JaSST'06

J2EEアプリケーション開発における ビルド／テスト自動化への実践的アプローチ

Software Engineering, IBM
ITA北嵐 直樹

アジェンダ

- ビルド／テストの自動化とは
- UTレベルのリグレッション・テストとは
- 適用プロジェクト概要
- 使用したオープンソース
- 単体テストの完了基準
- ビルド／テストの実行環境
- 実施タスク一覧
- ビルド／テスト自動化の効果について
- 適用のツボ

ビルド／テストの自動化とは

サーバー上の常駐モジュールが、リポジトリ内のソースコードを対象に、自動的にビルドとテストを実施

- 継続的インテグレーションの実施
- UTレベルのリグレッション・テストの実施
- メトリクスの収集
- テスト結果のレポート化とWebサーバーでの公開

モジュール品質の見える化

UTレベルのリグレッション・テストとは

- JUnitテストケースをサーバー上で自動実行し、常時、UTのリグレッション・テストを実施すること
  - キャプチャー／プレイバックツールと違って、別途スクリプトを作成／維持する必要はない
  - 開発時に作成したJUnitテストケースを、そのままサーバーで実行
- 開発の効率化よりは、品質の向上に貢献する
  - デグレードの防止
  - モジュール修正（リファクタリング）の促進

適用プロジェクト概要

- 大規模J2EE基幹システムの基盤フレームワーク開発
  - J2EE 1.3/EJB2.1、Servlet 2.3、Struts 1.1、SQLJ、JMS
- プラットフォーム
  - IBM z/OS v1.6, AIX 5L
- ミドルウェア
  - IBM WebSphere Application Server v6, MQ v5, DB2 v8, SVF v6
- プロジェクト期間
  - 要件定義 ～ サービスインまで約4年
- 開発者数
  - 基盤フレームワークの開発で最大100名以上
- Javaクラス数
  - 1000クラス以上

使用したオープンソース（1）

- CVS
  - 開発クライアントは eclipse 3.0 （Rational Software Architect）
- Ant 1.6.2
  - Javaによるタスク自動化のデファクト・ツール
- Maven 1.02
  - ビルド／テスト／デプロイ／レポート生成の自動化ツール
- CruiseControl 2.3
  - リポジトリを監視して、ビルド・タスクの実行を自動化
- xUnit
  - JUnit 3.6, DjUnit 0.7.2, DbUnit 2.1, Cactus 1.6, Struts TestCase for JUnit 2.1

使用したオープンソース(2)

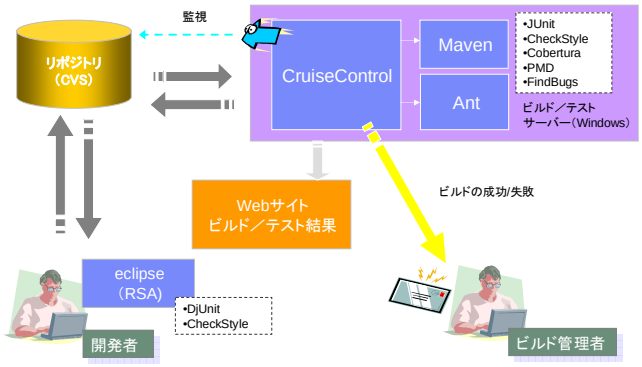
- CheckStyle 3.6
  - 静的コードインスペクション・ツール
- Cobertura 1.6
  - 実行時のコードカバレッジを取得
- PMD、FindBugs
  - Maven Plug-inによる コードインスペクションを実施

単体テストの完了基準

- カバレッジ基準の達成
  - DjUnit、Coberturaによるカバレッジの測定と基準の達成
- CheckStyleによる静的コードインスペクションの実施
  - プロジェクトのカスタマイズルールによるソース・フォーマットの検証
- JUnitによるテストケースの作成とサーバでの自動実行の成功
  - 常時、サーバー側でのJUnitの自動テストにパスすることが必要

- **OSS利用による全開発者への統一されたCD/UT環境の提供**
- **サーバー／クライアント上で同じツールを使用し、サーバー上で単体テストの受入確認を実施**

ビルド／テストの実行環境



実施タスク一覧

- MavenとAntの使い分け方針
  - FWのデプロイ用にビルド用Antファイルを準備しているため、これを使用(二重メンテとなるため、Mavenのビルド機能は使用しない)
  - Mavenのプラグインが正常に動作しない場合、同等機能のAntタスクで代替
  - Maven提供のプラグインでは機能が不足する場合、Mavenプラグインをカスタマイズせずにantタスクを追加
- 継続的ビルド(jarの更新)は更新の度に実施
- サーバーでの自動ビルド／テストは実行時の負荷が高いため、2回/日で実施

タスク名	Maven/Ant	実行間隔
リポジトリからのチェックアウト	Ant	更新の度
ビルド(コンパイル,jar作成)	Ant	更新の度
カバレッジレポート	Ant	2回/日
単体テスト(JUnit)	Ant	2回/日
コードインスペクション(CheckStyle)	Maven	2回/日
CVS 変更ログ一覧作成	Ant	2回/日
PMD(コードインスペクション)	Maven	2回/日
FindBugs(コードインスペクション)	Maven	2回/日
Javadoc生成	Ant	2回/日
ソースコード公開(Xref)	Maven	2回/日

ビルド／テスト自動化の効果について

実施したこと

- 継続的ビルド
- JUnitテストケースによる自動リグレッションテスト
- 各種メトリクスの自動収集
- 開発環境とサーバー環境でのツールの統一
- テスト結果レポートの公開

プロジェクトでの効果

- リリース間際のビルドエラーの防止
- 最新のJar配布による、ビルド環境構築の効率化
- デグレードの防止
- リファクタリングの促進
- 単体テスト作業の検証の効率化
- モジュール品質の見える化

適用のツボ(1)

- **管理者はビルド／テスト結果の結果を常時監視する必要がある**
  - ビルド／テストはよく失敗する
  - ちゃんと監視しておかないと、いつの間にかガラクタになっているかも
- **エラーはすぐに修正する。後回しにしない**
  - すぐに修正すれば工数は少ない。
  - 自発的に修正してくれる人ばかりではないので、管理者は絶大な権限を持って依頼すること
- **ビルド／テストの成功／失敗は、PULLではなくPUSHすることが大事**
  - 常時、結果をチェックするのはたいへん。テスト結果(成功／失敗)はメールで送信する
  - 失敗だけでなく成功も通知すること
  - CruiseControlを使えば、XMLの設定だけなので簡単
- **開発者を巻き込まないと、うまく運用できない**
  - 管理者一人で孤軍奮闘するのはつらい・・
  - エラーは放置せず、サーバーで発生したエラーも即時に修正する必要があることを、プロジェクトのルールにしよう
  - テスト結果をWebサーバーで公開し、Publicなものとする

適用のツボ（2）

- **ビルドの自動化（継続的ビルド）は、比較的簡単**
  - リポジトリの監視にはCruiseControlを利用
  - ビルドにはAnt、Mavenを利用
- **自動テストは全クラスを対象にしなければ効果が出ないわけではない**
  - 始めから全クラスを自動テスト対象にするのはたいへん
  - 一部の重要なモジュールだけでも、自動テストによるリグレッションテスト環境を整備することは、モジュール品質の点で効果あり
  - チームのスキルレベル、モジュールの重要性、JUnitとの相性などを考慮して
- **JUnitテストの自動化は難しい。。。**
  - 単純にUTが実施できないモジュールの扱い（画面系、EJB等々）
  - きちんとした標準化なしでは、サーバーではテストケースがうまく動かないことが多い
  - サーバー側での実施確認は、ビルド管理者になりがちなのでクラス数が増えるとたいへん

適用のツボ（3）- JUnitテストケースについて -

- **テストケース・クラスを標準化された形で作成してもらうことが必要**
  - プログラマのスキルに依存してしまうのが現実
  - 教育としっかりとした標準化が必要
  - テストケースの作成に慣れてしまえば、開発者の工数は変わらない
- **開発者の数が多いとテストケースの質をキープするのが難しくなる**
  - 50名以内が1つの目安
- **CD/UTの期間が短いと、教育している時間がない**
  - JUnit未経験者ばかりなら、CD/UT期間で2～3ヶ月は欲しいところ
  - JUnitに慣れた開発者なら、立ち上げ期間は短くて済む
- **短期間なプロジェクトや大規模プロジェクトでの適用はしきいが高い**
  - 開発者の成熟度に因る部分が多いので、一概に判断はできない