Vol.2011-ARC-195 No.21
Vol.2011-OS-117 No.21
2011/4/14

IaaS 上での Continuous Integration による運用支援フレームワークの提案

株式会社 PFU　原田暢彦
株式会社 PFU　山元隆弘
株式会社富士通研究所　今井祐二
株式会社富士通研究所　福井恵右

ハードウェアおよびソフトウェアの組合せ結果として完成した業務システムは，運用開始後も機能追加や故障／不具合解消のために常に更新され続ける．クラウドの特性を活かして，更新される業務システムを自動ビルド&テストで全てのテストシナリオを実行することで，常に要求仕様を満足する機能提供を保証するシステム運用ツールを提案し，試作制作を通じて得た成果とシステム開発に課せられる課題を報告する．

Operation framework for cloud application systems based on IaaS with Continuous Integrations

PFU LIMITED Harada Nobuhiko
PFU LIMITED Yamamoto Takahiro
FUJITSU LABORATORIES LTD. Imai Yuji
FUJITSU LABORATORIES LTD. Fukui Keisuke

Application systems, that are composed of a number of hardware parts and software modules, must be continuously modified to repaired from failures and kept updated for individual components remodeling. We propose the framework for continuous operation of ICT systems that realized automatic building, deployment and regression tests of application systems on top of plentiful virtual resources from IaaS clouds. With the framework, sufficiency of ICT system services can be verified every time before update.

1. はじめに

ITLCM(IT ライフサイクルマネジメント)は，ICT システムを企画／開発／運用から運転停止に至るまで，機能／品質／コストを管理するための枠組である．今日の典型的な ITLCM では，企画から最初のサービス開始までを初期開発フェーズに，それ以降を運用フェーズと分け，それぞれに作業品質を向上させる体系に基づいて管理される．初期開発フェーズは PMBOK や CMMI，運用フェーズでは ITIL が著名である．

初期開発フェーズに対しては，開発体系の元での作業を具体的にサポートする手法が多数存在する．近年は，Agile 開発，CI(Continuous Integration)，TDD(Test Driven Development)が注目されている．一方，運用フェーズでは ITIL のもとで具体的な作業を支援する手法の整備が，開発手法に比べて遅れている．

本報告では，運用フェーズを ITIL の体系にもとづいて支援する手法 Durable ICT を提案する．Durable ICT は開発フェーズの手法 CI と TDD を，運用フェーズに拡大適用したものである．運用フェーズのすべての変更において，クラウドを用いた CI により，検証系の構築とリグレッションテストを実施する．変更実施後の本番系と完全に同一となる検証系のシステムを，初期開発完了時の受入テストで検証することでサービス品質を継続的に保証する．

2 章では運用フェーズにおける課題と初期開発フェーズにおける背景技術を説明する．3 章で提案手法 Durable ICT を詳述する．4 章では提案のプロトタイプ実装について述べ，提案のねらいの実現性を議論する．

2. 課題と背景技術

2.1 課題

(1) ITLCM における初期開発と運用の溝

現在の ICT システムにおいては，IT ライフサイクルがサービス開始を境界に初期開発フェーズと運用フェーズに分けて管理されるケースが多い．両者を担当する組織を別にした場合，サービス開始時の受入作業で管理が移管される．移管後の品質劣化は，障害対応や保守開発などの形で運用組織が管理責任を担う．

受入時の品質に関する議論は開発組織よってプロダクト品質の保証という形でなされる．一方，運用組織はユーザへ供給するサービスの品質に責任を負う．システム運転期間中にサービス品質は，プロダクト品質以外の様々な要因で劣化する．受入の時点で，保守開発に必要な情報や外部環境の変化に対応するための機構を事前に考慮することは難しく，運用コスト負担の増大の一因となっている．

一方で，初期開発と運用の作業には多くの類似性を見いだすことができる．初期開発フェーズでのシステムテストでは，統合用の実行環境にシステムをインストールし

Vol.2011-ARC-195 No.21
Vol.2011-OS-117 No.21
2011/4/14

た後に各種の設定が実施される．これは，運用フェーズで稼働する本番環境の構築と同じ作業である．同様に，運用開始後の障害監視と初期開発フェーズの統合テストは，設計機能が正常に動作していることを判断する作業としては同一のものである．また，バグ回避のための保守開発と初期開発フェーズでのバグ修正は，障害要因の特定とプログラム改良と捉えることで同一のものとみなすことができる．しかし，初期開発フェーズで開発担当者が容易に実施していた作業でも，サービス開始後に運用担当者が実施することは類似であるとしても著しく困難である．これが ITLCM の初期開発と運用の間の深い溝となっている．

(2) リスクコントロールの観点での初期開発と運用の違い

ICT システムが高機能化／複雑化していく現状において，開発の効率化を図るためには，システムの構成要素となるハードウェア／ソフトウェアの一部または全てを外部から調達することになる．これら外部調達のモジュールはそれぞれサポート期間が異なり，個別に修正パッチが提供される．

初期開発には開発期限が設定されており，この期限を遵守する限り，途中でリグレッションを発生させることができる．そのため，パッチ提供などの外部要因が発生した場合でも柔軟に対応が可能である．

しかし，サービス開始を境にシステムの変更は著しく困難になる．修正モジュールの品質は提供元で確認された範囲までしか保証されないため，修正により他のどの部分にどのような影響が波及するかという予測が困難である．しかも，システムの停止は直接ユーザに影響を与える．

このようなことから，一旦運用に入ったシステムは，容易に変更が行えない状況となり時の経過とともに「塩漬けのシステム」となってしまう．

2.2 背景技術

(1) IT ライフサイクルで考える初期開発と運用の関連

初期開発は今日でもシステム開発のＶ字モデルで表されるような設計と結果検証の対応付けをして活動が行われるのが一般的である．このため，システム／サービスに対する要件は事業／顧客から与えられ，受入テストによって要求仕様／機能要件の充足性の確認が行われることとなる．

運用は近年では IT サービス運用のベストプラクティスとなる ITIL の提唱するサービス・ライフサイクルに従って活動を捉えるのが一般化しつつある．本論文では，この ITIL の考え方を中心に運用の活動を捉えている．ITIL では，事業／顧客から与えられたサービス要件をもとに IT サービス戦略を立案し(サービスストラテジ [1])，その実現方法を設計し(サービスデザイン [2])，システム開発プロジェクトの成果物を統合しつつ本番環境へ投入される(サービストランジション [3])と定義している．本番環境に投入されたシステム／サービスは利用者との対話や日々のオペレーションによってサービス提供が行われ(サービスオペレーション [4])，サービス・ライフサイクルを通じて行われる全ての活動について監査し提供価値を最大化する取り組み(継続的サービス改善 [5])が行われることとなる．

図 1 はシステム開発の V 字モデルと ITIL が定義しているサービス・ライフサイクルとの対応を簡略化して表したものである．

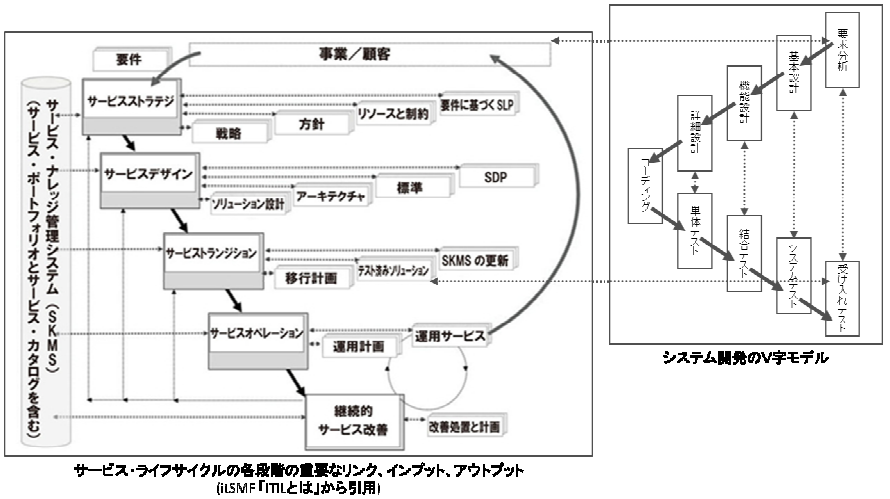

図 1　システム開発のＶ字モデルとサービス・ライフサイクルの対応 [6][7]

この図の中からサービス・ライフサイクルには，２つの PDCA サイクルが存在することが読み取れる．１つは，開発プロジェクトとして活動する要件定義からサービス提供開始までの PDCA サイクルである．事業／顧客の要件が初期開発では要求分析，運用ではサービスストラテジのインプットとなり，それぞれが並行して活動しながらサービストランジションで初期開発の成果物が運用と統合され本番環境へ投入されるまでのサイクルとなっている．

もう１つは，本番稼働中のシステムがサービスオペレーションで日々のサービス提供レベルを維持するとともに，継続的サービス改善により不具合の解消や品質向上に取り組む PDCA サイクルである．この小さな PDCA サイクルはサービス終息を迎えるまで何度も回り続けることとなる．

初期開発では，システム／サービスに対する要件は，事業／顧客から与えられ，システム開発のＶ字モデルに従って受入テストフェーズまで進む．受入テストでは，要

Vol.2011-ARC-195 No.21
Vol.2011-OS-117 No.21
2011/4/14

求仕様／機能要件の充足性の確認が行われる．

運用は同じく事業／顧客から与えられたサービス要件に基づき環境や運用体制などをサービスストラテジ／サービスデザイン／サービストランジションを通じて準備する．

このようにライフサイクルを通して見ると，初期開発と運用の接点はサービストランジションにあり，初期開発から運用に渡る直前のタスクがテストであることが良く分かる．

(2) 初期開発の改善手法に学ぶ

初期開発では，PMBOK／CMMI などの体系のもと様々な改善手法が存在し効果をあげている．特に前述した初期開発と運用の接点となるテストに着目すると，TDD と CI による成果が数多く報告されている．

TDD は，設計仕様を実行可能なテストコードとして記述し，開発中のプログラムが仕様どおりに動作するかを自動的なテストを実行して検証しながら開発を行なう手法である．XUnit／Selenium などのツールの他，Rails などのアプリケーションフレームワークがテストの雛形を用意するものもある．TDD により開発すべきプログラムの仕様が明確になるとともに，何度も自動実行可能なテストにより変更／リファクタリングによる影響を確認しながら開発を進めることができる．これによって短期間な開発を繰り返しながら実装を行う Agile 開発も安定した品質で行うことが可能となり，事業環境の変化に柔軟に対応しながら開発プロジェクトを運営することができるようになった．

CI は，開発作業中に発生する作業のうち頻度が高く定型的な，コンパイル／ビルド／システム統合／テスト実行を自動化し継続的かつ高頻度に実施する仕組みである．CI によって，個別に開発が進められる各コンポーネントが頻繁に結合され，テストされることになり，コンポーネント単独のテストでは検出できなかった変更による影響をいち早く検出し，コンポーネントの集合として構成されるシステムに不具合を発生するコードの混入を防ぐことが可能になった．前述の TDD による開発は，CI を効果的に行うためにも必要な技術である．

CI を普及させる推進力として，IaaS の利用によりサーバリソースをオンデマンドに大量かつ安価に供給できるようになったことが挙げられる．複数のコンポーネントにより構成されるシステムに対する CI を実現する場合などは，同時並行で試験を行うことも多く，試験のために必要な物理的なサーバリソースの調達を待つ必要がなくなり，大量のテストケースを効率よく実行することができる IaaS の登場が CI を現実的な費用で実現することを可能とした．

3. 解決方法

初期開発から運用へスムーズに移行し運用中に発生する小さな PDCA サイクルを円滑に回し続けるためのフレームワークとして Durable ICT を提案する．Durable ICT は IaaS を活用し，CI と TDD の適用範囲を運用フェーズに拡大することで，サービス開始後でも初期開発で実施した品質確保作業を適宜実施することができる．品質確保作業は，本番環境とは独立に，しかも本番環境と厳密に同一な手順でつくられた環境で実施できる．このため，本番環境に変更を実施した結果の影響確認を稼働中の本番環境に影響を与えることなく確認することが可能となり，変更に起因したサービス品質劣化のリスクから解放される．

3.1 CI の運用への拡張

運用まで拡張された CI では，ビルド&テストの対象がシステム／サービスを構成するコンポーネント／プロダクトの品質保証から，それらコンポーネントの組み合わせによって提供されるサービスの SLA の保証まで拡大される．また，各コンポーネントが個別にエンハンスを実施するため，テスト実行のトリガーが複数かつ散発的に発生する．

このため，テスト対象となるシステム／サービスは，各コンポーネントのバージョンの相違や同様機能を提供するプロダクトの違いを含めた複数の組合せで構成することを要求されることになる．

Durable ICT では，予め検証されたビルドスクリプトによって周辺環境を含めたシステム／サービスをビルド&テストする．テストは，各コンポーネントのインターフェースに対するものから，業務アプリケーションの機能仕様や運用ツールによる運用オペレーションを含むシステムテスト，SLA に対するサービス妥当性確認テストまでを含む．

これにより運用組織は，小規模な変更から保守開発に至るまであらゆる変更に対して，変更がサービスに与える影響を確認しながらサービストランジションの各プロセスを実行するとともに，本番環境で動作する各コンポーネントが完全にテストされ，動作検証されたものだけで構成されていることを保証することが可能となる．

3.2 TDD の運用への拡張

我々は TDD を運用まで拡張することで，初期開発で使用されたテストコードをそのまま変更要求に対する変更影響の確認に使うとともに，運用中のサービス妥当性確認や，保守開発に対する受入テストで使用することが可能になると想定した．

現在稼働中の本番系と厳密に同一な手順で検証系を構築し，その検証系に対して変更を実施し初期開発で使用されたテストを適用することは，変更によるサービス影響

Vol.2011-ARC-195 No.21
Vol.2011-OS-117 No.21
2011/4/14

を確認するための確実な手段となる．事前検証でこの手順を踏むことで，変更要求実現に必要な追加開発つまり保守開発の仕様を明確にすることができる．また，当然保守開発の成果物に対する受入テストにも利用できる．

これにより運用組織は，初期開発で要件定義にもとづき作成されたテストコードにより全ての変更が常にサービス要件を満たしていることを保証することが可能となる．さらに，テストコードは初期開発で作成されたものを利用していることで，変更の都度個別に受入テストのためのテストコードを準備する必要もなく，安価にこれを実現することができる．

3.3 IaaS の利用

運用組織が Durable ICT で提案するシステム／サービスの環境をこれまでと同様にオンプレミスな環境で実現するためには，本番稼働環境に加えて，テスト環境／検証環境を本番環境と同一な構成で複数保有する必要がある．このような機器を保有することは，環境構築からテスト実施までを自動化して効率化しても，実現コストが得られるメリットに対して過度に高価なものとなる．

しかし現在では，IaaS を利用することで環境保有のコストはゼロとなり，必要なリソースをオンデマンドで確保することで時間的制約もコントロール可能となった．

Durable ICT は，各環境を IaaS 上で実現することを前提に構成し，IaaS が提供する API を利用してテストを実行するために必要なサーバリソースを都度作成することで安価に実現することを可能としている．

3.4 Durable ICT の提案

Durable ICT フレームワークは, システム／サービスを運用する ICT 組織を支援するためのフレームワークである．

Durable ICT とは Eternal Service (永久不変のサービス)ではなく，Ephemeral Module（儚い，短命なモジュール) で Durable Service (永続性のあるサービス) を提供することを目指す概念である．

Durable ICT フレームワークでは ICT 組織全体を支援するフレームワークであるため, 単に CI によるシステムのビルド&テストによる機能要件の充足性確認だけではなく，運用作業のタスク管理や CAB(変更諮問委員会 Change Advisory Board 一般的に取締役会に相当する意思決定機関)への意思決定の指標などの情報提供といった機能も必要となる．

また, OSS の活用なども含むマルチベンダ化など, システムの構成要素が多様化し，異なるプロダクト・ライフサイクルを持つ製品の組合せへの対応も必要である．

図 2 は，Durable ICT フレームワークとして想定するシステム構成案である．

サービスマネージャ／サービスオーナは，提供するサービス運営に責任を持つ人物の役割であり，CIO や情報システム部長に相当する人物となる．サービスマネージャは, Durable ICT を通じて各変更要求の処理状況や本番環境へ投入されようとしているコンポーネントのテスト状況を確認することができる．

構成管理／変更管理他のマネージャとスタッフは，各運用プロセスを担うプロセス責任者や担当者となる．これらの人物は，サービスデスクなどを通じて報告される変更要求や各コンポーネントのベンダなどから上がってくるエンハンス情報などをもとに変更を計画し，それぞれの作業を Durable ICT のチケット管理の機能を通じて状況更新しながら実施する．また，運用に拡大された TDD / CI による作業品質の保証を得ることが可能となる．

コード開発者は運用組織が入手可能な形でリポジトリを提供し，それぞれの成果物を提供する．

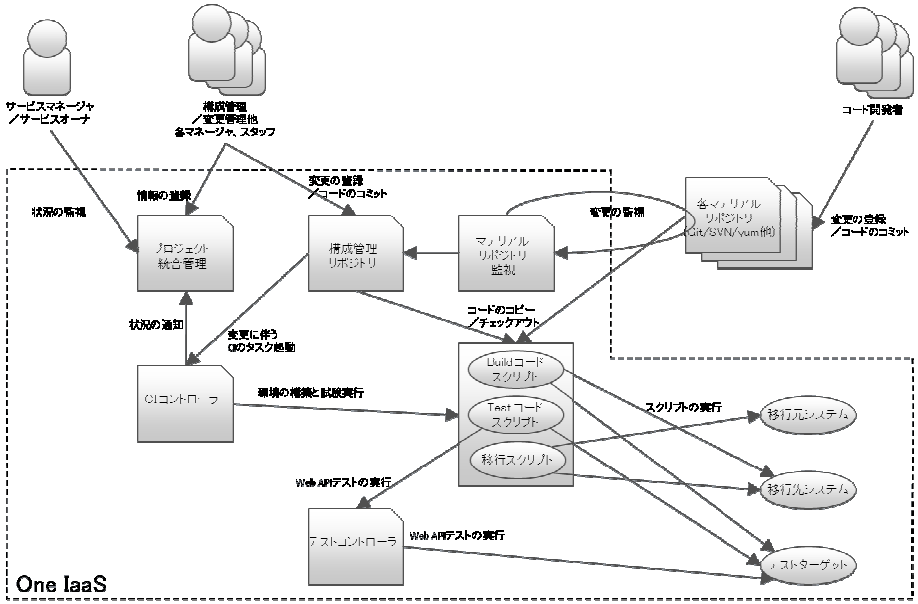

図 2 Durable ICT システム構成案

プロジェクト統合管理は，初期開発／保守開発といった開発プロジェクトの支援にとどまらず，運用組織側のサービストランジションのプロセスに対するタスク管理の提供やナレッジ管理システムとして利用することを想定している．CI コントローラによるテスト結果などの情報も統合し，変更要求に対する事前検証や受入テストなども

作業履歴とともに追跡可能なものとする．

構成管理リポジトリは，各マテリアルを組み合わせてシステムとして構成するためのビルドスクリプトやテストコードの格納場所として利用するほか，いわゆるCMDB(構成管理データベース Configuration Management Database)としての利用も想定される．

マテリアルリポジトリ監視は，外部調達を含むシステム構成要素の提供元が提供する変更情報などを監視し，構成管理リポジトリに検討可能な組合せ候補として登録し，CI コントローラによるビルド実行のトリガーとなることを想定している．

CI コントローラは，各マテリアルを組み合わせてシステムを構築するビルドコードと一連のテストの実行を制御するとともに，API を通じてプロジェクト統合管理へ結果の通知を行う．

テストコントローラは，CI コントローラから起動され全てのテストを実行する．

4. 実装

Durable ICT の評価のため，前述したフレームワークの構成をもとに，モデルシステムを実装した．また，サービストランジションのサンプルシナリオを作成し，このモデルシステム上で取り扱えることを確認した．

4.1 実装に用いた要素技術

実装に用いた要素技術は表 1　試作で実装したの通りである．

表 1　試作で実装した要素技術

構成要素	試作で使用した要素技術	試作で実現内容
プロジェクト統合管理	Redmine	チケット管理機能を利用したタスク管理
構成管理リポジトリ	Git	ソースコード管理システムによるビルドスクリプト／テストコードのリポジトリを提供
マテリアルリポジトリ監視	未実装	
CI コントローラ	Jenkins	CI ツールによるテスト環境構築および稼働環境構築．テスト結果集約
テストコントローラ	Selenium	自動テスト実行
ターゲットシステム ・移行元 ・移行先 ・テストターゲット	Nifty Cloud	クラウドを利用したサーバリソースの確保
各マテリアルリポジトリ	CentOS YUM リポジトリ Jenkins YUM リポジトリ Ruby GEM リポジトリ	

各要素技術の概要は次の通り．

(1) **Redmine**[8)]

チケットベースのバグトラッキングシステム，wiki，リポジトリビューアなどを備えたプロジェクト管理ソフトウェア．

ロールベースでチケット管理を行うことにより，柔軟なワークフロー定義を行うことができる．

(2) **Git**[9)]

分散型リポジトリによるソースコードなどのバージョン管理システム．

派生バージョンを容易に作成できる分散型リポジトリの利点を活かし，様々な変更の候補を作成することができる．

(3) **Jenkins**[10)]

自動ビルド／自動テストを実行および結果の集計を行う，継続的インテグレーション支援ツール．

容易な WEB UI による機能設定や豊富なプラグインによる機能追加により，目的に応じた柔軟なカスタマイズが可能である．

(4) **Selenium**[11)]

WEB ブラウザに対し予め指定した動作を行わせることにより，サービス利用者の視点で動作確認を行う，WEB アプリケーションのテストツール．

テストコードおよびテスト結果は HTML を用いて表現することができ，そのまま WEB ブラウザで確認することができるため，テストに携わらない者へ内容を説明することが容易である．

(5) **Nifty Cloud**[12)]

日本国内にサービスを展開する IaaS 型パブリッククラウド．

API を使用することで，ユーザの使用状況と連動したサーバリソースの追加／削除

Vol.2011-ARC-195 No.21
Vol.2011-OS-117 No.21
2011/4/14

を柔軟に行うことが可能である．

4.2 サービストランジションの具体例としてのサンプルシナリオ

ここでは，サンプルシナリオとして，業務システムを構成するデータベースエンジンを変更するサービストランジションを例にとり処理の流れを解説する．

サンプルシナリオの全体的な流れは図 3 サンプルシナリオの全体的な流れの通りである．

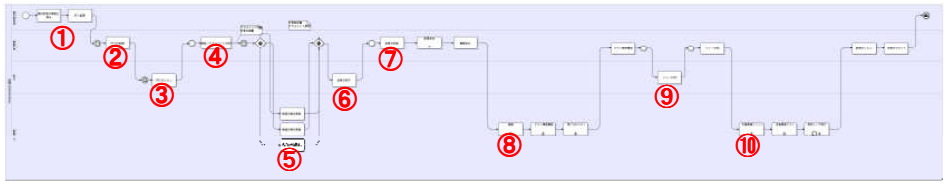

図 3 サンプルシナリオの全体的な流れ

運用組織は，Durable ICT を用いて作業タスクを管理するとともに，以下の CI のジョブを実施している．

- 変更に伴う影響確認のための事前検証
- 変更に対する受入テスト
- 本番環境への適用

以下にそれぞれの作業タスクのポイントを説明する．

(1) サービストランジションの契機

技術担当者は，使用中のデータベースエンジンのサポートが終了するとの情報を得て，サービスで使用中のデータベースエンジンの変更が必要である旨，新規チケットを発行し，変更要求(RFC)を起票する．

(2) RFC の記録

変更管理マネージャは，RFC を台帳に記録する．

(3) RFC のレビュー

変更諮問委員会(CAB)により，技術的視点だけでなく経営的な視点も加味して，RFC のレビューが行われる．その結果，データベースエンジンの変更が必要なこと／変更にあたりその影響の検証が必要なことが決定される．

(4) 検証／アセスメントの計画

変更マネージャにより，検証の具体的な計画が策定される．計画の概要は次の通り．

- 検証する対象：データベースエンジンのみ変更
- 検証する環境：仮想マシンを利用
- 検証で使用するテストプログラム：現バージョンに対するテストプログラム一式
- 検証項目：機能，手順，コスト

(5) 変更に伴う影響範囲の特定のための自動ビルド＆自動テスト

リリース管理マネージャにより，テストターゲットがビルドされ，テストされる．テストターゲットは，データベースエンジンを変更した以外は本番稼働環境と同一であり，テストは受入テストと同一のものを実行する．

(6) 変更の妥当性確認

変更許可委員はテストの結果から，追加開発が必要な個所の特定や，変更イベントそのものの妥当性を精査する．精査の結果，DB バックアップの個所がテスト NG であり，DB バックアップに改修が必要であるとの判断を行う．

(7) 追加開発物のコミット

変更マネージャは DB のバックアップツールの改修を計画し，実際に作業を発注する．改修による成果物は構成管理リポジトリにコミットされる．

(8) 受入テストのための自動ビルド＆自動テスト

リリースマネージャは，追加開発物受入の観点の他，追加開発物がシステム全体に及ぼす影響を確認するため．同一の条件で再度テストターゲットを自動ビルドし，テストターゲットに対し自動テストを行う．

(9) 本番リリースの許可

変更許可委員はテストの結果，特に不具合が見受けられないことから，本番投入しても問題ないとの判断を行う．

(10)本番リリース

リリースマネージャはテスト環境を自動ビルドしたときと同一のビルドスクリプト／テストコードを用い，本番環境の自動ビルドおよび自動テストを実行する．

4.3 ツールの使用イメージ

前述したサンプルシナリオをモデルシステム上で動作させた．画面イメージは次の

Vol.2011-ARC-195 No.21
Vol.2011-OS-117 No.21
2011/4/14

通りである．

(1) サービストランジションの契機

サービストランジションの進捗は，Redmine 上のチケットにて管理される．新規チケットの発行画面は，図 4　新規チケット発行画面のようになる．

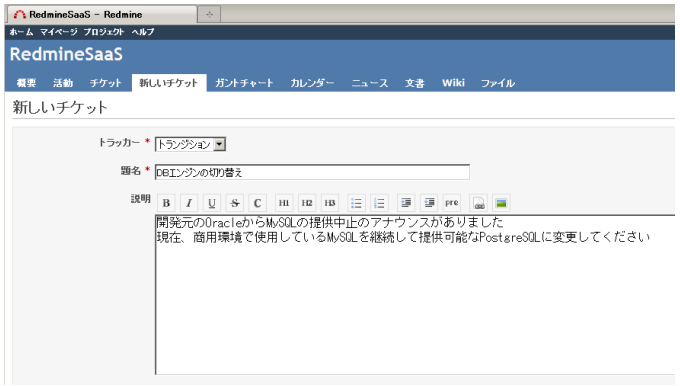

図 4　新規チケット発行画面

(2) チケットベースの進捗管理と CI ツールの呼び出し

Redmine にてチケットの現在のステータスが管理され，ステータスにひも付き担当者が明確となる．

チケットのステータスが 「検証環境でテスト」まで遷移すると，テストの担当者（リリースマネージャ）が CI ツールを起動しテストターゲットのビルドとテストを開始する．CI ツールは構成管理リポジトリから取り出されたビルドスクリプトをもとにテストターゲットを自動ビルドし，構成管理リポジトリから取り出されたテストコードにより自動テストを行う．

テストの内容は，Ruby on Rails の rake test のようなモジュールに付属されるテスト，Selenium による WEB UI のテスト，付属ツールを実際に走行させた結果の確認の 3 種のテストを行っている．

テストの結果は CI ツールにより集計され，集計画面により一元的に確認できる．リリースマネージャはテストの結果をチケットに追記し，ステータスを「変更の妥当性確認」に遷移する．

CI ツールは Jenkins により実装され，構成管理リポジトリは git により実装されている．

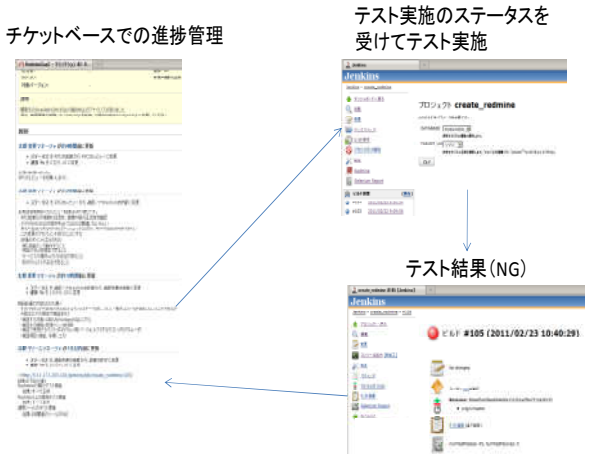

図 5　チケットのステータス遷移の記録

(3) 追加開発物のコミット

リリースマネージャは追加開発を計画し，作業の発注を行う．追加開発されたモジュールは，構成管理リポジトリにコミットされる．

構成管理リポジトリは git にて実現される．

```
[loot@ebizo remote_runner]$ git commit -a
Created commit b8aafad: DB backup tool has been modified.
 1 files changed, 2 insertions(+), 2 deletions(-)
[loot@ebizo remote_runner]$ git push DurableICT master
git@localhost's password:
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 463 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To ssh://git@localhost/~git/repository/DurableICT
   8deedfc..b8aafad  master -> master
[loot@ebizo remote_runner]$
```

図 6　追加開発物のコミット

(4) 受入テストのための自動ビルド＆自動テスト

追加開発の本稼働への影響を検証するため，ステータスを「テスト環境でテスト」に遷移させる．

Vol.2011-ARC-195 No.21
Vol.2011-OS-117 No.21
2011/4/14

チケットのステータスを受けて，テストの担当者（リリースマネージャ）が CI ツールを起動しテストターゲットのビルドとテストを開始する．

今回は追加開発により全てのテストが OK となる．

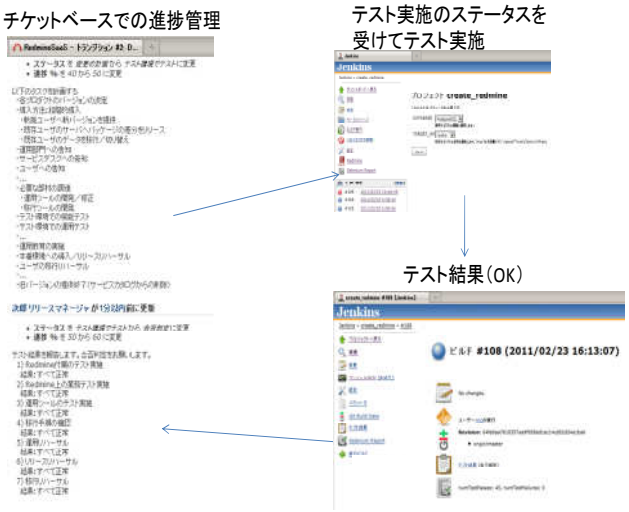

図 7　自動ビルド&自動テスト画面

(5) 本番リリースの許可

Jenkins のテスト結果集計ページを参照し，本番リリースが妥当であるか検討を行う．問題が無いと判断されると，ステータスを「本番リリース」に遷移する．

(6) 本番リリース

本番環境への自動ビルドは Jenkins により行われる．

本番環境の自動テストは Jenkins から呼び出された Selenium により行われ，テスト結果は Jenkins により集計される．

自動ビルドのためのビルドスクリプトおよび自動テストのためのテストコードは構成管理リポジトリから取り出される．構成管理リポジトリは git にて実現される．

5. 評価

今回の試作を通じ次のような評価を得た．

(1) 運用における TDD の有効性の確認

今回の試作では，次のテストを実行した．

- ミドルウェア(Redmine)付属の機能テスト
- WebUI を利用した業務フロー(業務 APL)に対する機能テスト
- バックアップジョブを実行する運用ツールに対する機能テスト

これらのテストをテストコードには全く手を加えずに次の作業タスクで実行した．

- 変更実施前の本番環境に対するテスト
- 変更に対する影響範囲の事前検証を行うために検証環境でデータベースエンジンのみを実施したテスト
- 保守開発により影響範囲の対応を済ませたテスト環境での受入テスト

TDD を運用まで拡張することにより，事前検証では変更による影響範囲を提供機能レベルだけでなく運用ツールに至るまで確実に検出することができた．また，その結果を受けた保守開発の成果物も本番環境に入れる前にサービス要件を満たすものとして保証し，安心して変更を実施することが可能となった．

これに加え，テストコードには全く追加／変更を行っていないため，追加開発のコストは変更箇所のみに限定できた．

この結果，現実的なコストで TDD を運用まで拡張することに対して，Durable ICT が有効な手段であると考えられる．

(2) 運用における CI の有効性の確認

今回の試作では，IaaS 上の標準 VM を利用して CI のジョブとして次のビルド&テストを実行した．

- 依存関係を持つソフトウェアを含むミドルウェアのインストール
- 業務ロジックを含むサービス環境の構築
- 業務ロジックを含むテストの実行

この CI ジョブを前述の 3 つの作業タスク(本番環境のテスト，事前検証のテスト，受入テスト)の中で実行した．

CI を運用まで拡張することにより，各作業タスクで作業ミスの混入は防止され，作業結果の証跡も確実に記録することができた．また，同一の作業を繰り返し実施しても同一の結果が得られることを確認し，作業品質の確保が行えていることが確認できた．

この結果，全ての作業を自動化するという課題はあるが，CI を運用まで拡張することの有効性を確認するとともに，Durable ICT がサービス品質の確保に有効な手段であると考えられる．

(3) IaaS 利用の有効性の確認

Vol.2011-ARC-195 No.21
Vol.2011-OS-117 No.21
2011/4/14

前述の通り今回の試作は全て IaaS 上の VM を利用して作成した．

IaaS を利用することにより，必要な場合にのみリソースを迅速に調達可能になり，低コストでこれらの成果を得ることができた．

また，提供される VM の環境や品質が標準化されているため，常に一定の条件で環境利用が行えるため，CI を行う上でも環境要因による変更や失敗が起こらず，作業品質の確保も行えた．

検証／テスト環境のみならず本番稼働環境にも IaaS を利用することで本番環境のビルドもテスト環境ビルドと同一のビルドスクリプトが利用できたため，一連のサービス・ライフサイクルを通じてオペレーション上のインターフェース統一が図れるという点においても，IaaS 利用は有効であると考える．

6. 関連研究

開発フェーズのテストにクラウドを適用する試みについて参考文献に資料名を記載した [13)14)15)]．これらの試みはテストの網羅性の向上にクラウドによる潤沢な資源供給を活用することで,大規模なシステムプロジェクトでの生産性向上に寄与している．IBM は開発フェーズに特化したパブリッククラウドを顧客のシステム構築に提供している [16)]．

運用まで含めたサービス品質向上にクラウドを適用する試みについて，Google が報告している [17)]．Google は自身のサービスの開発／運用のためのプライベートクラウドを構築して，サービスのライフサイクル全般でソースコードリポジトリ，クラスタ管理，Continuous Integration を連携させたサービス品質保持を実施している．

7. まとめ・今後の課題

IaaS を利用し，CI と TDD を，運用フェーズに拡大適用したフレームワーク Durable ICT を提案する．

Durable ICT では IT ライフサイクルの中で継続して要求される変更を，初期開発時と同一の自動試験で機能要件充足性を確認しながら確実に実施する．検証系構築は IaaS を用いて低コストに行なえ，CI を用いて稼働系との完全な同一性も保証される．

これにより，従来は制御が難しかった，変更に伴うサービスレベル低下のリスクを管理しながら，必要な修正や機能追加を適宜実施することが可能となる．これによりいわゆる「塩漬けのシステム」を一掃することができる．

今後の課題としては，現在未実装である外部マテリアルリポジトリ監視の実現，データ移行を含めた変更，PaaS/SaaS へ展開，経営層の判断が容易になるテスト結果の表現方法といったテーマに取り組む必要がある．

参考文献

1) OGC: ITIL V3 コア書籍 サービスストラテジ，TSO，日本語版，2008 年 4 月
2) OGC: ITIL V3 コア書籍 サービスデザイン，TSO，日本語版，2008 年 5 月
3) OGC: ITIL V3 コア書籍 サービストランジション，TSO，日本語版，2008 年 6 月
4) OGC: ITIL V3 コア書籍 サービスオペレーション，TSO，日本語版，2008 年 7 月
5) OGC: ITIL V3 コア書籍 継続的サービス改善，TSO，日本語版，2008 年 8 月
6) itSMF Japan: ITIL とは http://www.itsmf-japan.org/itil/index.html
7) itSMF UK: An Introductory Overview of ITIL V3，第一版，2007 年
8) Redmine - Overview – Redmine http://www.redmine.org/
9) Git - Fast Version Control System http://git-scm.com/
10) Welcome to Jenkins CI! | Jenkins CI http://jenkins-ci.org/
11) Selenium web application testing system http://seleniumhq.org/
12) クラウドならニフティのパブリック型クラウドサービス http://cloud.nifty.com/
13) Antawan Holmes, Marc Kellogg: Automating Functional Tests Using Selenium. AGILE 2006:270-275
14) George Candea, Stefan Bucur, Cristian Zamfir: Automated software testing as a service. SoCC 2010: 155-160
15) Toshihiro Hanawa, et al "ent Using Cloud Computing Technology for Dependable Parallel and Distributed Systems," Software Testing Verification and Validation Workshop, IEEE International Conference on, pp. 428-433
16) IBM Smart Business Development and Test on the IBM Cloud, HYPERLINK "http://www-935.ibm.com/services/us/igs/cloud-development/" http://www-935.ibm.com/services/us/igs/cloud-development/
17) Patrick Copeland: Google's Innovation Factory: Testing, Culture, and Infrastructure. ICST 2010: 11-14