IBM Software Group

JaSST'09 Tokyo
B3 ： もしかすると、もしかする？
モデル駆動テスティングから考える開発パラダイムの変化

日本アイ・ビー・エム　ラショナル事業部
テレロジック事業開発技術部・モデリンググループマネージャ
鈴木尚志

Rational. software

Go to IBM

AGENDA

- 障害の経済学
- 開発プロセスを改善しよう - MDDについて
  - MDDがROIに与える影響
- MDDとテストの関係
- テストプロセスを改善しよう – MDTについて
- Validation and Verification – 構造ベーステストとの連携
- IBMのトータルテストソリューションについて

この数年で、組込みソフトウェア業界ではMDD（モデル駆動開発）による実製品の開発が進み、MDDがROIの向上に大きく寄与することが分かってきました。そして、モデル駆動テスティング（MDT）が、この向上に大きく影響しているのです。なぜ、MDTがよりROI向上に貢献するのでしょうか？そもそも、モデルのテストとコードのテストは何が違うのでしょうか？本セッションでは、MDTについて説明、デモを行います。MDD,MDTを一気に体験し、開発パラダイムの変化を体感してください。

組込みSW開発で何が起こっているのか？

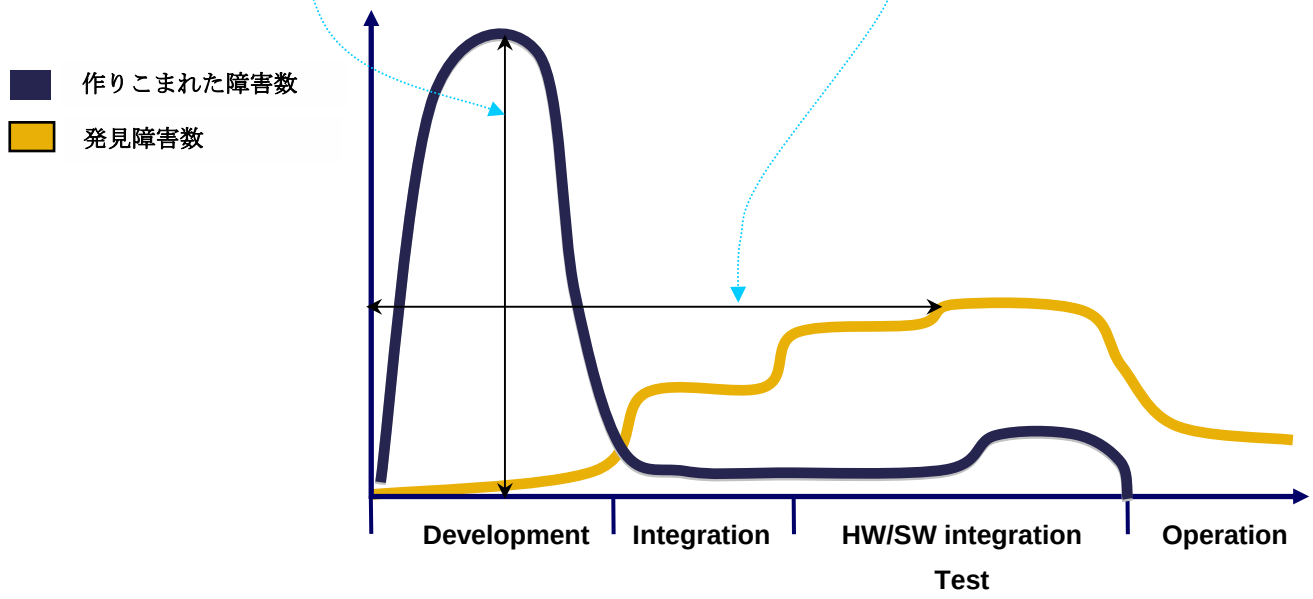
- **開発メンバーを追加した (37%) またはモジュールの書き換えを行った (74.9%)**
- **機能削減を行った(56%) 、又はプロジェクトを中止 (18%)**
- **出荷遅れが発生した (72.8%)**

Source: Embedded Market Forecasters

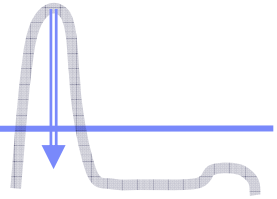
障害の経済学

あまりに多くの欠陥が開発初期に作りこまれている!

欠陥が発見された時点では遅すぎる!



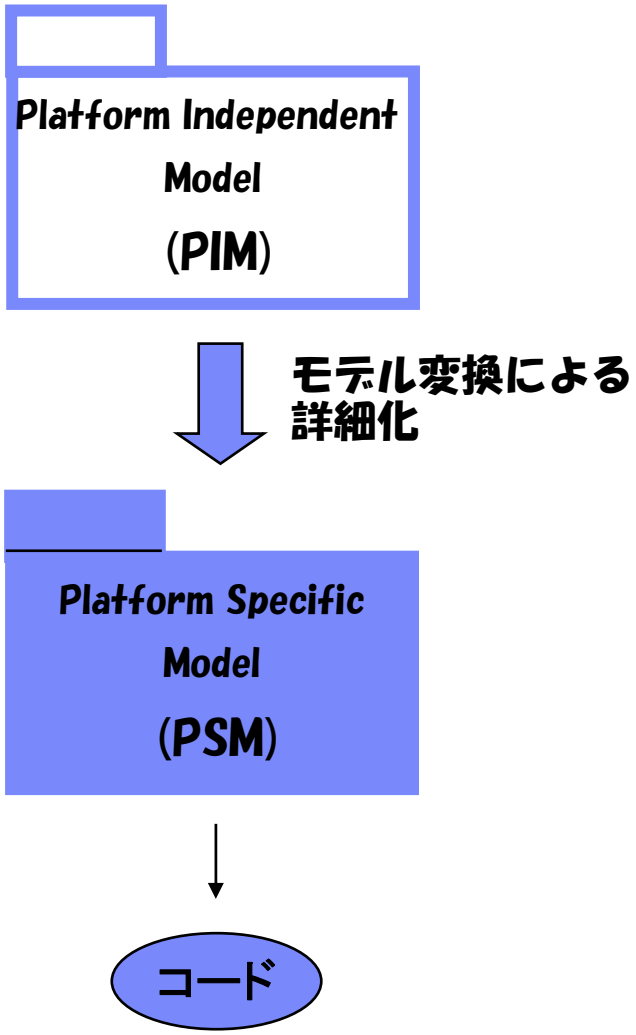
“あまりに多くの欠陥が作りこまれている”



開発プロセスを改善しよう!

- 明確で、追跡可能で、検証可能な要求
- よいアーキテクチャ設計
- 効果的なコミュニケーション
- 自動文書化
- グラフィカルな設計レビュー
- 進化的で、反復的なプロトタイプ
- 実行可能な設計
- 効果的なデバッグ/テストツール

MDAとモデル駆動開発

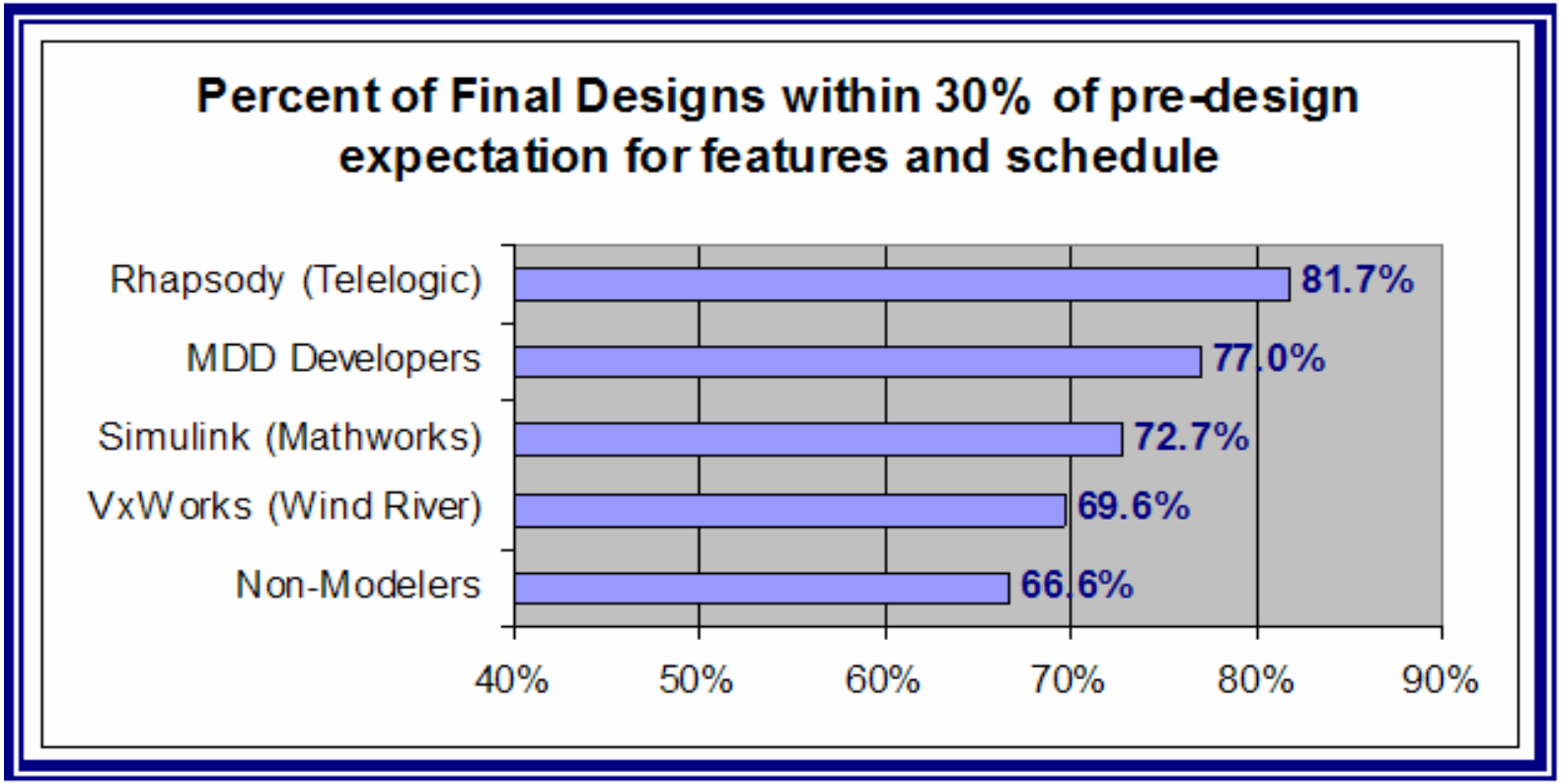


Telelogic® Rhapsody ®

最新技術の自動設計機能がベースの組込み開発環境

要求分析からテストまでソフトウェア開発の全工程をサポートするツール

MDDの優位性



[1]*from Embedded Market Forecasters 2006 Survey Results*

"What Do You Do When the Horse You're Riding Drops Dead?: Why Model Driven Design is Emerging as a Preferred Best Practice"

レポート：MDDがROIに与える影響調査

- 2008に行われた通信機器開発の455人（プロジェクト人数は12-14人）からの調査
- 様々なコストについて計測
  - 正しい納期 – 技術コストの節約
  - 期待どおりのパフォーマンス、機能、要件と開発期間
  - 開発経費のコントロール
    - SW・システムズエンジニア・HW・テストエンジニアのコスト
    - 納期遅れ/キャンセルのコスト
    - コンポーネントの選択
    - ツールの選択
    - テスト手法の選択
  - 市場機会逸失のコスト
  - フィールドサポートのコスト

Source: Embedded Market Forecasters

レポート：MDD-Non MDDの開発コスト比較

Time to Marketの向上

開発直接コスト	Non MDD	MDD	MDDによる向上
アプリケーションのコスト	$1,589,200	$1,165,600	36.3%
サポートスタッフのコスト	$1,607,180	$1,318,350	21.9%
開発直接コストの平均	$3,196,380	$2,483,950	28.7%
キャンセルコストの平均	$138,887	$70,396	97.3%
期限遅れコストの平均	$230,690	$93,703	146.2%
全ソフトウェア開発コスト	$3565947	$2648049	31.8%

開発者コスト $10,000/Month、サポートスタッフコスト $8,500/Monthで試算

機能落としが少ない

外因性コスト	Non MDD	MDD	MDDによる向上
パフォーマンス	60.0%	73.3%	22.2%
システム機能	55.0%	66.7%	21.3%
基本要件とスケジュール	65.0%	66.7%	2.6%

Source: Embedded Market Forecasters

レポート：結論

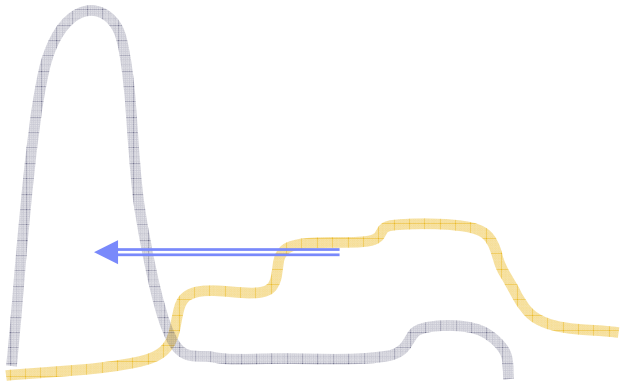
- **MDDを用いた開発は、開発コストを下げる。**
- **また、MDDは、Time-to-Marketの短縮、キャンセルの減少、納期遅れの減少、よりよい設計成果物などといった重要な成果を上げることも明らかになった**
- **そして、実現には、UML Drawing toolでは不十分。なぜなら・・・**
  - **設計シミュレーションが実行できないので初期の設計妥当性の検証が不能**
  - コードの大部分を手書きし、また、コードとモデルとの関係が疎になるため、プログラミングエラーを減少させることができない
  - **テストを手作業で記述するため、モデルからテストを生成するのに比べ、テスト時間を減少できない。**
  - 要求の追跡がモデルまでにとどまり、コードや、要求の妥当性を検証すべきテストまで届かないため、要求の追跡可能性が弱くなる。
  - Autosar, DoDAF, Net Centric Operations, Telecom Handsets, Medical FDA certification supportのような業界独自のソリューションサポートがない（極めて限定的）
  - 組込みOSとのリンクがなく、ターゲット上での検証が不可能

Source: Embedded Market Forecasters

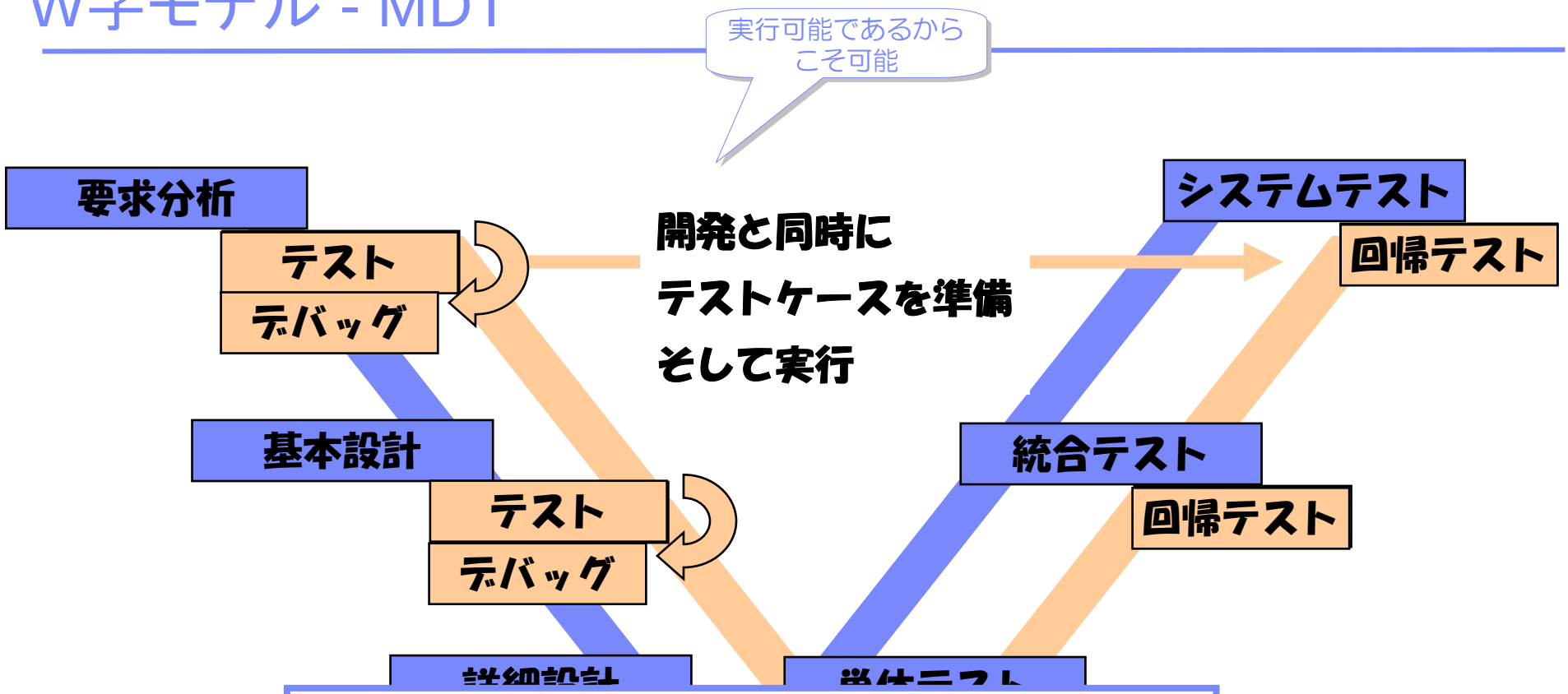
“欠陥が発見された時点では遅すぎる”

テストプロセスを改善しよう！

- Test Early; Test Often!
- 設計を実行可能としよう　・・・　実行しなくてはテストはできない!
- 自動化；自動化；そして自動化!



W字モデル - MDT



システムとテストの開発初期の統合

ミスをシステム開発初期から摘出可能に

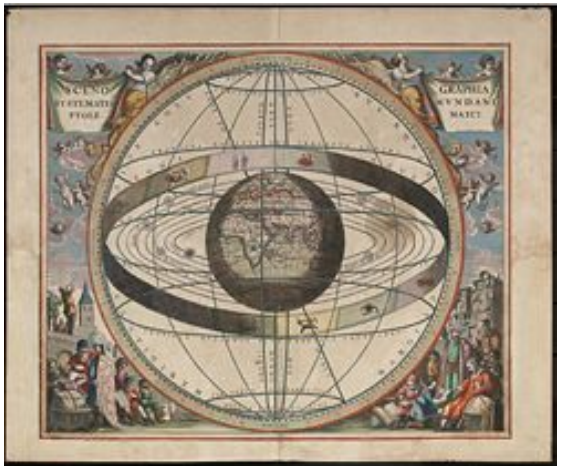
開発時間とコストを短縮

モデル駆動テスト

- モデル駆動テストとはテスト対象システム（SUT）の性質を記述したモデルから全部または一部のテストケースが生成されるソフトウェアテスト
- テストのモデルとは
  - テストのツクリ　ー　構造のモデル化
  - テストのカラクリ　ー　動作のモデル化
  - テストに使うデータ　ー　入出力のモデル化
  - テストの結果　– 成功/失敗のモデル化

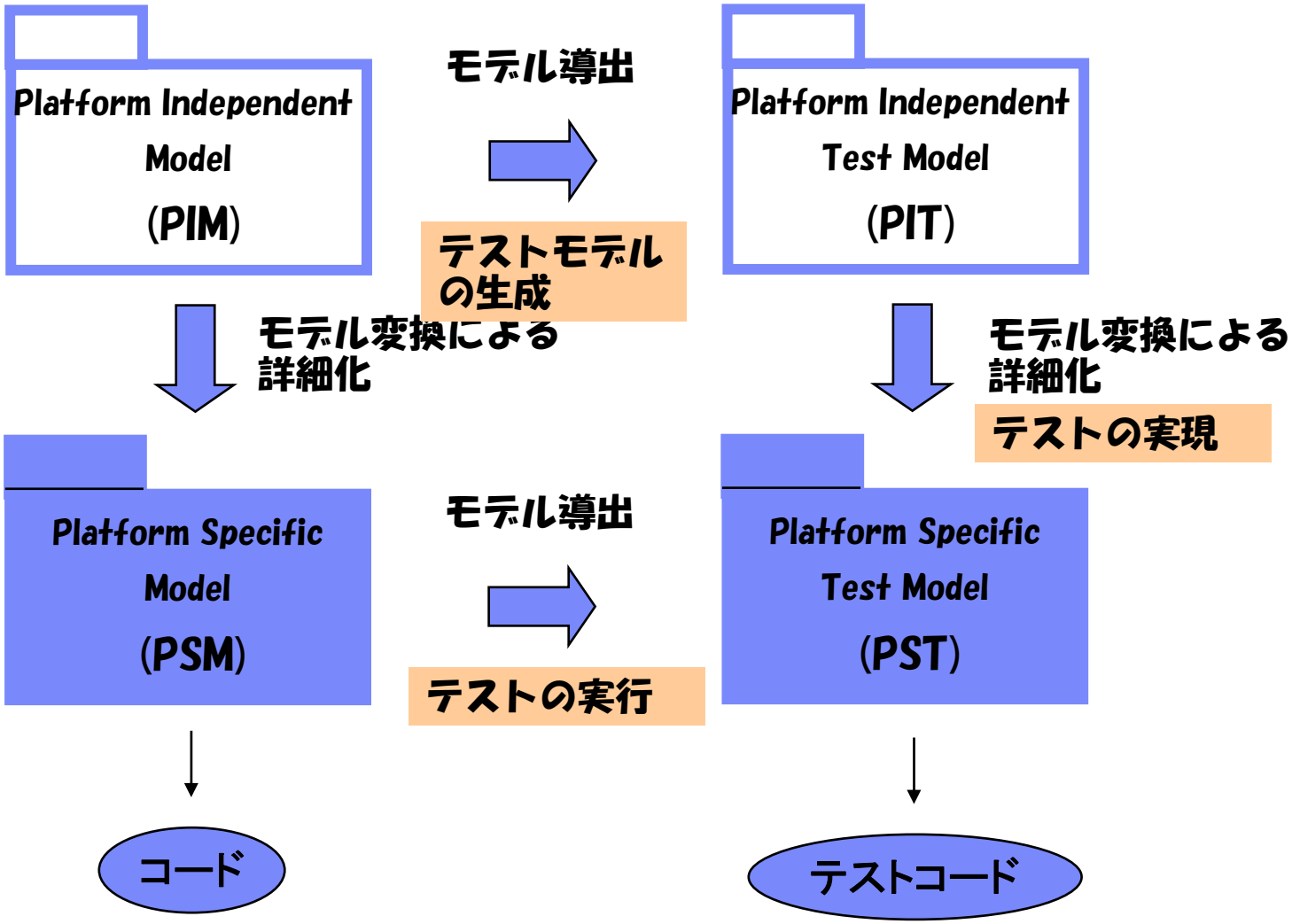
  などなど

「テストの」ツクリやカラクリに関するわかりやすい記述



MDAとモデル駆動テスト

モデルとして表現可能なものはテストケースの導出も可能



UML2.0 Testing Profile(U2TP)

- テストの内容やテスト結果をUMLにて記述するために定義されたUML Profile
- UMLテスティングプロファイルは、テストシステムの成果物を設計、ビジュアル化、仕様化、分析、構築、ドキュメント化するための言語を定義するテストモデル言語の1つ。
- テストの成果物を扱う、システムとテストの成果物を一緒に扱うなどの使い方がある
- UMLテスティングプロファイルは、テストコンポーネント、判定、デフォルトなどのようなテスト特有の概念でUMLを拡張している。
- UMLのメタモデルをベースとし、UMLの文法を再利用
- UMLテスティングプロファイルは、UML 2.0仕様をベース

U2TPのグループの概要

Test Architecture
テスト対象、テストの構造、
テストの構成、実行、制御など
規定するメタクラスが含まれる

SUT(System Under Test),
test components,
test context,
test configuration,
arbiter, scheduler
など

Test Data
保存やコミュニケーションのための
テストデータのメタ概念を保持する

Wildcards, data pools, data selectors, coding rulesなど

Test Behavior
テストの動的側面に関する
概念が含まれる

test objective,
test cace,
defaults,
verdict
validation
action
conceptsなど

Time
テストのふるまいを制御したり、
制約したりするような
時間に関する概念

Timer,
timezoneなど

ソフトウェアテスティングのための2つの戦略

- フィーチャ駆動テスト
  - 別名 要求ベーステスト
    - 「要求」は、準備されていないことも多い。
    - 要求が準備されていても、それはテスト可能なフィーチャまでブレークダウンされていない
    - 「小さな」フィーチャは開発者によってテストされ、システム全般にわたるフィーチャはテスタによりテストされる。
- 構造ベーステスト
  - 別名 プログラムベーステスト

“テストはバグの存在を示すことができるが、それがないことは決して示せない！”

Edsger Dijkstra

設計の正しさをチェックする構造ベーステスト

- ユニットテストのために用いられることが多い
  - ホワイトボックステスト
  - プログラムの制御フローやデータフローのような、コード内部の知見を用いたプログラムの内部構造の分析をもとにしたテスト
- 極めて普通で、多くのツールによってサポートされている
- 以下を含むことが多い
  - コードカバレッジ分析
  - 自動化（テストベンチの作成、レポートの生成など）
  - ランタイムトレース、パフォーマンスとメモリプロファイリング、複雑度分析

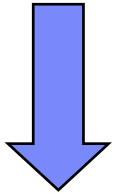
- しかし、「この設計は要求を満たすのか？」
  - コーヒーメーカーはコーヒーを作るのか？MP3プレーヤは音楽を演奏するのか？

妥当な設計をチェックするフィーチャ駆動テスト

- 統合テストやサブシステム/システムテストで用いられることが多い
  - グレイ/ブラックボックステスト
- テストは、検証されることが必要なフィーチャによって駆動される
  - システム外部から観察可能なふるまいにフォーカスすることが多い
- 以下を含むことが多い
  - モデルカバレッジ分析（グレイボックステスト）
  - HIL、SILテスト環境上で実行
  - 自動化
- しかし「この設計は、良い設計か？」
  - 全てのボタンを一緒に押したら、クラッシュしない？
  - デッドコードはない？
  - 全てのコードは一度は実行されたことがある？最初に実行するのがお客様だったりしないか？

Validation and Verification（妥当性検証と正当性検証）

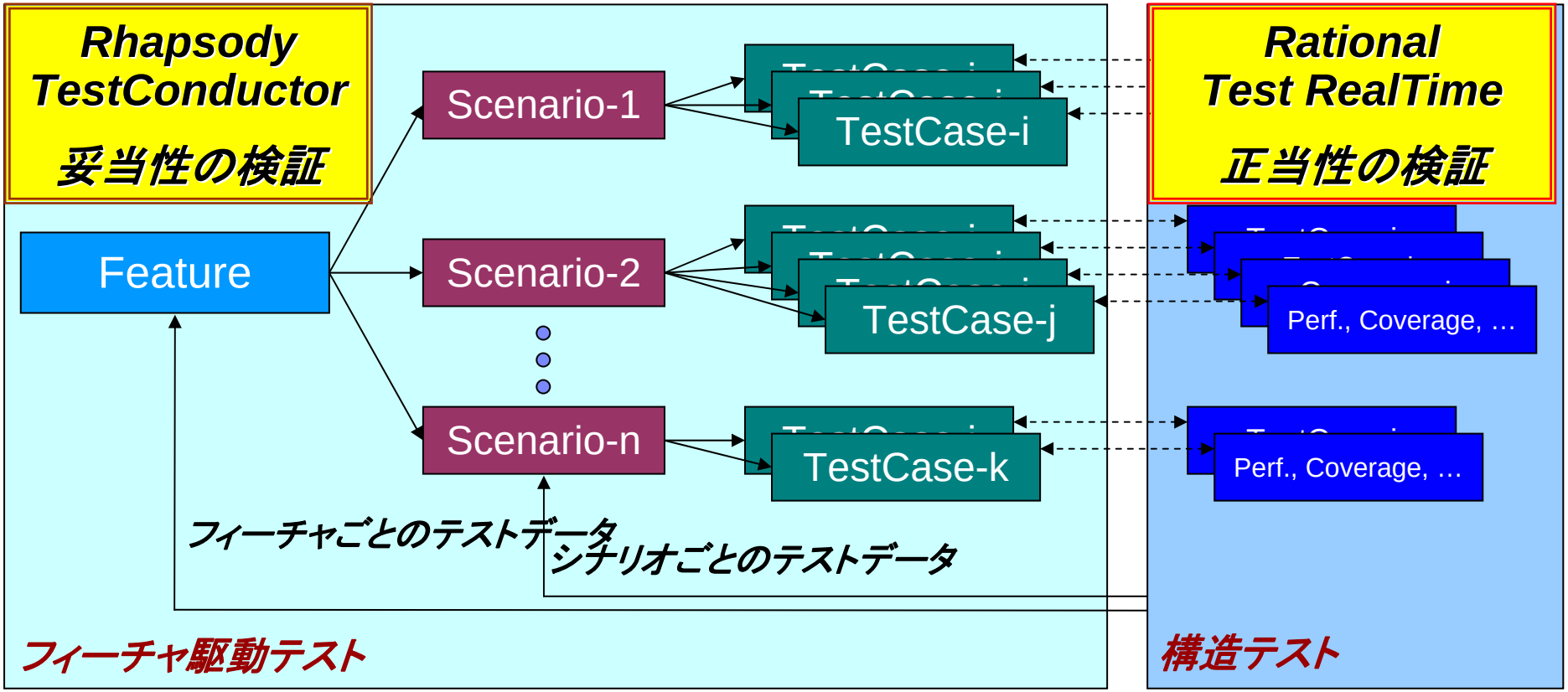
- フィーチャ駆動テスト - 設計妥当性の検証（ Design Validation ）を目的
  - 正しいフィーチャを開発しただろうか？
- 構造ベーステスト- 設計正当性の検証（ Design Verification ）を目的
  - そのフィーチャを正しく開発しただろうか？



- 妥当性と正当性の検証によって、次の問いに答えることができる
  - 本当に「正しいフィーチャ」を正しく開発しましたか/していますか？

  - 上記のフレーズは、通常一般的な「製品」というコンテキストで語られる。しかしテストは本来フィーチャが判明すると同時にできるだけ早く、製品化を待つことなく行われるべき。

IBMの包括的テストソリューション



- *テストデータは、パフォーマンスプロファイル、コードカバレッジ、実行時複雑度のようなテストケース実行に関連付けられたランタイム情報を含まない*

Telelogic® Rhapsody ®

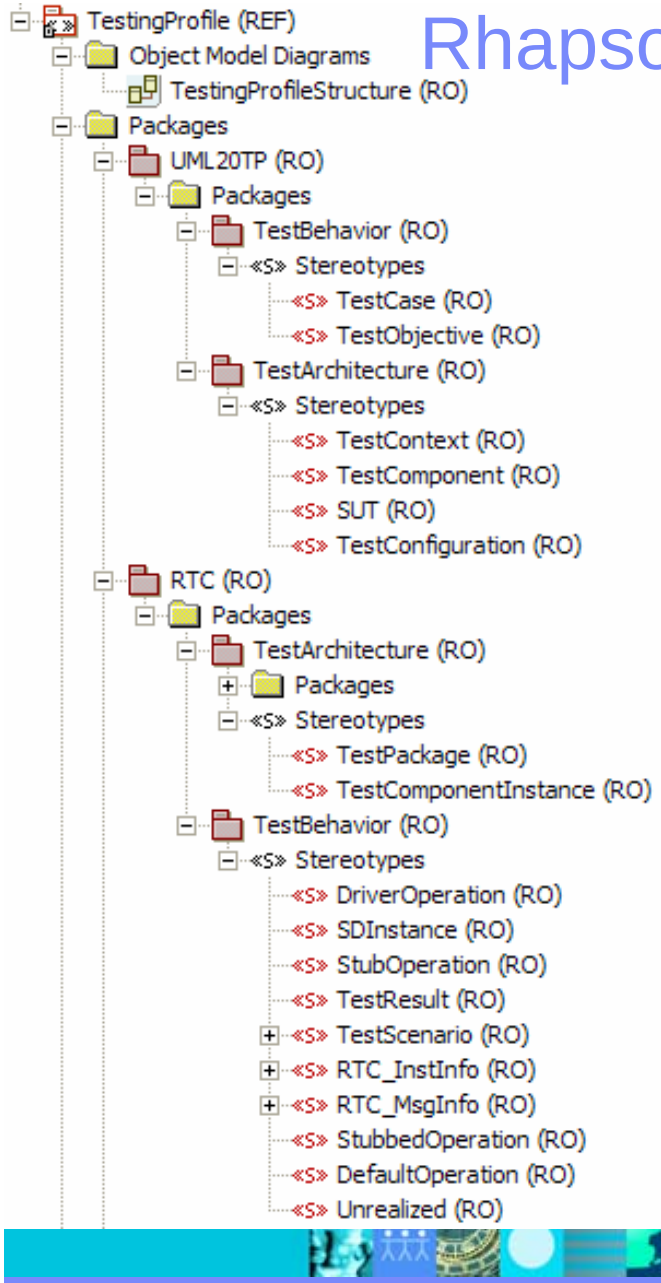
最新技術の自動設計機能がベースの組込み開発環境

要求分析からテストまでソフトウェア開発の全工程をサポートするツール

Test Conductor

- Test Conductorは、スケジューラと調停の双方の役目を果たす。選択したSUTのために、必要とするテストコンポーネントを加えて、テスト設定とテストコンテキストを自動的に作成することが可能。
- 1つのテストケースで、あるいは、テストケースの集合でスケジューリング/実行し、それぞれのテストログと判定結果を生成可能
- テストケースは、シーケンス図上のメッセージや、手書きのコードや判定マクロのなどで、刺激と観察として定義
- 結果分析には、ワイルドカード(Tolerance)を使用可能。
- 決定されたテストケースを、異なったテスト実行で再利用可能
- テストケースは、テスト目的からテスト計画の要求までトレース可能

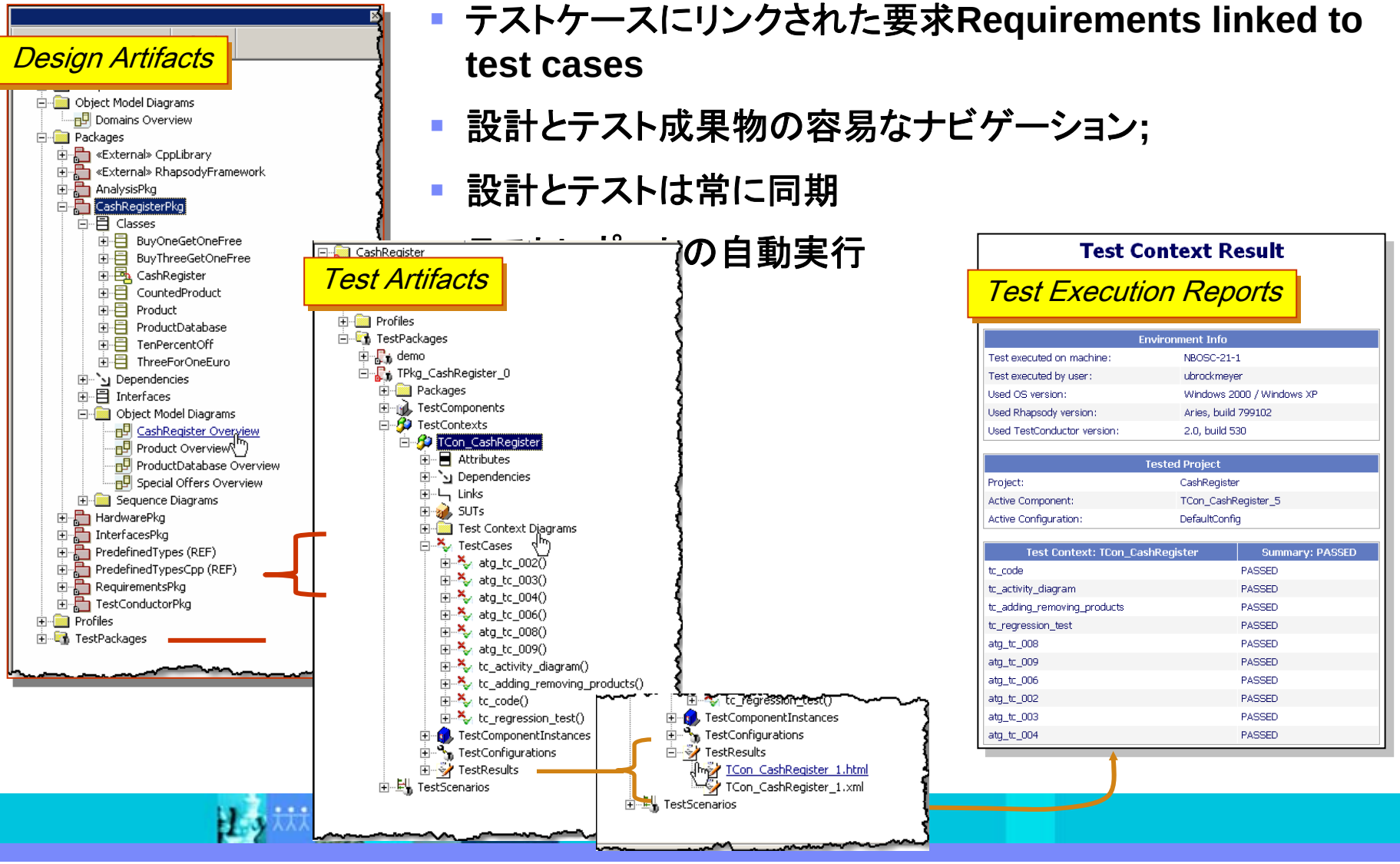
Rhapsody テスティングプロファイル



- ***Rhapsody UML** テスティングプロファイル*
  - UMLテスティングプロファイルをベース
  - Rhapsodyでモデルテストの成果物に適用することができる新しい用語とステレオタイプを含む
  - UMLテスティングプロファイルで定義された要素のいくつかは、Rhapsodyテストプロファイルにはありません。
  - UMLテスティングプロファイルにはない補助的な要素も含みます。たとえば、UMLテスティングプロファイルによって指定されないテスト活動のために使われる付加的な要素はその1つです。

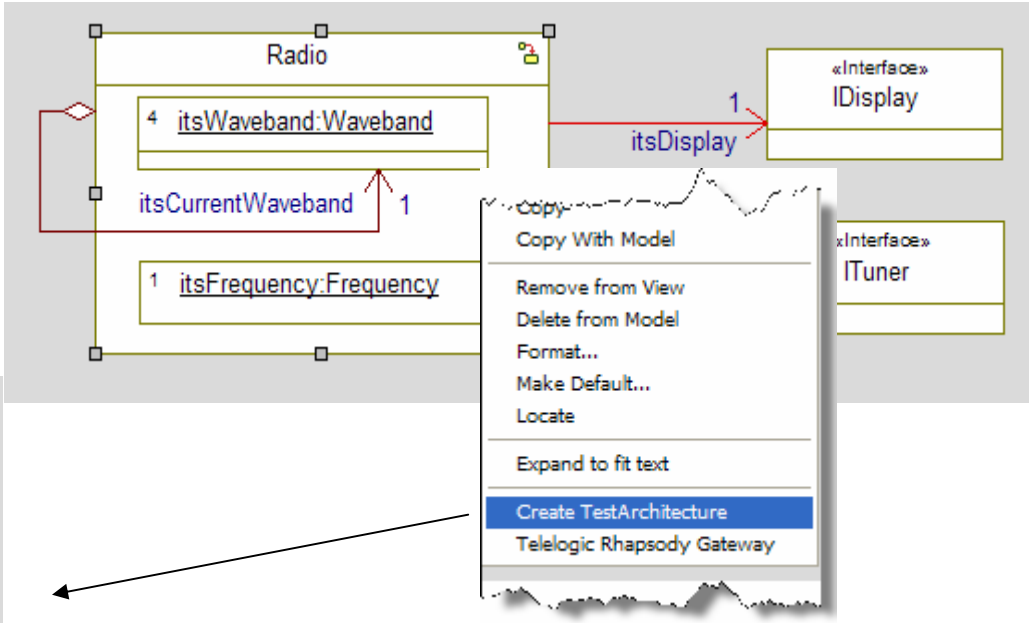
設計とテストプロセスの統合

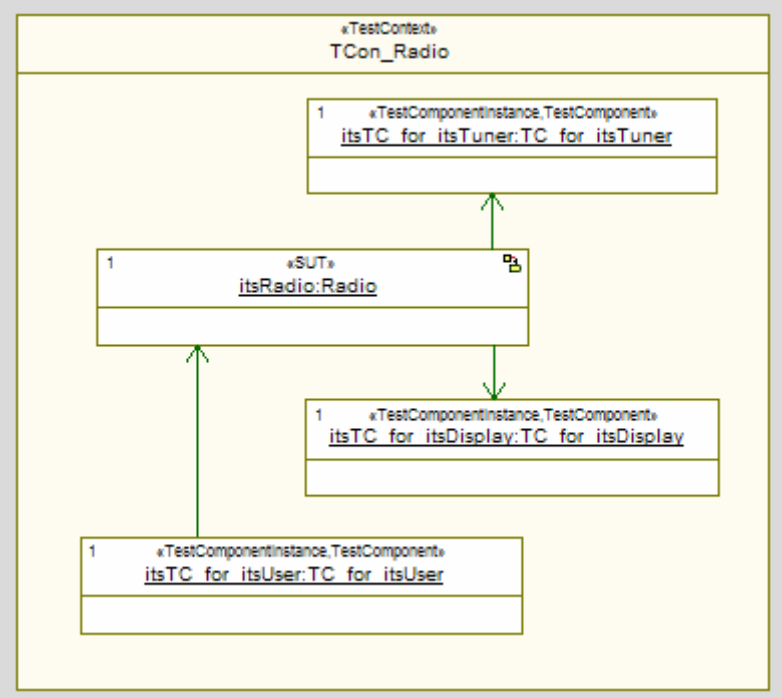
- 共通のブラウザ
- テストケースにリンクされた要求Requirements linked to test cases
- 設計とテスト成果物の容易なナビゲーション;
- 設計とテストは常に同期
- の自動実行



グラフィカルにテストアーキテクチャを作成

- テストアーキテクチャは、コードベースのテストベンチのモデル
  - 自動生成される
  - モデルが更新されると更新される





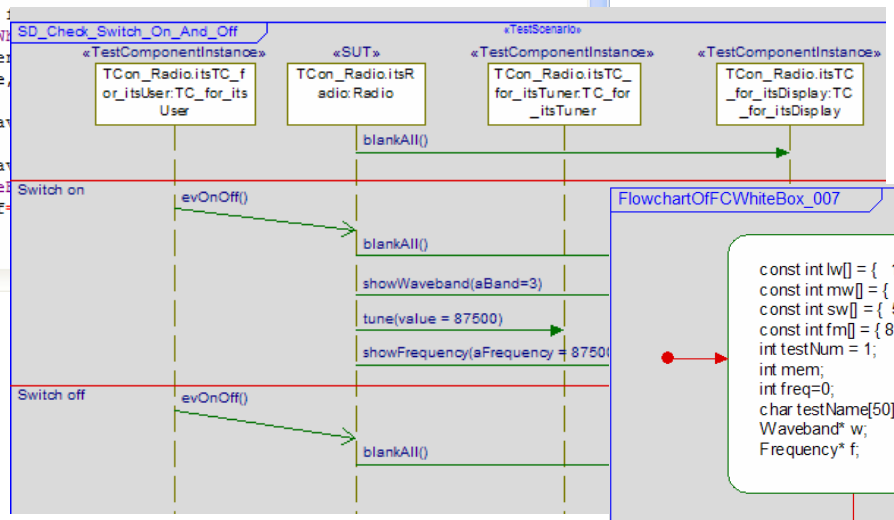
- 生成されたアーキテクチャには、
  - テストケースはまだない
  - ただコンパイルできるだけ

モデル成果物のテスト

- モデルを設計したようにテストを設計する

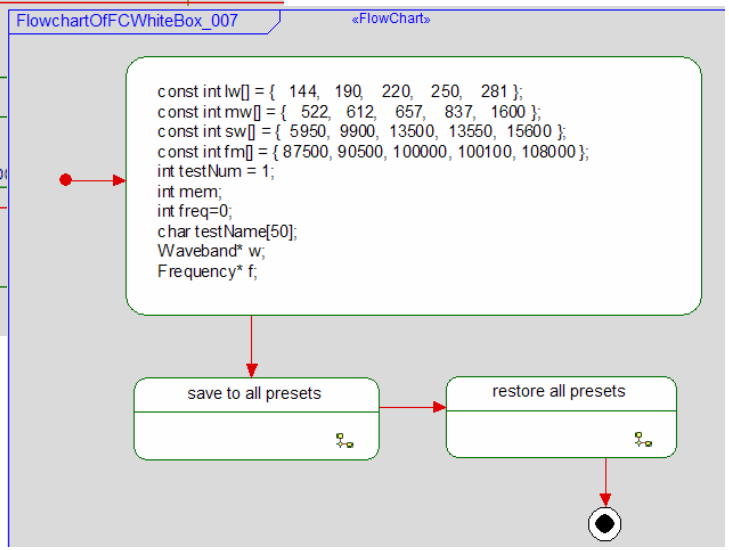
コード

```
int f;
int freq;
int testNum = 1;
char testName[50];

itsRadio.nextWaveband();

// Test MW  522KHz to 1620KHz step 9KHz
itsRadio.nextWaveband();
for ( freq=522; freq<=1620;
    sprintf ( testName, "CDW
    f = itsRadio.getItsCurre
    RTC_ASSERT_NAME(testName,
    testNum++;
    itsRadio.getItsCurrentWa
}
f = itsRadio.getItsCurrentWa
sprintf ( testName, "CDWhiteE
RTC_ASSERT_NAME(testName, (f
```



シーケンス図

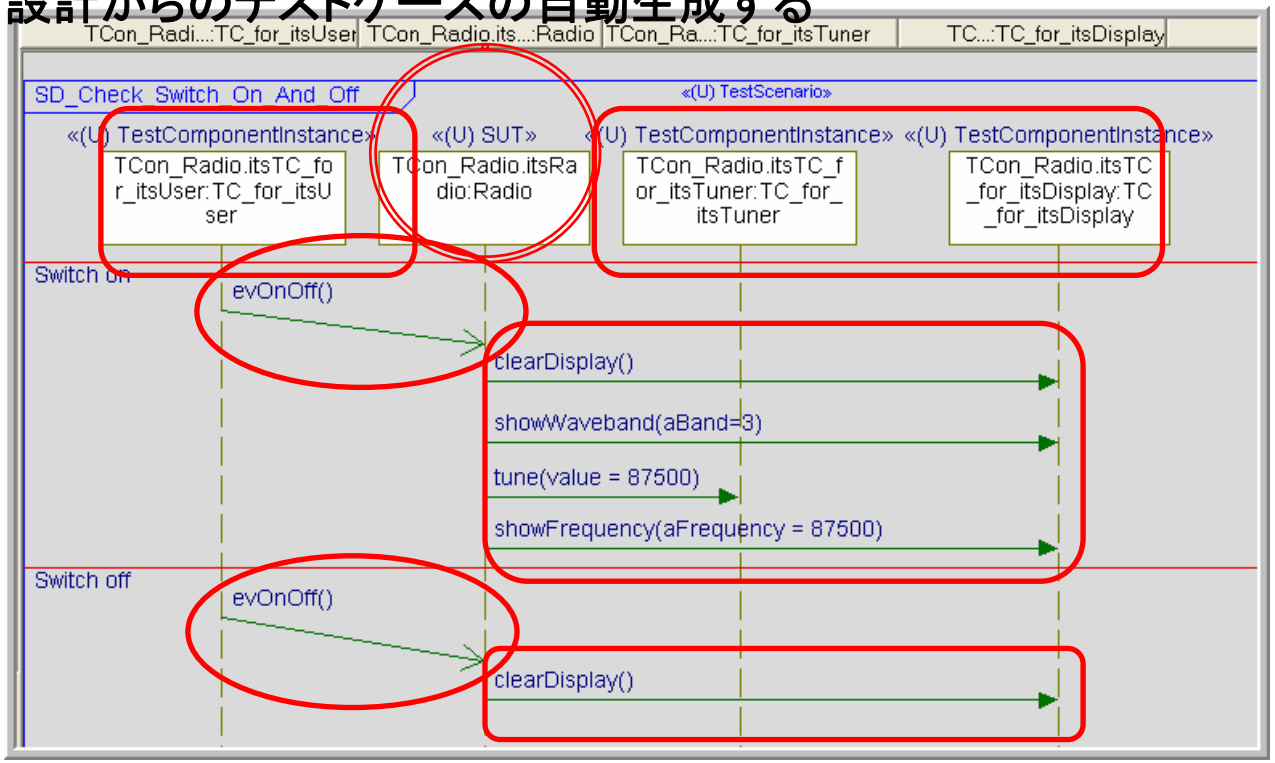
フローチャート



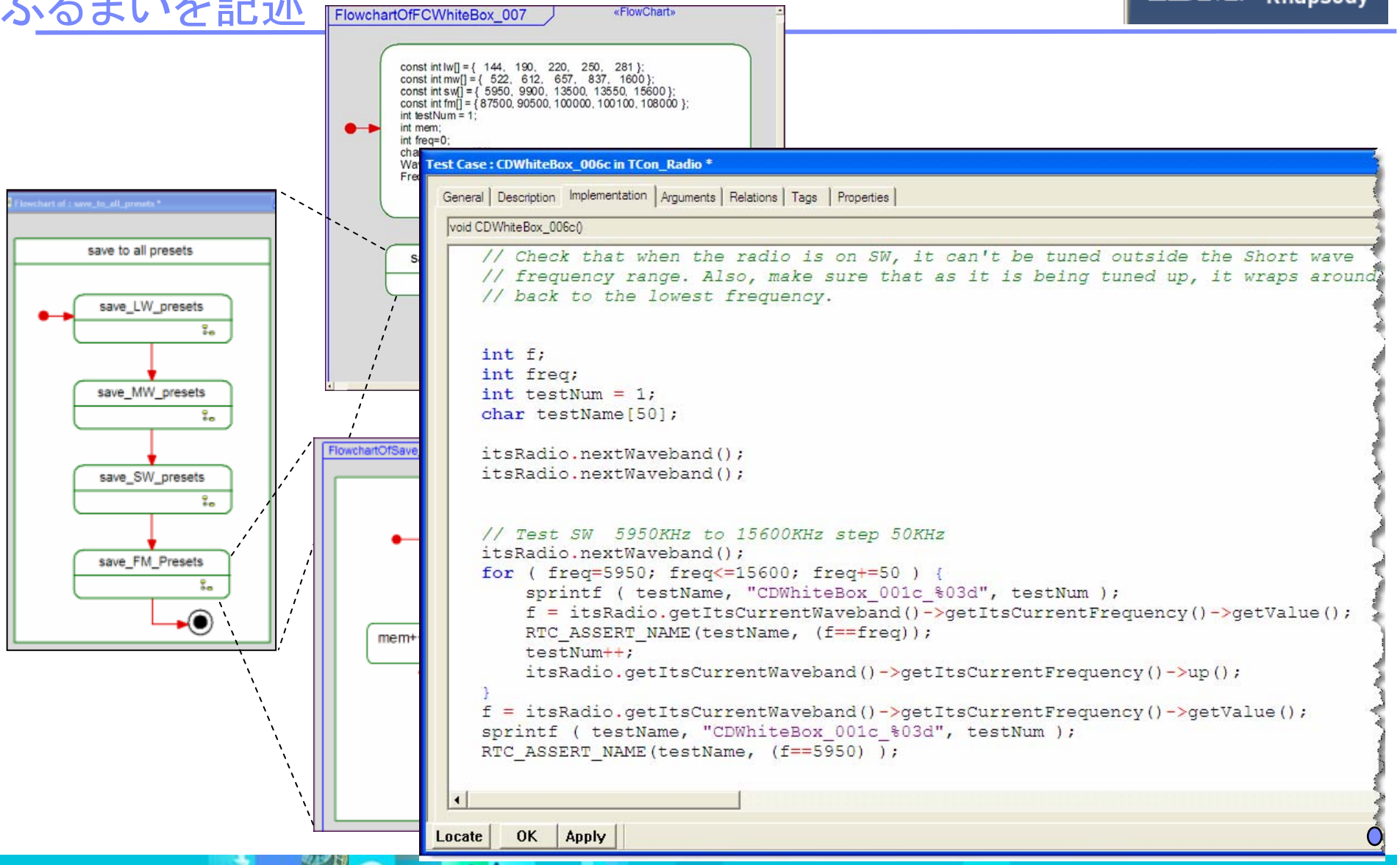
- テストをデスクトップ上でもターゲット上でも実施する

シーケンス図でテストケースのふるまいを記述

- シーケンス図を、*要求されたふるまい*を記述するグラフィカルなテストスクリプトとして使用する
- *リグレッションテスト*のテストケースとしてアニメーション化されたシーケンス図を記録し、再利用する。
- ATGを用い、設計からのテストケースの自動生成する

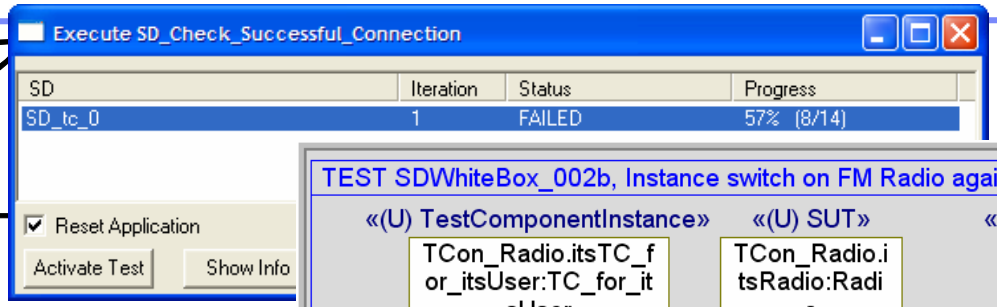


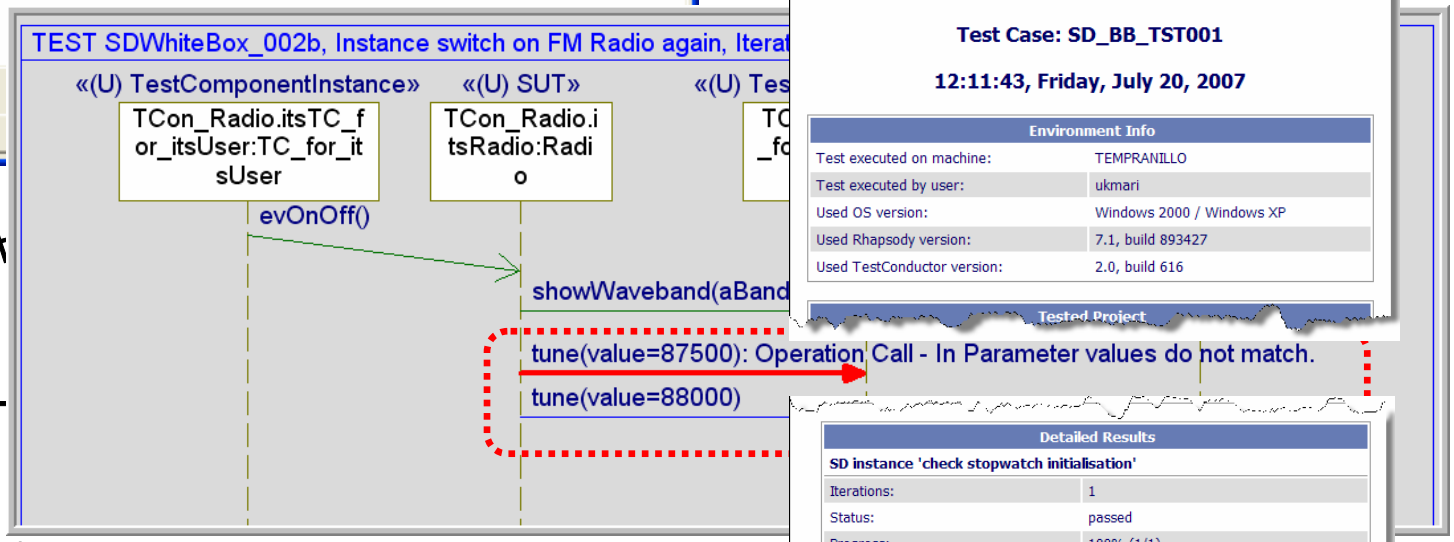
フローチャートやコードでテストケースのふるまいを記述



テスト実行とテストレポート

- テ
- シ
- コードとフローチャ
- コードでテストケー
- テスト実行し、レポート作成
  - SUTへの入力とスタブのふるまいは自動で動作
  - 予期せぬふるまいはハイライトされる
  - テスト実行結果レポートは会社/プロジェクトの標準に適応するようカスタマイズ可能





Test Case Result

Test Case: SD_BB_TST001

12:11:43, Friday, July 20, 2007

Environment Info	
Test executed on machine:	TEMPRANILLO
Test executed by user:	ukmari
Used OS version:	Windows 2000 / Windows XP
Used Rhapsody version:	7.1, build 893427
Used TestConductor version:	2.0, build 616

Tested Project

Detailed Results	
SD instance 'check stopwatch initialisation'	
Iterations:	1
Status:	passed
Progress:	100% (1/1)
SD instance 'check stopwatch start'	
Iterations:	1
Status:	passed
Progress:	100% (8/8)
SD instance 'check stopwatch stop'	
Iterations:	1
Status:	passed
Progress:	100% (3/3)
SD instance 'check stopwatch restart'	
Iterations:	1
Status:	passed
Progress:	100% (7/7)
SD instance 'check stopwatch reset'	
Iterations:	1
Status:	passed
Progress:	100% (3/3)

まとめ：なぜモデル駆動テスティング？

- **早期のフィーチャ駆動テストが手戻り防止に重要**
  - 早期のテストは、エラーを後工程に広めてしまわないために重要
  - テストを何回でも繰り返せることが重要
  - デバッグしなければならない範囲を限定することが重要
  - 認可を得るために重要
- **これまでのテストの困難さを解決**
  - 入出力用のテストクラスやテスト用インフラを作る (i.e., Logger, Reporter, Executer)
  - ターゲットなしのテスト
  - 開発チームで一貫性を保つ
  - 開発スケジュールに合わせる
  - システムモデル設計時からテストする

定期健診と
予防が大切です！

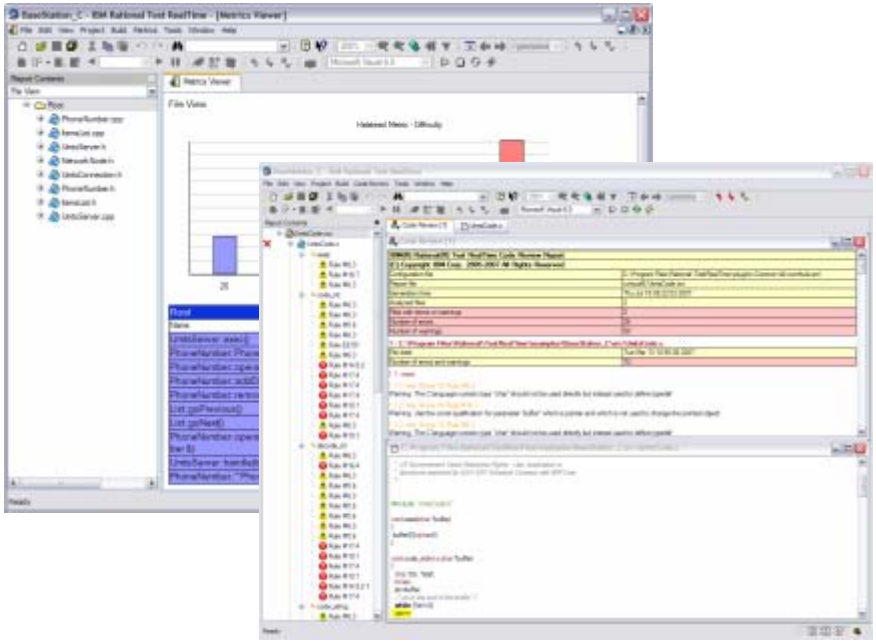
手軽に検診
できることも重要
です

- ところで、コードのテストはしなくていいの？
  - 全てのボタンを一緒に押したら、クラッシュしない？
  - デッドコードはない？
  - 全てのコードは一度は実行されたことがある？最初に実行するのがお客様だったりしないか？

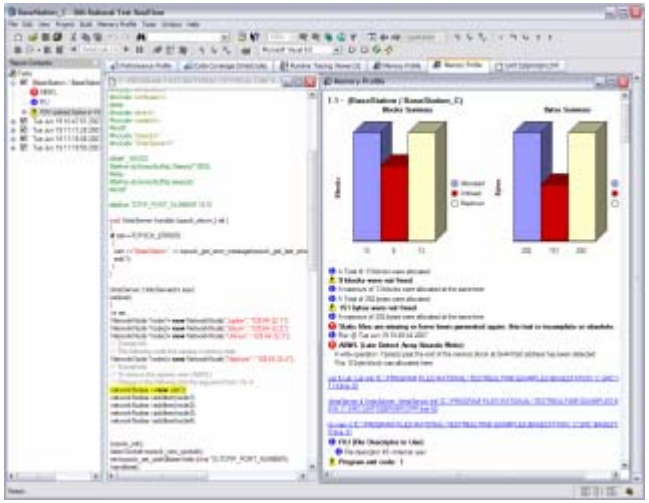
Test RealTimeによる構造ベーステスト

アプリケーションの品質を保証するためにコードを分析

- 静的分析
  - メトリクスレポート
  - MISRA-C:2004 コンプライアンスレビュー

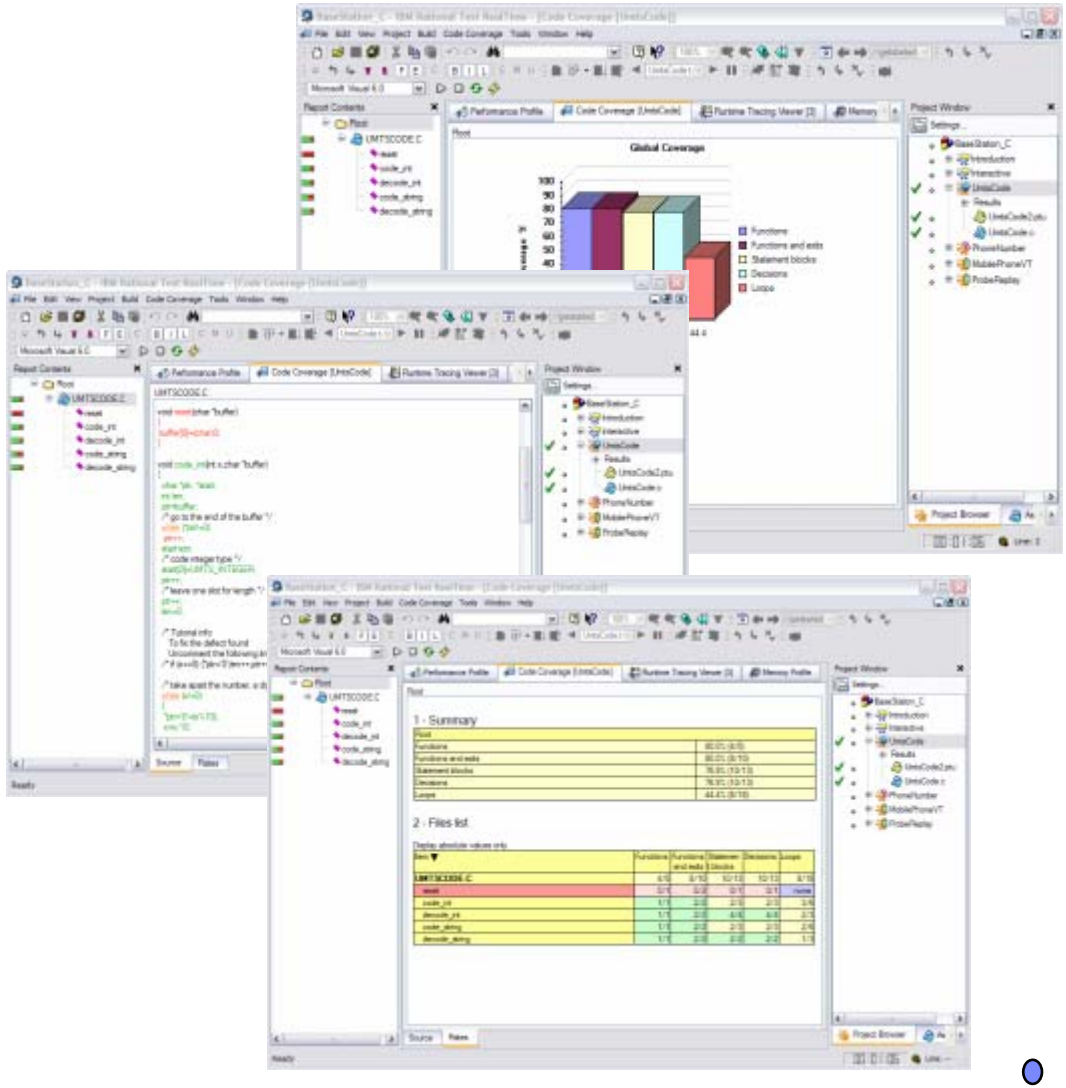


- ランタイム分析
  - コードカバレッジ
  - メモリプロファイリング
  - パフォーマンスプロファイリング



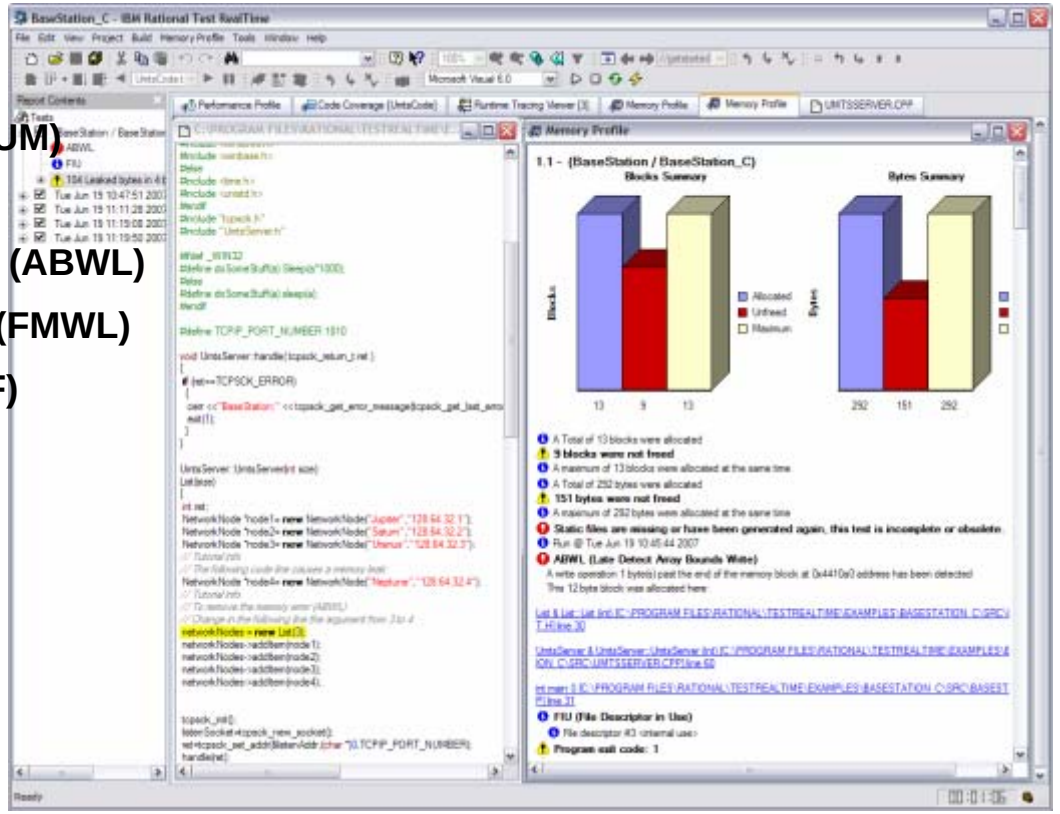
ランタイム分析：コードカバレッジ

- プログラム実行の内部を探る
- 使用されない　かつ/または　テストされなかったコードの特定
- C/C++ではMC/DC カバレッジレベル
- テストケースやコードと、全ての分析との直接相互関係



ランタイム分析：メモリプロファイリング

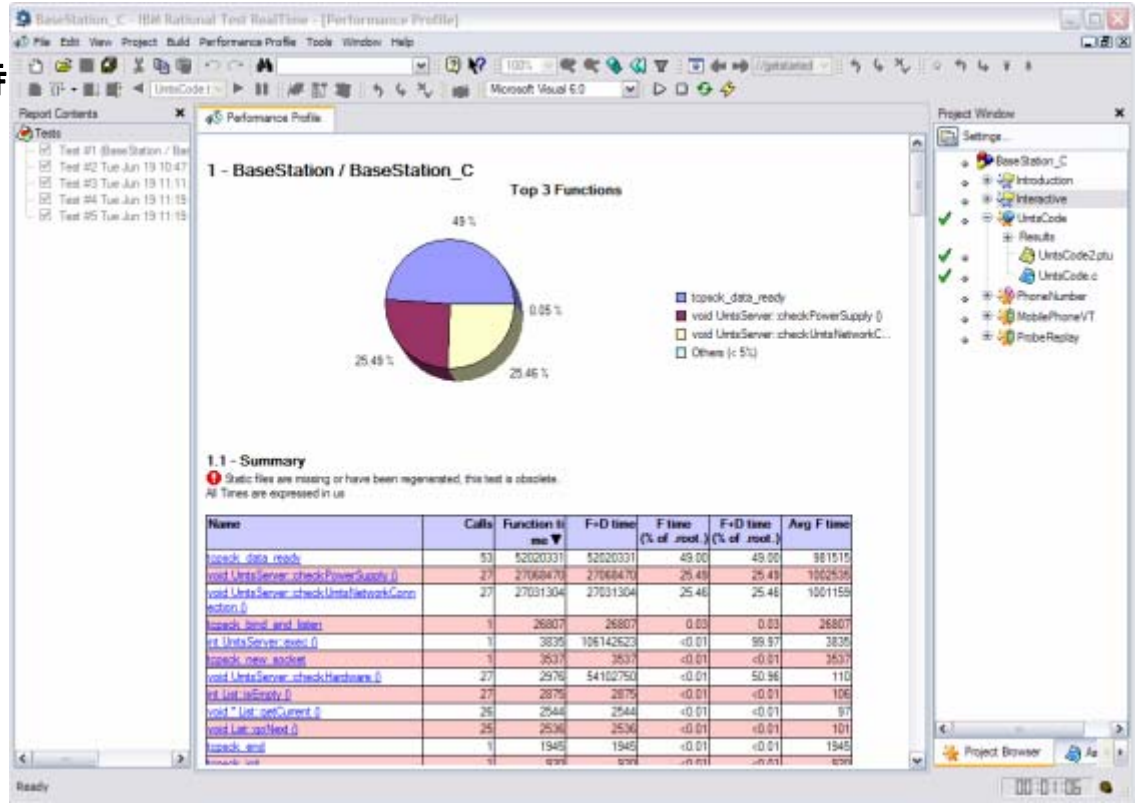
- 発見困難なメモリ問題を発見
  - エラーメッセージ
    - Freeing Freed Memory (FFM)
    - Freeing Unallocated Memory (FUM)
    - Freeing Invalid Memory (FIM)
    - Late Detect Array Bounds Write (ABWL)
    - Late Detect Free Memory Write (FMWL)
    - Memory Allocation Failure (MAF)
    - Core Dump (COR)
  - ワーニングメッセージ
    - Memory in Use (MIU)
    - Memory Leak (MLK)
    - Potential Memory Leak (MPK)
    - File in Use (FIU)
    - Signal Handled (SIG)



- テストケースやコードと、全ての分析とが直接相互関係

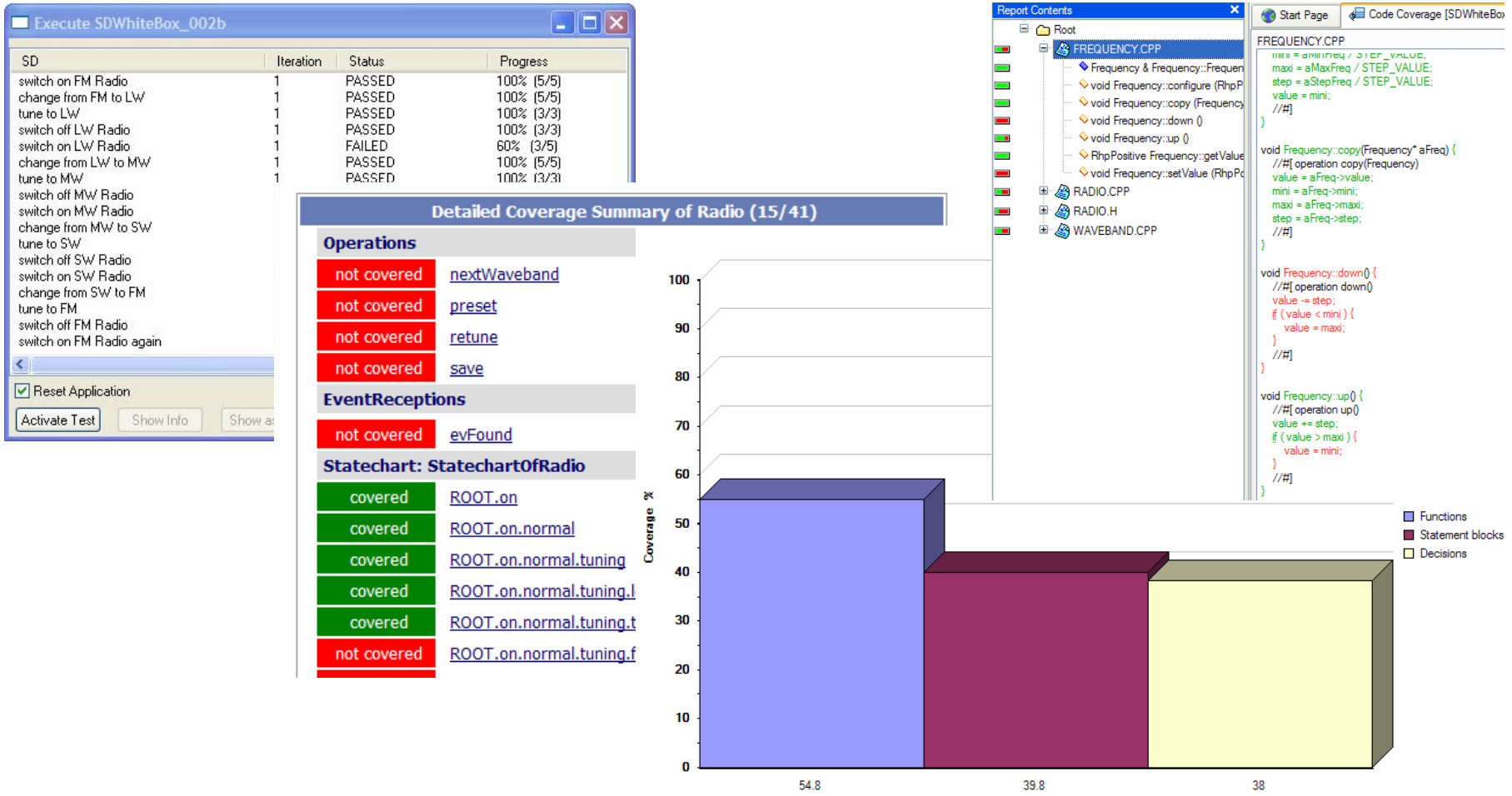
ランタイム分析：パフォーマンスプロファイリング

- アプリケーションのどの部分が実行時間を消費しているかをグラフィカルに表示
  - トップ関数グラフ
  - パフォーマンス概要レポート

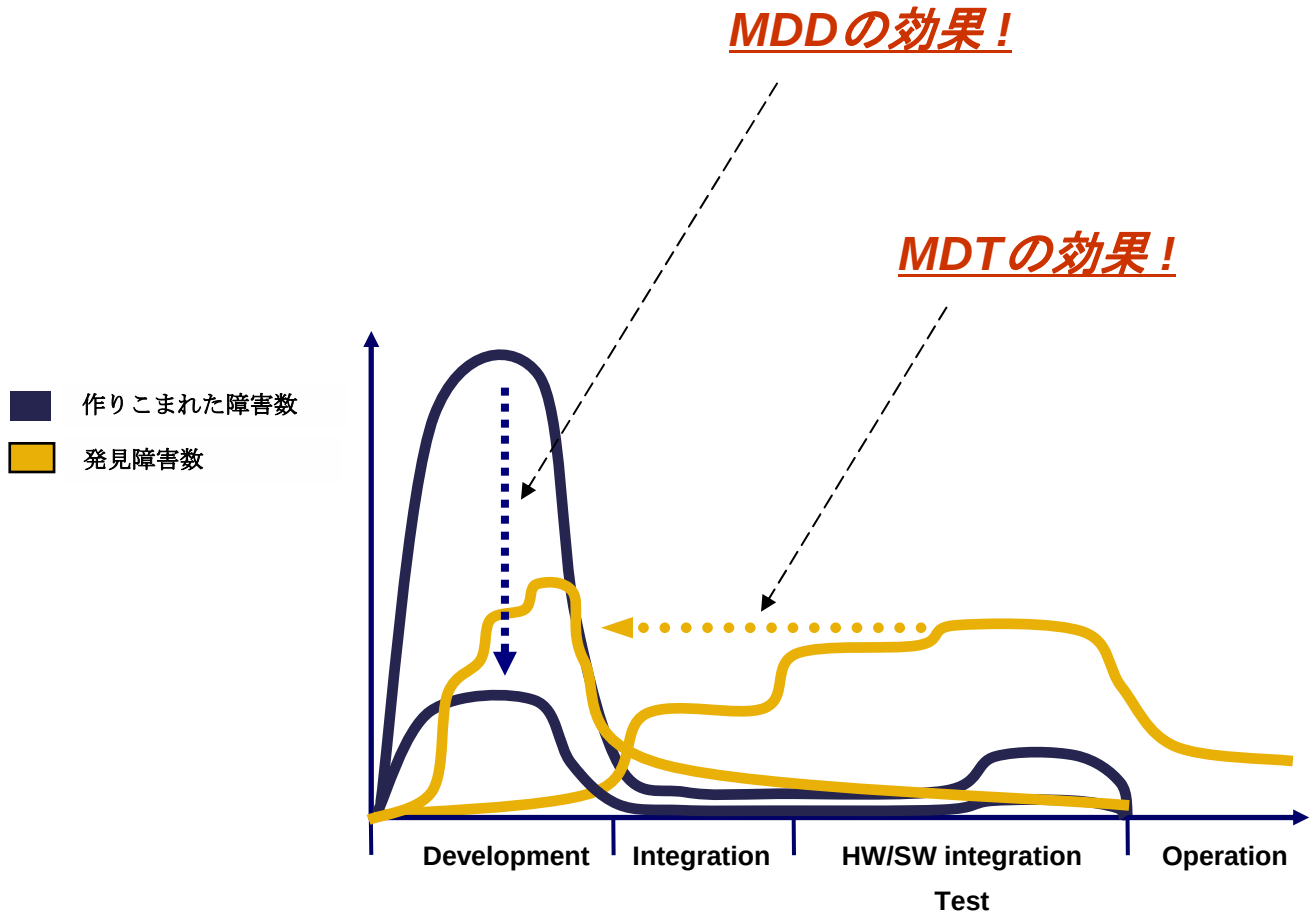


テスト結果、モデルカバレッジ、コードカバレッジ

- シミュレーション実行、ターゲット実行、それぞれから得られたテスト結果



まとめ – MDD & MDTと障害の経済学



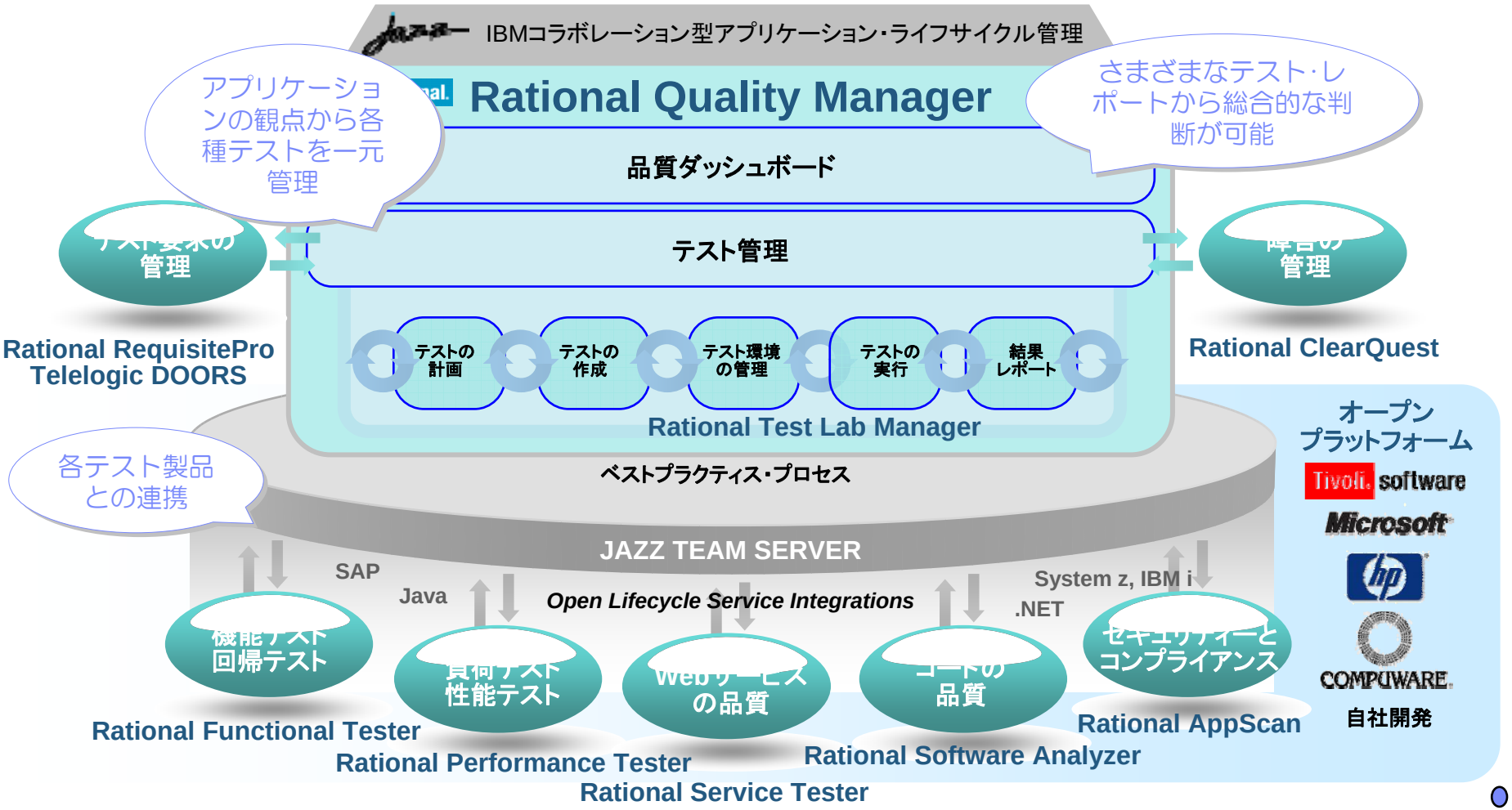
まとめ

- MDD、MDTは、すでに実行可能、実用段階です
- MDDによって
  - 「あまりに多くの欠陥が作りこまれている」→「欠陥の作りこみ」が減少します
- MDTによって
  - 「欠陥が発見された時点では遅すぎる」→開発初期に欠陥を除去可能です
  - MDDによる開発プロセスをより効率的にする効果があります
- Telelogic Rhapsodyによって、
  - 要求、設計、テストなど開発に必要な各モデルが有機的に連携・組織化されます
  - 実行可能モデルによるシミュレーションがMDDを強く推進します
  - テストアーキテクチャの自動生成、テストの自動実行がMDTを強く推進します
- Telelogic Rhapsody – Test Realtime連携によって
  - 設計の妥当性と正当性が共に検証可能となります
  - 「本当に「正しいフィーチャ」を正しく開発しました/していますか？」

IBMのトータルテストソリューションについて

Rational トータル・テスト管理ソリューション

あらゆる種類のテストとテスト・プラットフォームを、ライフサイクル全体にわたってサポートする



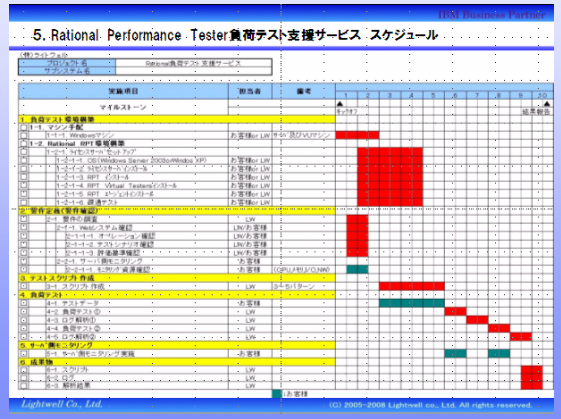
品質ダッシュボード


Mozilla Firefox
File Edit View History Bookmarks Tools Help
Getting Started Latest Headlines
Rational Quality Manager
Welcome Katie | Project 001
Sign Out
Home Planning Construction Lab Management Execution Analysis Help
Configure this page
Katie's Home
My Tasks
Newly Assigned (3 Items)
Verify Build Verification test
Review Properties Test Case
Create GUI Install Test Plan
Today (3 Items)
Complete Project ABC Test Plan
Review Properties Test Case
Complete GUI Install Test Plan
Future (1 Item)
Review Lab Quality Test
Event Log
View all
Build 0986 has successfully finished 30 min ago
Machine 0892-x2 has been reserved 1 h 20 min ago
Build 0985 is ready 5 h ago
Build 0984 is ready 10 h ago
Machine 098-Linux has been reserved Yesterday, 10:00 PM
Build 0983 encountered errors Nov. 12, 1:20 PM
Build 09261 is ready Nov. 10, 10:00 AM
Reservation expired for Machine Sierra Nov. 5, 1:00 PM
Build 98142 encountered errors Nov. 5, 10:00 AM
Team Load
Katie Testlead (20 Items Assigned, 10 Complete) Behind
Ted Tester (13 Items Assigned, 10 Complete) On Time
Terry Tester (15 Items Assigned, 10 Complete) Behind
Uma Requester (21 Items Assigned, 16 Complete) On Time
Overall Progress
Test Case Points
Projected Complete
Projected Attempted
Actual Complete
Actual Attempted
Milestone
Project ABC Milestone 4 Start: Nov. 15, 2007 Target End: Dec. 31, 2007 Behind
© IBM Corporation 1996, 2007
Powered by jazz
新しく割り当てられたタスク
本日の作業
イベントログ（自動テストの結果 レビュー ステータス など）
チーム全体の進捗状況（テストケース数の予定と実績）
個人ごとの進捗状況（テストケース消化率）
マイルストーンの達成状況
Web
Eclipse
Visual Studio

IBM Rational SaaS型 テスト・ソリューション

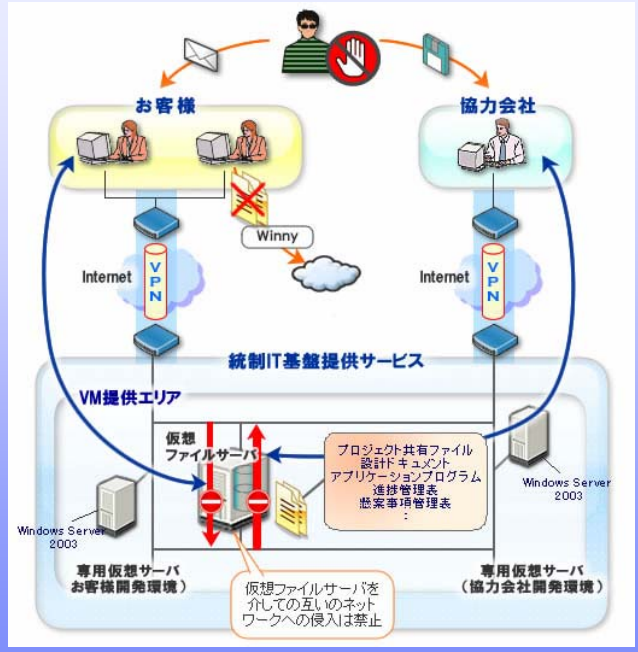
- **SaaS型テスト・ソリューションのメリット**
  1. **比較的すぐ使える**（H/W、S/Wのセットアップを簡素化）
  2. **すきな期間 使える**
  3. **コスト削減**

ビジネス・パートナー・ソリューション
（株）ライトウェル：負荷テスト支援サービス



負荷テストそのものを支援します

ビジネス・パートナー・ソリューション
日立ソフト（株）：Secure Online



仮想マシン上で機能テスト、負荷テスト・ツールのご利用が可能となります

ありがとうございました

hisashi@jp.ibm.com