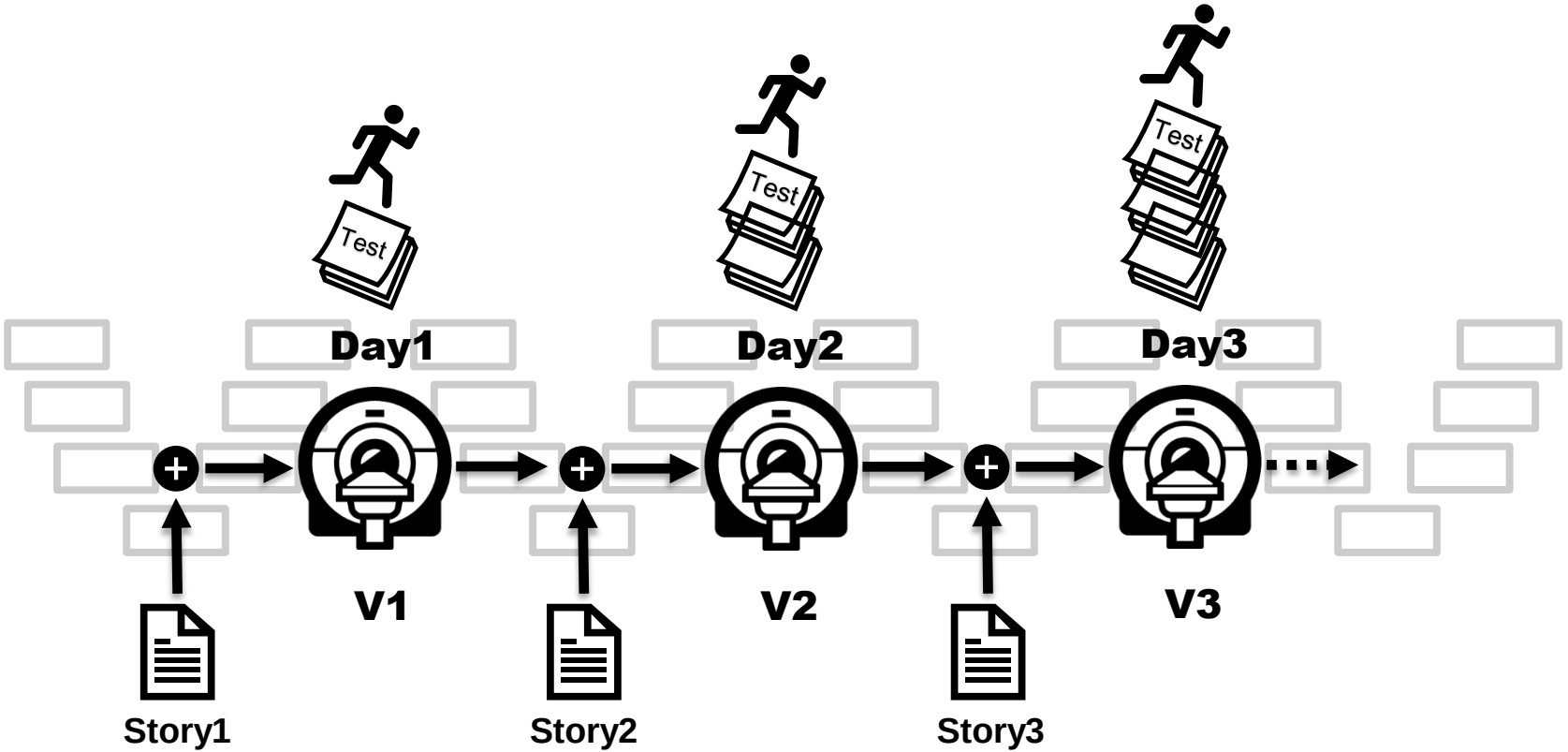
- 結合した状態で確認するという原則を守っていれば、ノイズ程度
- まだ結合してない部分は「何か問題があるかも」と認識しておく
- チーム間でも反復がきれいに揃うことは重要でない



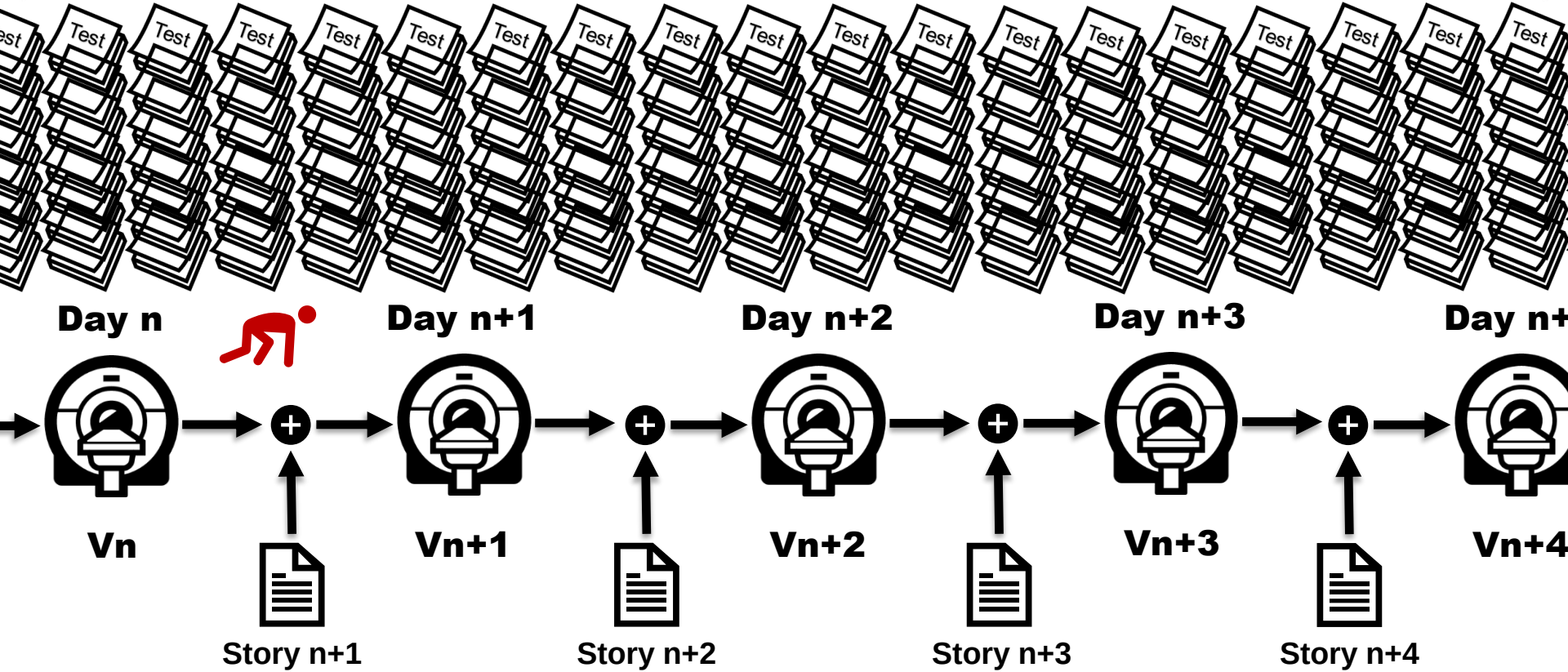
今日の話

- 大規模ソフトウェア開発の難しさ
- 忍者式テスト
- 反復開発をうまくやるには
  - ストーリーを考えるときのヒント
  - バグの修正を後回しにしない
  - ペースの違いをどう考えるか
  - 規模と向き合う
- 忍者式テストの効果

【図解】忍者式テスト



たいへんなことになります



規模と向き合う

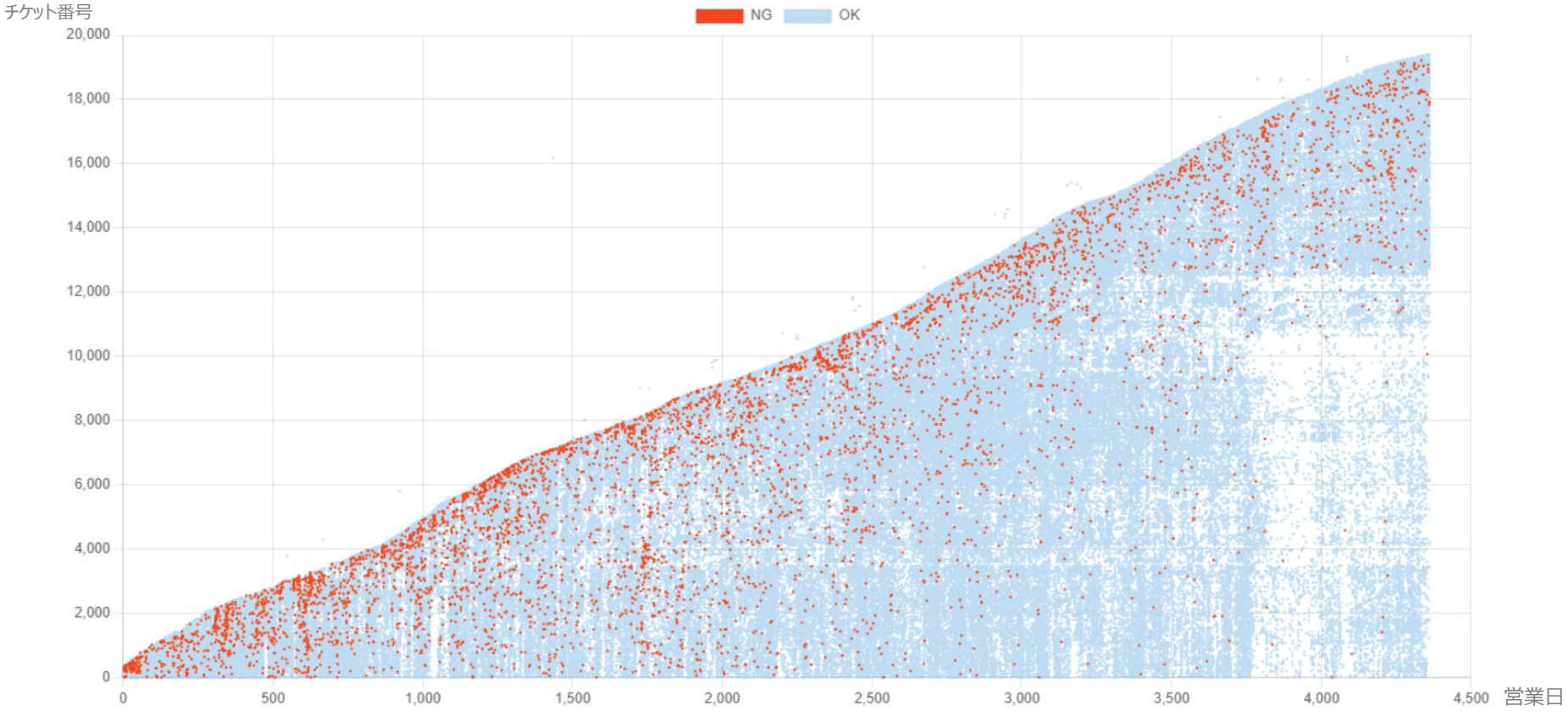
- 直近の変更は早く確かめたい／すべてのテストケースを確かめたい

- 一日ではなく、ある期間でテストケースをすべて回す作戦に切り替えた

- まんべんなく、かつ、効果の高いテストケースを抽出するアルゴリズムの開発
  - 新しいストーリー、修正したばかりのチケットのテストケースを最優先
  - 前回のテスト結果がパスしたテストケースの出現頻度を徐々に落とす
  - 開発の状況に応じて機能ごとに出現頻度を調整
  - ある期間で見ると、すべてのテストケースが実行できる

- アルゴリズムにより抽出した今日のテストスイート →「本日のおすすめテスト」
  - 一日にできそうな量のテストケースしか表示しない（量に圧倒されないようにする）

テスト実施の記録（20年分）



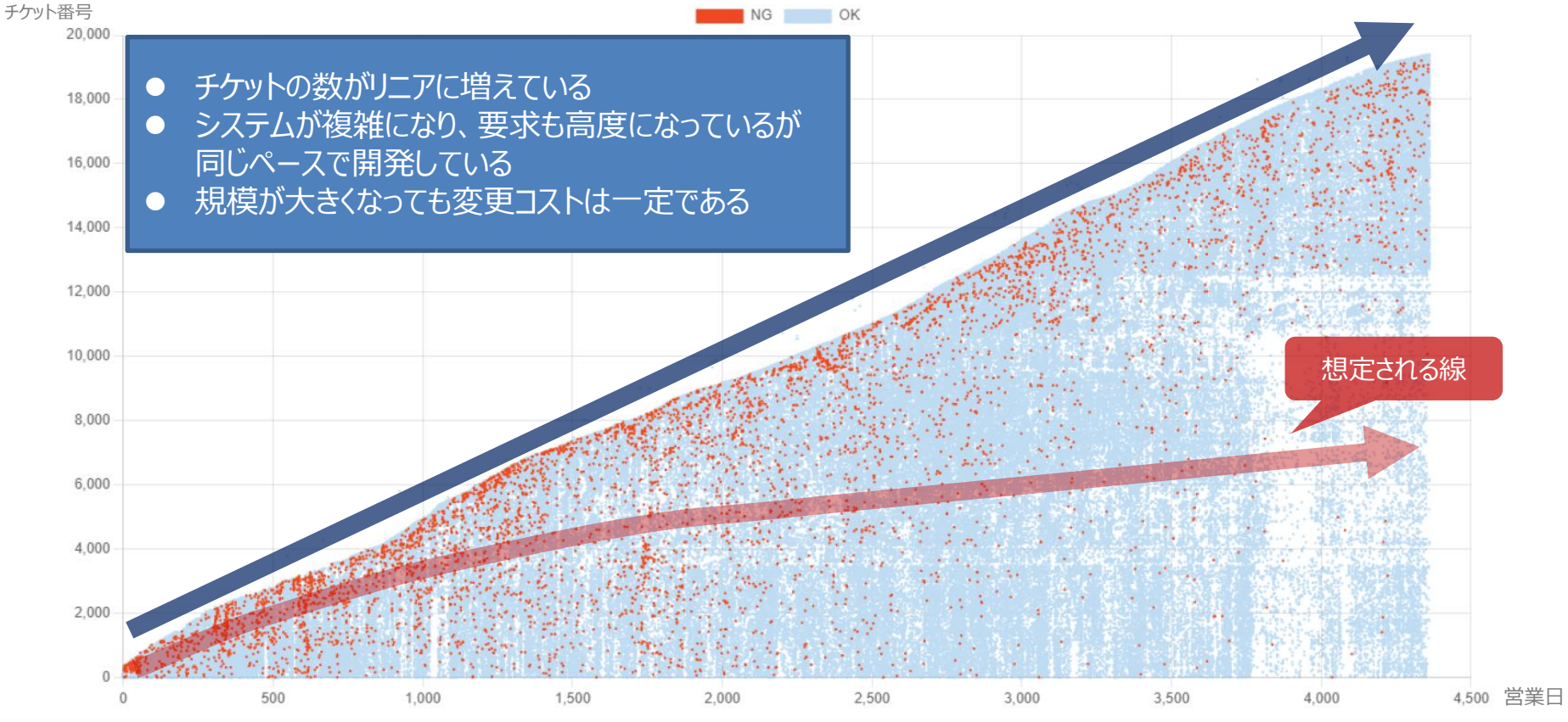
NGの点はよく見えるように強調しています

今日の話

- 大規模ソフトウェア開発の難しさ
- 忍者式テスト
- 反復開発をうまくやるには
  - ストーリーを考えるときのヒント
  - バグの修正を後回しにしない
  - ペースの違いをどう考えるか
  - 規模と向き合う

- 忍者式テストの効果

開発の難しさに釣り合うように、チームの開発能力も向上



忍者式テストの効果

- 難しそうな問題も躊躇せずに修正できるようになった
- 不具合や性能などの実装レベルの問題に積極的に対応できるようになった
- 直せる幅が広がった
- 理想の製品をイメージして、それとの差分も積極的に探しだす者が現れた
- 全員がずっと「良い製品とは何か」を問いながら開発するようになった
- あとからチームに参加した開発者にも同様の変化が起きた

- この効果が開発能力の向上に寄与している

今日の話

- 大規模ソフトウェア開発の難しさ
- 忍者式テスト
- 反復開発をうまくやるには
  - ストーリーを考えるときのヒント
  - バグの修正を後回しにしない
  - ペースの違いをどう考えるか
  - 規模と向き合う
- 忍者式テストの効果

ヒントのまとめ

- 結合して試すのを後回しにしない
- 後回しを誘う入れ子を避ける
- 「システムで試せる一番小さな変化はどこか」を探す
- すぐに結合して試せるよう小さなストーリーに解釈しなおす
- バグの修正を最優先にしても開発のペースは遅くならない
- 大規模で複雑になると完成のペースが揃わなくなるが無理に同期しなくてもよい

Made For life

患者さんのために、あなたのために、そして、ともに歩むために。

人々の健やかな生活の実現のために、「いのち」と向き合う。

「Made for Life」はキヤノンメディカルシステムズの経営理念を象徴するスローガンです。

Canon

キヤノンメディカルシステムズ株式会社