# フレームワークを活用した安定稼働の実現

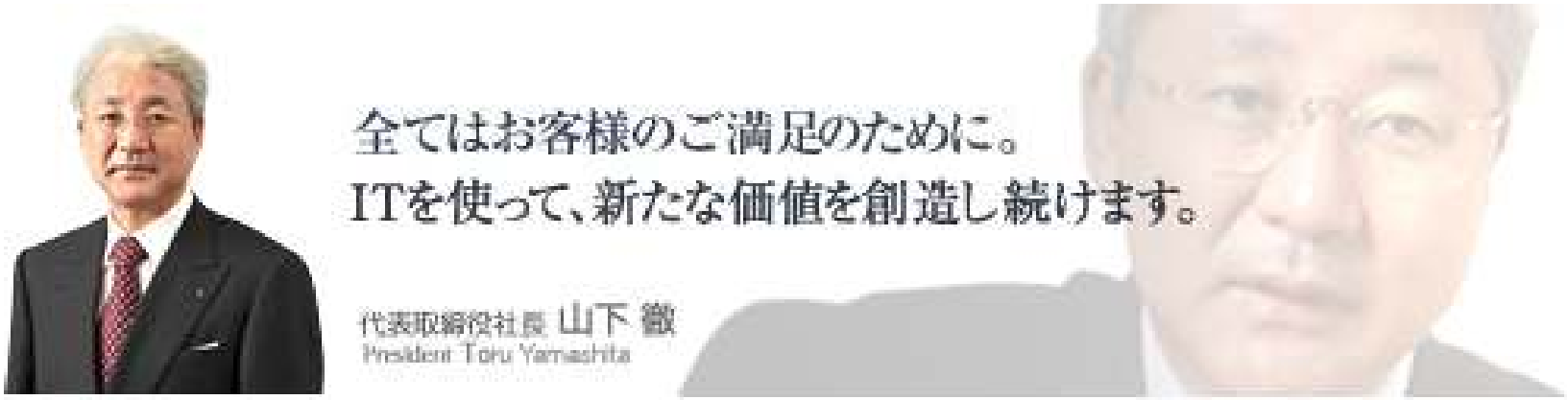
平成20年9月19日

株式会社 NTTデータ
技術開発本部
ソフトウェア工学推進センタ
木村　利幸

# 会社紹介

## NTTデータ概要

- 主な事業内容：システムインテグレーション事業
- 設立年月日 ：1988年（昭和63年）5月23日
- 売上高：1,074,405百万円（平成19年度）
- 経常利益 ：94,342百万円（平成19年度）
- 従業員数：8,550名（平成20年3月31日現在）



# 本日のアジェンダ

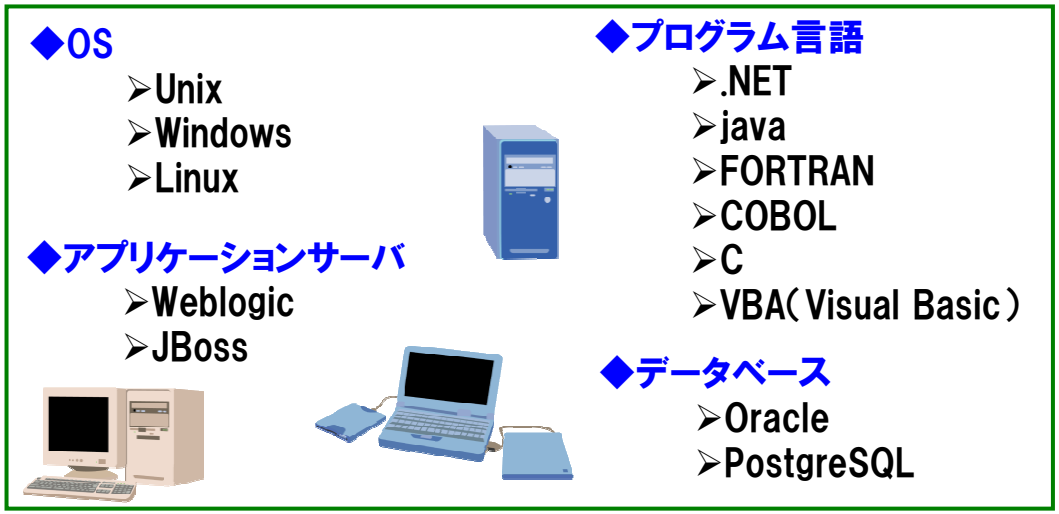
- 開発の秩序と規約
- 安定稼働とフレームワーク
- TERASOLUNA

- **開発の秩序と規約**
- 安定稼働とフレームワーク
- TERASOLUNA

# システム開発における秩序

- システム開発の選択肢は広がる一方
  - OS、言語、サーバの選択肢が豊富
  - SOA、マッシュアップなどの開発手法が次々と台頭
  - AJAX、JAXBなどの開発技術も次々と台頭

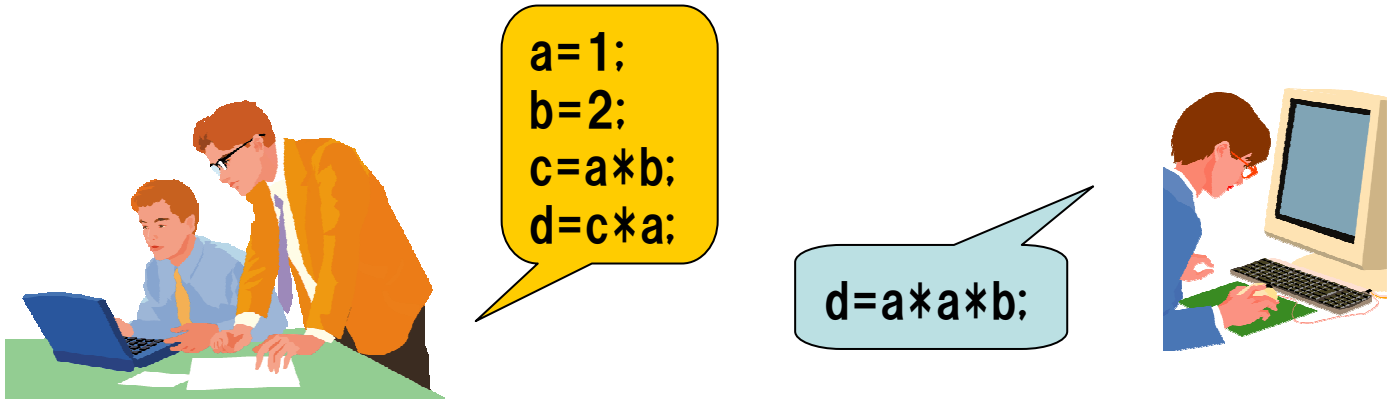
**システム構成要素の選択肢の例**




**近年のシステム開発は、秩序が保たれない状況に陥りやすい**

# システム開発における規約

- システム開発の大枠が決まった後にも落とし穴が
  - 特定の手法や技術の中にも複数の表現や方法が存在
  - 同じことを実現するための手段が多数
    - データベースとのやりとりの方式
    - プログラムの記述表現やコメントの付け方、変数の命名方法
  - 解釈や手段がバラバラ化し、確認が困難、再利用性が失われる




**開発を一定の品質に保つことが困難になっている**

# 品質確保の課題

## ■ 秩序と規約が保たれず、品質で問題が発生した事例

### バッチ処理が遅い

- ある基幹系Webシステムで性能評価を実施したところ、夜間バッチ処理の一部が目標を6時間もオーバーするという問題が発覚しました。しかしDBサーバ上で完結するシンプルな処理だったことも幸いし、DBテーブルにインデックスを追加するだけで処理時間が短縮し、性能目標を達成することができました。

### メモリの無駄使い

- あるWebシステムでの性能評価でのこと、負荷生成ツールから多重でリクエストを送信すると、JavaのGC(ガベージコレクション)が多発して目標の性能が出ないことが判明しました。Javaのプロファイラを利用して原因を特定し、プログラム修正で問題を解決することができました。

**いずれも、処理方式や実装方法に関する規約を整備することで早期発見・回避が可能**

※@IT記事より引用　NTTデータ　岡安一将氏
攻めの性能マネジメント　性能問題の失敗事例集
http://www.atmarkit.co.jp/im/carc/serial/attack02/attack02.html

# 安定稼働への貢献

- システム開発の品質を確保するには
  - 処理方式や実装方法に関する規約を整備
    - コーディング規約の徹底
    - 実装規約のドキュメント化
    - 社内標準や業界標準への準拠
    - *フレームワークの利用　→　本日のテーマ*

**フレームワークでアプリケーション基盤を統一**

↓

**バラバラな設計・実装を抑え、一定の品質を確保**

↓

**安定稼働に貢献！**

- 開発の秩序と規約
- **安定稼働とフレームワーク**
- TERASOLUNA

# 安定稼働の定義

## 安定稼働とは

- RASIS：安定稼動の要素
  - **信頼性** (Reliability) 故障が少ない
  - **可用性** (Availability) 稼働率が高い
  - **保守性** (Serviceability) 障害箇所の発見が容易
  - **保全性** (Integrity) データの矛盾が発生しない
  - **機密性** (Security) 不正アクセスを防止

- 安定稼働を実現するには
  - RASISの各要素について対策が必要
  - 性能設計やチューニングの実施も不可欠

# 安定稼働の実現手段

## ■ RASISの各要素に関する対策と実現手段

| RASIS要素 | 対策例 | 実現手段例 |
|---|---|---|
| **信頼性**<br>故障が少ない | 故障要因を排除 | ・高信頼性のハードウェアを採用する<br>・実績のあるアーキテクチャを採用する |
| | 運用開始時までのバグ刈り取りを徹底 | 試験工程に時間をかけ、バグを収束させる |
| **可用性**<br>稼働率が高い | 障害発生時の影響範囲の局所化、障害回復の迅速化 | サーバ構成を2重化して切り替え可能にする |
| | 点検などの運用停止スケジュールの低減 | メンテ頻度を抑えたシステム設計を行う |
| | 障害時のサービス切り替えまでの時間を短縮 | 障害時の運用切り替え手順書を作る |
| | 冗長可し、障害の影響範囲を縮小化 | 各ハードウェアを複数台構成にする |
| | データとプログラムを分散保管 | 分散管理可能な設計を行なう |
| | 障害時の損失データを削減、復旧容易性向上 | エラー時の復旧をサポートする機能を持ったサーバやDBを使用する |
| | リカバリ時の復旧ポイントを多数設置 | リカバリ頻度を増やす設計を行う |
| | サービス停止時の代替運用 | 代替となる環境や人員を用意しておく |

# 安定稼働の実現手段

## ■ RASISの各要素に関する対策と実現手段

| RASIS要素 | 対策例 | 実現手段例 |
|---|---|---|
| 保守性<br>障害箇所の発見が容易 | 障害検知方法や障害情報収集方法の確保 | ・障害検知が可能な機能を付けたり、検知可能な構成で設計する<br>・オープンソースを採用して開発する |
| | 障害からの回復時間を短縮 | ・障害発生時の対処フローを整備しておき、調査分析のスペシャリストを手配する。<br>・障害を想定した設計を行っておく |
| | 運用監視の範囲を拡大、詳細度を高度化 | 監視ソフトウェアの監視範囲や詳細度を高いレベルに設定しておく |
| | 障害時の配備人数の増員、現場到着の迅速化と役割分担の徹底 | 障害時の対処にむけて組織体制を整える |
| | 障害時の運用を規定 | 障害時の分析調査・復旧マニュアルを作る |
| | 致命的な故障に至る前に予兆を検出 | 監視ソフトウェアを常時動かすことでアラームを上げる |
| 保全性<br>データの矛盾が発生しない | データ保存方式の統一<br>記録媒体の多重化　など | ・排他制御の方式を規定する<br>・データをミラーリングして管理、随時比較修正する構成にする |

# 安定稼働の実現手段

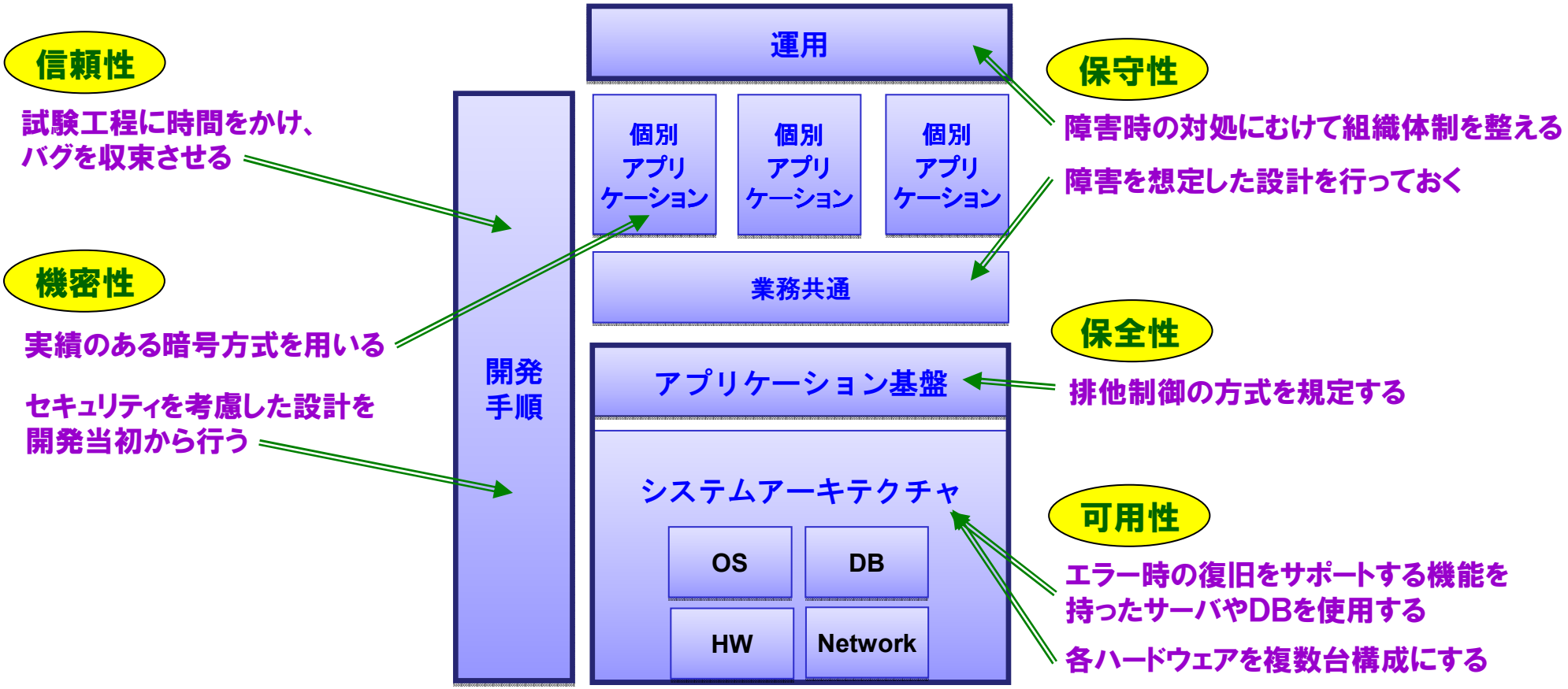
## ■ RASISの各要素に関する対策と実現手段

| RASIS要素 | 対策例 | 実現手段例 |
|---|---|---|
| **機密性**<br>不正アクセスを防止 | セキュリティを強化 | セキュリティ担当者を設け、セキュリティを考慮した設計を開発当初から行う |
| | 各種セキュリティ標準に準拠 | 社内標準や業界標準、国際規格などで指定されている基準を満たすよう設計する |
| | 災害や人災への対策 | ・災害時の対処方法を運用ルールとして定める<br>・入室管理や認証チェック機能を用いる |
| | 監査ログの取得 | ログデータの作成頻度を詳細に設定し、いつでも取得できる仕組みを作る |
| | パスワードを設定 | パスワードのルールを厳しく設定する |
| | 暗号化 | 実績のある暗号方式を用いる |
| **その他**<br>チューニング | 負荷の把握と制御<br>処理制御<br>リソース制御<br>環境設定　など | ・業務量の最大をあらかじめ把握しておき、負荷に耐えられるよう設計する<br>・暗号化の範囲を最適に設定し、処理を減らす<br>・データの保存期間を最適に設定し、リソース不足を防ぐ<br>・CPUやメモリの性能拡張性を確保しておき、後々の要望や問題に答えられる環境を整える |

# 実現手段の範囲

## ■ 安定稼働を実現する手段の分散具合

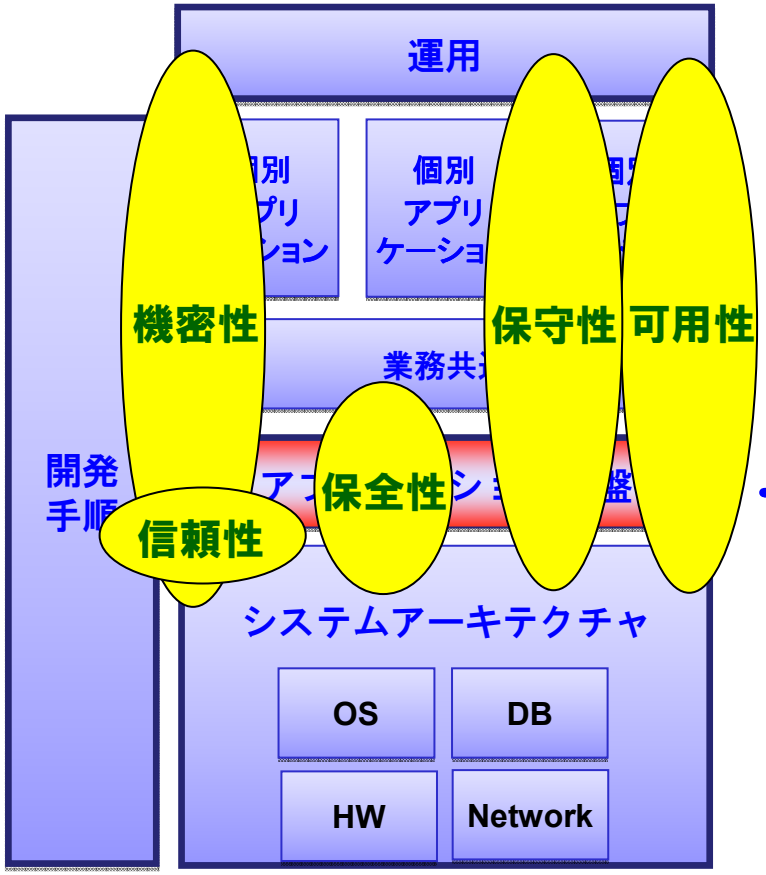
- 様々なレイヤーに関係している



# フレームワークの貢献範囲

## ■ フレームワークが安定稼働に貢献する部分

- 全ての要素に関連している




**アプリケーション基盤 ＝ フレームワーク**

- ■信頼性：実績のあるアーキテクチャ
- ■可用性：障害の影響範囲を局所化
- ■保守性：故障原因の絞込みが容易
- ■保全性：制御方式を統一
- ■機密性：最新のセキュリティに対応

**規約に則った開発で安定稼働を実現！**

# フレームワークの定義

## フレームワークとは

- IT用語辞典( http://e-words.jp/ )より引用
  - アプリケーションソフトを開発する際に頻繁に必要とされる汎用的な機能をまとめて提供し、アプリケーションの土台として機能するソフトウェアのこと。開発にフレームワークを利用すると、独自に必要とされる部分だけを開発すれば済むため開発効率の向上が見込める。具体的なソフトウェアだけでなく、汎用的に適用できるプログラムの設計モデルや典型的な処理パターンなどを含めてフレームワークと呼ぶ場合もある。
- フレームワーク誕生の背景

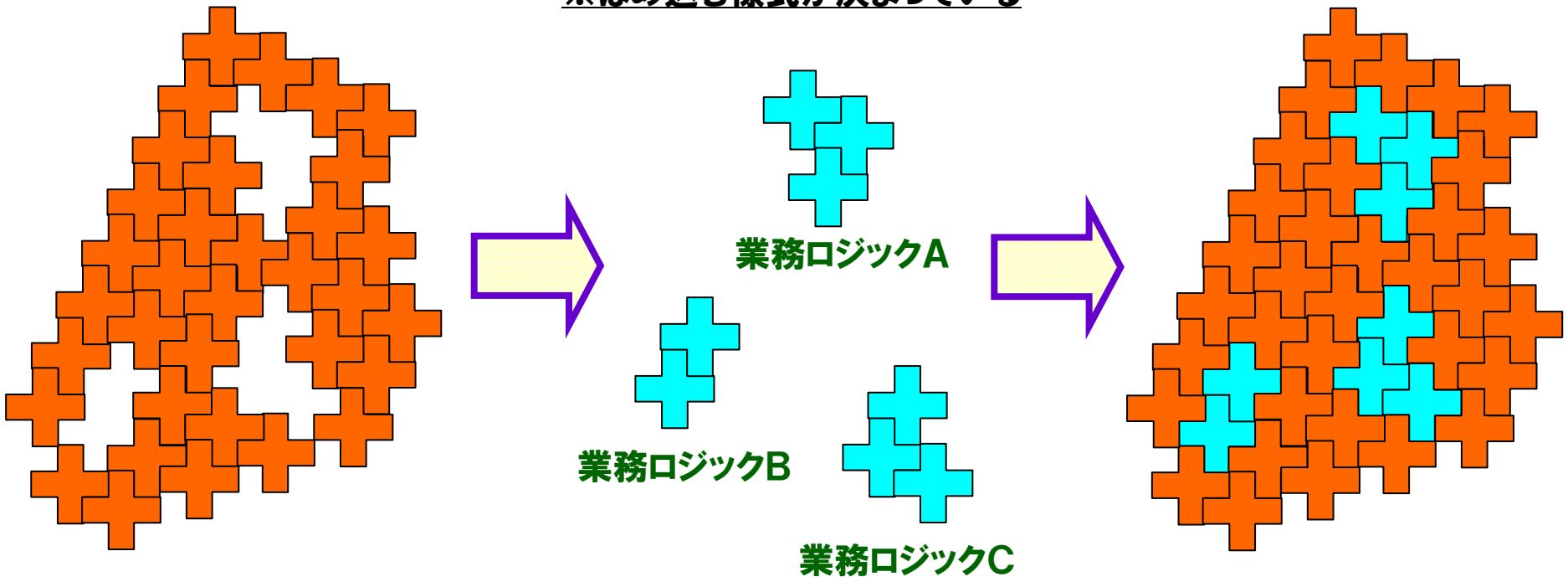
# フレームワークの使われ方

## フレームワークの利用イメージ

- アプリケーションの雛形

1. フレームワーク選択　　2. 必要箇所のみを開発　　3. アプリケーション完成

※はめ込む様式が決まっている



# フレームワークのメリット

## フレームワークを使うことで得られるメリット

- **アプリケーション基盤のアーキテクチャが統一される**
  - 細かい処理方法、手順、仕組み、管理方式などが統一され、実装のばらつきが抑えられることで、エラーや不具合の発生を抑制
  - 開発者ごとのスキルのばらつきを吸収、インタフェースを規定
- **必ず用いる共通的な機能をフレームワークが代行してくれる**
  - 業務に特化する部分のみを注力して開発できるので、作り込みの工数が減り、高品質なアプリケーションを構築できる
- **規約に則ることで、品質レベルが保証される**
  - 複数人数での作業時もプログラムコードが共有可能、再利用性が向上する
- **【デメリット】FWの学習コストがかかる**
  - FWの特徴を把握し、効率的に利用できるまでの学習コストが必要

# フレームワークの種類

■ 言語の分類と、オープンソースか商用かの分類がある

## .NET系フレームワーク例

■.NET　■ASP.NET

## Java系フレームワーク例

■Struts　■Spring
■Tapestry　■Turbine
■JSF　■WebWork2

### フレームワーク利用環境の比較


@IT：特集　ASP.NET vs Strutsフレームワーク徹底比較
著者：山田 祥寛　　Web記事より図を引用
http://www.atmarkit.co.jp/fdotnet/special/aspstruts01/aspstruts01_02.html

## 商用フレームワーク例

■intra-mart（NTTデータイントラマート）
■InterStage Apcoordinator（富士通）
■ObjectWorks+ /STDIO（野村総合研究所）

# フレームワークの代表的な機能

- Strutsが提供する機能の例
  - コントローラサーブレット機能
  - JSPで利用するタグライブラリ
  - JavaBeansプロパティの自動設定機能
  - 表示の国際化を支援する機能
  - ログ出力機能
  - 入力データ検証機能(Validator)

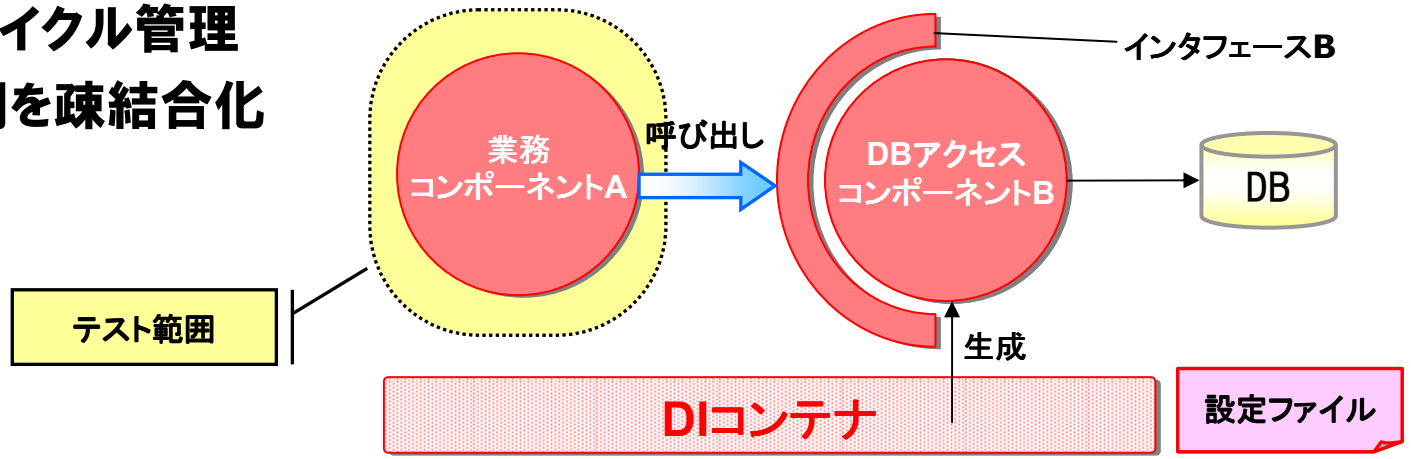
# フレームワークの代表的な機能

- Springが提供する機能の例
  - DIコンテナ
  - AOP(Aspect Oriented Programming)
  - MVCモデル実装機能
  - JDBC抽象化
  - Webインテグレーション機能
  - ORMインテグレーション機能

# 代表的な機能の特徴やメリット

## DIコンテナ（Spring）

- オブジェクト間依存関係の管理機能を備えた実行基盤
  - ライフサイクル管理
  - クラス間を疎結合化




**＜メリット1＞ メンテナンス性の向上**

ロジック変更やコンポーネント差し替え時の影響範囲を極小化できる

**＜メリット2＞ テスタビリティの向上**

モックを利用したテストが行えるため、単体試験でのテスタビリティが向上する

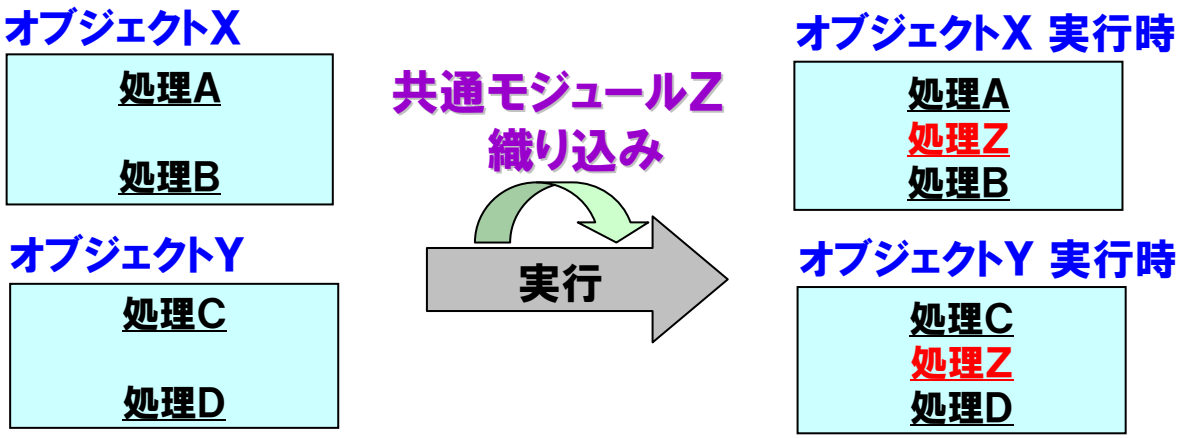
**＜メリット3＞ 再利用性の向上**

コンポーネントが特定のAPIに依存しないため、再利用の機会が増える

# 代表的な機能や特徴のメリット

## AOP（Spring）

- 複数のオブジェクトにまたがる処理をモジュール化する機能
  - 共通処理を分離
  - 後付で処理を織り込み




**＜メリット1＞ 保守性の向上**

共通モジュールとして切り出しているため、変更・修正箇所が絞り込める

**＜メリット2＞ 記述忘れの防止**

共通処理を各オブジェクトに埋め込む必要がなくなるため、記述忘れを防止

# 安定稼働と機能や特徴の対応

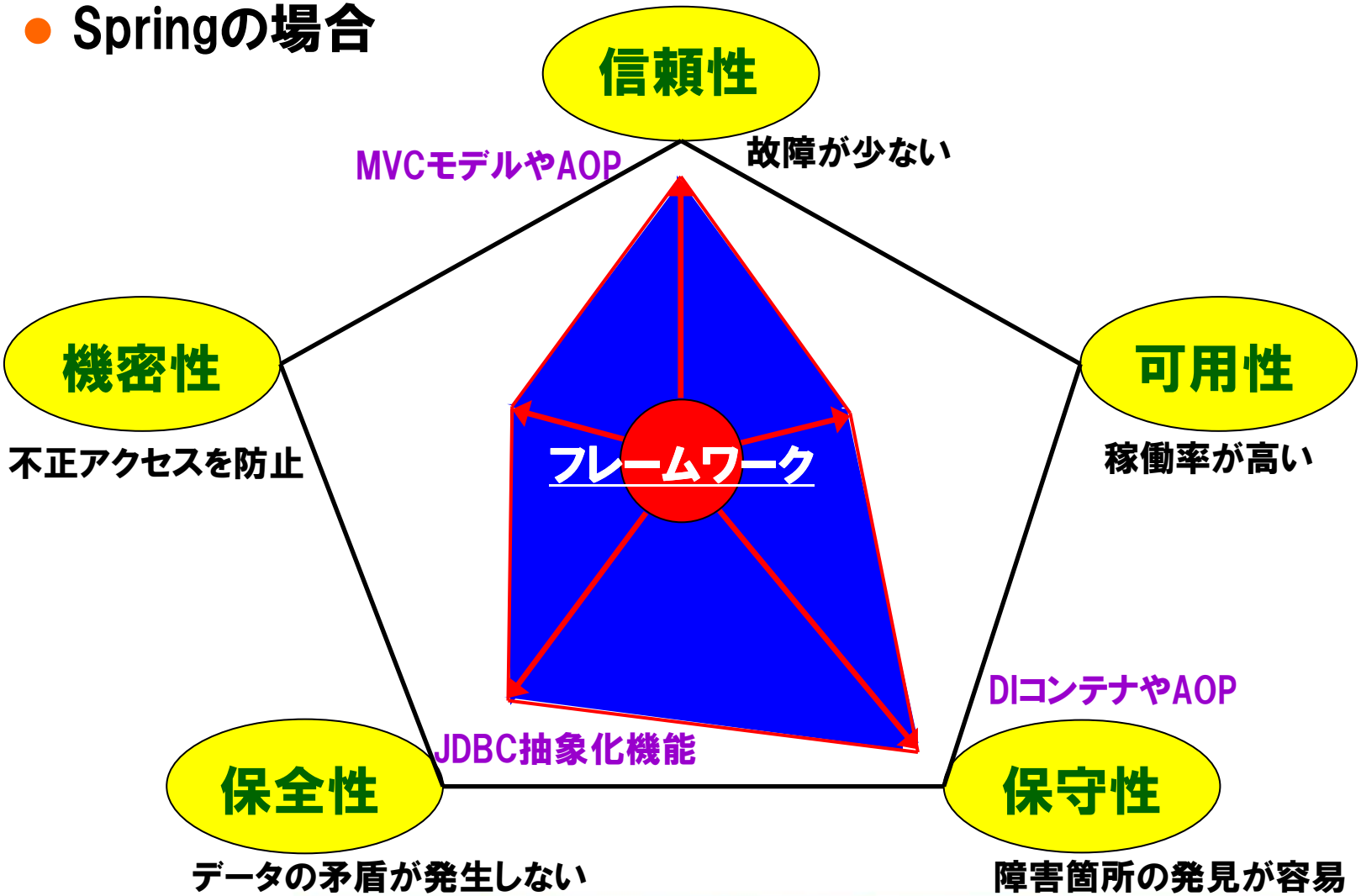
## ■ 安定稼働とフレームワークの関係（springの場合）

| RASIS要素 | フレームワークの貢献 | 対応する機能や特徴の例 |
|---|---|---|
| **信頼性**<br>故障が少ない | 実績のあるアーキテクチャ | MVCモデルやAOPにより、処理の構造を規定することで、コーディングの記述もれやミスを防止 |
| **可用性**<br>稼働率が高い | 障害の影響範囲を局所化 | ベースフレームワークでは不足気味<br>※商用フレームワークや機能拡張に依存 |
| **保守性**<br>障害箇所の発見が容易 | 故障原因の絞込みが容易 | DIコンテナやAOPにより、共通処理を切り出して管理することで、修正変更箇所の特定や絞込を効率的に実施 |
| **保全性**<br>データの矛盾が発生しない | 制御方式を統一 | JDBC抽象化機能により、データベースへのアクセス方式を統一することで、制御誤りを予防 |
| **機密性**<br>不正アクセスを防止 | 最新のセキュリティに対応 | ベースフレームワークでは不足気味<br>※商用フレームワークや機能拡張に依存 |

# 安定稼働と機能や特徴の対応

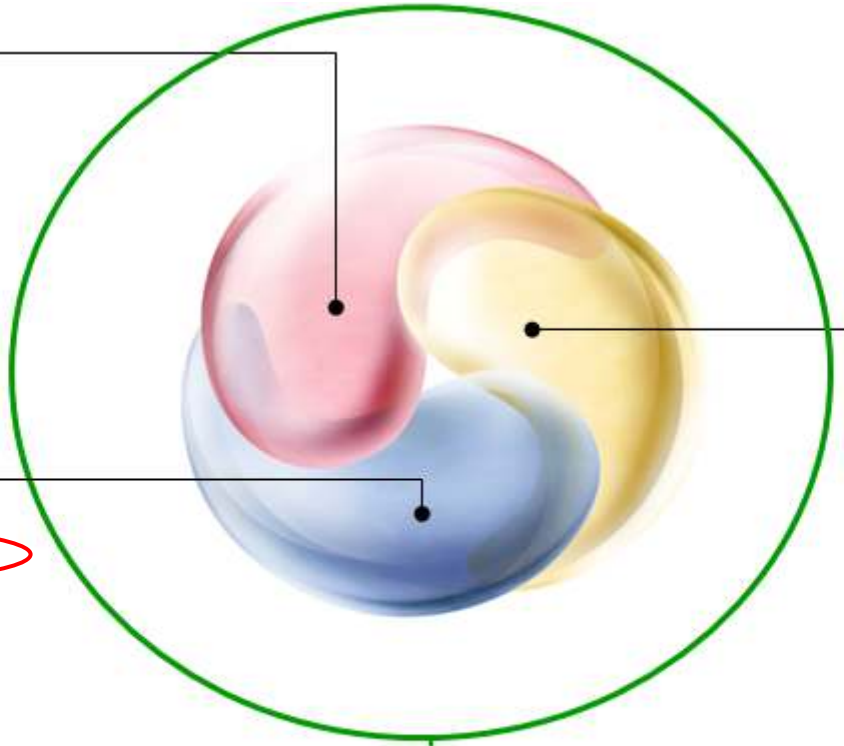
## フレームワークの貢献度合い

- Springの場合



- 開発の秩序と規約
- 安定稼働とフレームワーク
- **TERASOLUNA**

# TERASOLUNAとは



**開発プロセス**

- ソフトウェアアーキテクチャ中心
- IT基盤構築プロセスを明確化
- 業務改革をサポート
- ユニバーサルデザイン対応

**開発環境**

- オープンソースフレームワーク
- Java/.NET/Ajax対応
- リッチクライアントをサポート
- バッチをサポート
- 開発支援ツールの提供

**サポート**

- 開発メンバによるサポート
- 研修サービスを実施

TERASOLUNA®は「開発プロセス」「開発環境」「サポート」を一体としたNTTデータのトータルソリューションです。

# TERASOLUNAフレームワーク

## ラインナップ

- 2000年前後から整備を進め、現在のラインナップは以下の通り。

| | サーバ<br>フレームワーク | クライアント<br>フレームワーク | バッチ<br>フレームワーク |
|---|---|---|---|
| Java | TERASOLUNA<br>Server Framework for Java | － | TERASOLUNA<br>Batch Framework for Java |
| .NET | TERASOLUNA<br>Server Framework for .NET | TERASOLUNA Client Framework for .NET | － |
| Ajax | － | TERASOLUNA Client Framework for Ajax（マスカット） | － |

- 全てオープンソースとして無償公開中

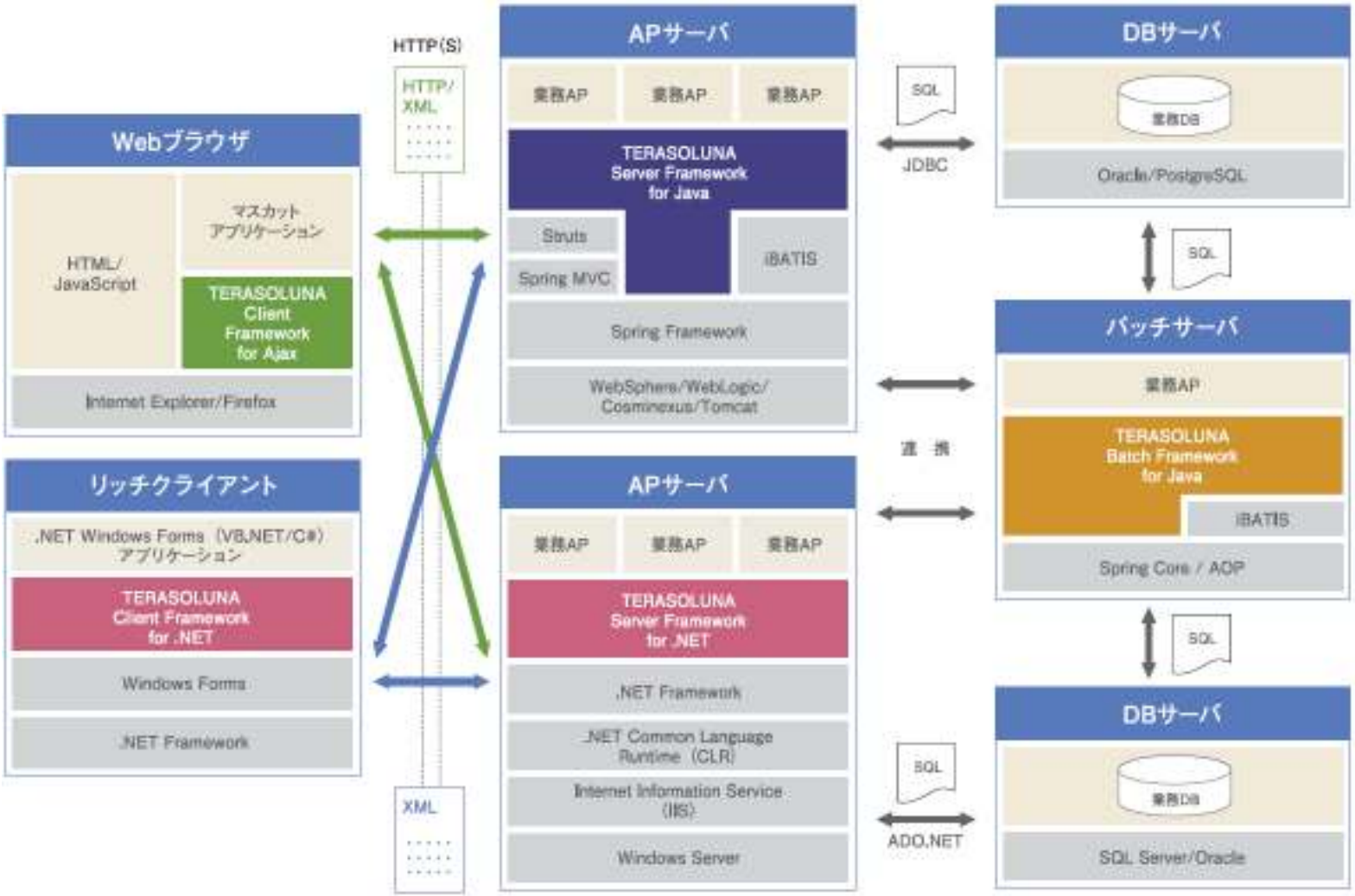
http://sourceforge.jp/projects/terasoluna/

公開後9ヶ月でダウンロード数は約40000

# TERASOLUNAフレームワークの組み合わせ

## ■ 様々な組み合わせが選択可能

◎ フレームワークの組み合わせモデル



# TERASOLUNAフレームワークの機能

## ■ ベースフレームワーク(Spring)に追加した独自機能例

### ● TERASOLUNA Server Framework for Java の場合

| 機能分類 | 機能名 | 機能概要 |
|---|---|---|
| トランザクション管理 | トランザクション管理機能 | Springの宣言的トランザクション管理機能を利用して、起動したビジネスロジックのトランザクションを制御する。 |
| ユーティリティ | 日付操作機能 | 西暦⇔和暦変換などの日付を操作するユーティリティクラスを提供する。 |
| アクセス制御 | ログオン済みチェック機能 | リクエストのたびにユーザがログオン済みかどうかをチェックする。 |
| | アクセス権限チェック機能 | リクエストのたびに正しい権限でアクセスされているかどうかチェックする。 |
| データ保持 | ユーザ情報保持機能 | ログオン中のユーザ情報を保持し、セッションに格納されるバリューオブジェクトを提供する。 |
| | アクションフォーム切替機能 | "_"付のアクションフォームをセッション内に一つしか保持しないように管理する機能。 |
| 例外処理 | Action実行時システム例外処理機能 | ・Action内でシステム例外発生時に、メッセージリソースを用いて、メッセージ変換を行う。<br>・例外情報をログ出力する。 |
| ファイル管理 | セッションディレクトリ機能 | サーバサイドで生成されるファイルを格納する為の一時ディレクトリをログオンユーザ毎に作成する。 |
| アクション | アクション拡張機能 | ログ出力やトランザクショントークンチェックが出来るアクション基底クラス。 |
| | 標準ディスパッチャ機能 | リクエストパラメータによってフォワード先を振り分ける標準アクション。 |
| 入力チェック | 拡張入力チェック機能 | Strutsの入力チェックルールを拡張し、典型的なチェックルールを提供する。 |
| メッセージ管理 | 拡張メッセージリソース機能 | システム共通メッセージ、業務共通メッセージの読み込み、保持を行う。メッセージをデータベースから取得する。 |
| | メッセージ順序保持機能 | メッセージの順序を保持する。 |
| ビジネスロジック | ビジネスロジック入出力機能 | ビジネスロジックの入出力データの変換を設定ファイルに従い自動的に実行する。 |
| 一覧表示 | 一覧表示機能 | ページ遷移を行う一覧表示画面を表示する機能。 |
| 画面表示 | アクセス権限チェック機能 | ユーザの権限によって表示/非表示を切り替える。 |

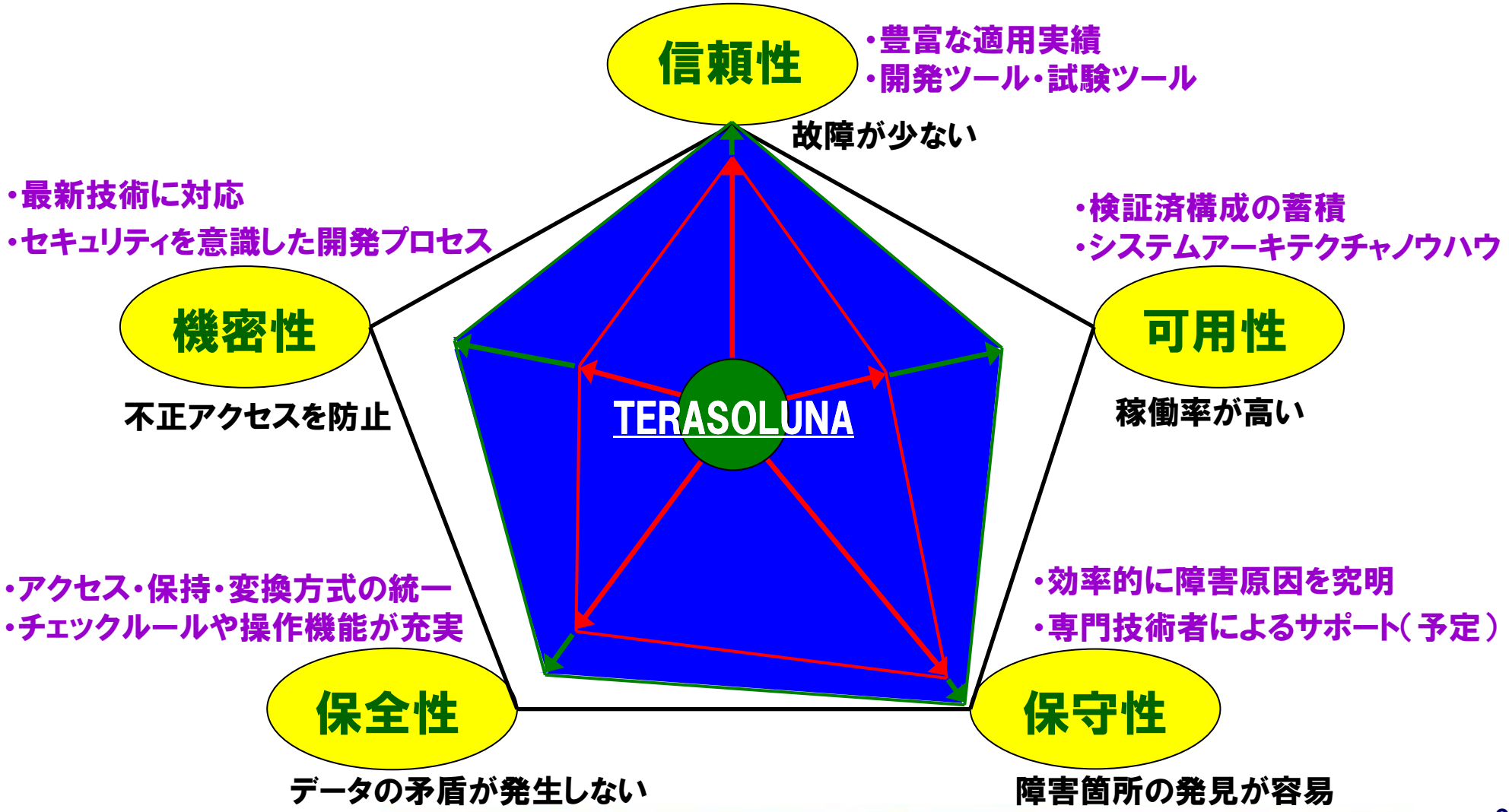
# 安定稼働と機能や特徴の対応

## ■ 安定稼働とフレームワークの関係（TERASOLUNA）

| RASIS要素 | フレームワークの貢献 | 対応する機能や特徴(＋α分) |
|---|---|---|
| **信頼性**<br>故障が少ない | 実績のあるアーキテクチャ | 300プロジェクトを超える適用実績。開発ツールや試験ツールをセットで利用することで、事前にバグを刈取り可能 |
| **可用性**<br>稼働率が高い | 障害の影響範囲を局所化 | 検証済構成の蓄積。システムアーキテクチャのノウハウを開発プロセスとして提供 |
| **保守性**<br>障害箇所の発見が容易 | 故障原因の絞込みが容易 | 適用実績に基づき、効率的に障害原因の究明が可能、専門技術者によるサポート有(予定) |
| **保全性**<br>データの矛盾が発生しない | 制御方式を統一 | 各種データのアクセス・保持・変換の方式を統一。チェックルールや操作機能も充実 |
| **機密性**<br>不正アクセスを防止 | 最新のセキュリティに対応 | 維持メンテを継続することで、常に最新技術に対応。セキュリティを意識した開発プロセスをセットで利用することで相乗効果 |

# 安定稼働への更なる貢献

## ■ TERASOLUNAフレームワークで安定稼働を実現

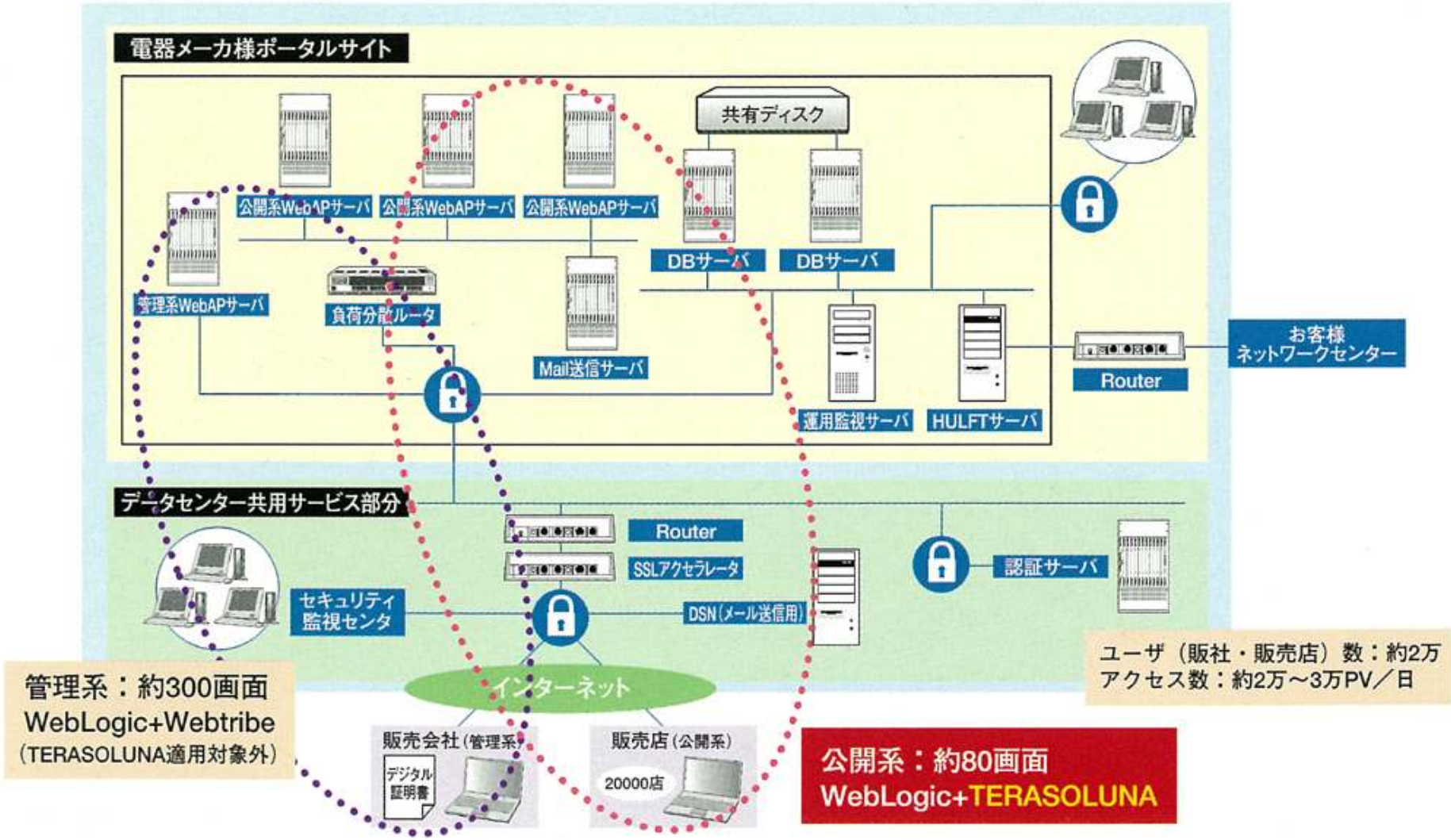


# TERASOLUNAフレームワーク適用事例

- 安定稼働を実現したプロジェクト
  - お客様
    - 大手電気メーカ
  - システム内容
    - 全国の販売店向けに、商品の価格や在庫情報などをWebで提供
  - お客様のご要望
    - 1日2～3万ページビューの負荷に耐えられるシステムにしたい
  - 開発規模
    - 約51KStep
  - 開発期間
    - 約20ヶ月

# TERASOLUNAフレームワーク適用事例

## ■ システム概要図



# TERASOLUNAフレームワーク適用事例

## TERASOLUNA適用の効果

- **オフショア開発の成功**
  - フレームワークの適用により、アプリケーション基盤開発（日本）と業務アプリケーション開発（中国）を分離
    - 開発の作法がおのずと決まるため、オフショアにおいても均質な開発が可能となり、コストメリットを発揮

- **処理方式を統一**
  - ロジックの配置がパターン化され、設計とコードの対応関係が容易にわかる

- **品質低下のリスクを回避**
  - 実装の難しい基盤部分のコードをフレームワークが提供
    - 高スキル者が確保できない状況でも品質が担保

# TERASOLUNA + Cosminexus の連携について

**TERASOLUNA FW の検証済構成に Cosuminexus を追加！**

## Cosminexusへの期待

NTTデータ昨年度実績で10プロジェクトに適用中

- 安定動作
  - TERASOLUNA FWと連携するAPサーバとして、安定した性能を発揮して欲しい
    - Cosminexusは、業務ロジック単位での流量制御が可能
- トラブル解析
  - 障害発生時に問題箇所を素早く特定できることが望ましい
    - Cosminexusは、スレッドダンプ出力機能で原因の特定が容易

## 協業

- 共同動作検証
  - 日立ハーモニアス・コンピテンス・センターにて動作検証を実施済
    - 2007年10月1日～2007年10月22日
    - TERASOLUNA FW + uCosminexus Application server
    - FWの構成モデルに Cosminexus を正式認定！
- 共同R&D
  - TERASOLUNAフレームワーク(Batch)との連携
    - 2008年度下期、TERASOLUNA FW(Batch) + Cosminexus の動作検証を予定

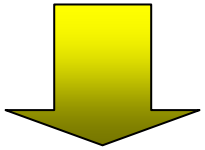
## 連携事例

- TERASOLUNA + Cosminexus を適用したプロジェクトをご紹介

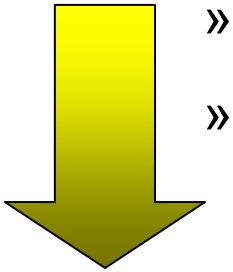
# まとめ

## ■システムの安定稼働を実現するには

- **規約に基づいた開発が効果的**



- **フレームワークの活用**



» 開発時の規約が自ずと定まる

» 一定の品質を確保

- **TERASOLUNAフレームワークによる改善**

» 独自に機能を追加

» 開発プロセスや開発ツール、サポートも充実

Insight for the New Paradigm

未来のしくみを、ＩＴでつくる。

株式会社ＮＴＴデータ

**技術開発本部　ソフトウェア工学推進センタ**
**TERASOLUNA担当**

**Phone: 050-5546-2482**
**Mail: terasoluna@kits.nttdata.co.jp**
URL: http://www.terasoluna.jp