自動化で楽してハッピー！ ゲームタイトル開発の現場から

2014/03/07 JaSSTソフトウェアテストシンポジウム JaSST'14 Tokyo 東洋大学白山キャンパス

株式会社アクワイア 開発本部 尾中 竜雄 ＜B2) エンターテイメントとテスト ～時代と共に～＞

目次

はじめに

紹介

自己紹介

尾中 竜雄

- 組み込み系ソフトウェア開発2年
- ゲームタイトル開発約7年
  - 株式会社アクワイアへ2011年入社
  - 内製ツール開発、UI、データベースほか
- （居酒屋調理3ヶ月）

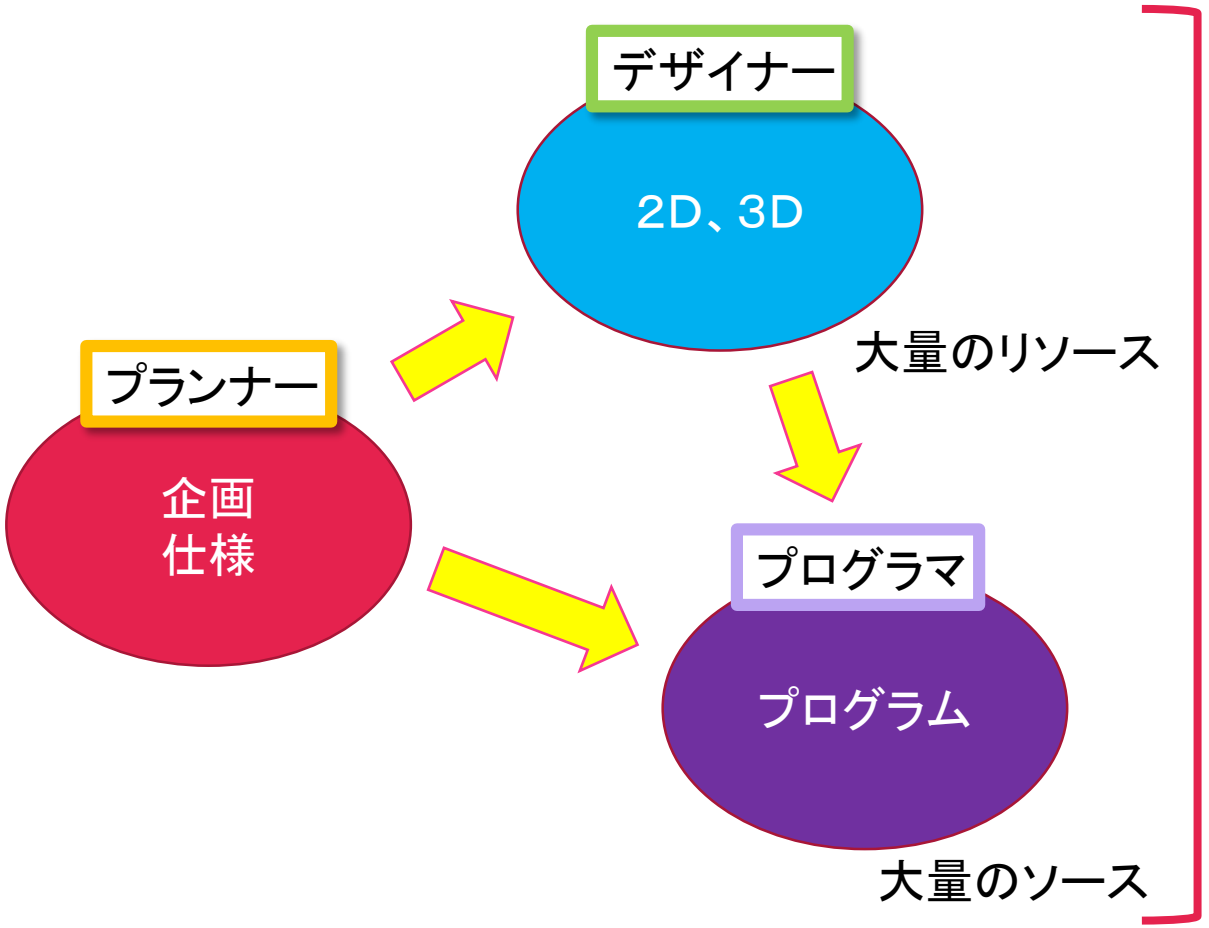
会社紹介

株式会社アクワイア

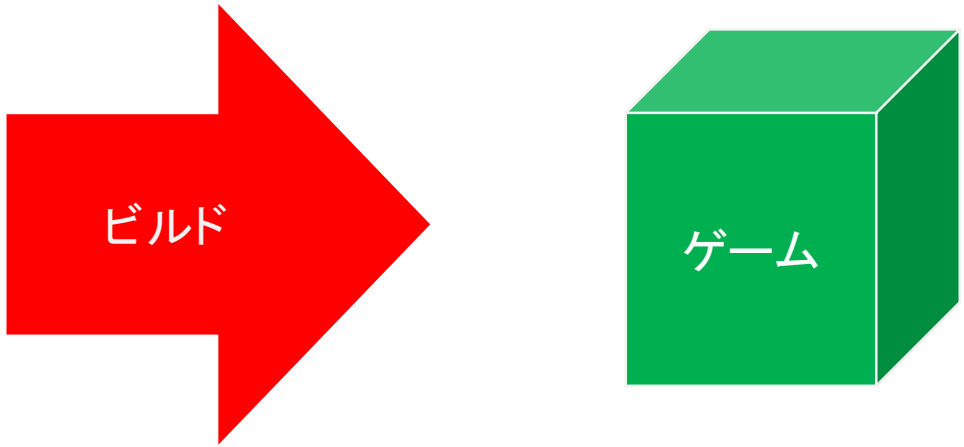
- 開発タイトル
  - 「天誅」「侍道」「忍道」「ととモノ」「勇者のくせになまいきだ」「rain」「AKIBA'S TRIP」「ロード・トゥ・ドラゴン」「ディバインゲート」など
- 創業
  - 平成6年12月6日（今年は20周年）
- 従業員数
  - 84人
- ゲーム以外の事業
  - モーションキャプチャースタジオ運営（両国にモーションが取れるスタジオがあります）
  - デジタルサイネージ系制作（テレビ東京等の株式市況等表示・病院やホールの液晶表示）

ゲーム開発の特徴

ゲーム開発の特徴1

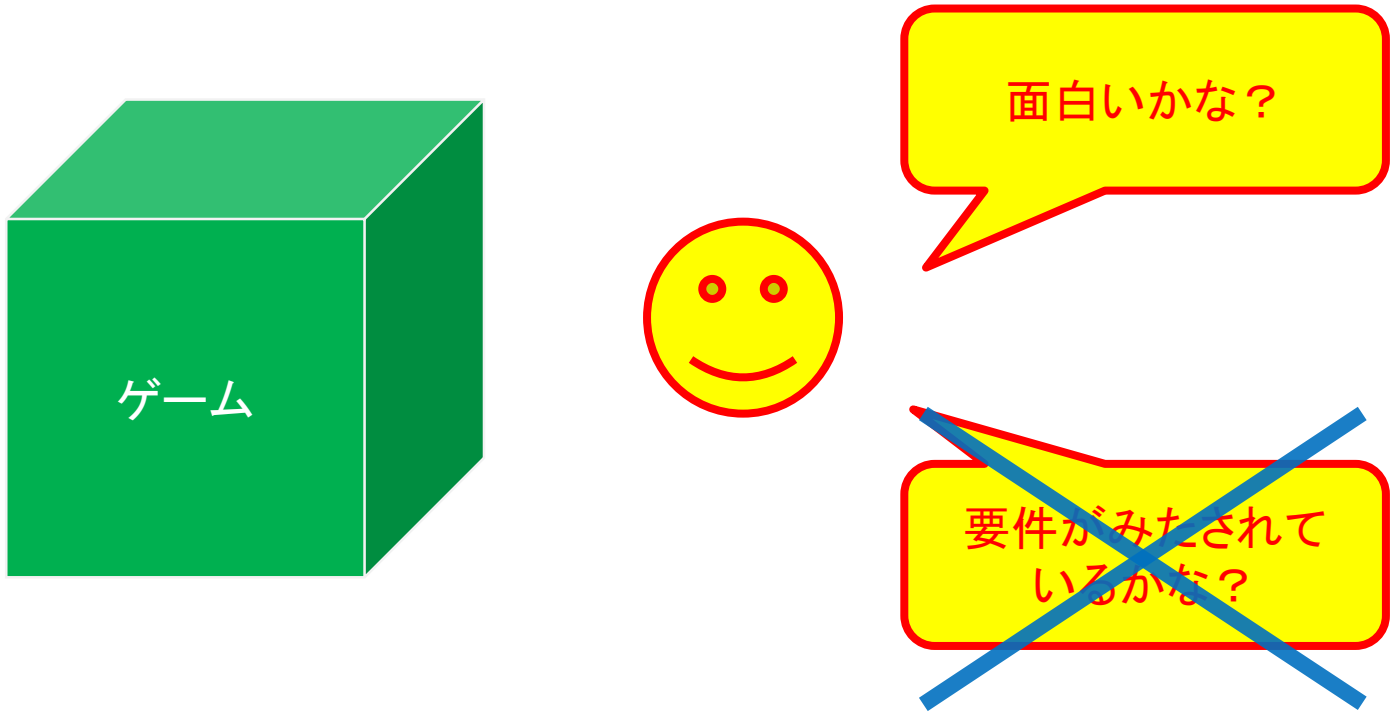


1回のビルド時間が長い！
（数分～数十分）



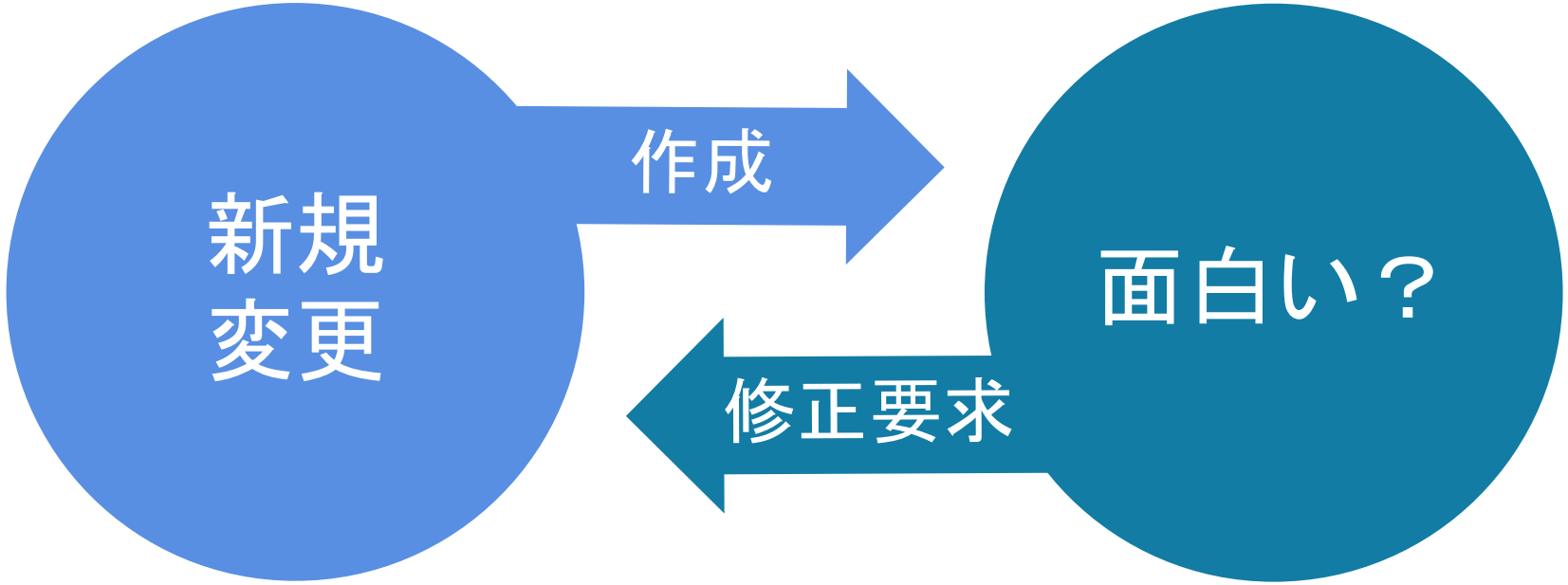
ゲーム開発の特徴2

面白いかが重要！



ゲーム開発の特徴3

いつでも早く正確に
ビルドの内容を確認したい



ゲーム開発の特徴4

ほか重要なこと

- 見た目
  - 動けば中身は重要ではない
- パフォーマンス
  - 60(30)fpsを安定させる
- メモリ
  - 大量のデータを限られた領域にどう読み込むか
    - キャラクター、街、イベント、サウンドなどなど
  - 読み込み時間も制限あり

本題へ

導入

導入の前

導入の前

1. なんとなく学んでみる
   - CEDECやネット上の資料を調べてみる

   →効率化、コスト削減には大きく役に立つ
     - 最速でビルドを作成してくれる
     - 作成ミスをほとんどなくせる

2. 自動化について聞いてみて回った
   - 個人で簡単な自動化をやっている人はちらほら
   - 部署やプロジェクト単位で導入していなかった

   →「自分でやってみれば？」という流れになった

やってみた

まずは時間を見つけて自分のマシンへ

- Jenkinsを導入
  - CI (継続的インテグレーション)ツール
  - 導入がすごく楽
    - 面倒なサーバー構築必要なし
    - Javaで動作する
  - 使いやすい、わかりやすい
    - わからなくても情報が検索しやすい

→自動化の環境はJenkinsさんに決定！

※（社内では「さん」付けされて呼ばれてます）

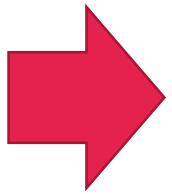
Jenkinsさんすげぇ！！！！

初めての導入

プロジェクトへの導入チャンス

海外版プロジェクトの開始

- 小規模
  - 最大7人で期間は3ヶ月ほど
- 地域数は3つ（言語は7くらい）
  - ビルドの数は3倍
  - やることの殆どは翻訳テキストの対応
- 国内版の修正と同時開発
  - ソースのマージが発生する



ビルドコスト削減には
とても良いプロジェクト

早速使用されていないPCを確保

- CPU　Core2 Duo E8500
- メモリ　4GB
- HDD　500GB
- OS　WinXP

導入した内容

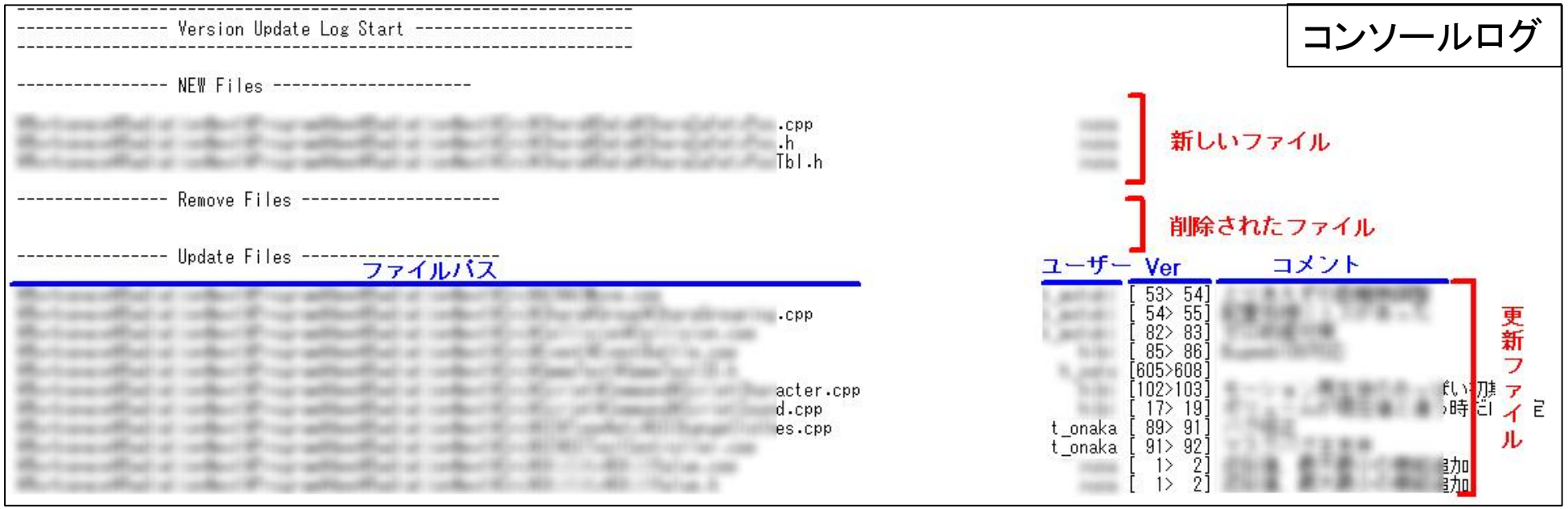
- ビルドチェック
- 社内向けリリース
- 提出用ビルド作成
- 国内版のソースマージ

ビルドチェック

- コードのerrorやwarningチェック用に1時間毎に実行

手順

1. 前回とのファイルの差分を表示
    1. Javascriptで最新ファイルを取得し、パス、Ver、ユーザー、コメントをファイルに出力する。
    2. 前回と今回のファイルを比較し、差分のあるファイルをコンソールログ出力する。
       C++で作成（ Javascriptだと遅いため）



ビルドチェック

2. ビルドエラーがないかのチェック
    1. バッチコマンドでビルドし、ビルドログをファイルに保存
    2. ログファイルから「warning」と「error」を含む行をコンソールログに表示する。
       C++で作成。
       「error」が発見されたらmain関数をreturnを-1にする（ERRORLEVELの値に）
    3. JenkinsのバッチコマンドでERRORLEVELが0でない時は直ちに終了
       （失敗の状態のままで終了する）
3. 失敗があった場合はメールを送信する

バッチコマンド

```
echo _
echo ****************************************
echo Build : Win32 Release
echo ****************************************
echo _
start /wait /min c:                Build_Win32Relese.bat     — 1
c:\jenkins\FindError.exe C:                Win32Release.log — 2
IF not %ERRORLEVEL% == 0 goto END  ———————————————————— 3
```

コンソールログ

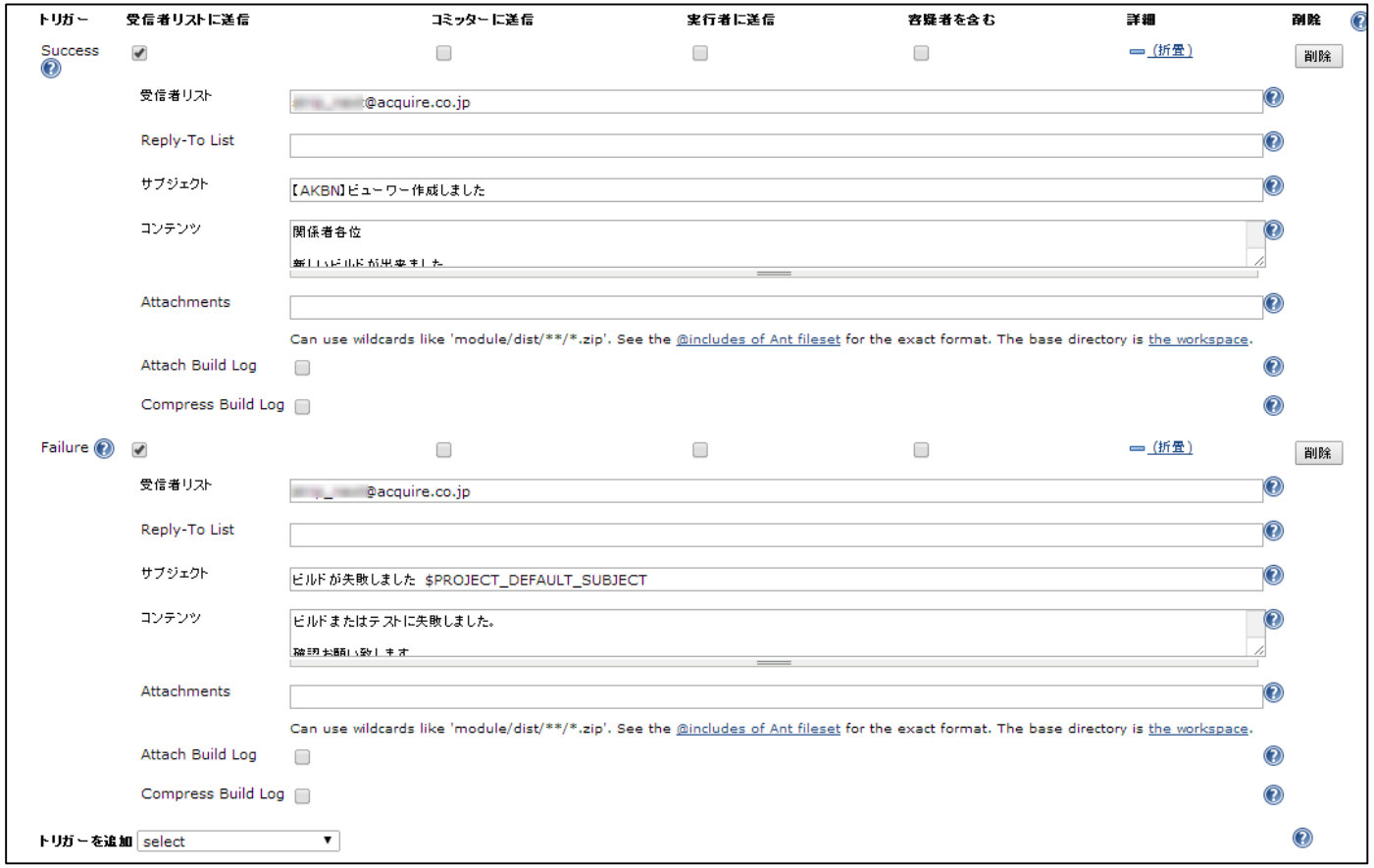
```
_
****************************************
Build : Win32 Release
****************************************

== Error or Warning Log Start. ============================================================
                                .cpp(192): warning C4100: 'bUseMainMemory' : 引数は関数の本体部で
                                .cpp(602): warning C4189: 'nPhase' : ローカル変数が初期化さ
                                .cpp(84): warning C4100: 'bUseMainMemory' : 引数
== Error or Warning Log End. ==============================================================
```

社内向けリリース

- 社内チェック用
- 毎朝作成(+手動)

手順

1. Release版をビルドしチェックイン
    1. 最新のファイルを取得し、ビルド
        - ここまではビルドチェックと同じ
    2. 作成したファイルをチェックインする
        - Javascriptにてチェックイン。
        - コメントはVersionと日時
2. メールを送信する
    - 成功と失敗でメールの内容を変更。Jenkinsの拡張E-mail通知のプラグインを使用

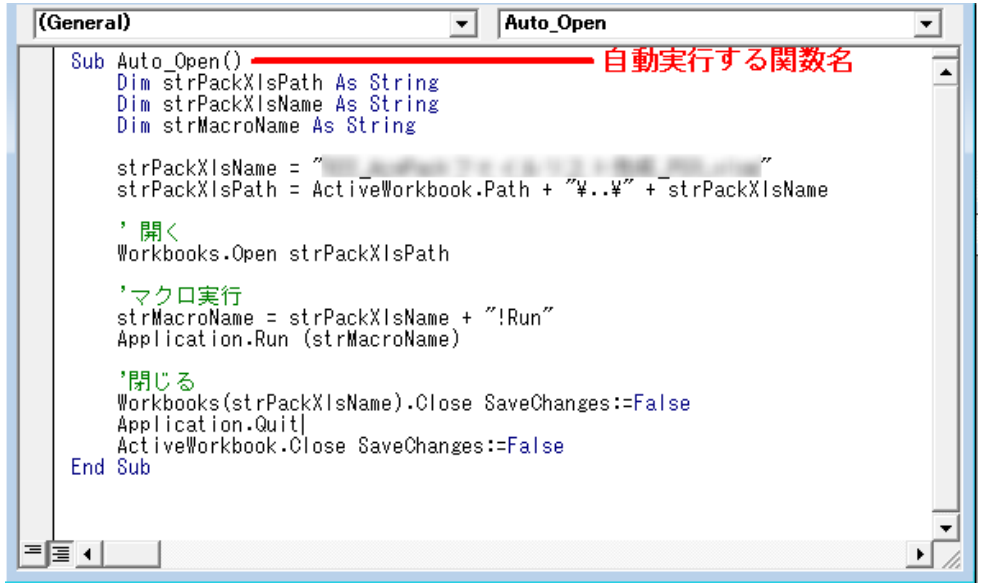


提出用ビルド作成

- 週に数回提出
- 手順が多く、時間がかかる
  - すべてを作成するのに1時間程度（手動では1時間半ほど）
  - ヒューマンエラーが発生しやすい
- 手順

1. プログラムをビルド
   - 社内向けリリースと同じ手順
2. リソースファイルをパッケージ化
   - パッケージ化はもともとあり、エクセルで起動していたため、自動で実行し終了するエクセルのマクロを作成。
3. 提出用ファイルの作成
   - 規定に従ったファイルを作成する。
   - コンソールコマンドが使用できない箇所があったがSikuliという画像認識できるスクリプトツールで実装

エクセルの自動実行マクロ



国内版のソースマージ

- 数週間に1回（トータルで5，6回実行）
- 国内版のパッチ取り込み

手順

1. 前回のマージから更新のあったファイルを検出し、テキストファイルへ保存
   - C++で作成（最新版取得の出力をログからファイル保存へ変更）
2. 1で検出したファイルにマージを実行
   - Kdiff3というフリーツールを使用
3. マージの失敗があったら怖いので
   自分ですべて確認しながら手動でチェックイン

導入結果

導入結果

- ビルドエラーの早期発見
  - ビルドしないでチェックインは多いです
    - 2プラットフォーム×3コンフィグレーション=6回もやらない
  - 普段使用しないMasterビルドのエラー通知が助かった
- すぐチェックできる体制になった
  - 毎朝最新のビルドがあるのは便利！！
  - ほしい時にだれでも作成できる
    - PGにビルドのお願いをしなくても良くなった
- コスト大幅削減ができた（次ページ）

コスト

- 自動化導入コスト(20～35時間 : 開発の初期に集中)※
  - 初めてJenkinsを導入に半日
  - ビルドチェックの自動化が出来るまでは2日程度
  - ほかのものを作るには1つにつき数時間
- 自動化なしのコスト(70～100時間 : 開発の後期に集中) ※
  - 社内リリース作成:[10分/回] × [7回/週] × [12週(3ヶ月)] = 840分 = 14時間～
  - 提出用ビルド作成:[90分/回] × [2回/週] × [12週(3ヶ月)] = 2160分 = 36時間～
  - 国内版からのマージ:[120分/回] × [5回/週] = 600分 = 10時間～
  - ヒューマンエラーによる作りなおし:多数 ← ここがなくなるのは大きい！

短期間プロジェクトで
35時間以上コスト削減！

※大体です

追加した自動化

更に効率化

追加した自動化

- 追加のチャンス到来！！「AKIBA'S TRIP2」のプロジェクトへ参加
  - 期間が長いプロジェクトなのでなにか効率的なものを入れたくなった

新たに2つを入れてみた

- 起動チェック
- メモリチェック

起動チェック

作られたビルドが起動できるかのチェック

手順

1. 社内向けリリースの最後に追加して作成
    1. Win版起動
        - コンソールコマンドに引数付きで起動し
          起動チェックモードであるフラグを立てる
    2. 各機能が初期化したらゲームを自動終了させる
        - 例外処理に引っかかったら、main関数を-1でreturn
    3. エラー終了していたらチェックインをキャンセル

メモリチェック

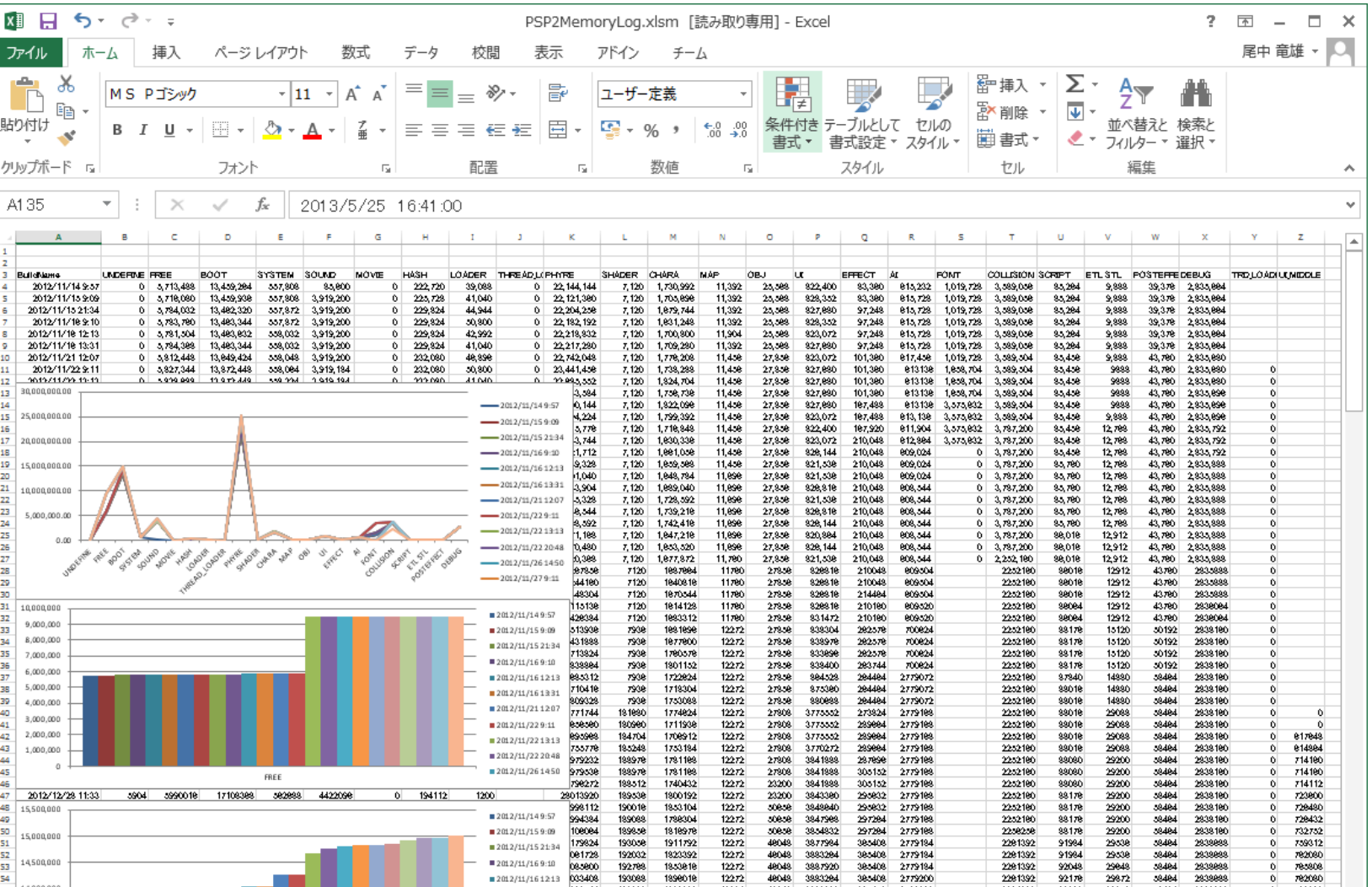
起動チェックの拡張

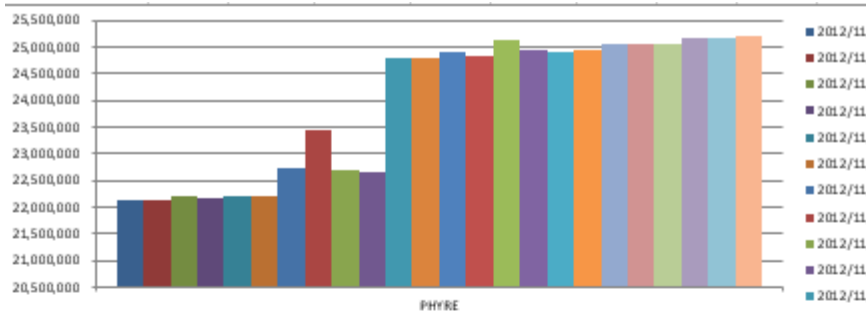
Game Mainに入った時のメモリ使用量のチェック

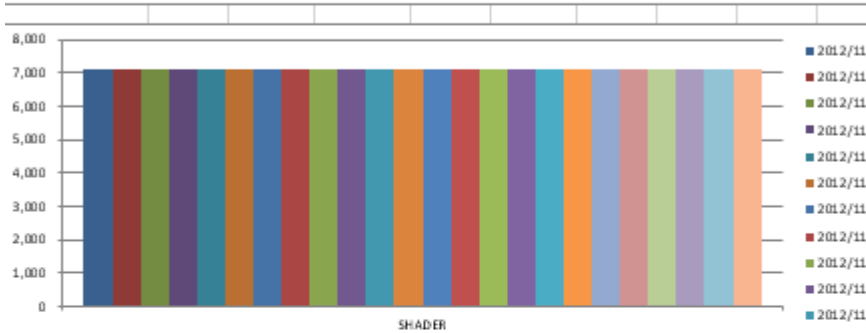
手順

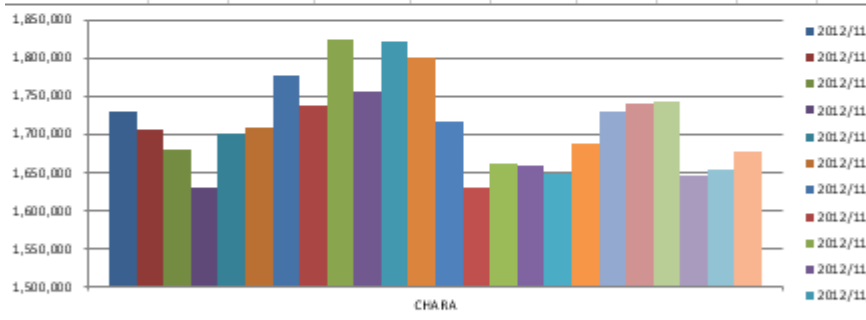
1. 起動チェック中後、メモリ使用量のログ出力
   - ログ出力はすでにあるものを使用
   - Game Mainですべてロードし終わったあとに実行
2. ログ出力から使用量を計測し、エクセルにデータを追加する
   1. ログからメモリ使用量だけを抽出したファイルを作成（C++で作成）
   2. 1で作ったファイルからデータを表に追加するエクセルのマクロを作成

メモリチェック


PSP2MemoryLog.xlsm [読み取り専用] - Excel
ファイル
ホーム
挿入
ページ レイアウト
数式
データ
校閲
表示
アドイン
チーム
尾中 竜雄
貼り付け
MS Pゴシック
11
ユーザー定義
条件付き書式
テーブルとして書式設定
セルのスタイル
挿入
削除
書式
並べ替えとフィルター
検索と選択
クリップボード
フォント
配置
数値
スタイル
セル
編集
A135
2013/5/25 16:41:00


PHYRE


SHADER


CHARA

追加した自動化の結果

追加した自動化の結果

- 全然効果ありませんでした・・・
  - 起動チェック
    - 起動できない事態は殆ど起こらない
  - メモリチェック
    - そもそも決まりごとがしっかりしていなかった
      - 問題がある度に修正されていた
      - グラフにしただけで予測していなかった

→ゲーム自体に影響する自動化はもっとよく検討してから実装しよう

普及

自分で使うだけではもったいない

普及に向けての活動

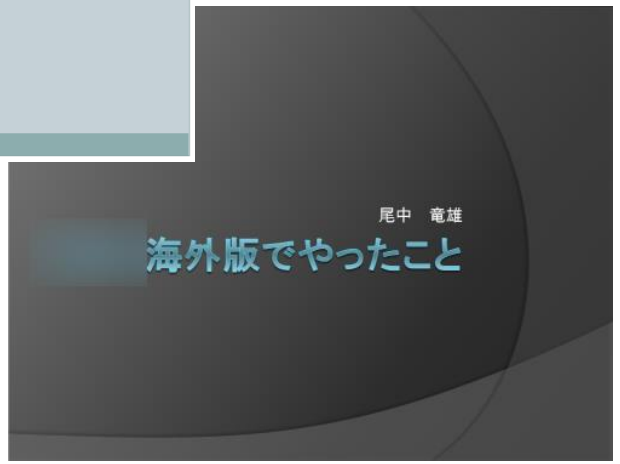
- 社内の勉強会にて発表
- 自ら場所を用意して社内で発表
- マニュアルを作っておく
- キーマンに個人的に接触
- 普及結果

社内の勉強会にて発表

社内勉強会（定期開催）

- CEDECのレポートの一部として発表
  - ほかの講演も含む

- 導入したプロジェクトの紹介
  - 自動化のほかにも様々なチャレンジを紹介

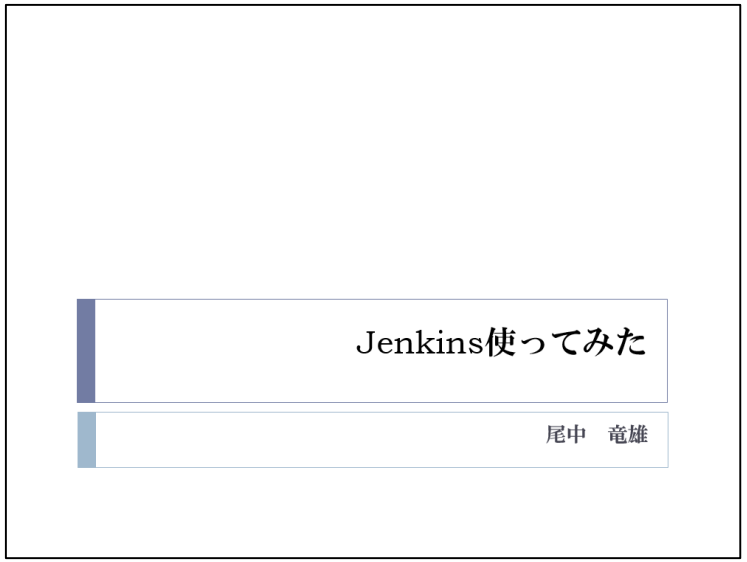


→自動化って実現できたら便利そうだねぇ～

自ら場所を用意して社内で発表

自動化にフォーカスを当てて発表

- 実績を交えて自動化の素晴らしさを発表
- Jenkinsの導入はすごい簡単なことをアピール

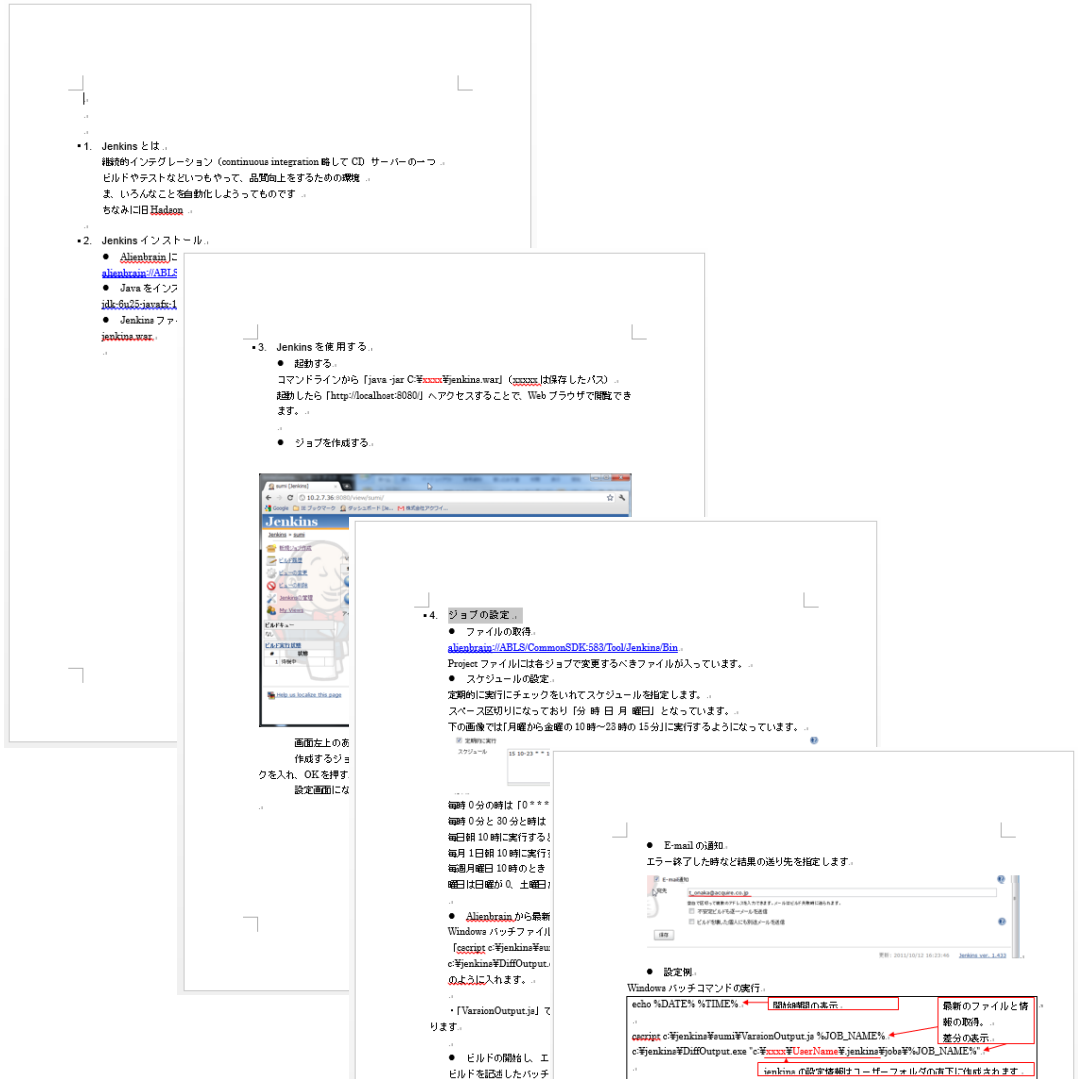


→自動化出来るならやってみたいかなぁ～

マニュアルを作っておく

- 作りこむのは面倒なので、Jenkins導入のみ
  - 結果4ページ
    - Jenkinsとは
    - インストール
    - 起動方法
    - ジョブ作成方法
- 導入マニュアルがあることをメールで通知する

→機会があれば導入してもいいかなぁ～



キーマンに個人的に接触

- 実際に効果の有りそうなプロジェクトの人に直接話しかける
  - 導入には8割くらい手伝う感じでとにかくスタート
  - 1週間位続けて手伝う

→便利で簡単を実感して普及の仲間に！

普及結果

- 私の関わったプロジェクトすべてで自動化へ
  - 3プロジェクトで導入
    - やった内容はほとんど同じ
- それ以外では
  - 2プロジェクトで導入
  - 1つはプラットフォームが違うため独自に色々組んでくれました

→便利なので自動化は今後も入れるべき！

伝えたいこと

伝えたいこと(導入)

自動化の導入は簡単

- Jenkinsを入れてみよう
- ほとんどのことはフリーツールで出来る
  - 情報収集が重要
    - 英語で検索
    - いろんなことに参加
- 自作ツールは単純なことばかり
  - データをログ出力→ログから抽出(比較)→結果のログの出力

→自分を助けることから始めてみましょう

伝えたいこと(普及)

しかし普及は結構大変

- 継続して普及活動しないとならない
  - すぐに「導入しよう！」とはなりにくい
- 自動化の信頼を勝ち取るために
  - 結果を出す
    - 自分でやるのが手っ取り早いです
  - 自動化それ自体でミスを起こさない
    - ちゃんと動くまで、自分で十分に試用してから広める

→気長に根気よく活動しましょう

ツール紹介

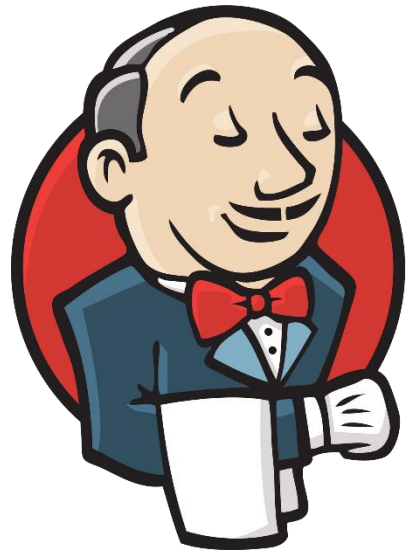
実際に使用したツール

ツール紹介

- Jenkins
- Autoit
- Sikuli
- kdiff3
- 自作ツール
  - ファイル差分検出
  - ビルドログの抽出
  - マージファイルの検出

JENKINS

- http://jenkins-ci.org/
  - 日本語 https://wiki.jenkins-ci.org/display/JA/Jenkins
- CIツール
- 導入がかなり簡単です
  - Javaの環境があれば「java -jar Jenkins.war」を行うだけ
- 実装にはバッチコマンドを使用
- Email-ext plugin
  - メール送信のプラグイン。成功や失敗などの内容を書き換えられる



AUTOIT

- http://www.autoitscript.com/site/autoit/
- GUI自動操作が出来るスクリプト。
- エラーダイアログが出てしまう際にボタンを押してくれるために使用

SIKULI

- http://www.sikuli.org/
- 画像認識できるスクリプトツール
  - キャプチャーしながらスクリプトを書けて面白い
- Javaの環境があれば動く

→CUIコマンドが出来ないものも自動化が可能になります

KDIFF3

- http://kdiff3.sourceforge.net/
- テキストの差分解析ツール

- 自動マージツールの機能が優秀である
  - 7，8種類のツールを調べた中で一番良かった

自作ツール

- ファイル差分検出
  - 2～3万ファイルのログを検索し、バージョンの差のあるファイルをログ出力する
  - C++で作成
- ビルドログの抽出
  - ビルドされたログファイルからwarningとerrorをログ出力
  - errorがある場合はエラー終了する
  - C++で作成
- マージファイルの検出
  - ほぼファイル差分検出と同じ
  - C++で作成

以上ありがとうございました

http://www.acquire.co.jp/