



# 短期システム構築に伴う品質管理 サービス・ソリューション(案)

---

株式会社 ステラーソリューション



# << 目次 >>

● <u>短期開発や技術の高度化により不具合がすり抜ける</u> .....	4
● <u>他のユーザーのデータが表示されてしまうケース</u> .....	5
● <u>テストの盲点</u> .....	6
● <u>テスト・ケースの盲点(1/3)</u> .....	7
● <u>テスト・ケースの盲点(2/3)</u> .....	8
● <u>テスト・ケースの盲点(3/3)</u> .....	9
● <u>テスト環境の盲点(1/3)本番のハードウェアやソフトウェア環境を完全に再現不可</u> .....	10
● <u>テスト環境の盲点(2/3)本番データや設定の完全な再現が難しい</u> .....	11
● <u>テスト環境の盲点(3/3)本番環境とテスト環境のギャップ</u> .....	12
● <u>負荷テストと障害テストの盲点(1/2)</u> .....	13
● <u>負荷テストと障害テストの盲点(2/2)問題のある負荷テストの例</u> .....	14
● <u>テスト・ツールの盲点(1/3)</u> .....	15
● <u>テスト・ツールの盲点(2/3)</u> .....	16
● <u>テスト・ツールの盲点(3/3)</u> .....	17



# << 目次 >>

● <u>品質向上のご提案</u> .....	18
● <u>提案 開発工程全体でテストの死角をなくす(1/2)</u> .....	19
● <u>提案 開発工程全体でテストの死角をなくす(2/2)</u> .....	20
● <u>提案 システムや機能の重要度に応じて厳密さを設定する</u> .....	21
● <u>提案 テスト工程で増員する</u> .....	22
● <u>提案 進捗は必ず把握する</u> .....	23
● <u>提案 コミュニケーションを重視する</u> .....	24
● <u>提案 (お打合せ後、お見積書提出) 単体テストで実施すべきテスト・ケース</u> .....	25
● <u>提案 (お打合せ後、お見積書提出) テスティング・フレームワークで単体テストを自動化</u> .....	26
● <u>提案 (お打合せ後、お見積書提出) 各テスト工程で実施すべきテスト・ケース(1/3)</u> .....	27
● <u>提案 (お打合せ後、お見積書提出) 各テスト工程で実施すべきテスト・ケース(2/3)</u> .....	28
● <u>提案 (お打合せ後、お見積書提出) 各テスト工程で実施すべきテスト・ケース(3/3)</u> .....	29
● <u>提案 (お打合せ後、お見積書提出) 構築中のシステムに対する品質管理(1/2)</u> .....	30
● <u>提案 (お打合せ後、お見積書提出) 構築中のシステムに対する品質管理(2/2)</u> .....	31
● <u>提案 (お打合せ後、お見積書提出) 夜間にビルドと回帰テストを自動的に行う</u> .....	32



## 短期開発や技術の高度化により不具合がすり抜ける

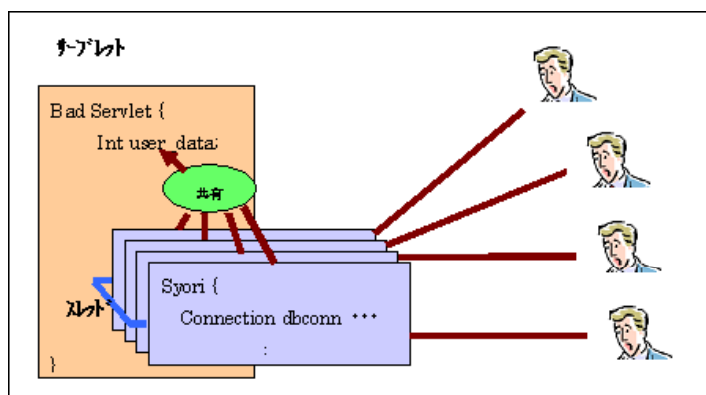
人間のミスをゼロにすることはできない。

さらに、経験の浅い開発者の増加、システムの複雑化一様な要因が不具合の増産を後押しする。

さらに、開発期間の短期化によりテストに十分な時間をかけられないプロジェクトが増加している。

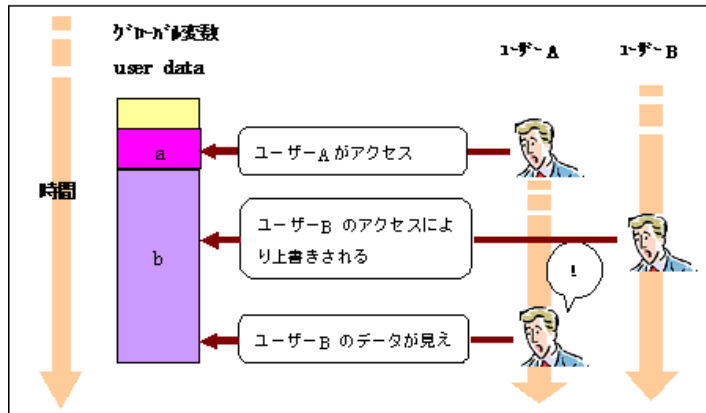
技術の高度化も不具合の発見を困難にしている。

# 他のユーザーのデータが表示されてしまうケース



不具合が発生したケースの例。

Javaサーブレットはマルチスレッドで動作しているため、複数の処理が、データを上書きしてしまう。そのため、アクセスすると、別のユーザーのデータが表示されてしまう。これは、一例であり、他にも同様な不具合が発生するケースはいくつも存在する。



# テストの盲点

## テスト・ケースの盲点

想定されていない入力データ

想定されていない操作

時間や時刻によって異なる挙動

外部システムや別モジュールの動作

## テスト環境の盲点

本番のハードウェアやソフトウェア環境を完全には再現できない

本番データや設定の完全な再現が難しい

## 負荷テストと障害

本番の負荷を再現できない

障害時の予期せぬ挙動

## ツールの盲点

テスト・ツールにはデメリットや限界が存在する

すべてのケースを100%テストすることは不可能だ。しかし、どのようなポイントに大きな「漏れ」が発生しやすいかを把握することで、重大な不具合を効果的に検出できる。

## テスト・ケースの盲点(1/3)

稼働後や後工程でモレが発見された例

想定されていない入力データ	
データが“ない”ことを想定していない	単体テスト済みのはずモジュールに、null値を入力して検証してみるとnull pointer exceptionが発生し停止してしまうことが多い
想定外のデータ型が入力される	英数字だけ前提で型番データを定義していたが、最終段階のテストでエラーが発生。半角カナも使う場合があったことが判明した
すべてのケースをテストしきれない	プログラム中に、文字変換テーブルを配列の定数として保持していたが、1カ所だけ間違いがあった。ほとんど使用されない文字であったため、稼働後1年して発見された。
正常値以外想定していない	検収テストで、正常値以外の値を入力してみたところ、アプリケーションが停止、異常値を処理するためのELSE文が記述されていないモジュールがあった。
想定されていない操作	
きわめて同時に近いタイミングで複数ユーザーが操作する	マルチスレッド間の競合により、他のユーザーの処理をしているスレッドがデータを上書きしてしまう。クライアントの画面に他のユーザーの情報が表示されてしまう。
WWWブラウザから、本来ありえないはずのデータが送信される	いたずらのためか、URL中に埋め込んだパラメータが改変されて送り返されることがあり、エラーが発生する。

## テスト・ケースの盲点(2/3)

時間によって異なる挙動	
ある時間帯でテストが行われていない	10時にシステムの営業日付を切り替えるシステムがあったが、切り替え時間にシステムが停止していた。切り替え時間の後にシステムを起動すると、システムの営業日が切り替わっておらず、エラーが発生した。
例外的な運用を想定していない	夜間に一度停止させ、営業日付を切り替えるシステム。夜間にバッチのトラブルがあり、システムを停止させないまま営業時間を迎えた。システムの営業日付が切り替わっておらず、エラーが発生した。
1つの処理が長い時間にわたって行われることを想定していない	WWWシステムのセッション情報は、3ヵ月を経過したらバックアップ・ディスクへ移動していた。常時接続環境の普及により、3ヵ月以上ログアウトしていないユーザーが出てきた。異常処理を示す「センターに連絡してください」とのメッセージを表示させたとの問い合わせにより判明。以後、このような場合には「再ログインしてください」と表示するようになった。
長期運用により領域やディスクがあふれる、デフラグが発生する	1日約1Kバイトのログが蓄積され、稼動して3年経過後にディスク領域があふれてしまい、アプリケーションがエラーにより停止した。

## テスト・ケースの盲点(3/3)

外部システムや別モジュールの動作	
別モジュールから渡されるデータは、チェック済みの正常値のみという前提で実装やテストを行う	入力データの半角と全角の統一を、お互いに相手のモジュールで行うものと期待して、自分のモジュールに実装していなかった。
シミュレータで外部システムを完全に再現できない	カード決済システムのシミュレータを作成し、連携のテストを行ったが、エラーデータを送信した場合のふるまいまで再現できなかった。
データ形式や解釈の細部が食い違う	日付データの形式が、こちらで開発したモジュールではYYMMDD(年月日)だったが、相手側モジュールがDDMMYY(日月年)だった。ホストのシステムでは年の項目を利用していないことを示す特別な値として「9999」を使用していた。C/Sシステム側ではこれをそのまま「9999年」と解釈してしまった。
文字コードが食い違う	ホスト・システムとC/Sシステムでは使用する文字コードが異なり、ソート順もことなっていることが、結合テスト時に判明した。

## テスト環境の盲点(1/3)

# 本番のハードウェアやソフトウェア環境を完全に再現不可

本番のハードウェアやソフトウェア環境を完全に再現できない	
環境が変わるとシステムの挙動が変わる	本番環境と開発環境のJavaVMのバージョンが異なり、本番環境ではSQL文をあらかじめコンパイルしおおく機能であるprepared statementに不具合があり、アプリケーションが動作しなかった。
本番のネットワーク環境の再現が難しい	本番では、通信相手との間にファイアウォールが存在した。ファイアウォールが、一定時間通信がないとセッションを切断してしまうことが、本番環境でのテストで発覚。必要が無くとも定期的に通信を行うように変更した。
大規模な本番環境では、機能追加などのテストに本番と同じハードウェアをそろえられない	32プロセッサ機をテスト用に用意するのは困難。追加機能などは、外部からアクセスできないようにしながら本番機で稼働させ、テストすることもある。
テスト環境を切り離しておかないと、テスト中に外部へのアクセスやデータ送信が発生する	メール送信機能を持つシステムでは、テスト中に外部へ送信してしまう可能性があるため、25番ポートをふさぐなどしてからテストを行っている
クライアントとなるWWWブラウザやOSの組み合わせが多すぎる	リスト・ボックスなどを多用した、WWWブラウザ画面。Windows2000では動作したが、Windows98ではリソース不足によりWWWブラウザがハング・アップしてしまうことが、最終段階になって判明。ブラウザ画面の設計を変更した。

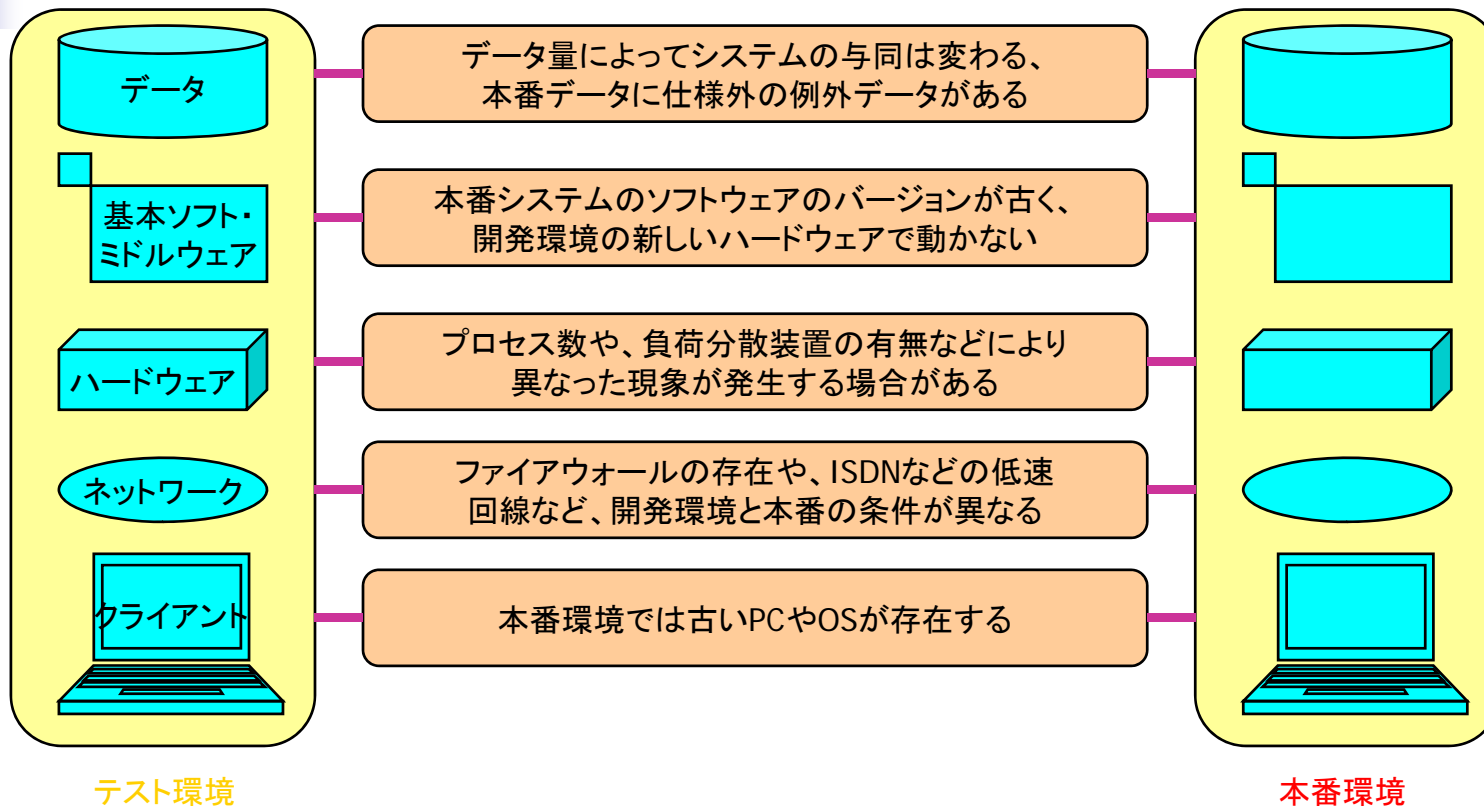
## テスト環境の盲点(2/3)

### 本番データや設定の完全な再現が難しい

本番データや設定の完全な再現が難しい	
本番データを持ち出したりすることにより情報漏洩が懸念される	住民データが開発中に持ち出された例もある。テストデータ中の個人情報にはマスクしている。
現行システムのデータに仕様外のデータが含まれている	データ不整合が発生し、原因を突き止めると移行データの問題だったことが多い。 例外処理の対応などで直接操作されたデータが存在すると、一括移行はまず不可能。
本番環境の設定と、テスト環境の設定が異なってる	本番環境で設定したデータベースの参照制約を、テスト環境に反映し忘れた。そのため、テスト環境では成功したデータベース更新処理が、本番では失敗した。

# テスト環境の盲点(3/3)

## 本番環境とテスト環境のギャップ



様々な理由により、テスト環境で本番環境を再現できない場合は多い。ギャップが生じやすい部分に注意し、重要な部分は本番環境でもテストを行う。

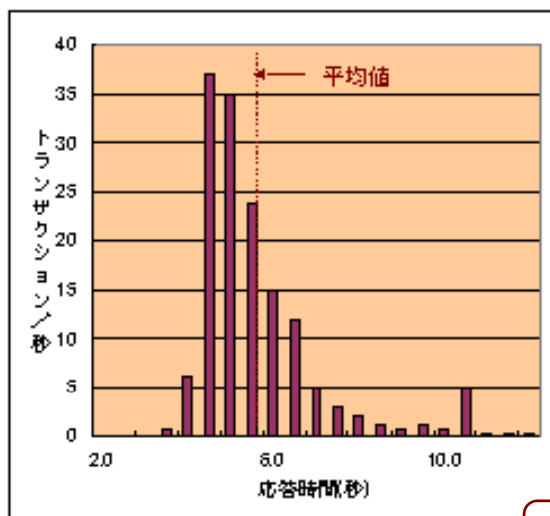
## 負荷テストと障害テストの盲点(1/2)

障害テストの問題	
バックアップ・サーバーへの切り替え時の挙動は、負荷やデータ量によって変わる	テストでは正常に動作したバックアップ・サーバーへの切り替えが、本番ではうまくいかなかった。切り替え時にデータベースのロールフォワードが30分以上かかったために、切り替えを監視しているプログラムが異常と判断してシャットダウンしてしまった。
高付加時にのみ発生する、プラットフォームの挙動や障害が存在する	Enterprise JavaBeansの一種であるStateful Session Beanは、高付加時には新規インスタンスを生成する代わりに既存のインスタンスを使いまわすことがある。そのため、他のユーザーの画面が表示されてしまったことがある。Stateful Session Beanを利用する前に必ず初期化することで発生しなくなった。
負荷テストの問題	
負荷テスト・ツールの出した値が必ず正しいとは限らない	負荷テスト・ツールにバグがあり、複雑なアクセスを大量にかけるとテスト・ツール側がボトルネックになり、本来より低い性能値しか出なかった。
負荷のパターンにより、性能は大きく変わる	既存システムのWWWアクセス・ログを負荷テスト・データとして使用した。フロント・エンドのWWWアクセスだけでなくバックエンドのバッチ的な負荷も同時にかけた。
ISDNなどの低速回線を介すると、表示速度やサーバー負荷が変わる	低速回線が介在すると、クライアントからの応答パケット待ちによりサーバー側の処理待ちプロセスが増加し負荷が高くなる。また、データ量が多いページは転送に時間がかかり、ユーザーから見た応答速度も下がる。ルーターの設定で低速回線を再現したテストを行うようにしている。

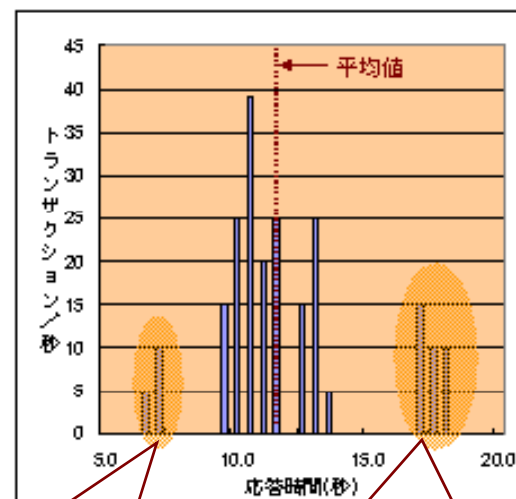
# 負荷テストと障害テストの盲点(2/2)

## 問題のある負荷テストの例

正常な負荷テスト



問題のある負荷テスト



ツールのバグなどによる異常値の可能性

キャッシュ・ミス多発などによる異常値の可能性

応答時間の平均値だけを見ていると問題を見逃す可能性がある。例えば右のグラフでは中央のピークと別に2カ所ピークが出ているが、ツールのバグなどによる異常値の可能性が高い。

# テスト・ツールの盲点(1/3)

## テスト・ツールの盲点と実例や対策

テスト・ツールのデメリットや限界	
ツールを導入すればテストを効率化できるとは限らない	自動操作テスト・ツールの場合、テスト・スクリプト作成のために、テスト自体と同等以上の工数が必要。実際に工数を計測したところ、4回以上同じテストを繰り返さなければメリットがでないという結果になった
システムの応答以外はツールでは発見できない	デザインの崩れや、メッセージの誤字などはツールでは発見できないため、システム・テストはツールを使わず人手で行っている
ツールの習熟に時間がかかる	初めてのツール導入だったため、スクリプトをベンダーに作ってもらい、その過程を見ながら勉強した
ツール導入にコストがかかる	100万円以上のツールがほとんどで、個別のプロジェクトの予算では導入するのは難しい。負荷テスト・ツールを作成したり、Microsoft Web Application Stress Toolなどの無償ツールを利用する企業も多い

## テスト・ツールの盲点(2/3)

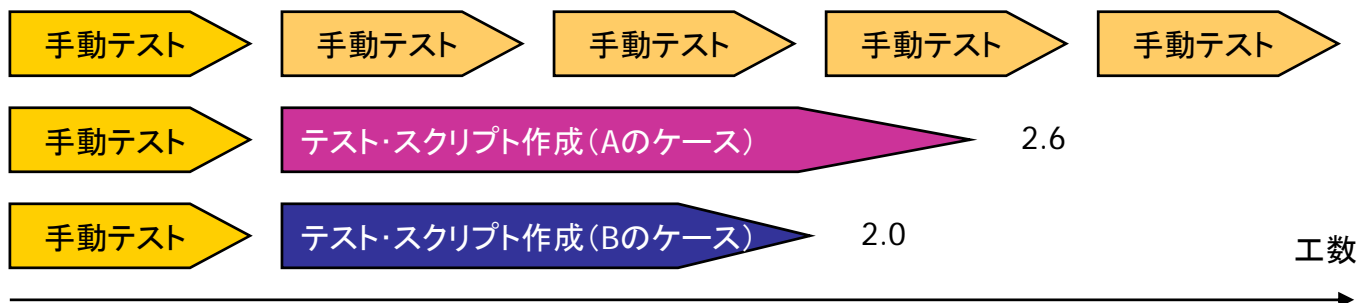
### テスト・ツールの種類と内容

テスト・ツール	
テスト管理／バグ管理ツール	登録されたテスト・ケースをいつ誰が実行したか、未実行のテストはどれかを記録し管理する。また発見されたバグとそれに対する修正を登録し、未解決バグをリスト・アップする。テスト・ケース消化率、モジュールや原因ごとに分類し集計する機能を持つ製品もある。
負荷テスト・ツール	クライアントからアクセスをプロトコル・レベルなどで再現し、大量に発生させることでサーバーやネットワークに負荷を与え、性能測定や安定性の検証を行う。
単体テスト・ツール	プログラムのソース・コードを調査し、メモリ・リークなどを発生させる恐れが無いか、規約に則って記述されているかなどを検査する(静的テスト)。個々のモジュールを起動し、入力データを与えて、出力データを予想される結果と照合するなどのテストを自動的に実行する(動的テスト)。ソース・コード中の分岐のどの部分がテストされたかを管理する機能(ガバレージ管理機能)を備える製品もある。
自動操作テスト・ツール	マウス操作やキーボード入力などクライアント画面での操作を記録して、再現することで機能テストを自動化する。記録した操作からスクリプトを自動生成するので、スクリプトを変更することで入力値を自動的に変更させたりすることが可能。

## テスト・ツールの盲点(3/3)

### 自動操作テスト・ツールの効果(某会社の事例)

測定した自動操作テスト・ツールの効果。テスト・スクリプト作成のために、テスト自体の2倍以上の工数がかかった。テストの繰り返し回数が3回以下の場合には効率化効果は得られない。全く同じテストを4回以上繰り返す場合は効果がある。Webアプリケーションについて測定した。



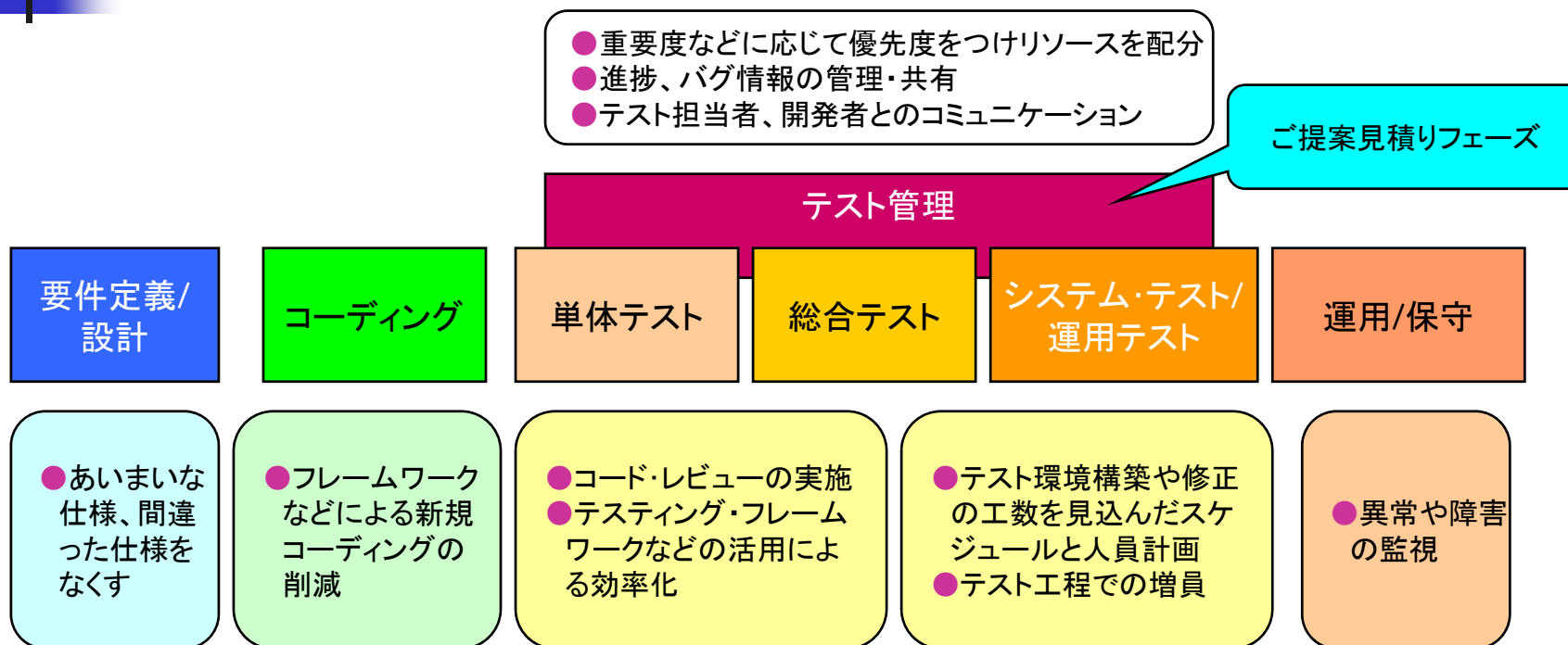
#### Aプロジェクト

- A省庁向けシステム
- Webアプリケーション
- 適用工数: 保守・運用工程

#### Bプロジェクト

- B省庁向けシステム
- Webアプリケーション
- 適用工数: 開発工程(新規開発——2段階リリース)

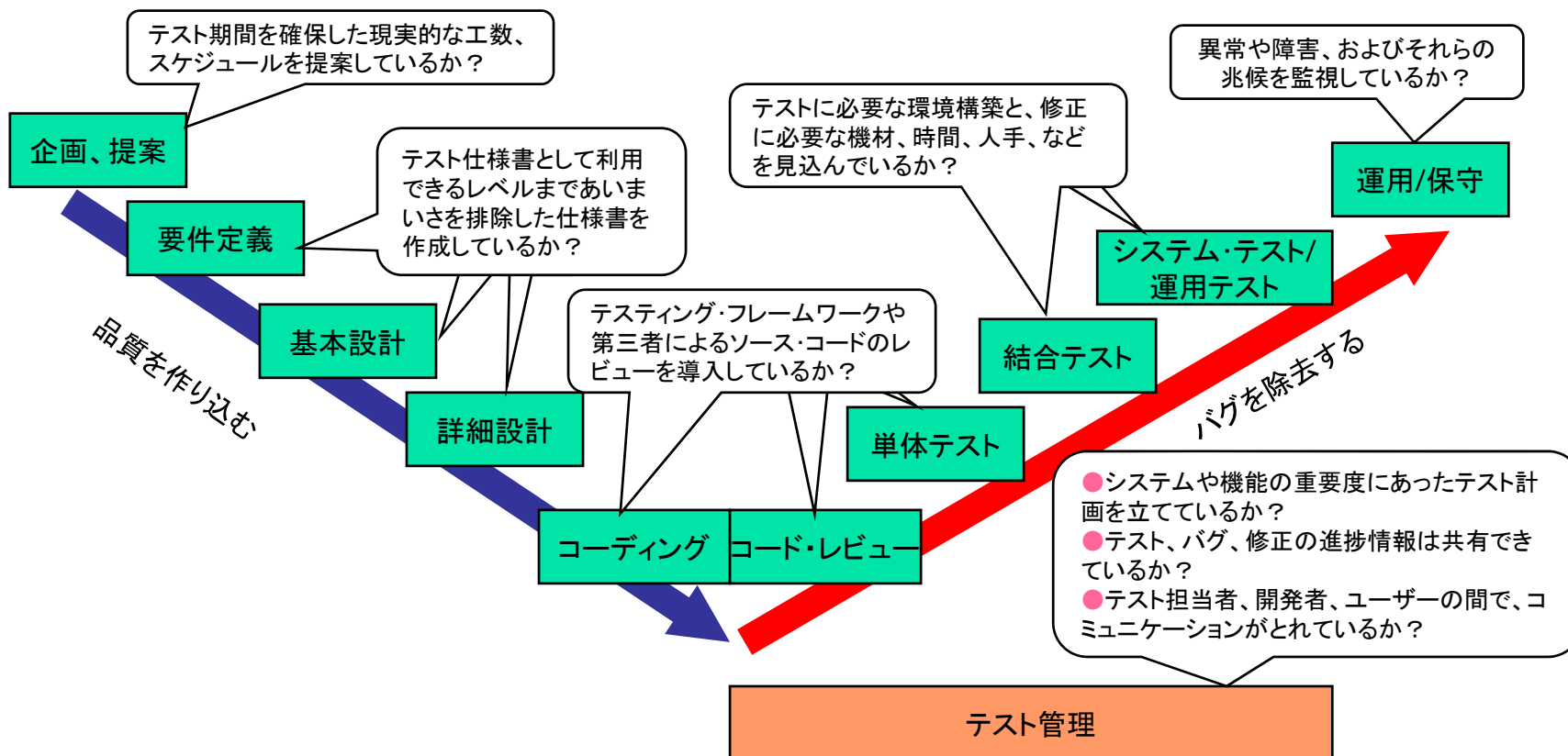
# 品質向上のご提案



実施すべくテストの数を予測し、不具合の修正やテスト環境構築のための工数を見込んだスケジュールを立てる。アプリケーションの重要度に応じて、限られた時間と人的リソースを優先度をつけて配分する。何より、テストで発見するより前に、変更の発生しない仕様書を作り、新規に記述するコードを減らすなど、開発や設計の工程で不具合を生み出さない仕組みを作らなければならない。

# 提案

## 開発工程全体でテストの死角をなくす(1/2)





## 提案

# 開発工程全体でテストの死角をなくす(2/2)

---

テスト工程だけではシステム品質は向上しない。

バグを除去するための工数は、下流になればなるほど高くなる。

テスト工程までに、

「テスト期間を確保したスケジュールを提案する」

「あいまいさを排除した設計書を作成する」

「無償のテスト・フレームワークを利用して開発効率を上げる」

などの工夫をすることで、より高い品質を生み出すことができる。


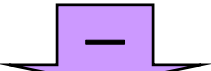
また、作成したシステムの監視も重要だ。

システムにはまだ除去しきれしていないバグが潜んでいるからだ。

# 提案

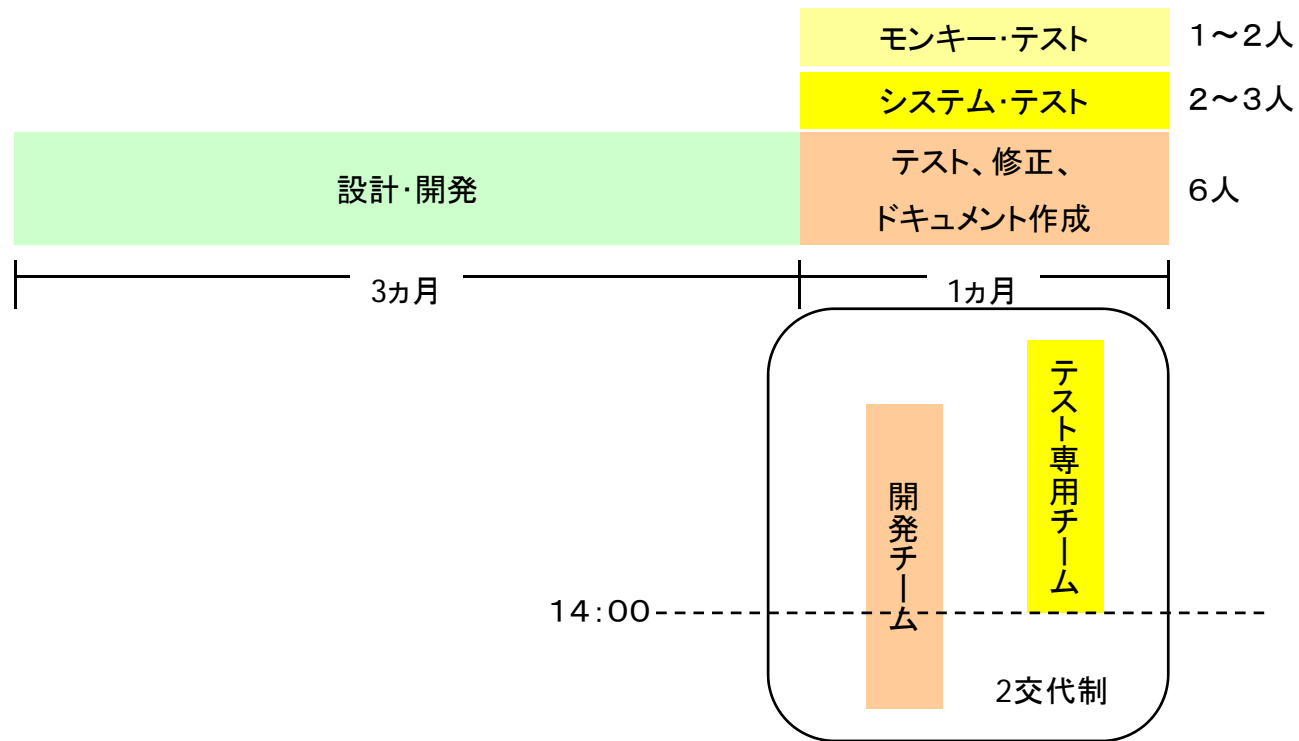
## システムや機能の重要度に応じて厳密さを設定する

リリースまでに実施すべきテスト基準

ランク	実施すべきテスト	対象となるシステム
特A	エンド・ユーザーも参加したテストを実施しないとリリースしてはならない	電話加入者が直接使用するシステム
		
A (基本となるランク)	ストーリー・テスト(仕様書に基づいたテスト)と、開発者の相互レビューを実施する	加入者が直接は使用しないが、複数の部署が使用するシステム
		
B	開発者によるテストのみ実施後、即日リリース	特定の部署だけが使用するシステム

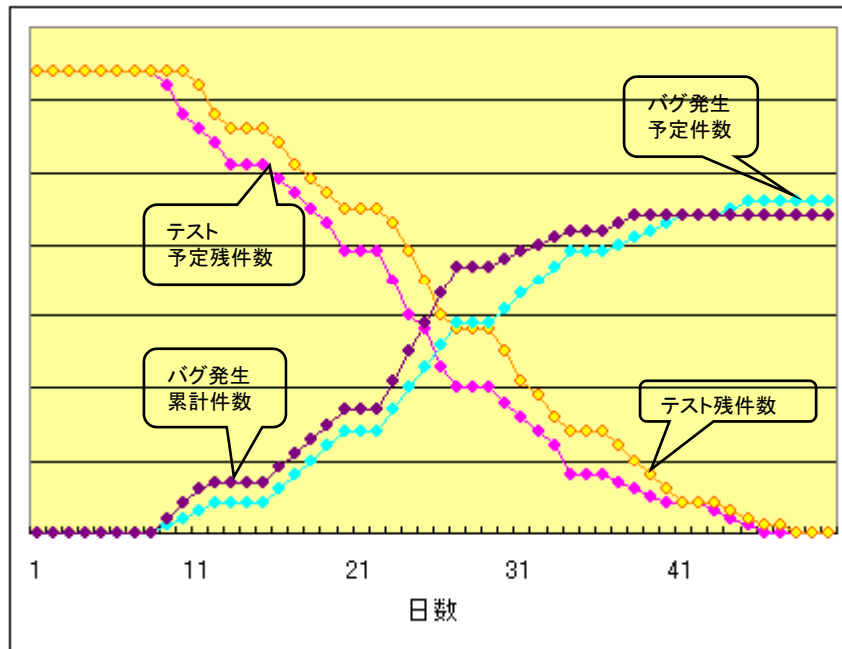
テストに投入できる時間と人は限られている。そのため全体の完成度よりもシステムや機能に重要度に応じたテスト密度を設定すること、スキルに応じた人員配置をすること、など、濃淡をつけることが需要だ。

# 提案 テスト工程で増員する

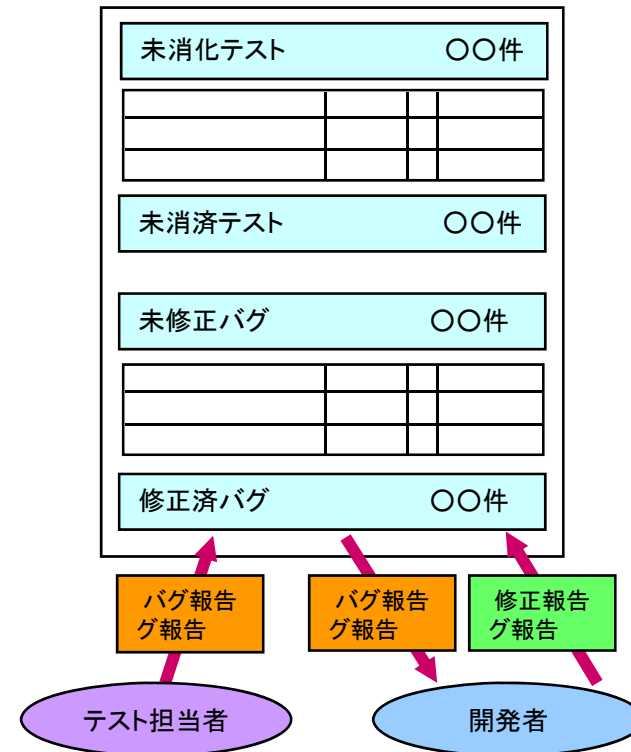


メンバーでは、ほとんどの開発プロジェクトで、テスト工程での増員を行っている。また、開発チームとテスト・チームは2交代制で勤務する。2交代制により、テスト工程を3分の2程度に短縮できるという。また、でたらめな操作をすることでユーザーの予想もつかない行動をテストする「モンキー・テスト」のための要員を置くこともある。

# 提案 進捗は必ず把握する



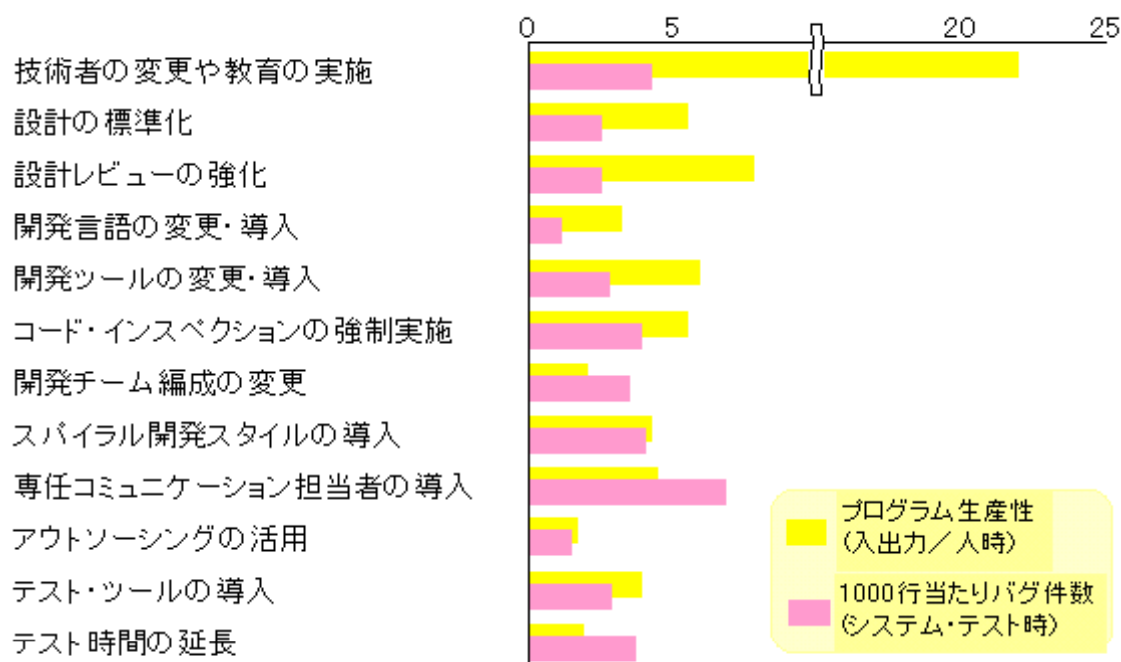
× × 会社が実際の開発で使ったバグ管理図



テスト項目の消化数や発生バグ件数など、テストの進捗は必ず把握しておかなければ、プロジェクトの最終局面であるテスト工程では混乱が生じやすい。また、バグ情報を共有することで、同様のバグの抑止や、バグ修正による影響範囲の把握をスムーズに行える。

# 提案

## コミュニケーションを重視する



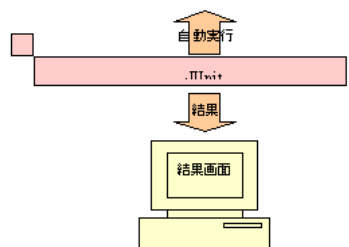
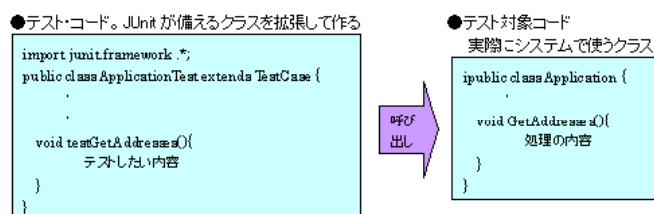
様々な品質向上対策の効果を測定した結果、開発者とテスト担当者、開発者とユーザーの仲介役となる「専任コミュニケーション担当者」の設置が最も効果が高かった。仕様に対する誤解や、行き違いなどが減少したためである。

## 提案(お打合せ後、お見積書提出) 単体テストで実施すべきテスト・ケース

目的	テスト内容	確認項目例	標準価格	
①詳細設計書の要件確認 ②分岐(ソース・コード)網羅 ③モジュール単体動作網羅	全分岐通過テスト	すべての内部処理と静的データ	80万円/月～	
	モジュール機能テスト	モジュール外部仕様		
		入力値チェック		
		出力編集		
		データ入出力(エラー含む)		
	モジュール・インタフェース	インタフェース入出力内容		
	ユーザー・インタフェース	GUI動作		
メッセージ内容				

# 提案(お打合せ後、お見積書提出)

## テストリング・フレームワークで単体テストを自動化



- JUnitのメリット**
- テスト環境やテスト手順が統一される
  - 無償である
  - 結果が分かりやすい

- JUnitではテストできない機能や、利用効果が低い機能**
- EJB:EJB コンテナ上で動作するため、JUnit ではテストできない
  - JSP: サーブレットに変換されるため、HttpUnit や Cactus を使用する
  - DB アクセス: SQL 文などでテストコードの可読性が低下

### オープンソースのテストリング・フレームワークの例

製品名	テスト対象	URL
JUnit	Java クラス	<a href="http://www.junit.org/">http://www.junit.org/</a>
HttpUnit	WWW ブラウザ	<a href="http://httpunit.sourceforge.net/">http://httpunit.sourceforge.net/</a>
Cactus	サーバー・サイド Java	<a href="http://jakarta.apache.org/cactus/">http://jakarta.apache.org/cactus/</a>
JsUnit	Java スクリプト	<a href="http://www.edwardh.com/jsunit/">http://www.edwardh.com/jsunit/</a>
Nunit	Visual Studio .NET	<a href="http://www.nunit.org/">http://www.nunit.org/</a>
VBUnit3	Visual Basic	<a href="http://www.vbunit.com/">http://www.vbunit.com/</a>
csUnit	Visual C# .NET	<a href="http://www.csunit.org/">http://www.csunit.org/</a>

その他のテストリング・フレームワークについては、<http://www.xprogramming.com/software.htm> を参照

JUnitはオープン・ソースのJava単体テスト用フレームワーク。まずテスト・コードを記述し、次にテスト・コードを通るコードを記述するという「テスト・ファースト」の概念と合わせて用いることが多い。

ただし、テスト・ファーストの強要は、慣れないうちは生産性を落とすこともある。開発規模やシステムの特性、メンバーのスキルに合わせて利用状況を変えることが大切だ。



# 提案(お打合せ後、お見積書提出)

## 各テスト工程で実施すべきテスト・ケース(1/3)

目的	テスト内容	確認項目例	その他
<b>運用テスト</b> ①ユーザーによる妥当性確認	ユーザーによる確認	実業務での機能の妥当性	別途お見積
	運用管理者による確認	運用管理機能の妥当性	
<b>システム・テスト</b> ①要件定義書の要件確認 ②運用マニュアルの要件確認 ③ユーザーの目的/システム間 に対する妥当性の最終確認 ④システム形態でのテスト	要求仕様テスト	機能要件の確認	別途お見積
		運用マニュアル手順の正確性	
	大容量テスト	大量データ(将来の増分を予測)	
	負荷テスト	高負荷	
	性能テスト	複数機能を統合した実用性能	
		性能目標値の確認	
	障害テスト/復旧テスト	耐障害性	
		障害レポートの妥当性	
		リカバリ処理(回復処理)	
		縮退運転処理	
バッチ再実行後のデータ整合性			
	システム・ダウン時のユーザー側対処		

# 提案(お打合せ後、お見積書提出)

## 各テスト工程で実施すべきテスト・ケース(2/3)

目的	テスト内容	確認項目例	その他
システム・テスト	安定化テスト	消費リソースが増大しないこと	別途お見積
		性能劣化のないこと	
	構成テスト/設置テスト	構成変更、再構築	
		実稼働環境での稼働確認	
	セキュリティ・テスト	機密保護機構の有効性確認	
		ハッキングの試行	
	サービス性テスト	保守用の機能の有効性	
		保守文書の正確性	
他システムとの接続テスト	実環境での接続試験		
コンFORMANCE試験	規格適合性試験		

# 提案(お打合せ後、お見積書提出)

## 各テスト工程で実施すべきテスト・ケース(3/3)

目的	テスト内容	確認項目例	その他
<b>結合テスト</b> ①基本設計書の要件確認 ②外部仕様網羅 ③モジュール間結合動作網羅 ④入出力整合性	機能テスト	すべての外部仕様の妥当性	別途お見積
		外部条件	
		機能単位の性能	
	モジュール間インタフェース確認	共有データを経由した整合性	
		同時実行制御(排他制御)	
		区分コードに依存する機能の差異	
		データ操作タイミング、状態遷移	
		移行データを使用した動作テスト	
	入出力データ整合性確認	トランザクションでの入力と出力での整合性	



## 提案(お打合せ後、お見積書提出) 構築中のシステムに対する品質管理(1/2)

---

### お打合せ

- 現状把握し、テスト方式および双方の作業範囲の提案
- テスト作業体制および作業工程の提案とお見積書を提出

### 貴社提出物資料

- ソースプログラム
- 各種設計書
- 操作手順書
- 運用マニュアル
- その他



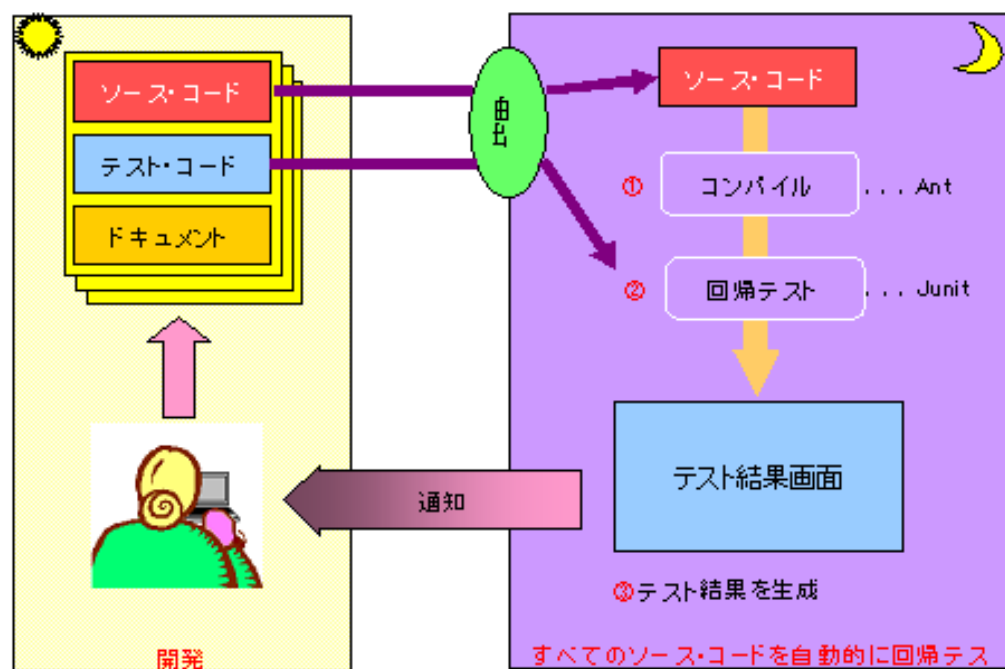
## 提案(お打合せ後、お見積書提出) 構築中のシステムに対する品質管理(2/2)

---

### 弊社提出物資料

- テスト計画書(貴社よりお預かりした資料等を調査分析し計画書作成)
  - 1) 単体テスト(具体的なテスト仕様作成および実施)
  - 2) 結合テスト(具体的なテスト仕様作成および実施)
  - 3) システム・テスト(具体的なテスト仕様作成および実施)
  - 4) 運用テスト(具体的なテスト仕様作成および実施)
- その他

# 提案(お打合せ後、お見積書提出) 夜間にビルドと回帰テストを自動的に行う



夜間にすべてのソース・コードの回帰テストを行っている。仕組みは、①まず、XMLデータに格納したソース・コードを抽出して、コンパイルする。②次に、XMLデータから抽出したテスト・データを基に回帰テストを行う。③結果はHTMLに出力される。オープン・ソースのビルド・ツール「Ant」とテスト・フレームワーク「JUnit」を利用し、開発した。