# 「龍が如く」の高速デバッグ術

## ～そびえ立つバグの山を踏破するための弾丸ワークフロー～

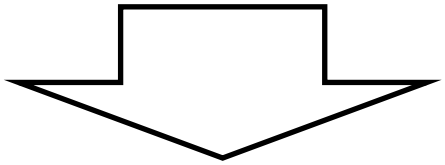
株式会社セガゲームス
第一CS研究開発部
アドバンスト・テクノロジー開発チーム
阪上　直樹

# 自己紹介

- 関わったゲームシリーズ
  - 「プロサッカークラブをつくろう！」
  - 「プロ野球チームをつくろう！」
  - 「龍が如く」
- ゲームプログラマから自動化や効率化の仕事に移行
  - QAプログラマ？
  - QAエンジニア？
  - ビルド職人？
- アドバンスト・テクノロジー開発チーム
  - 龍が如くエンジンのベース部分や各種ライブラリ・ツール等の開発

# 本セッションについて

- KYUSHU CEDEC 2015 での同名セッション
  - 概要
    - http://kyushucedec.jp/session.html#06
  - 発表資料
    - https://cedil.cesa.or.jp/cedil_sessions/view/1398



今回は完全版です！

# 本セッションのゲームタイトル


**龍が如く 維新！**
PS3/PS4
PS Vita（無料アプリ）
2014年2月22日発売


**龍が如くO　誓いの場所**
PS3/PS4
PS Vita（無料アプリ）
2015年3月12日発売


**龍が如く 極**
PS3/PS4
2016年1月21日発売

**複数プラットホーム×毎年リリース**

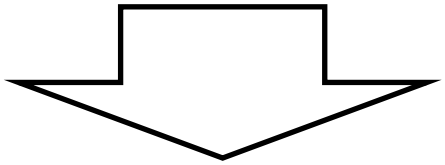
# 「龍が如く」を短いスパンで
# リリースするための秘密は？

ないです。がんばってるだけです。

でも…

# がんばる場所を限定

- ゲームの面白さ
- クオリティ
- ユーザーに感動体験を与えるための仕掛け

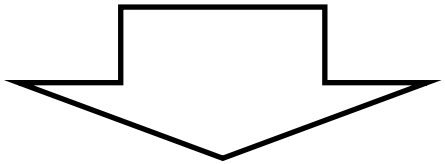


他にもやることはたくさんあるよ！
理想論じゃないの？！

# がんばらないために、がんばる

- 自動化できるものは、ためらわず自動化
- 作業の効率化を徹底して追求
- ワークフローの見直し

誰でもできる作業をクリエイターにさせない



本セッションのテーマ

# 本セッションの用語定義

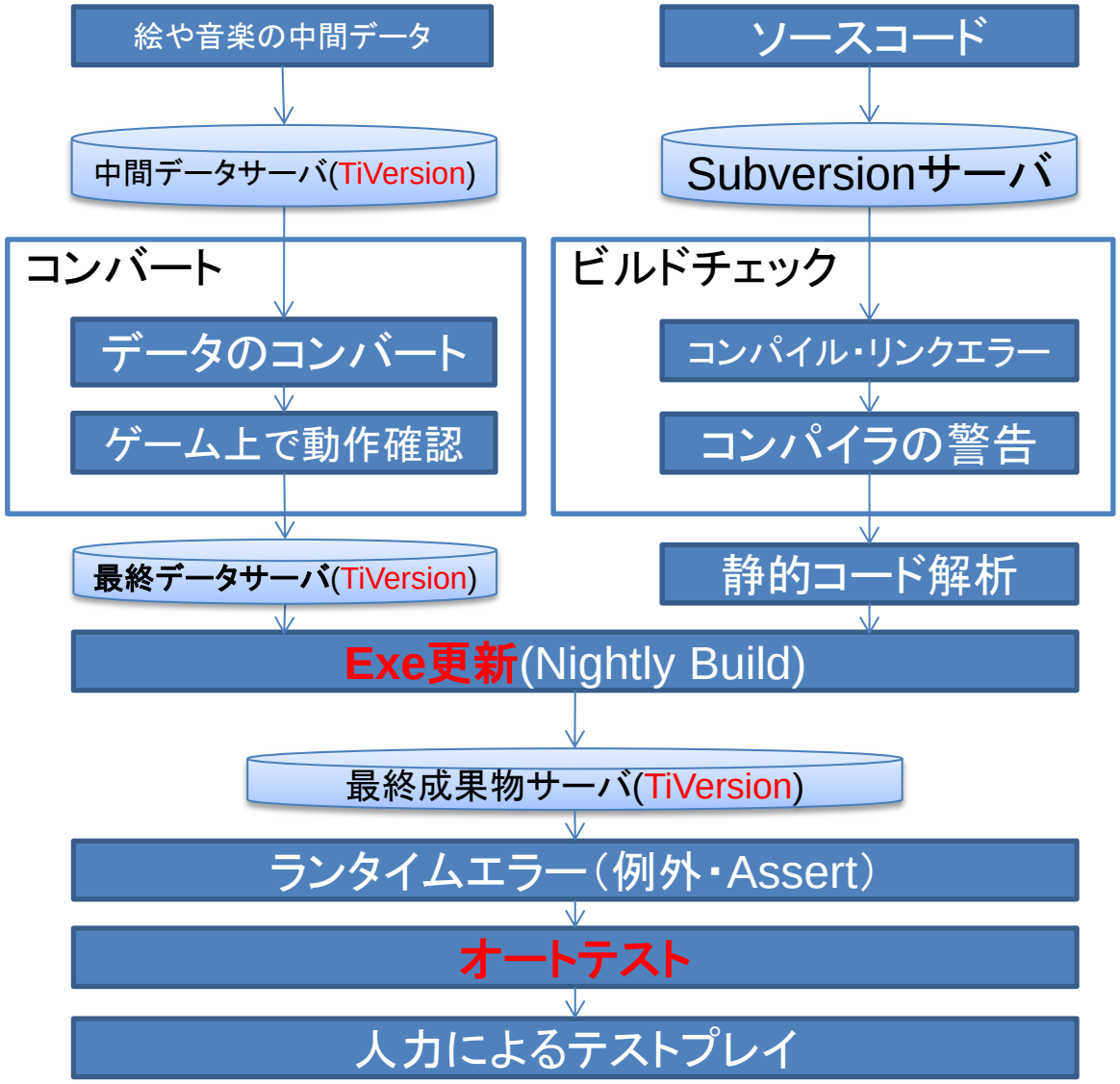
- デバッグ
  - バグを修正する
  - テストプレイを含まない狭義の意味
- テストプレイ
  - プレイチェック
  - バグ報告
  - ゲームバランス(難易度/面白さ)の指摘
- 赤字の用語
  - 龍が如くチーム用語

# 本セッションの概要

- 前半
  - 高速デバッグ術
    - 自動化
    - Jenkins
- 後半
  - そびえ立つバグの山を踏破するための弾丸ワークフロー
    - バグ管理
    - Redmine

# 高速デバッグ術

- 自動化
  - Jenkinsを使用
  - パイプライン
    - ビルド
    - コンバート
  - エラー検出
    - ビルドエラー検出
    - 静的コード解析
    - 例外検出
    - テスト



# Jenkinsによる自動化

- Jenkinsとは
  - CI(Continuous Integration:継続的インテグレーション)ツールの一つ
  - 日々の開発に必要なビルドやコンバートを自動化し、安定した開発環境を継続していくためのもの
- なぜ必要？
  - ゲームの動作が安定していない状態では、実装したものをすぐに確認できないので作業が滞る
  - ビルドが壊れた状態で放置すると、修正コストが増大し、他のバグを生む原因にもなる

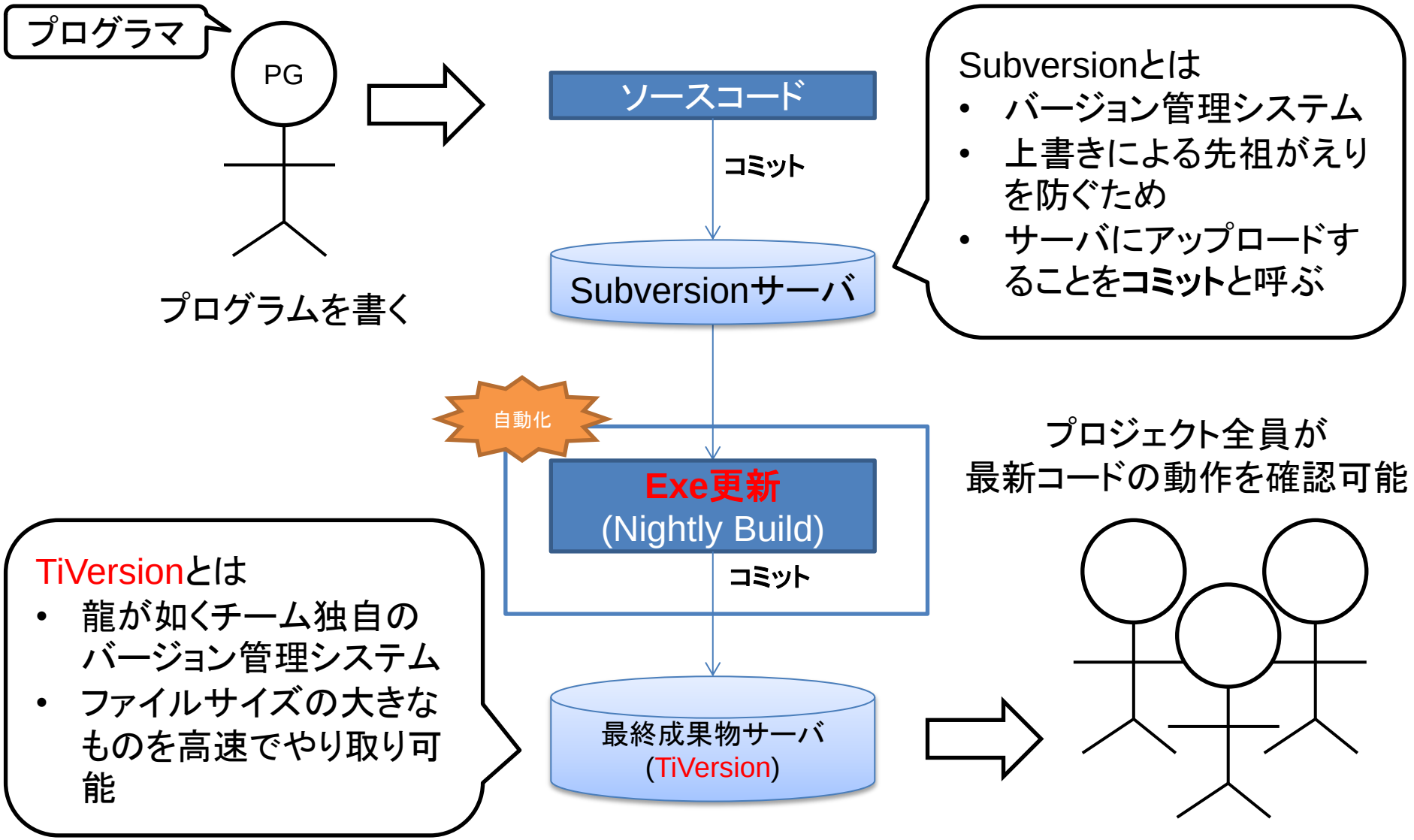
# パイプラインの自動化

- ビルド
- コンバート

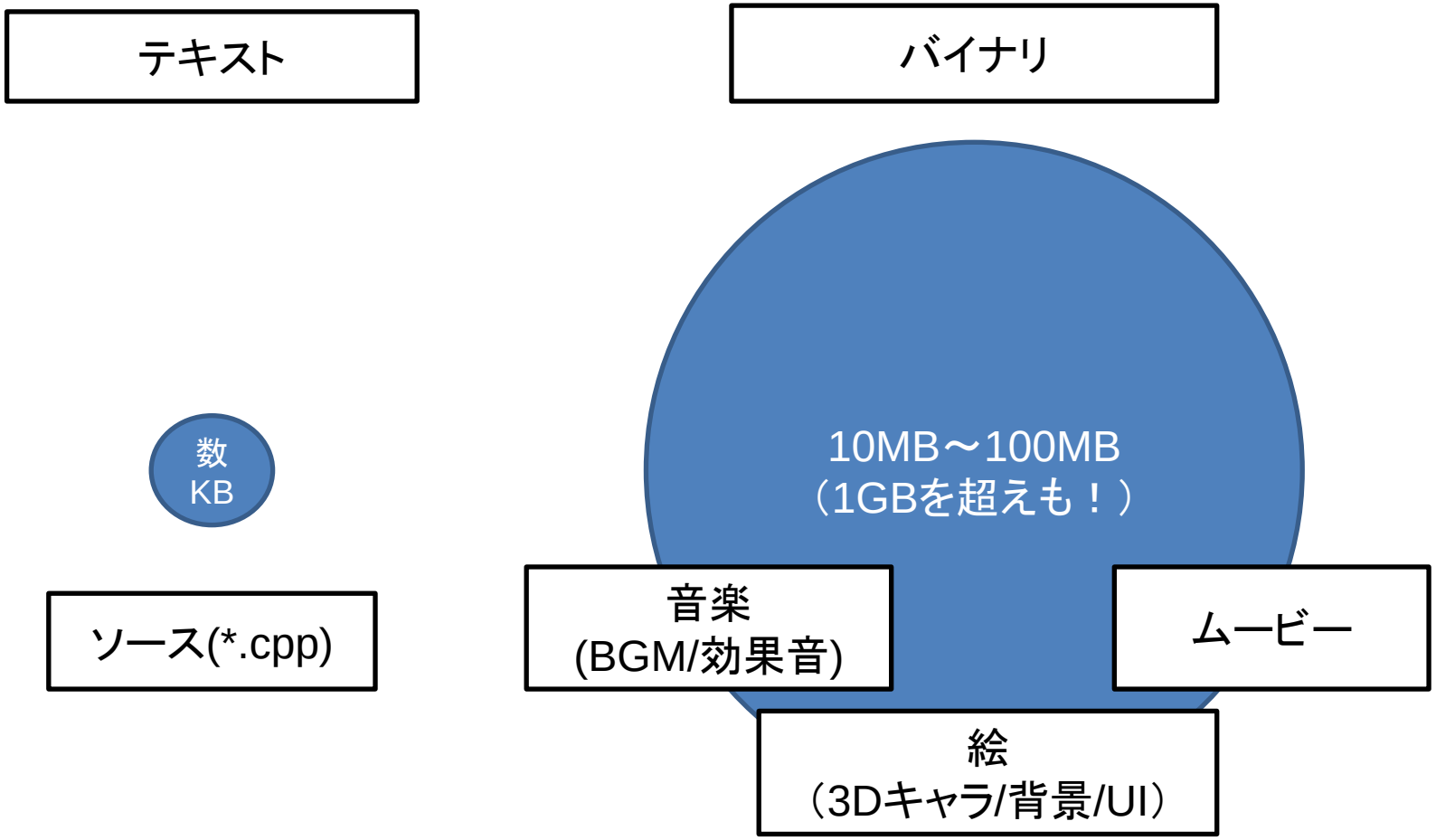
# ビルドの自動化 (1/2)

- ビルドとは
  - プログラムをコンパイル・リンクして実行可能な形式に変換すること
  - 自動で毎日ビルドした成果物は、Nightly Buildと呼ばれる
- Exe更新とは
  - Nightly Buildの龍が如くチーム特有の呼称
  - PS3/PS4/PS Vitaに加えて、それぞれのターゲットに対応したデバッグ用のWindows版の実行ファイルを定期的に生成
  - Windows版もあることから、「Exe更新」と呼んでいる

# ビルドの自動化 (2/2)


プログラマ
PG
プログラムを書く
ソースコード
コミット
Subversionサーバ
Subversionとは
• バージョン管理システム
• 上書きによる先祖がえりを防ぐため
• サーバにアップロードすることをコミットと呼ぶ
自動化
Exe更新
(Nightly Build)
コミット
プロジェクト全員が
最新コードの動作を確認可能
TiVersionとは
• 龍が如くチーム独自のバージョン管理システム
• ファイルサイズの大きなものを高速でやり取り可能
最終成果物サーバ
(TiVersion)

# バイナリ管理を分けている理由

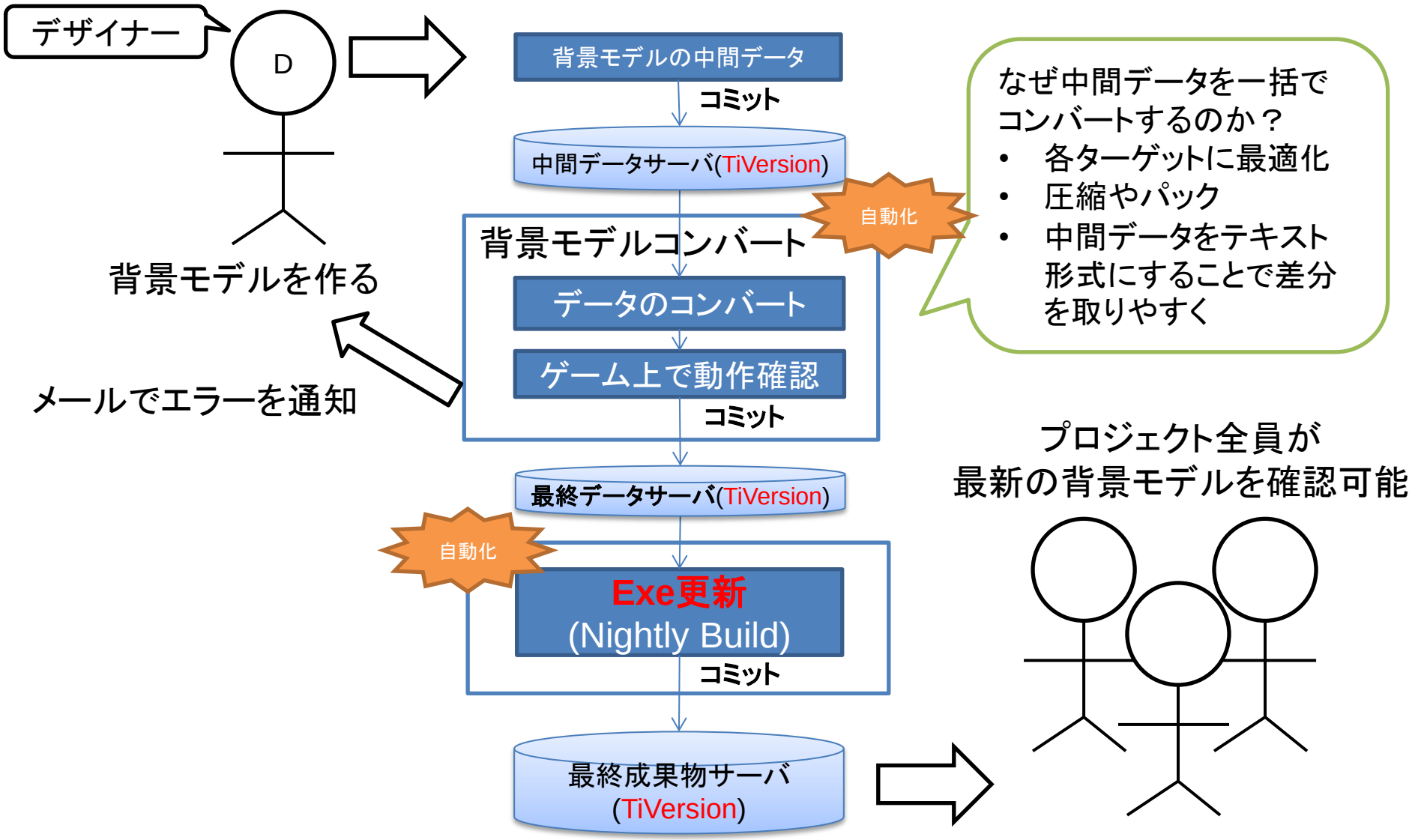



マージできなくていい！とにかく高速にやりとりしたい！

# パイプラインの自動化

- ビルド
- コンバート

# コンバートの自動化


デザイナー
D
背景モデルを作る
メールでエラーを通知
背景モデルの中間データ
コミット
中間データサーバ(TiVersion)
自動化
背景モデルコンバート
データのコンバート
ゲーム上で動作確認
コミット
最終データサーバ(TiVersion)
自動化
Exe更新
(Nightly Build)
コミット
最終成果物サーバ
(TiVersion)
なぜ中間データを一括で
コンバートするのか？
• 各ターゲットに最適化
• 圧縮やパック
• 中間データをテキスト
形式にすることで差分
を取りやすく
プロジェクト全員が
最新の背景モデルを確認可能

# ゲーム上で動作確認

- 実際にゲームを起動して、データが正しく読み込まれ、動作しているかを確認する簡易なテスト
  1. ゲームをテストモードで起動
  2. 背景データを読み込み
  3. 背景データだけを描画
  4. エラーが出ないことを確認し、終了して成功コードを返す

# パイプライン(ビルド・コンバート)の自動化

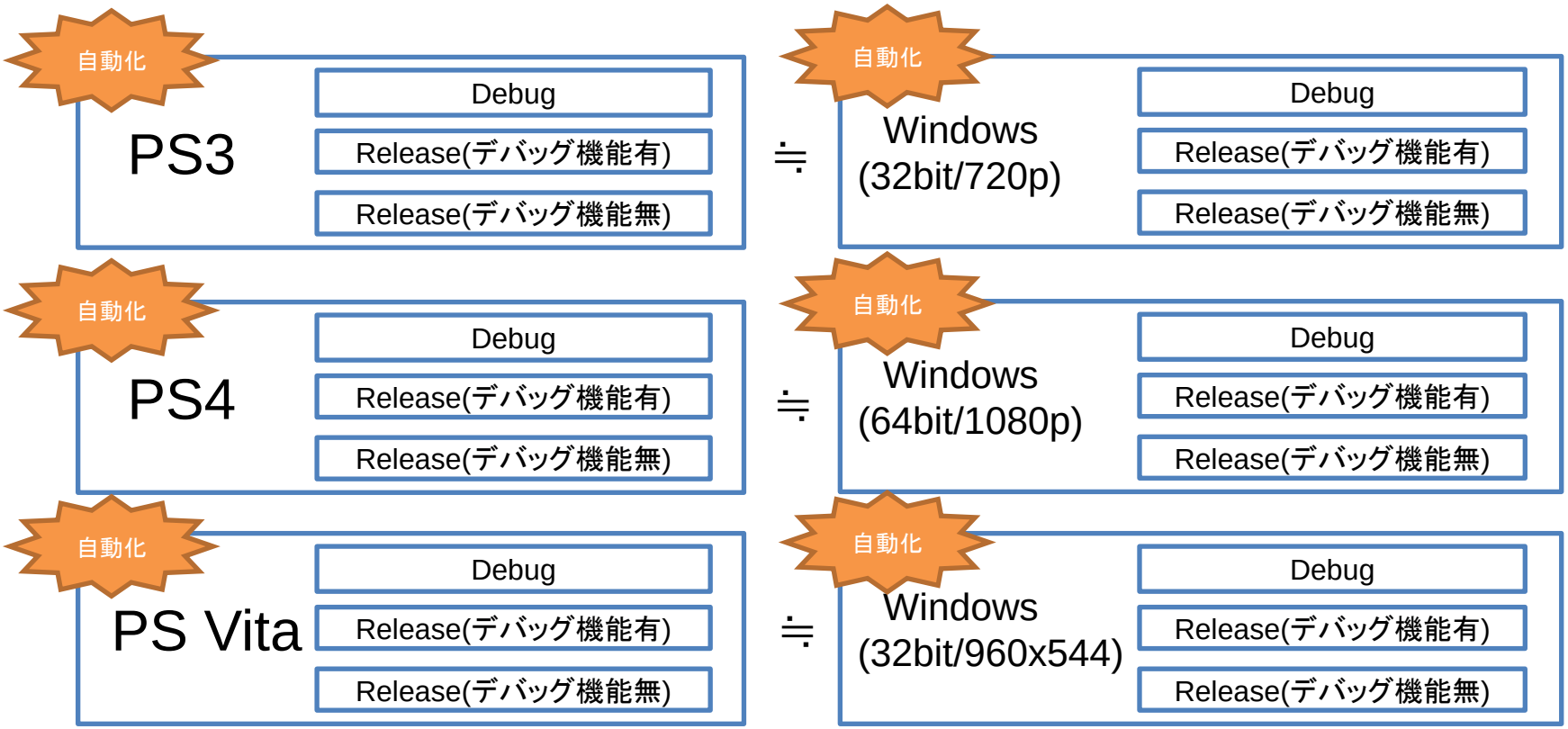
# エラー検出の自動化

- エラーの種類
  - ビルドエラー
  - 静的コード解析の指摘
  - ランタイムエラー
  - テストのエラー
- なぜ必要？
  - エラーを見つけて報告するのもコスト
  - 自動で不具合を見つけてくれるものは、どんどん使う
  - 手動だと欠落しやすいデバッグに必要なデータ（ダンプやログ、スクリーンショット等）を自動収集

# エラー検出の自動化

- ビルドエラー
- 静的コード解析の指摘
- ランタイムエラー
- テストのエラー

# ビルドエラー検出の自動化 (1/4)




手動で確認するのは大変！

# ビルドエラー検出の自動化 (2/4)

- Exe更新とは別にSubversionサーバを監視して、コミットごとにビルドが壊れていないかチェックを行う
- ビルドが壊れていたら、Jenkinsからメーリングリストにメールを自動送信
- ビルドエラーが出たら即時直すルール
  - ビルドエラーは出さないことが望ましいが、 ビルド構成が多く、各自がすべてを確認してコミットするのは無駄が多いため
- ビルドは分散ビルドを使用
- PCの台数を増やして高速化

# ビルドエラー検出の自動化 (3/4)

警告はエラーです

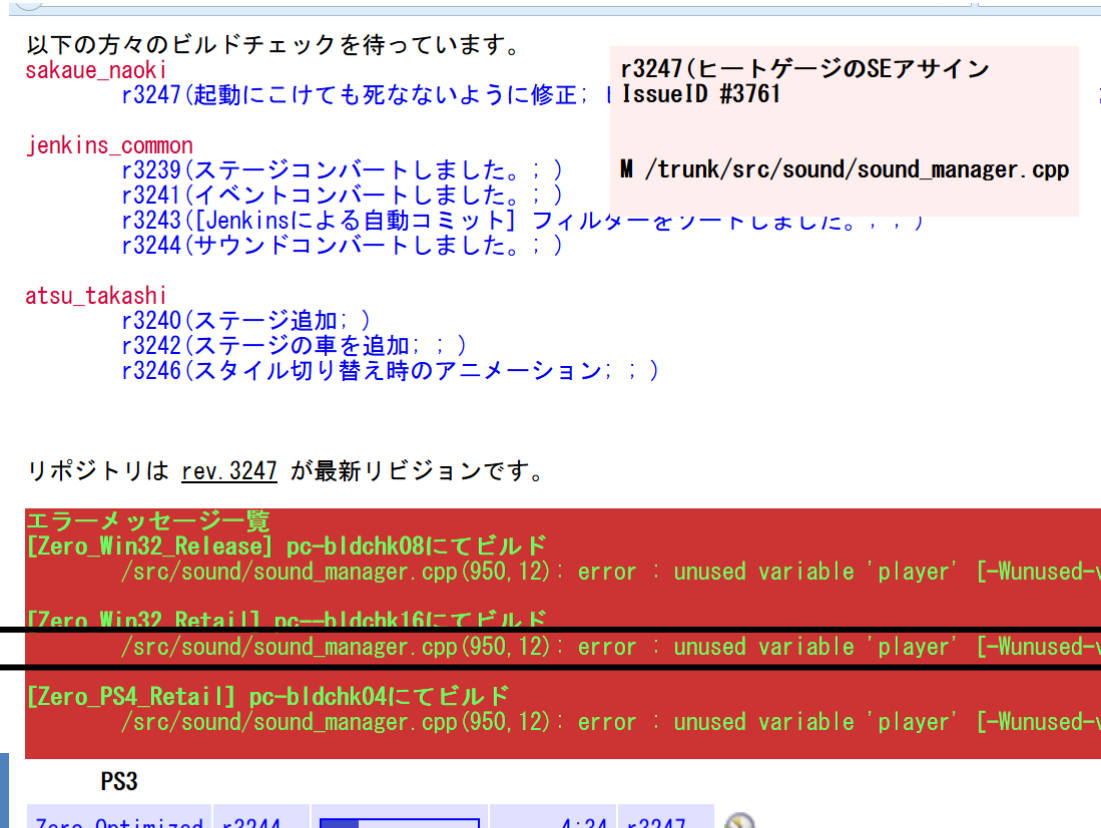
- コンパイラの警告は、エラー扱い
  - ビルドエラーになるので、即時修正が必要
  - コンパイラの警告レベルは最高に設定
  - 例外的に使いたいものは、設定で除外するか#pragmaで局所的に対応
  - まずは警告をゼロにするところから

# ビルドエラー検出の自動化 (4/4)

- ビルドが壊れた状態を短縮するためには
  - 声かけが一番効果的
  - チェック結果を分かりやすく表示する
    - ビルドチェック待ちページ

このリストから
自分の名前が
消えたら帰宅可能

unused variable
警告はエラー扱い



# エラー検出の自動化

- ビルドエラー
- 静的コード解析の指摘
- ランタイムエラー
- テストのエラー

# 静的コード解析の自動化

- 静的コード解析とは？
  - プログラムを実行せずにソースコードを解析し、不具合やその可能性となるものを検出するツール
- なぜ必要？
  - すべてのコードを人力でレビューするのは大変
  - 不具合の芽を先に摘んでおく
- 静的コード解析ツール
  - 使用ツール
    - Coverity（精度が高い/有償）
    - Visual Studioのコード分析（Visual Studioの標準機能）
    - Cppcheck（オープンソース）
  - 使用ツールは、管理できるなら多ければ多いほどいい
  - Jenkinsを使って毎日自動的に解析して、結果をメールやチケットでフィードバック

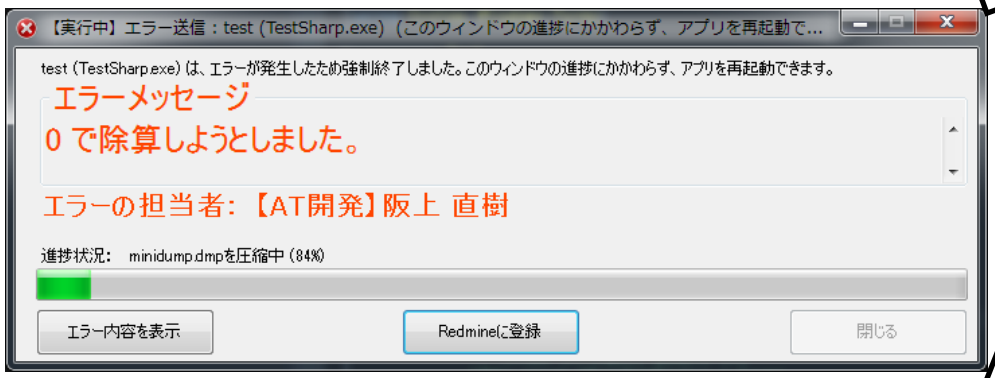
# エラー検出の自動化

- ビルドエラー
- 静的コード解析の指摘
- ランタイムエラー
- テストのエラー

# ランタイムエラー報告の自動化

- エラー送信（クラッシュレポート機能の龍が如くチーム用語）

ゲームやツール
実行中に
例外発生！




アップロード

ネットワークドライブ

メール送信

- ダンプ表示batのURL
- ログ表示batのURL
- コールスタック
- リビジョン
- ターゲット

バグ報告

バグ管理システム
(Redmine)

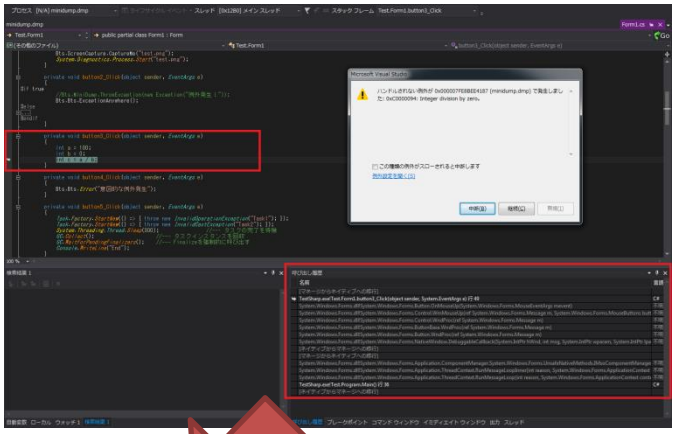
デバッグに必要な情報を自動収集
- ダンプ
- ログ
- コールスタック
- スクリーンショット … etc

# エラー送信の実装例 (Windowsの場合)

- C++
  - 未処理例外のキャッチ
    - SetUnhandledExceptionFilter
  - コールスタック
    - StackWalk
- C#
  - 未処理例外のキャッチ
    - System.Windows.Forms.Application.ThreadException
    - AppDomain.CurrentDomain.UnhandledException
    - System.Threading.Tasks.TaskScheduler.UnobservedTaskException
  - コールスタック
    - Environment.StackTrace
- ダンプ出力
  - MiniDumpWriteDump
    - C#ではアンマネージド・コード呼び出し
    - ダンプを開くときは、ダンプファイル(.dmp)以外に、exeファイルと対応するpdbファイルが必要

# ダンプを開いた例(C#)

```
private void button3_Click(object sender, EventArgs e)
{
    int a = 100;
    int b = 0;
    int c = a / b;
}
```



呼び出し履歴

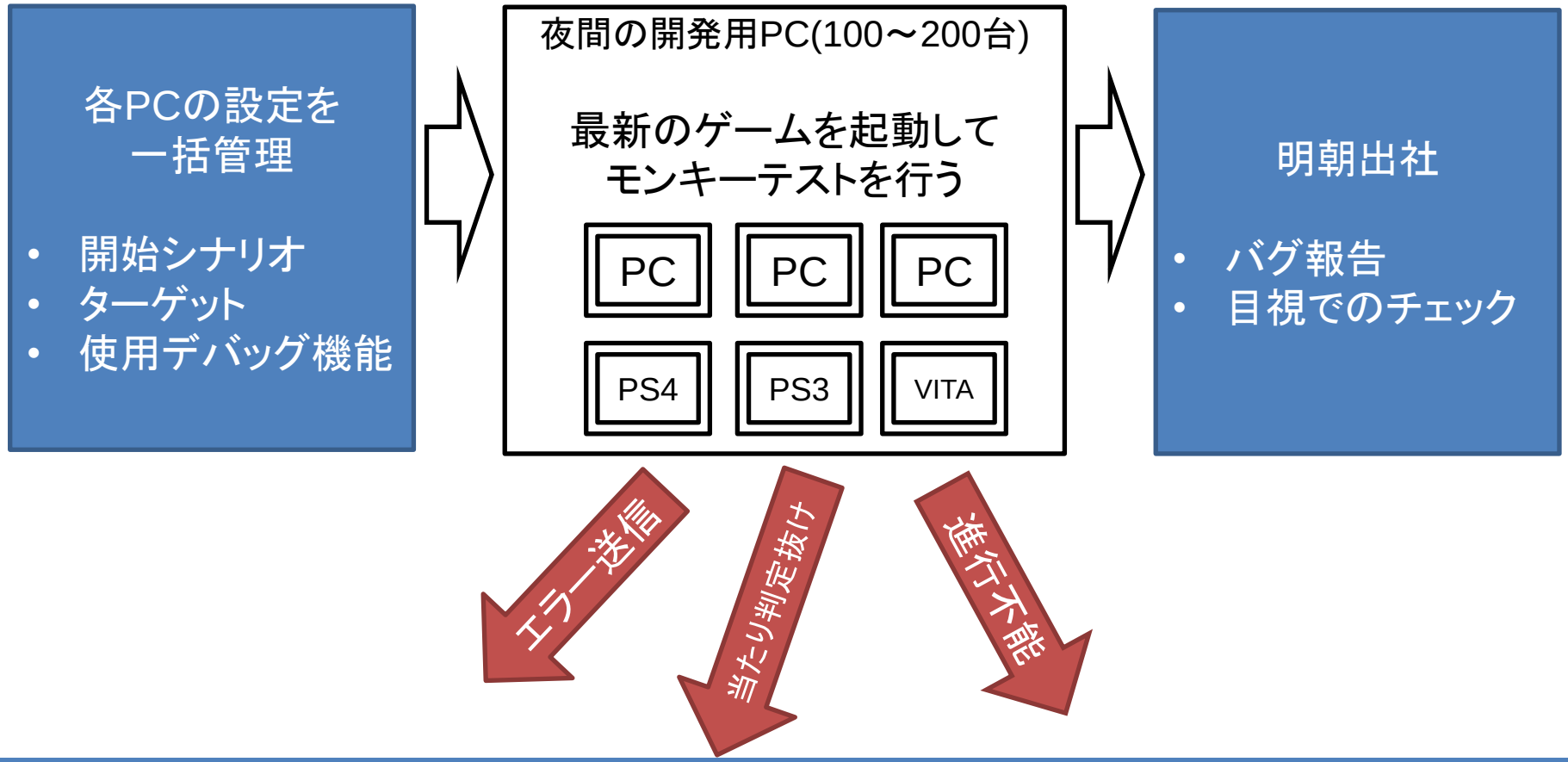
| 名前 |
| --- |
| [マネージからネイティブへの移行] |
| TestSharp.exe!Test.Form1.button3_Click(object sender, System.EventArgs e) 行 49 |
| System.Windows.Forms.dll!System.Windows.Forms.Button.OnMouseUp(System.Windows.Forms.MouseEventArgs mevent) |
| System.Windows.Forms.dll!System.Windows.Forms.Control.WmMouseUp(ref System.Windows.Forms.Message m, System.Windows.Forms.MouseButtons butt |
| System.Windows.Forms.dll!System.Windows.Forms.Control.WndProc(ref System.Windows.Forms.Message m) |
| System.Windows.Forms.dll!System.Windows.Forms.ButtonBase.WndProc(ref System.Windows.Forms.Message m) |
| System.Windows.Forms.dll!System.Windows.Forms.Button.WndProc(ref System.Windows.Forms.Message m) |
| System.Windows.Forms.dll!System.Windows.Forms.NativeWindow.DebuggableCallback(System.IntPtr hWnd, int msg, System.IntPtr wparam, System.IntPtr lpa |
| [ネイティブからマネージへの移行] |
| [マネージからネイティブへの移行] |
| System.Windows.Forms.dll!System.Windows.Forms.Application.ComponentManager.System.Windows.Forms.UnsafeNativeMethods.IMsoComponentManage |
| System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoopInner(int reason, System.Windows.Forms.ApplicationContext |
| System.Windows.Forms.dll!System.Windows.Forms.Application.ThreadContext.RunMessageLoop(int reason, System.Windows.Forms.ApplicationContext cont |
| TestSharp.exe!Test.Program.Main() 行 36 |
| [ネイティブからマネージへの移行] |

# エラー検出の自動化

- ビルドエラー
- 静的コード解析の指摘
- ランタイムエラー
- テストのエラー

# テストの自動化 (1/4)

- モンキーテストとは
  - ランダムに擬似パッド入力を行うテスト手法
- オートテスト（モンキーテストの龍が如くチーム用語）



# テストの自動化 (2/4)

- Excelファイルに設定を記述


| 種類 | シナリオチャプター | シナリオタイトル | AUTO | AUTO | AUTO | AUTO | AUTO | AUTO | AUTO | dip SYSTEM | dip SYSTEM | dip SYSTEM | dip SYSTEM | dip SYSTEM | dip SYSTEM | その他 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DIP力テゴリ | | | 自動スタート | W32 | W64 | PS3 | PS4 | VITA | ムービーキャプチャ | ランダムインプット | 非アクティブ時も実行する | パッドを振動させない | 主人公無敵 | 敵無敵 | 敵即死 | 最後にdbg_logを送る |
| 名前 | シナリオチャプター | シナリオタイトル | | | | | | | | | | | | | | |
| 名無し用 | PADV | プレミアムアドベンチャー | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name01 | PADV | プレミアムアドベンチャー | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name02 | PADV | プレミアムアドベンチャー | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name03 | PADV | プレミアムアドベンチャー | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name04 | PADV | 闘技場 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name05 | PADV | 闘技場 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name06 | PADV | 闘技場 | ○ | ○ | | | | ○ | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name07 | PADV | 闘技場 | ○ | ○ | | | | ○ | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name08 | 第一章 | 第一章 タイトル表示 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name09 | 第一章 | 第一章 タイトル表示 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name10 | 第一章 | 第一章 タイトル表示 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | | | | ○ |
| name11 | 第二章 | 第二章 タイトル表示 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name12 | 第二章 | 第二章 タイトル表示 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name13 | 第二章 | 第二章 タイトル表示 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | | | | ○ |
| name14 | 第三章 | 第三章 タイトル表示 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name15 | 第三章 | 第三章 タイトル表示 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | ○ | | | ○ |
| name16 | 第三章 | 第三章 タイトル表示 | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ | ○ | | | | ○ |

# テストの自動化 (3/4)

- 各PCでオートテストを実行

オートテストクライアントを実行

↓

設定ファイルから
iniを生成

↓

TiVersionから
最新のexe更新を取得

↓

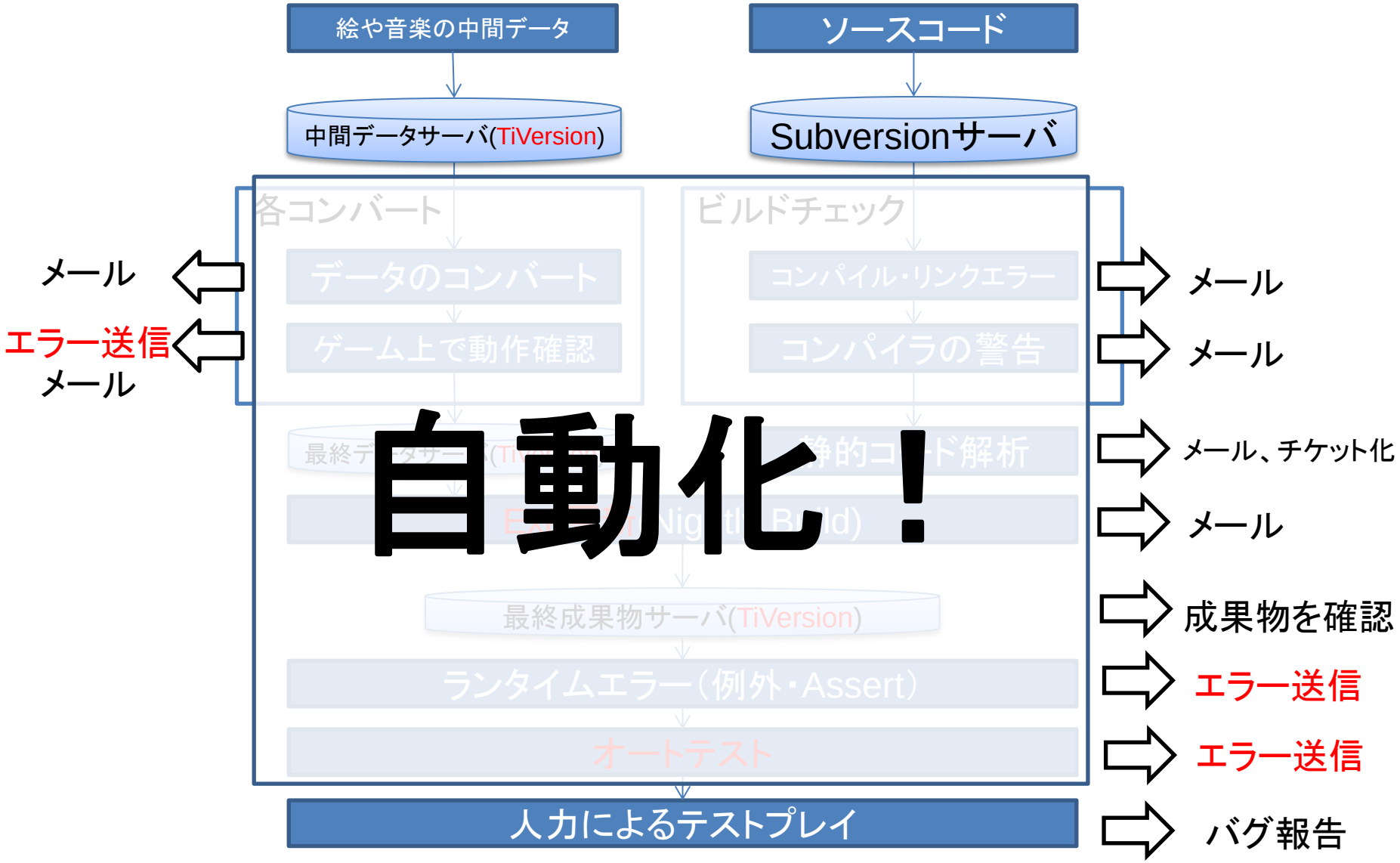
ゲームを自動起動
(iniファイルのシナリオ/条件)

最終成果物サーバ
(TiVersion)

# テストの自動化 (4/4)

- テストで検出されたバグを見逃さない
  - エラー送信
  - ログ出力
  - 動画撮影
- 人力で出せないバグの検出
  - 1/60秒単位の特定のパッド入力で発生
  - 長時間同じ動作を繰り返すことで発生
- 人力で出せないバグを見つける必要があるのか？
  - 数十人では出ないバグも、数十万人がプレイすると出てしまう可能性がある
  - 人力のチェックはリソースに限りがあるため、バグを自動で検出する仕組みは、大規模ゲーム開発には欠かせない

# 高速デバッグ術（自動化）のまとめ

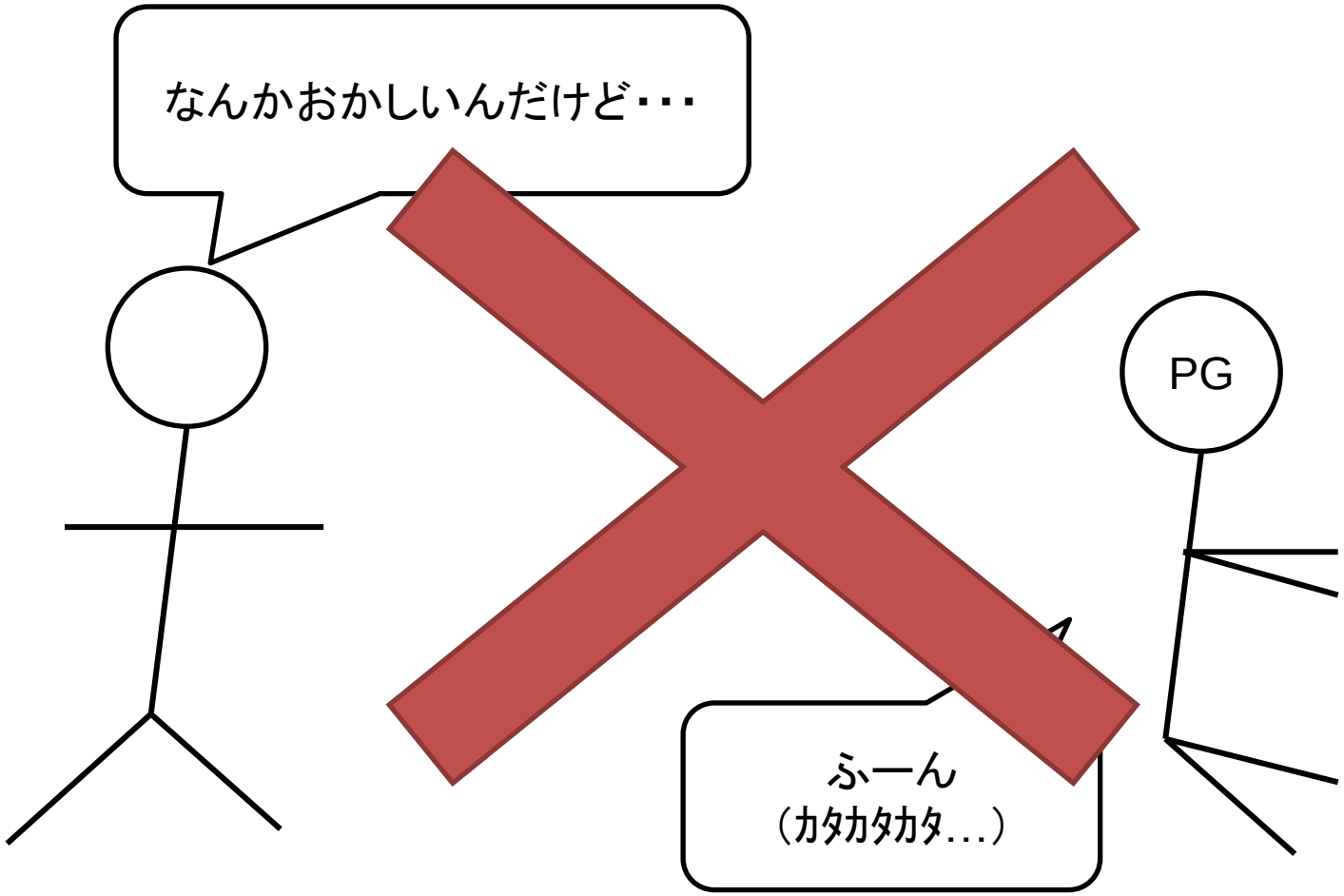



**自動化！**

# そびえ立つバグの山を踏破するための弾丸ワークフロー

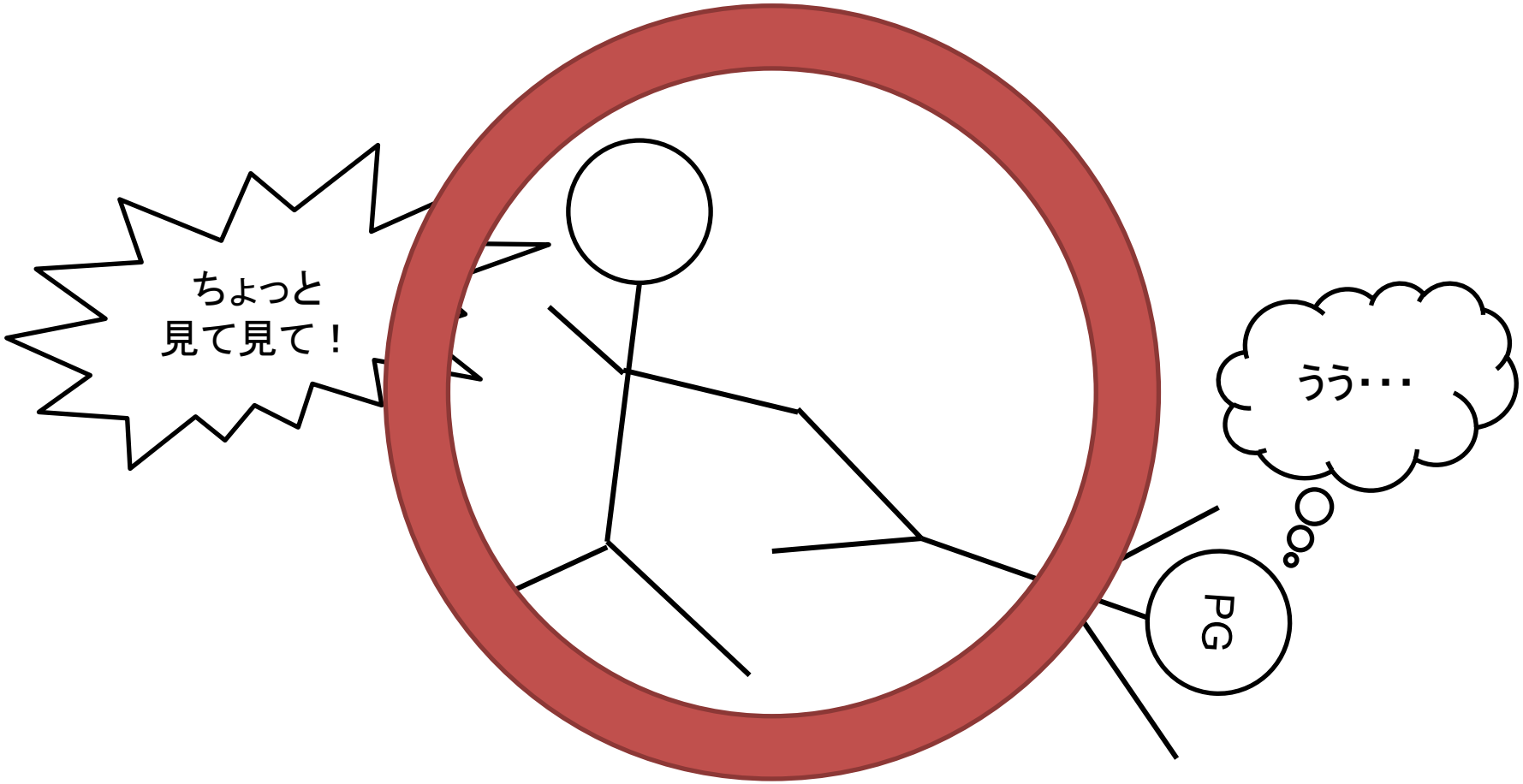
- バグ管理
  - Redmineを使用
  - バグ報告の掟
  - バグ管理システムの導入の歴史
  - バグ管理システムの問題と解決

# バグ報告の掟



# バグ報告の掟(破り)

なんとかしてプログラマにバグの現場を見せる！



# バグ管理システムの必要性

- 口頭でのやり取りは、数人程度が限界
- バグ管理システムが必要な場合
  - 数百人規模のプロジェクト
  - それぞれが別の場所で作業している

# バグ管理システムの導入の歴史

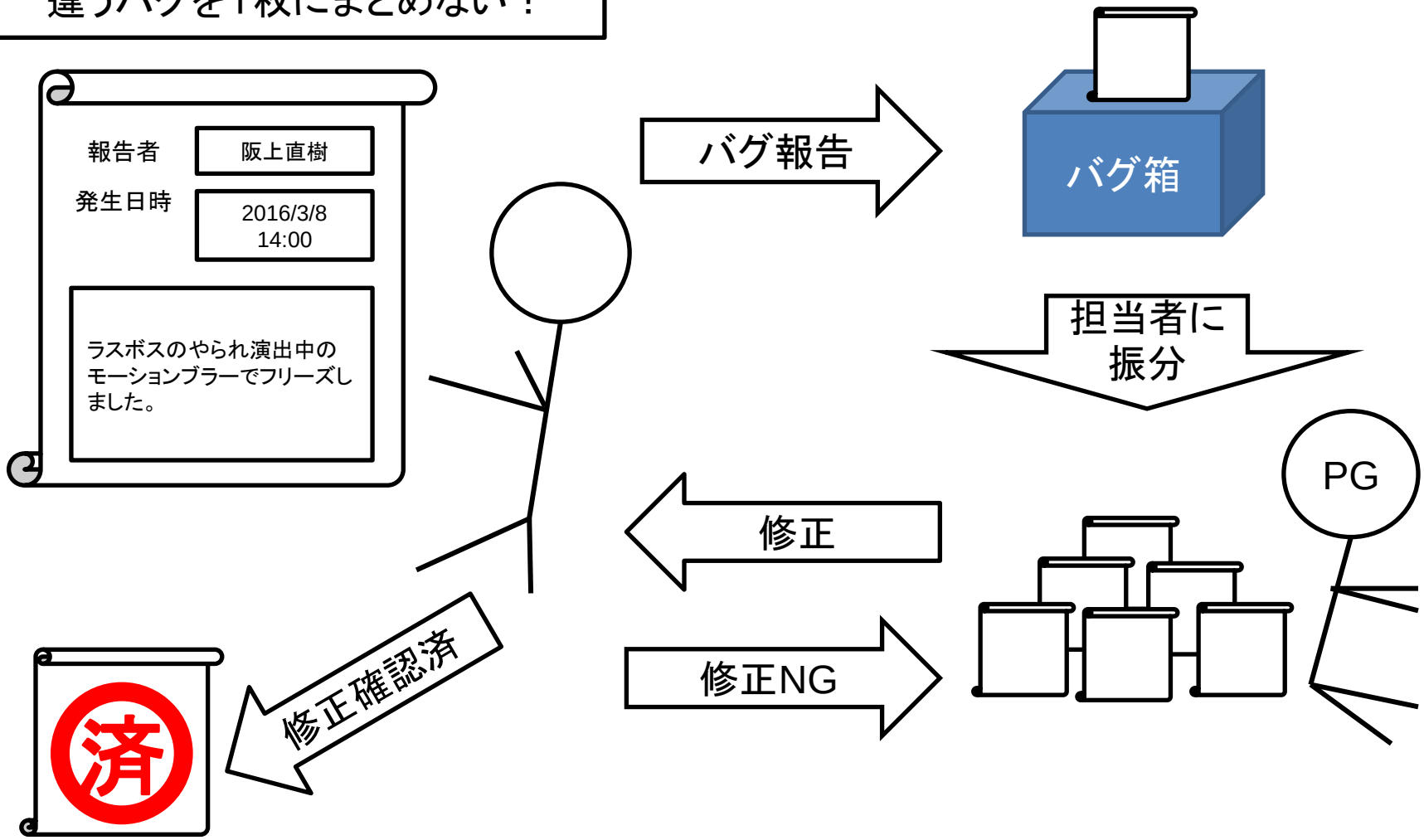
- ~~口頭だけ~~
- 紙で管理
- Excelで管理
- オリジナルツールで管理
- Mantisで管理
- Redmineで管理（現在）

# バグ管理システムの導入の歴史

- ~~口頭だけ~~
- 紙で管理
- Excelで管理
- オリジナルツールで管理
- Mantisで管理
- Redmineで管理（現在）

# 紙でバグ管理 (1/2)

バグ1つに紙1枚（チケット）
違うバグを1枚にまとめない！



# 紙でバグ管理 (2/2)

- メリット
  - バグの見える化
    - 各自の記憶に頼らない
    - 誰が何個バグを抱えているのか
  - アナログならでは
    - バグの報告者・担当者間のコミュニケーションがスムーズ
    - 少人数では一番高速
- デメリット
  - かかわる人数が増えると、チケットの管理が煩雑
  - チケット全体を管理できない
    - 総数が分からない
    - バグの傾向が分からない
    - 各バグのステータス（対応状況）が分からない
    - どのバグから手をつけていいのか分からない

# バグ管理システムの導入の歴史

- ~~口頭だけ~~
- 紙で管理
- Excelで管理
- オリジナルツールで管理
- Mantisで管理
- Redmineで管理（現在）

# Excelでバグ管理

※Office 365がなかったころの話

- メリット
  - 情報のデジタル化
  - 共有設定にして、1ファイルですべてのバグを管理
- デメリット
  - 誰が編集したのか分からない
  - 共有設定にすると、ファイルがたびたび壊れる
  - 部外とのやり取りが煩雑
    - マージ作業が発生してしまう
  - 情報が漏洩しやすい
  - バグのステータスが分からない

# バグ管理システムの導入の歴史

- ~~口頭だけ~~
- 紙で管理
- Excelで管理
- オリジナルツールで管理
- Mantisで管理
- Redmineで管理（現在）

# オリジナルツールでバグ管理

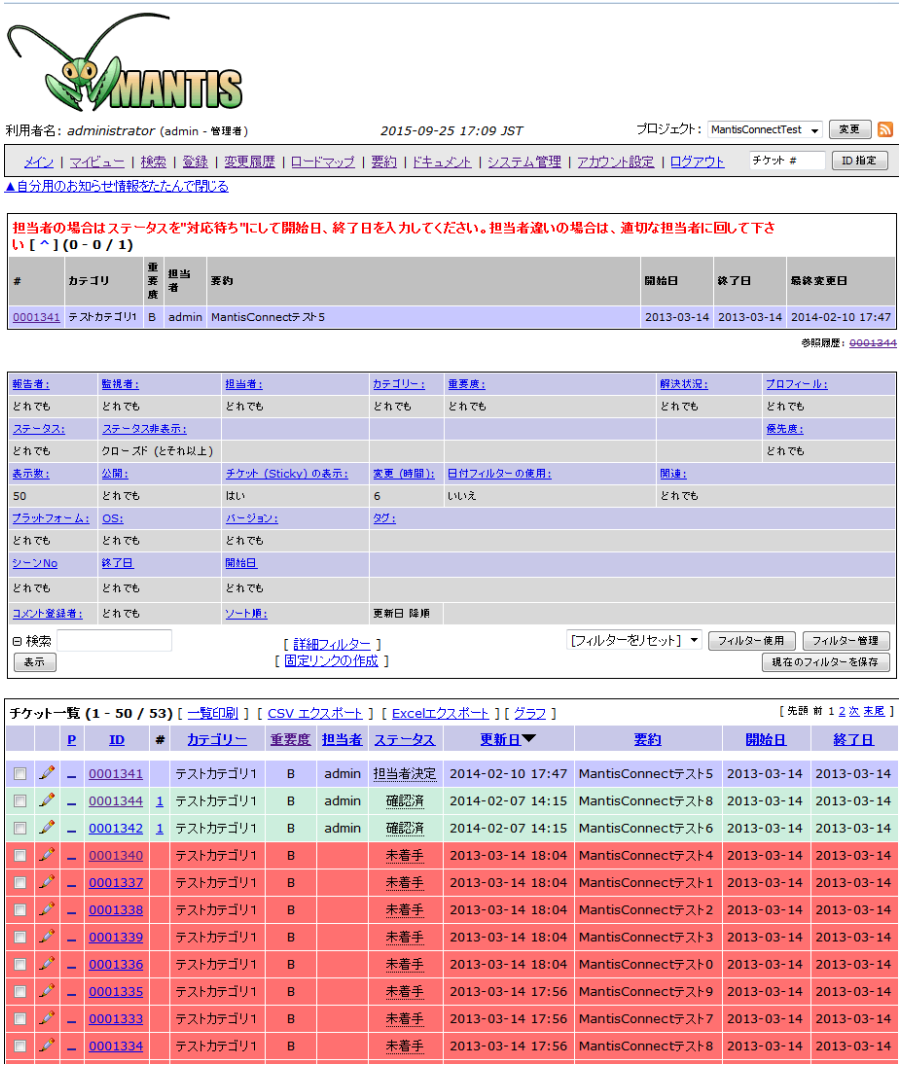
- メリット
  - サーバで一元管理
  - ツールの開発者が社内にいるので、要望を追加しやすい
- デメリット
  - ツールの属人化
    - 開発者がいなくなったらメンテナンスできない
  - ツールの学習コスト
    - 社内システムなので、ツールに慣れてもらう時間が必要

# バグ管理システムの導入の歴史

- ~~口頭だけ~~
- 紙で管理
- Excelで管理
- オリジナルツールで管理
- Mantisで管理
- Redmineで管理（現在）

# Mantisでバグ管理(1/2)

- Mantisとは
  - バグ管理システム(BTS)
  - Webブラウザでアクセス
  - オープンソース



# Mantisでバグ管理(2/2)

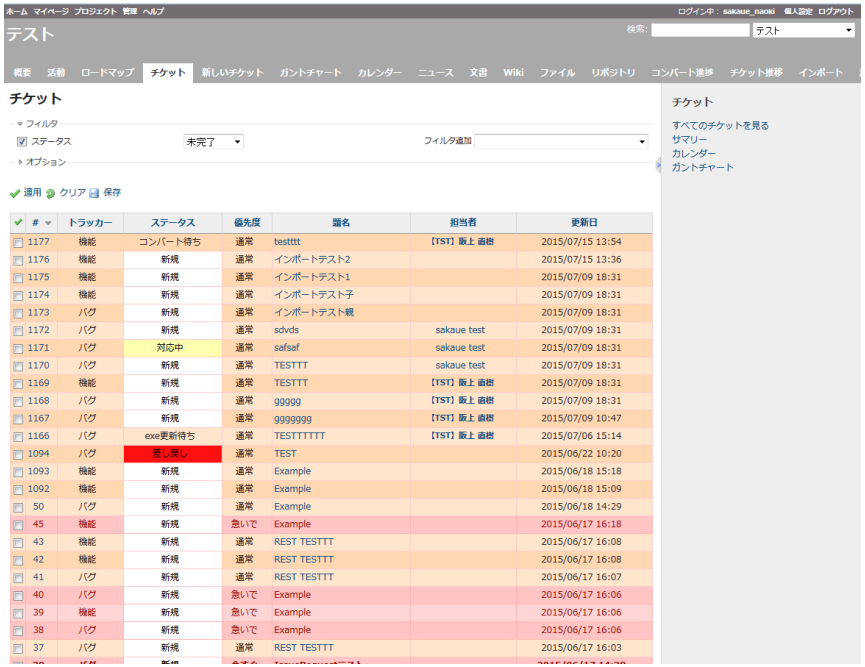
- メリット
  - サーバによる一元管理
  - スクリーンショットやログファイルを添付できる
  - コメント入力やステータス、優先度を設定できる
  - 有名なツールなので、学習コストを削減できる
- デメリット
  - バグ以外の管理には不向き（タスクなど）

# バグ管理システムの導入の歴史

- ~~口頭だけ~~
- 紙で管理
- Excelで管理
- オリジナルツールで管理
- Mantisで管理
- Redmineで管理（現在）

# Redmineでバグ管理 (1/5)

- Redmineとは
  - バグを含めた課題管理システム
  - Webブラウザでアクセス
  - オープンソース

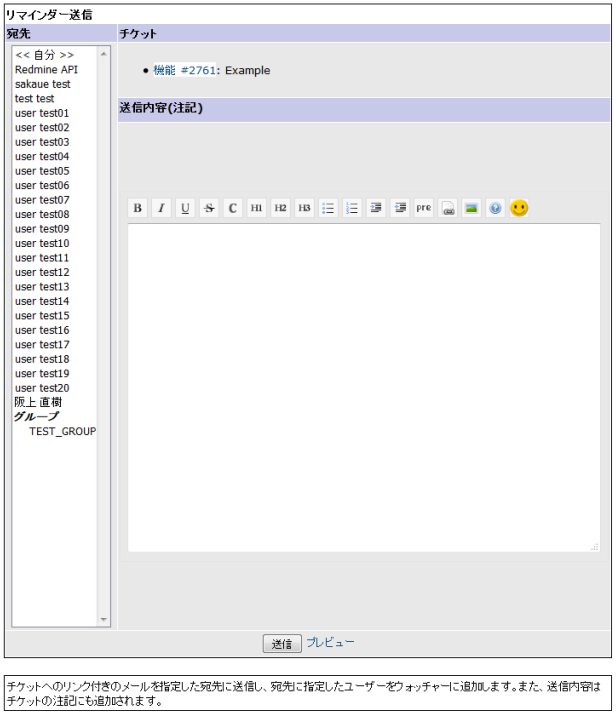



# Redmineでバグ管理 (2/5)

- タスクも管理するためにMantisからRedmineに移行
  - あれがない、これがない、使いにくいと言われる
    - 代替手順を提示して、なんとかツールに慣れてもらう
    - どうしても必要と思われる機能は、プラグインとして実装
      - リマインダー送信

Mantisのリマインダー送信


見た目もそのまま
Redmineに移植

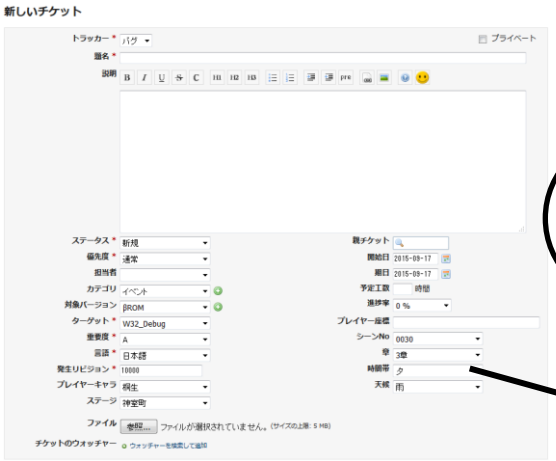


# Redmineでバグ管理 (3/5)

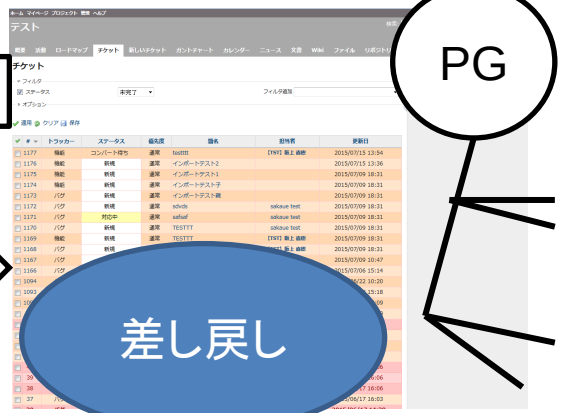
- メリット
  - サーバによる一元管理
  - スクリーンショットやログファイルを添付できる
  - コメント入力やステータス、優先度を設定できる
  - 有名なツールなので、学習コストを削減できる
  - APIやプラグイン等の拡張機能が豊富で、開発も活発
  - タスクも管理できる
- デメリット
  - 自由度の高いツールなので、運用ルール作りが必要
  - プラグインを多用すると、保守・メンテナンスコストが高くなる

# Redmineでバグ管理 (4/5)

## バグのステータスの一例





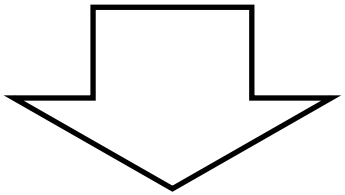


# Redmineでバグ管理 (5/5)

- なぜ「確認待ち」なのか？
  - 「修正完了」ではない？
    - バグ修正を実装した時点で、チケットは終わっていない！
    - 修正を確認して、はじめてチケットは完了する。
  - ステータス名は、見ただけで次に何をしなければならないかがすぐ分かるのがよい

# ツールを使えば、すべて解決？

もちろん、そんなことは無いです。



- バグ管理システムの問題点
  - Excel使いたいよ・・・の声
  - バグ報告の問題点
  - バグのワークフローの問題点
  - 各ツールにプロジェクトの情報が分散

# バグ管理システムの問題と解決

- Excel使いたいよ・・・の声
- バグ報告の問題点
- バグのワークフローの問題点
- 各ツールにプロジェクトの情報が分散

# Excel使いたいよ・・・の声 (1/2)

- RedmineにExcelのインポート＆エクスポートを追加
- 数百個のタスクを一括で登録したい
  - redmine_importプラグインを導入
    - Redmine 3.2で標準機能に追加
- Excelで見たい人にチケット一覧をxlsファイルに変換
  - redmine_xls_exportプラグインを導入

# Excel使いたいよ・・・の声 (2/2)

- インポート用のテンプレートを作成
  - Redmine REST API経由でトラッカー別に自動生成
  - Jenkinsで定期的に更新
    - カテゴリやメンバーの追加に対応するため

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | トラッカー | 題名 | 説明 | 優先度 | 担当者 | カテゴリ | 対象バージョン | 開始日 | 期日 |
| 2 | タスク | タスクテスト1 | タスクテスト1の説明です。 | 高め | 阪上 直樹 | イベント | α ROM | 2016/1/1 | 2016/2/1 |
| 3 | タスク | タスクテスト2 | タスクテスト2の説明です。 | 高め | sakaue test | イベント | α ROM | 2016/1/2 | 2016/2/2 |
| 4 | タスク | タスクテスト3 | タスクテスト3の説明です。 | 高め | 阪上 直樹 | プログラム | α ROM | 2016/1/3 | 2016/2/3 |
| 5 | タスク | タスクテスト4 | タスクテスト4の説明です。 | 通常 | 阪上 直樹 | プログラム | β ROM | 2016/1/4 | 2016/2/4 |
| 6 | タスク | タスクテスト5 | タスクテスト5の説明です。 | 通常 | 阪上 直樹 | プログラム | β ROM | 2016/1/5 | 2016/2/5 |
| 7 | タスク | タスクテスト6 | タスクテスト6の説明です。 | 通常 | 阪上 直樹 | プログラム | β ROM | 2016/1/6 | 2016/2/6 |
| 8 | タスク | タスクテスト7 | タスクテスト7の説明です。 | 通常 | 阪上 直樹 | プログラム | α ROM<br>β ROM<br>Final ROM<br>Master ROM | 2016/1/7 | 2016/2/7 |
| 9 | タスク | タスクテスト8 | タスクテスト8の説明です。 | 通常 | 阪上 直樹 | プログラム | | 2016/1/8 | 2016/2/8 |
| 10 | タスク | タスクテスト9 | タスクテスト9の説明です。 | 通常 | 阪上 直樹 | プログラム | | 2016/1/9 | 2016/2/9 |
| 11 | タスク | タスクテスト10 | タスクテスト10の説明です。 | 通常 | 阪上 直樹 | プログラム | Final ROM | 2016/1/10 | 2016/2/10 |
| 12 | タスク | タスクテスト11 | タスクテスト11の説明です。 | 通常 | 阪上 直樹 | プログラム | Master ROM | 2016/1/11 | 2016/2/11 |
| 13 | タスク | タスクテスト12 | タスクテスト12の説明です。 | 通常 | 阪上 直樹 | プログラム | Master ROM | 2016/1/12 | 2016/2/12 |
| 14 | タスク | タスクテスト13 | タスクテスト13の説明です。 | 通常 | 阪上 直樹 | プログラム | Master ROM | 2016/1/13 | 2016/2/13 |

# バグ管理システムの問題と解決

- Excel使いたいよ・・・の声
- バグ報告の問題点
- バグのワークフローの問題点
- 各ツールにプロジェクトの情報が分散

# バグ報告の問題点

- 必要な情報を入力するのは結構大変
  - テストプレイがメインの仕事でない時に、バグ報告を怠りがちになる
- ログの添付を忘れやすい
  - 手動だと、必要な情報を入力・添付し忘れる
- 入力ミスも発生する
  - リビジョン番号が違う
  - ターゲットの選択を間違える

# バグ報告に必要な情報

## 基本情報

- 原因
- 結果
- 再現率
- 重要度
- カテゴリ
- 報告者
- ターゲット
  (Win/PS3/PS4/VITA)
- 言語
  (日本語/中国語)
- リビジョン

## 添付ファイル

- ダンプファイル
- デバッグログ
- 使ったデバッグ機能
  （設定ファイル等）
- スクリーンショット
- セーブデータ

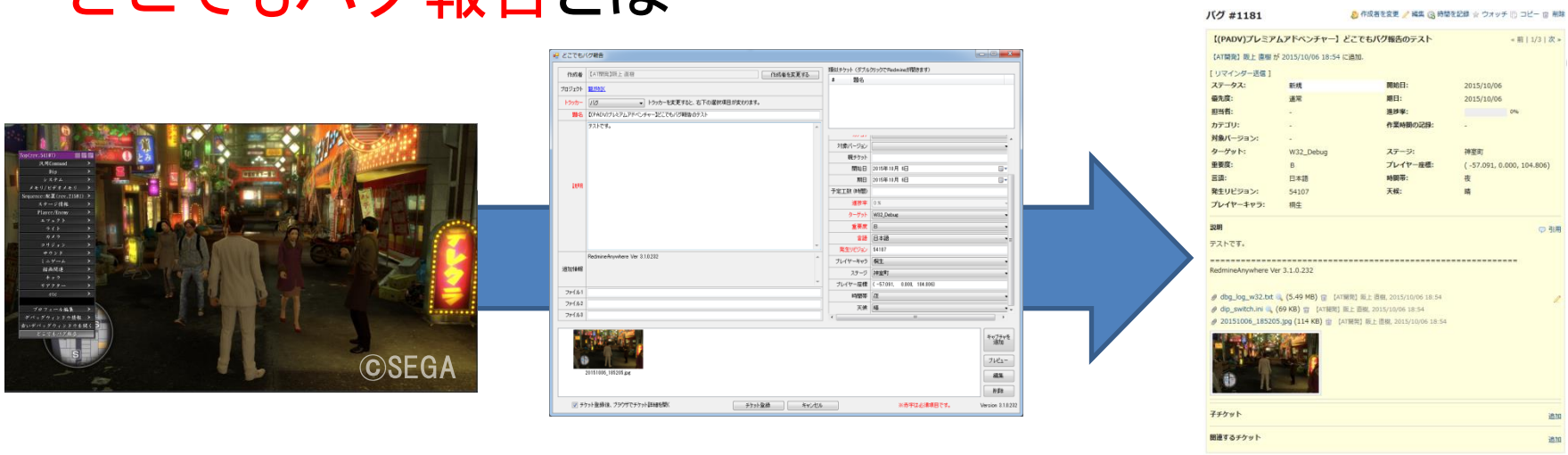
## 龍が如く独自

- プレイヤーキャラ
  (桐生/真島/秋山…)
- ステージ
  (神室町/蒼天堀…)
- プレイヤーの座標
- シナリオ進行度
- 時間帯(昼/夜)
- 天候(晴/雨/雪)

自動入力できる！

# バグ報告の自動化

- どこでもバグ報告とは

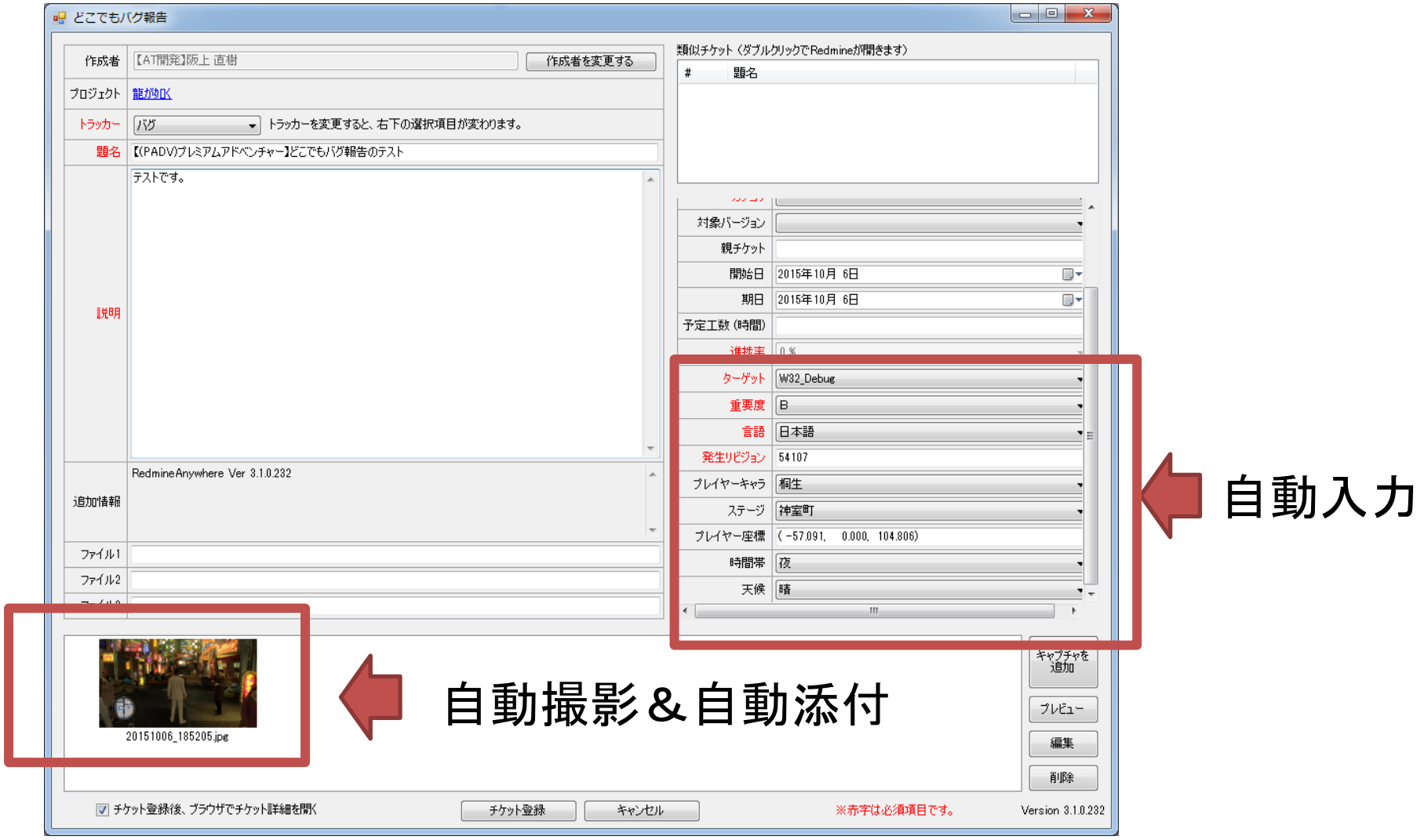



- Redmine REST APIを使用して、専用アプリ経由でゲームアプリから直接Redmineにチケット登録
- 全ターゲット(Win/PS3/PS4/PS Vitaに対応)
- エラー送信からも直接バグ報告が可能
- デバッグ機能がないと使えない

# どこでもバグ報告 (1/3)


デバッグメニューからどこでもバグ報告をクリック！

# どこでもバグ報告 (2/3)


どこでもバグ報告
作成者
【AT開発】阪上 直樹
作成者を変更する
プロジェクト
龍が如く
トラッカー
バグ
トラッカーを変更すると、右下の選択項目が変わります。
題名
【(PADV)プレミアムアドベンチャー】どこでもバグ報告のテスト
説明
テストです。
追加情報
RedmineAnywhere Ver 3.1.0.232
ファイル1
ファイル2
類似チケット（ダブルクリックでRedmineが開きます）
#
題名
対象バージョン
親チケット
開始日
2015年10月 6日
期日
2015年10月 6日
予定工数（時間）
進捗率
0 %
ターゲット
W32_Debug
重要度
B
言語
日本語
発生リビジョン
54107
プレイヤーキャラ
桐生
ステージ
神室町
プレイヤー座標
( -57.091, 0.000, 104.806)
時間帯
夜
天候
晴
20151006_185205.jpg
キャプチャを追加
プレビュー
編集
削除
チケット登録後、ブラウザでチケット詳細を開く
チケット登録
キャンセル
※赤字は必須項目です。
Version 3.1.0.232
自動入力
自動撮影＆自動添付

# どこでもバグ報告 (3/3)


自動入力

自動添付

# バグ報告自動化による効果

| | どこでもバグ報告<br>使用件数 | バグの<br>総数 |
|---|---|---|
| 龍が如く 維新！ | **9,338件** | 17,448件 |
| 龍が如くO　誓いの場所 | **6,307件** | 12,955件 |
| 龍が如く 極 | **1,925件** | 4,799件 |

3タイトル(9,338件+6,307件+1,925件) X 10分(短縮時間/件)

**＝ 約3,000時間を削減**

バグ報告の質が均一化して、精度も向上

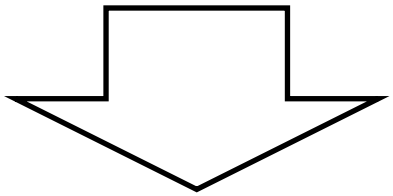
**テストプレイヤーは、バグ探しに専念！**
**クリエイターは、バグの修正・バランス調整・クオリティの向上に専念！**

# バグ管理システムの問題と解決

- Excel使いたいよ・・・の声
- バグ報告の問題点
- バグのワークフローの問題点
- 各ツールにプロジェクトの情報が分散

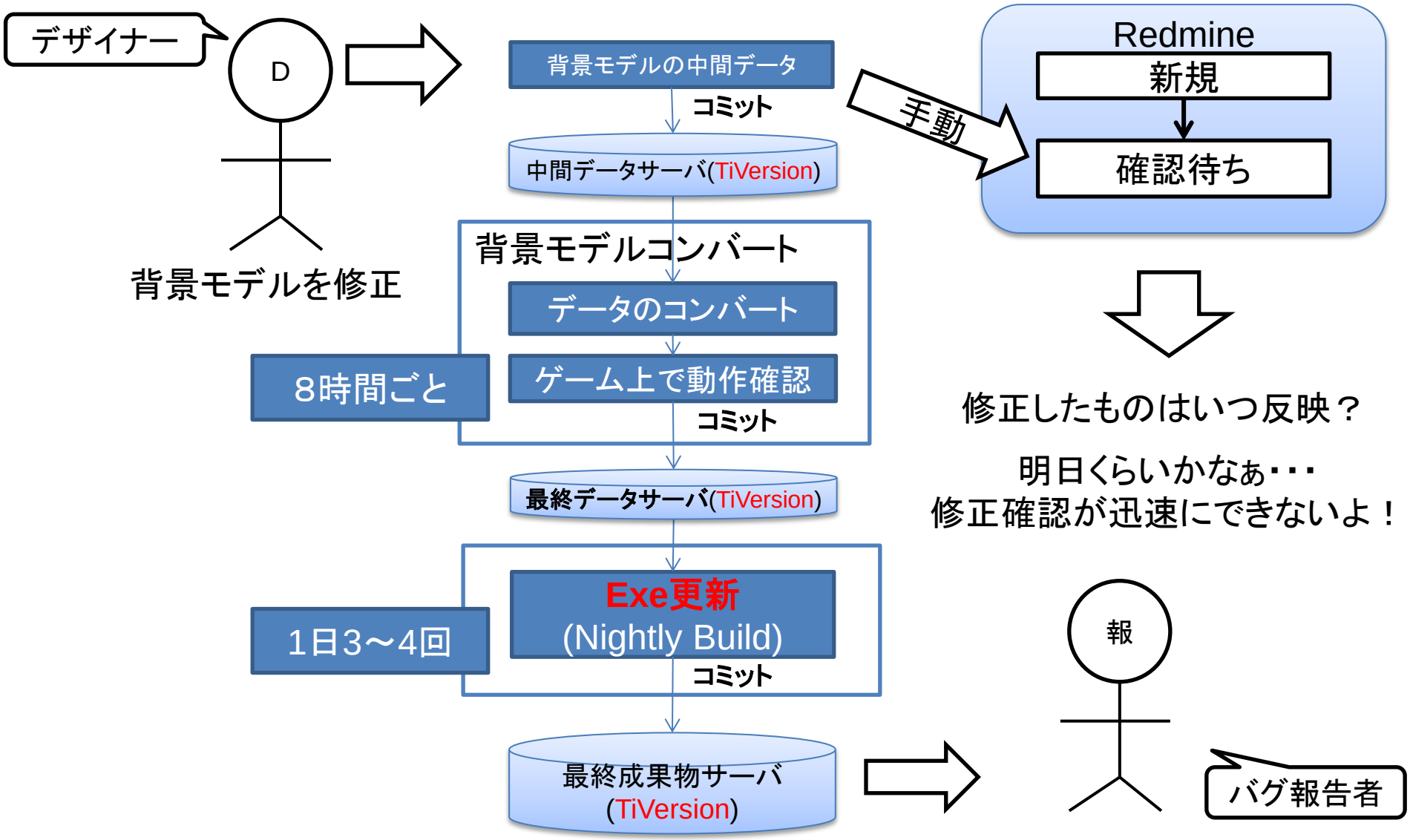
# バグのワークフローの問題点

- 修正したものがいつ反映されたか分からない
- バグチケットに対して、どういう修正が入ったのか分かりにくい

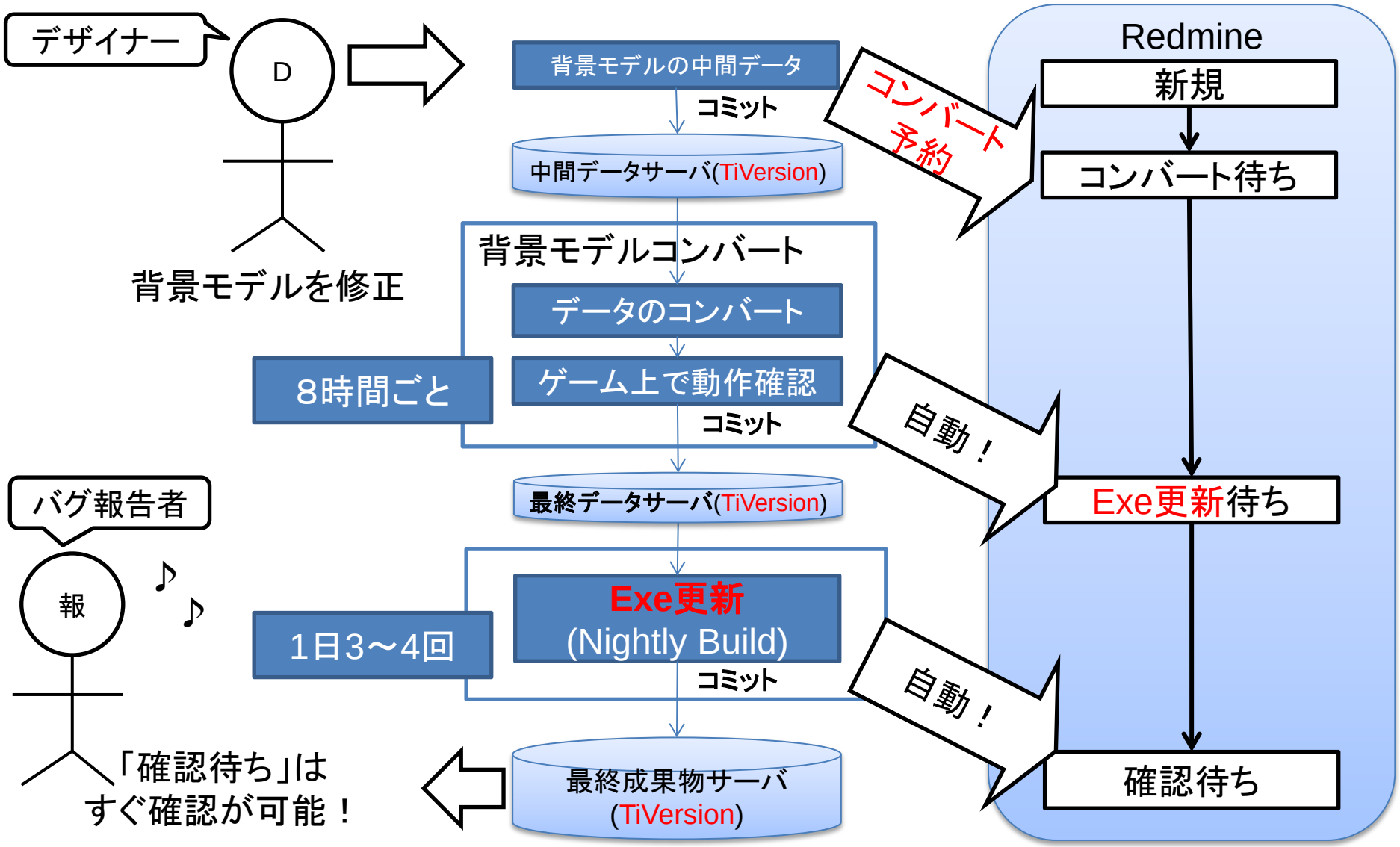


Redmineと各システム・ツールを連携して
ワークフローを自動化

# Redmineとパイプラインの連携 (1/4)

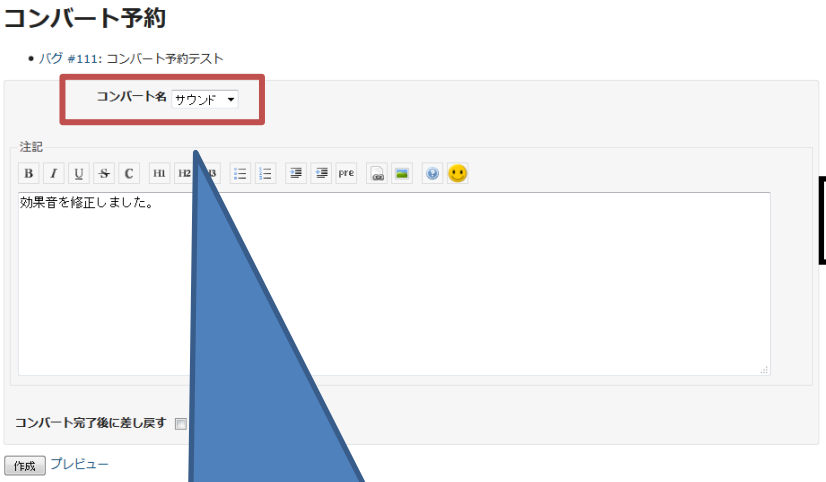


# Redmineとパイプラインの連携 (3/4)



# Redmineとパイプラインの連携 (4/4)

## コンバート予約とは



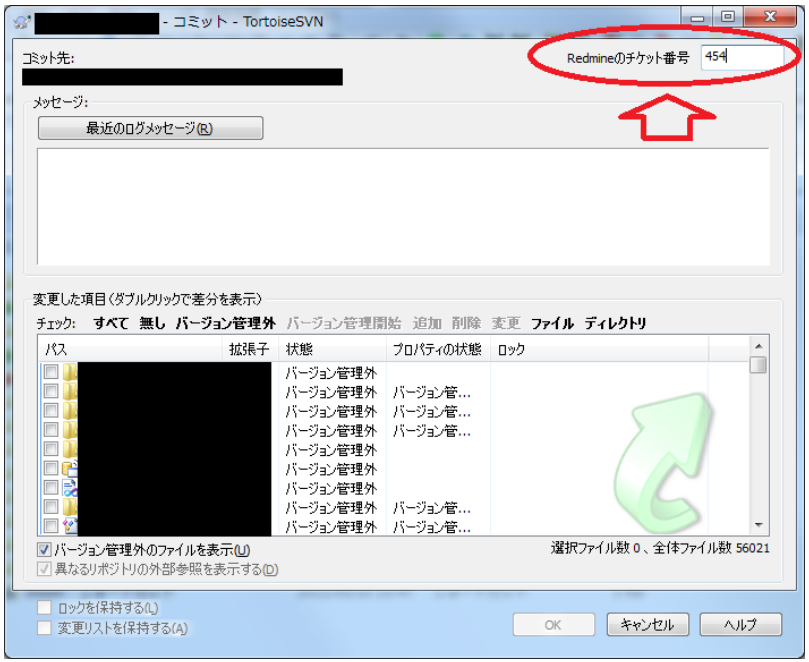

コンバートの種類を選択して
予約を作成

「確認待ち」になったら
確実に修正確認が可能

# RedmineとSubversionの連携 (1/2)

- バグチケットに対して、どういう修正が入ったのか分かりにくい

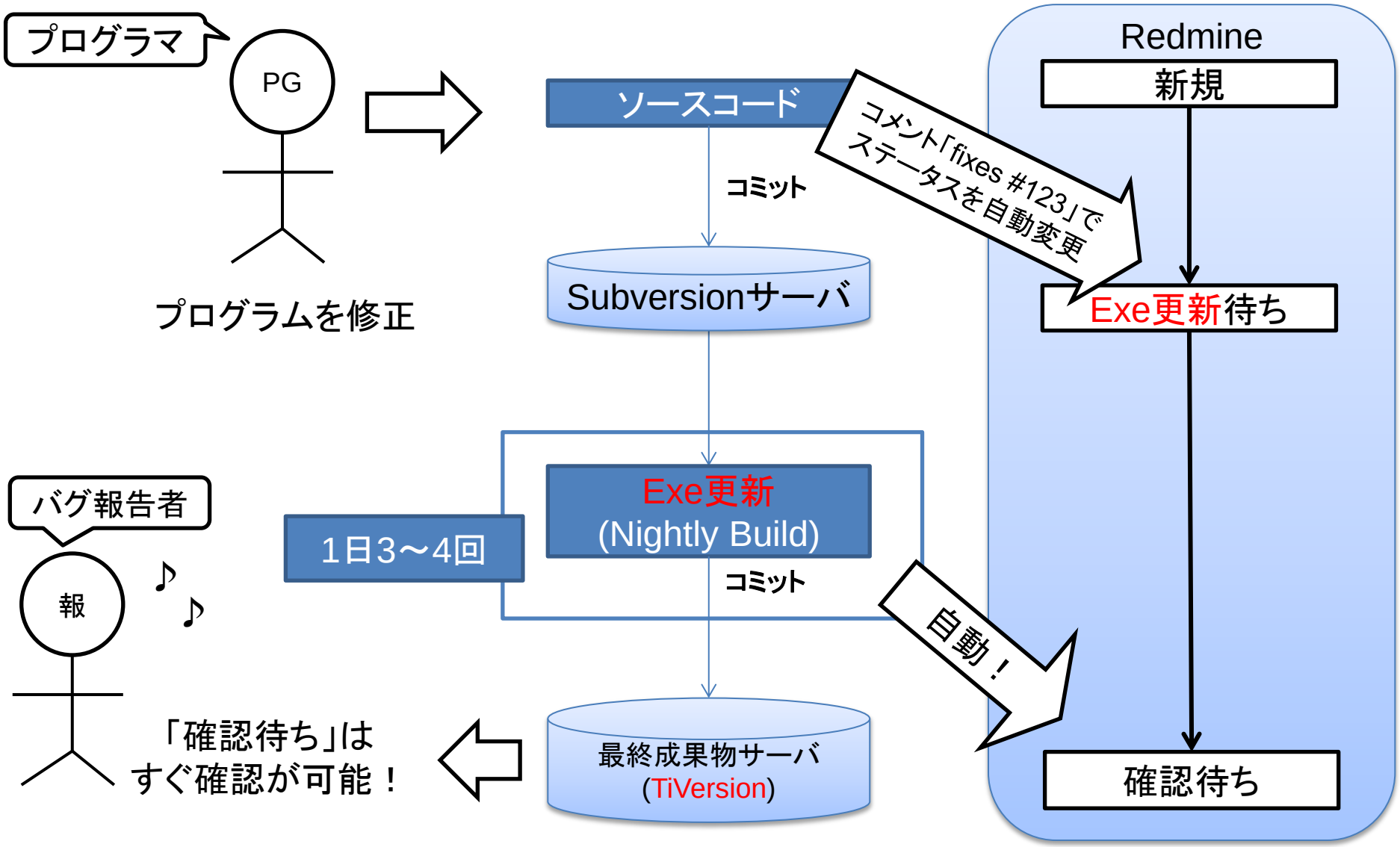
→Subversionのコミットログにチケット番号を入力





バグチケットに対して、どういう変更が入ったのか一目瞭然！

# RedmineとSubversionの連携 (2/2)

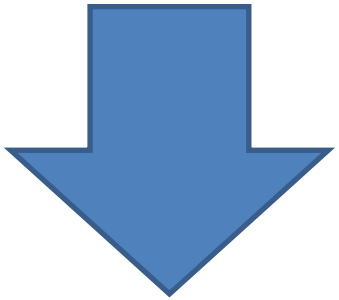


# バグ管理システムの問題と解決

- Excel使いたいよ・・・の声
- バグ報告の問題点
- バグのワークフローの問題点
- 各ツールにプロジェクトの情報が分散

# 各ツールにプロジェクトの情報が分散

- 様々なツールを行き来して、作業が煩雑になる
- バグとタスクの管理を分けると、チケット全体の優先順位が把握できなくなる

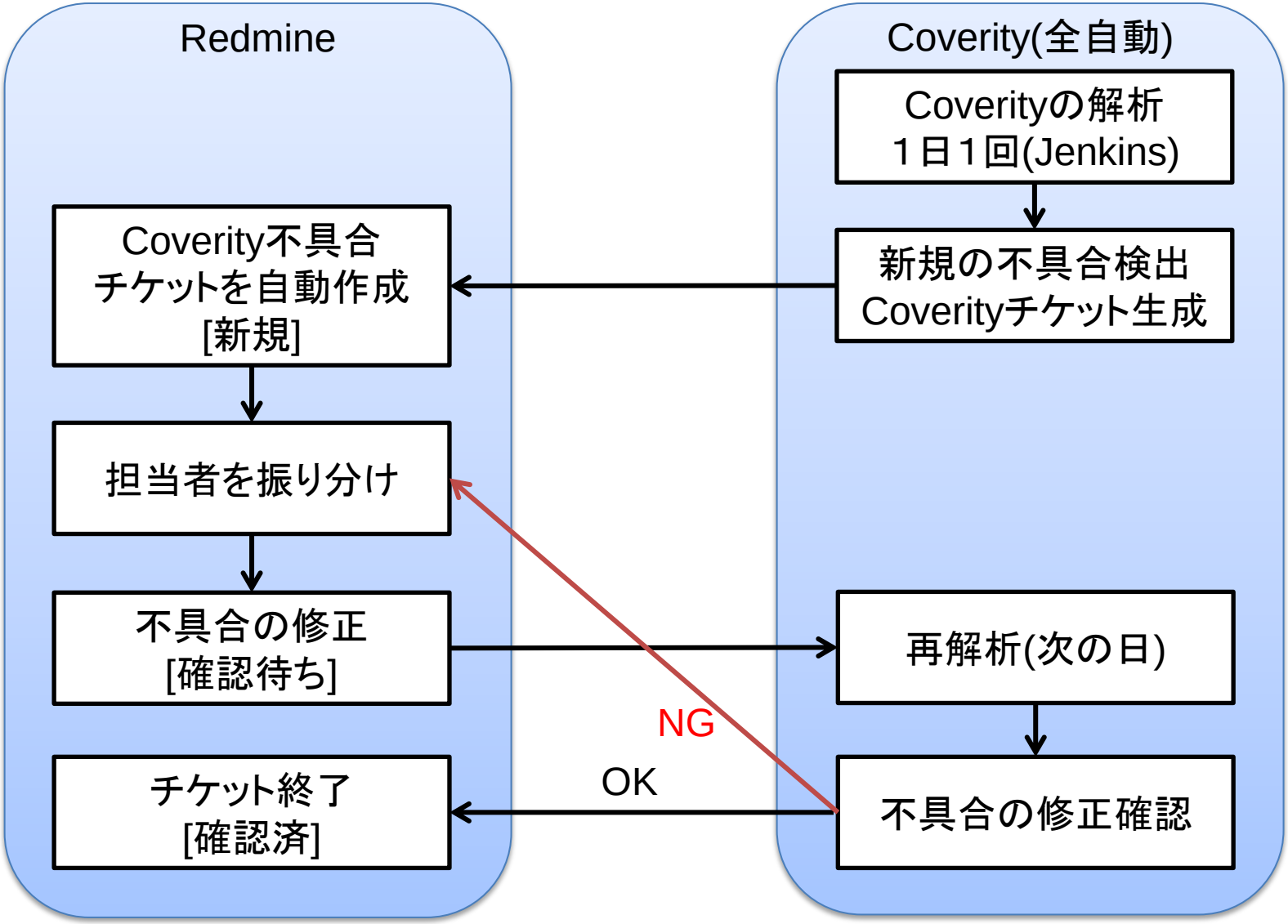


Redmineに情報を集約

# RedmineとCoverityの連携 (1/2)

- メリット
  - バグやタスクと静的コード解析の不具合を一括管理
  - Redmineのワークフローに統一して作業を効率的に
- 連携方法
  - RedmineとCoverityのそれぞれのAPIを使った同期ツール
    - セガゲームス開発技術部の粉川氏制作

# RedmineとCoverityの連携 (2/2)



# そびえ立つバグの山を踏破するための弾丸ワークフロー(バグ管理)のまとめ

- Redmineを使ってバグを一元管理
- バグ報告に必要な情報は自動入力
  - どこでもバグ報告
- バグ管理システムと他のシステムの連携
  - ワークフローを自動化
    - コンバート予約
    - Subversionと連携
  - Redmineにプロジェクトの情報を集約

# 本セッションのまとめ

- 大規模＆高速リリースの秘密は無い
  - 地道な自動化・効率化
- 自動化
  - パイプラインの自動化
  - エラー検出の自動化
  - バグ報告の自動化
  - ワークフローの自動化
  - ためらわず、小さなことから、こつこつと
- 効率化
  - 開発チームの規模や環境・風土に適したツールを導入
  - 情報を1つの場所(Redmine)に集約

# 最後に

私(QAプログラマ)がいないと
龍が如くは毎年出せない！

# 参考資料

- 「10ヵ月でHDゲームを開発する方法 ～龍が如くを支えたテクノロジ～」（CEDEC 2010）
  - 前半の高速デバッグ術の関連情報
    - TiVersion
    - エラー送信
    - オートテスト etc..
  - 概要
    - http://cedec.cesa.or.jp/2010/program/PG/C10_P0064.html
  - 講演資料(CEDiL)
    - https://cedil.cesa.or.jp/cedil_sessions/view/324