Ｗｅｂアプリケーション開発における
テストの効率化と生産性向上

～ Productivity improvement and efficient application testing for Web application development ～

日本アクチュアリー会
ＩＴ研究会第７グループ

＜担当委員＞
横井　俊明　（三井生命）

＜メンバー＞
小田　敏久　（日本生命）
沖本　和彦　（ＡＩＵ）
松崎　豊　（三井住友海上）
三田村　奈美（損保ジャパン）
内村　洋介　（住友生命）
工藤　将人　（太陽生命）
山崎　祐介（日本生命）
岩品　知明（アリコ）
井上　信明（三井生命）
桑原　正樹（ﾌﾟﾙﾃﾞﾝｼｬﾙ生命）
入江　寛　（太陽生命）
木村　純一（太陽生命）

＜目次＞

はじめに

企業におけるＩＴインフラがここ数年で急速に整備され、Ｗｅｂベース（インターネット・イントラネット・エクストラネット）のシステムを構築する企業が急増している。
個人においても、ブロードバンド環境の普及が急速にすすんでおり、２００５年６月末時点でブロードバンド総加入件数が２０５８万件を超えるとの調査結果もある。
こういった環境の変化の中、Ｗｅｂアプリケーション開発の効率化・開発期間の短縮・精度の向上が、経営戦略上においても非常に重要な課題となってきている。

これらの課題を解決するため、最近では開発ツールだけではなく、さまざまなテストツール・開発手法が各種メディアで取り上げられ注目され始めている。
「Ｗｅｂアプリケーション開発におけるテストの効率化と生産性の向上」というテーマを受けた当研究チームでは、保険業界のシステムの特徴・問題点を整理するとともに、
同業界でのテストツールの利用状況・問題点の検証・効果的な活用方法、各種テストツールの紹介、また、最近注目を集めている「ＵＭＬ/モデル駆動型開発」などについても考察し、「テストの効率化と生産性の向上」について提言していくこととする。

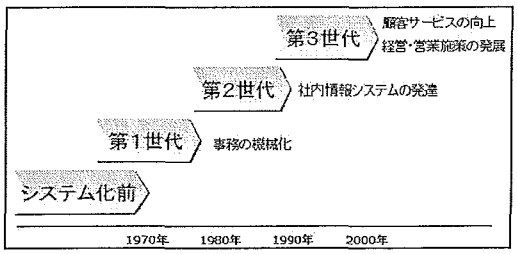
第Ｉ章　保険Ｗｅｂアプリケーション開発の問題点

第Ｉ章では保険Ｗｅｂアプリケーション（以降、「Ｗｅｂアプリ」と呼称）開発について「システム的観点」と「業務的観点」からの分析を行い、問題点を洗い出していくこととする。

１．システム的観点からの分析

（１）保険会社のシステムの歴史

まずは保険会社のシステムの歴史を第１世代・第２世代・第３世代と大きく３つに分けて説明する。



第１世代は１９７０年代で、主に社内事務が手作業から機械化へ移行が行なわれた時代と捉えることができる。現在のパソコンやメインフレームとは違い、オフコンでのコンピュータの導入が行なわれた。

第２世代は１９８０年代に入り、メインフレームの導入・ネットワークのオンライン化の時代と定義することができる。システム開発の面で見ると、ＣＯＢＯＬやＰＬ／Ｉでの開発が行われるようになり、開発規模が大きくなるにつれて「ウォーターフォール開発モデル」に代表される開発手法も確立されてくることになる。その結果、社内情報システム、特に契約管理系のような大規模システムの構築が可能になった時代でもある。

第３世代は１９９０年代からのインターネットの普及に伴う、Ｗｅｂ技術の発達の時代である。第２世代に導入されたメインフレームをベースにした業務システムのオープン化、サーバーサイドでのＷｅｂアプリ開発、インターネットを利用した、ＢｔｏＢやＢｔｏＣシステムが構築されるようになった。それに伴い、インターネットを使用したＷｅｂ関連技術も飛躍的に進歩してきた。たとえば、ＸＭＬを使用したシステム間通信を行うＷｅｂサービスや、携帯電話の普及に伴うｉモード等の携帯Ｗｅｂサイト等があげられる。

保険システムについて見てみると第３世代は、「顧客サービスの向上」と「情報システムの戦略的な使用」の時代とも定義することができる。インターネットでは、契約者の契約内容照会はもちろんのこと、保全業務も行えるようになってきている。その他にもコールセンターの構築や、個々の顧客のニーズに対応できるようにＣＲＭの構築などが行われてきている。また、情報システムが経営戦略にとって重要で欠かせない存在になってきているのが、昨今の特徴ということができるだろう。

（２）現在の保険システムとシステム開発の状況

当研究チームでは、保険Ｗｅｂアプリ開発の現在の状況を正確に捉えるために、日本ア

クチュアリー会の協力の下、各保険会社にアンケートを実施した。

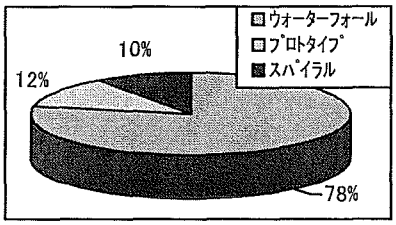
アンケートの実施要領は以下の通りである。

■	実施時期	：　２００５年６～７月
■	実施対象	：　アクチュアリー会賛助会員会社のシステム担当者
■	回答社数	：　６５社中４５社(回答率６９%)
■	回答手段	：　電子文書にて授受
■	内容	：　Ｗｅｂアプリ開発の現状など
■	方式	：　選択式，記述式

※以降、特に記述がないものは上記のアンケート結果である。

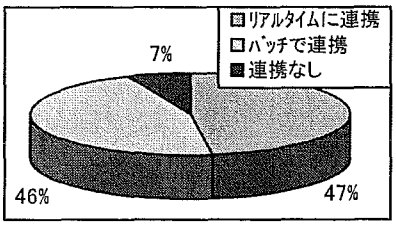
○「ソフトウェア開発モデルの採用状況」

保険会社システムの歴史の第２世代頃から登場してきた「ウォーターフォール開発モデル」（アンケート：７８%）を採用しているところが多い。これは第２世代の「メインフレーム中心の開発」の流れをそのまま引き継いで使用していると言うことがわかるとともに、「メインフレーム中心の開発」を長年行ってきたことにより他の開発手法への移行が難しくなってきている現状を表している、と捉えることができる。



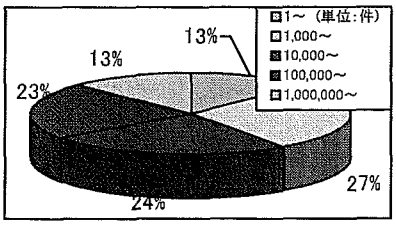
○「Ｗｅｂアプリとメインフレームの連携」

リアルタイム（アンケート：４７%）、バッチ（同：４６%）で連携している状況を見ると、第２世代で中心的な存在であったメインフレームも廃れることなく、第３世代で登場してきたＷｅｂアプリと連携しながら使用され続けていることがわかる。



○「Ｗｅｂアプリへのアクセス数」

社内からのアクセスだけでなく、インターネットからの利用が多くなってきていることに伴い、アクセス件数も月に１００万を越えるシステム（アンケート：１３%）も決して少なくない。



（３）分析結果による３つの問題点

「保険会社のシステムの歴史」と「現在の保険システムとシステム開発の状況」を踏ま

えて、「システム的観点」からの分析で問題点を３つに整理した。

ａ．テスト期間が短い

これは第２世代より使われ続けている「ウォーターフォール開発モデル」中心の開発に起因した問題である。「ウォーターフォール開発モデル」は開発工程を「要件定義／設計／製造／テスト」と言ったようなフェーズに分けて開発を進めて行くものであるが、Ｗｅｂアプリでは、実際にユーザーがテスト使用する段階でユーザーインターフェースの変更等の要件変更が多く発生している。そのため、前工程の開発フェーズに戻ることが余儀なくされ、結局はテスト期間が充分に取れなくなっているのが現状のようだ。

ｂ．システムの大規模・複雑化　影響範囲の特定が困難

２つ目の問題は、第２世代のメインフレームとの密接な連携にある。
大規模かつ複雑メインフレームのシステムとの連携は、１つのシステム変更がどこまでの影響範囲があるか特定するのを困難にさせている。昨今メディアを賑わせている「２００７年問題」に、大規模なシステムをすべて把握している人が少なくなってきている状況とは無縁ではない。

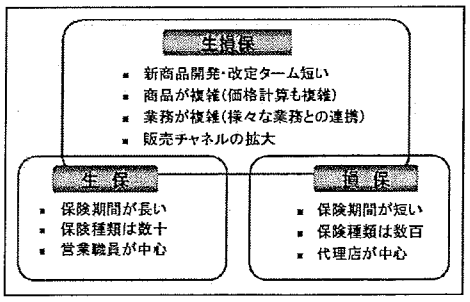
ｃ．システム負荷が増大

３つ目の問題は、インターネットやイントラネットからのアクセス数の増加に起因する。限られたユーザーしか使用されていなかった社内システムとは異なり、インターネットを経由して不特定多数のユーザーが利用するシステムがＷｅｂアプリでは多く開発されている。このような環境では、そのアクセス増加に伴うシステム負荷の増加に耐えられるシステム構築が必要になる。

２．業務的観点からの分析

（１）保険業務の特徴

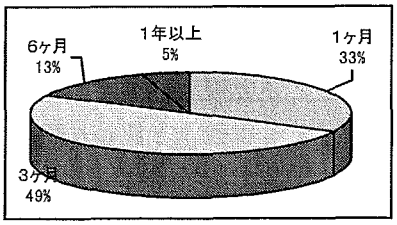
まずは保険業務の特徴を生命保険、損害保険、２者の共通という３つの括りで整理してみる。生損保の相違点には、保険期間、保険種類、メインとなる販売チャネルの違いなどはあるものの、「新商品の開発タームの短さ」「商品・業務の複雑性」「販売チャネルの拡大」など、多くの共通点があることがわかる。



保険会社へ行ったアンケートから保険業務の特徴を分析してみる。

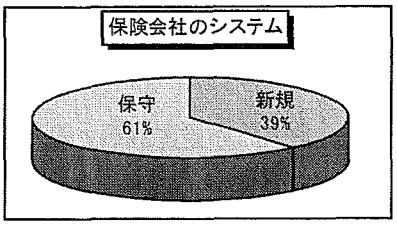
○「開発保守サイクル」

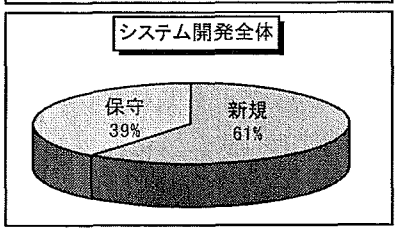
Ｗｅｂアプリの開発タームを見ると、３ヶ月未満の開発が８割を超えている。これは各社が新商品開発のタイミングにあわせてシステム対応を行っており、そのため、３ヶ月という短期開発を行っていることがわかる。



○「新規開発と保守の割合」

新規開発と保守の割合を「ソフトウェア開発業界全体」と「保険会社のシステム」で比較してみた。生損保共通の特徴である「新商品開発・改定タームが短い」ことからわかるように、保守案件の割合が６割に達している。「ソフトウェア開発業界全体」では比率が逆になり、「保守業務が多い」という保険会社システムの特徴が現れている。

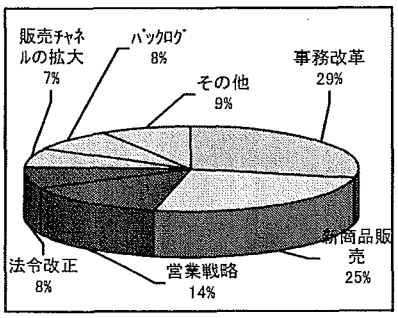




※ソフトウェア開発データ白書より

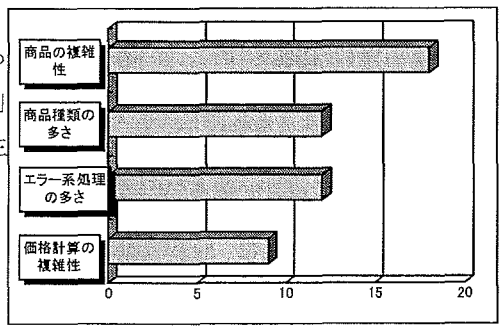
○「保守案件の内容」

保険会社システムの６割を超える保守案件の内訳を見てみると、事務改革（アンケート：２９％）と新商品対応（同：２５％）だけでも開発の５割以上を占めていることと、他にも営業戦略，法令改正など、多彩な保守案件があることがわる。



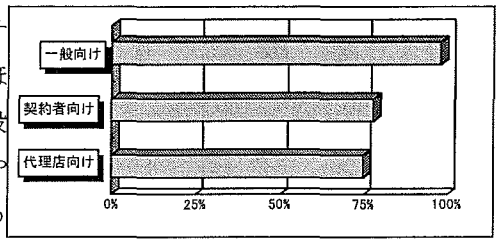
○「業務特性からくる複雑性の要因」

「商品の複雑性」「商品種類の多さ」さらにそれに伴う、「エラー系のチェック処理」「価格計算の複雑さ」がシステムの複雑性を生み出している。



○「社外向けサービスの提供（インターネット）」

インターネット系のシステムに絞ってサービスの提供状況を見てみると、保険会社のほとんどが社外向けサービスを行っており、一般ユーザー向けのサービスだけでなく、代理店や契約者向けのシステムも広く提供されていることがわかる。



（2）分析結果による５つの問題点

「業務的観点」の分析から問題点見つけ出し整理してみた。

a．テスト期間が十分に確保できない

アンケート結果の「開発保守サイクル」からわかる通り、開発期間が３ヶ月以内と短い。これは同時にテスト期間も短くなっていることを示している。このことから、全ケースを網羅するようなテストが時間的に困難であることで、十分なテストを行えていないと思われる。

b．プログラムなどのソース管理が難しい

「保守業務の多さ」と「保守案件の多彩さ」から同じプログラムソースを同時に複数案件で変更することがあり、さらにそれらのリリースタイミングが異なることによる、並行開発を強いられている。つまり、プログラムソースが複数世代存在する状態が多々あり、ソース管理をうまく行わないと、修正されたプログラムコードが数世代前に戻ってしまう、「デグレード」が発生する可能性が高い。

c．類似のテストパターンの繰り返しが多い

短期開発のうえ、開発期間を重複させながら行うことを常に行っているため、同じような検証を何度も行っている。これらを何の工夫もなしに実施することは非常に効率が悪いため、生産性に悪影響を及ぼしている。

d．精度を求められる

アンケート結果からもわかる通り、インターネットによる社外向けサービスの提供状況が高いことと、保険業務は基本的に「お金」を取り扱うものであるため、常に高い精度が求められる。

e．ロジカルチェックが多い

「業務システムの複雑性の要因」からもわかる通り、商品種類の多さや複雑性はプログラ

ムのロジックにそのまま反映され、ロジカルチェック、例えば商品規定のチェックやエラーチェック処理等が多くなることがわかる。これも短いテスト期間でいかに効率的に行えるかが課題である。

３．まとめ

「システム的観点」と「業務的観点」から洗い出された問題点を７つに整理した。

①テスト期間が短い
②他システムとの連携（大規模／複雑）
③類似のテストパターンの繰り返し
④精度を求められる
⑤ロジカルチェック（商品規定のチェック）が多い
⑥プログラムなどのソース管理が難しい
⑦不特定多数ユーザーのため負荷が増大

これらの問題点を解決することが今後の保険会社のＷｅｂアプリの課題であり、テストの効率化と生産性向上には欠かせない要因である。

第Ⅱ章　ツールによる解決方法

第Ⅰ章で提示された問題点を分析すると、以下の３点により問題の解決が可能と考えられる。

解決策	解決可能な問題
１．テストの自動化	①テスト期間が短い ②他システムとの連携（大規模／複雑） ③類似のテストパターンの繰り返し ④精度を求められる ⑤ロジカルチェック（商品規定のチェック）が多い
２．適切な資源管理	⑥プログラムなどのソース管理が難しい
３．正確な負荷測定	⑦不特定多数ユーザーのため負荷が増大

これらについて検討した結果を下記に提示する。

１．ツールによる解決方法の検討

（１）自動機能テストツールによる解決

自動機能テストツールを利用したテストとは、テストケースに沿ってテスターが確認事項をこなしていく方法ではなく、テストケースをスクリプト化し、テストツールで実行させるものである。

自動機能テストツールを利用するメリットは、短時間で多数のテストケース（スクリプト）が実行可能であること、同一のテストケース（スクリプト）を何度も繰り返し行えることがあげられる。そのため、テストケースを作成し、各テスターが手動で行うテストに比べて均一な品質のテストを実施することができる。

上記の理由により、以下の問題について解決できる。

①テスト期間が短い
②他システムとの連携（大規模／複雑）
③類似のテストパターンの繰り返し
④精度を求められる
⑤ロジカルチェック（商品規定のチェック）が多い

（２）資源管理ツールによる解決

資源管理ツールとは、システム開発に係わるプログラム資源を管理するツールである。資源管理ツールを利用することで、ソースコードやデータといったプログラム資源をバージョン毎に一元的に管理することが可能である。一元的に管理しているため、複数人の開発者からの変更履歴も管理することが可能になる。

上記の理由により、以下の問題について解決できる。

⑥プログラムなどのソース管理が難しい

（3）負荷テストツールによる解決

負荷テストツールとは、大量に仮想ユーザーを発生させ、ネットワークやサーバにアクセスし負荷をかけることで、耐久性や問題点を検証するテストツールである。
ツールを利用しない場合、想定されるアクセス数分の人およびＰＣを用意する必要があるが、ツールを利用すれば１台のＰＣで同等のテストが可能となる。

小規模システムであればさほど気にならないパフォーマンス劣化でも、大規模システムでは致命的な問題となりうる。１，０００，０００アクセス／月を超えるような保険Ｗｅｂアプリも存在しており、そのような件数に耐えうるシステム構築が不可欠になる。いまや、高い信頼性を要求されるシステムでは負荷テストは必須とも言える。

同時１００アクセス程度であれば、開発者を総動員することで負荷テストは可能となるが、１０００アクセスとなると現実的とはいえない。このようなテストを実施する場合、負荷テストツールを使用することで開発者を総動員することなく、システムの耐久性・問題点(ボトルネック)を検証する事ができる。また、限界点を把握することが可能なので、現在のシステム構成でどの程度のトランザクション数を処理できるか予め想定することも可能となる。

上記の理由により、以下の問題について解決できる。

⑦不特定多数ユーザーのため負荷が増大

２．ツールの概要

「１．ツールによる解決方法の検討」であげた「自動機能テストツール」「資源管理ツール」「負荷テストツール」について、概要や主な機能をまとめ、ツールで行えるＷｅｂアプリの問題の解決方法について分析を行った。

（1）自動機能テストツール

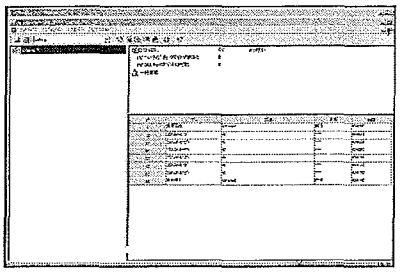
ａ．概要

自動機能テストツールとは、テストをスクリプト化し、それを再利用しテストを実施することでテストにかかる負荷を軽減するツールである。
代表的なツールとして、Mercury WinRunnerがある。

ｂ．主な機能

○自動比較機能

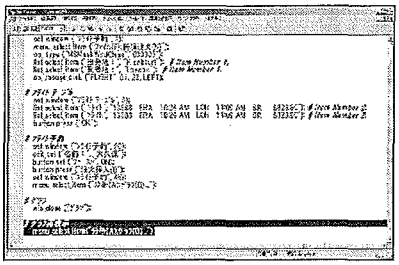
Ｗｅｂアプリで行うＤＢ更新や画面表示内容の整合性確認などデータ比較・検証が自動化できる。



※WinRunner／テスト結果表示画面

○自動入力機能

エミュレータから大量のコマンドを入力して出力ログの確認を行うスクリプトを作成することでコマンド入力を自動化できる。



※WinRunner／自動実行する際のスクリプト作成画面

ｃ．問題解決の方法

テストを自動化してテスト実行にかかる時間を短縮し、短い期間でも必要十分なテストを行うことが可能となる。

また、テストで使用したスクリプトを次回のテスト時に再利用することでテストにかかる負荷を軽減させることで問題の解決を図ることができる。

（2）資源管理ツール

ａ．概要

資源管理ツールはプログラムソース資源をバージョン管理するツールである。代表的なツールとして、ConcurrentVersions System、Microsoft Visual SourceSafeがある。

ｂ．主な機能

○資源管理機能

資源管理ツールでは、各プログラム資源を一括して管理できる。ツール単体でも稼動は可能だが、eclipse等の開発ツールと連携して利用することも可能である。

○バージョン管理機能

資源管理ツールではプログラム資源をバージョン毎に管理することができ、以前の状態に簡単に戻すことが可能である。

ｃ．問題解決の方法

資源を一括集中で管理することで、マスターや、最新状態のプログラム資源を把握することが容易である。さらにプログラム資源をバージョン管理することで、デグレード発生時のプログラム資源の復旧が容易に行うことが可能となる。

（３）負荷測定ツール

ａ．概要

負荷測定ツールとは、擬似的に大量ユーザーの負荷を測定できるツールである。
代表的なツールとして、Mercury LoadRunnerがある。

ｂ．主な機能

○負荷測定

負荷測定ツールを導入することにより、1台のマシン上で大量の仮想ユーザーの実行が可能である。それにより、擬似的に大量ユーザーの負荷を測定することが可能になる。仮想ユーザーの動きについてもテストシナリオとして保存できるので、再テスト実行やテストシナリオの再利用が容易にできる。

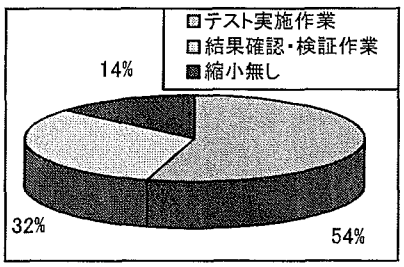
ｃ．問題解決の方法

従来、負荷テストは１００人の人が集まって同時に実行するというような方法をとっていた。それが、１人が負荷測定ツールを実行することで擬似的に１００人のアクセスを実現可能としている。負荷測定ツールを用いることで、様々な負荷テストを行うことが容易になる。

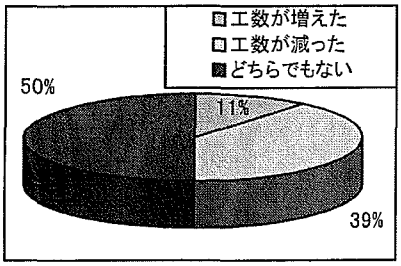
３．まとめ

現在のテストツールの利用状況及び利用局面において、保険会社にアンケートを実施した結果を紹介する。

○「テストツールを利用した結果大幅に短縮された作業はありますか」との問いにテスト実施作業（アンケート：５４％）、結果確認・検証確認（同：３２％）あわせて８６％が作業の縮小に繋がったとの結果になった。この事からツールを利用することでテスト時の効率化が図れ、浮いた工数を別の作業に割り振ることも可能になる。



○「利用する場面・局面で効果がありましたか」との問いは、工数が減った（アンケート：３９％）、逆に工数が増えた（同：１１％）となっており、テストツールの使用頻度・利用局面などはさまざまであるが、ツールを利用することにより、一定の効果は認められていると言える。



テストツールを使用する事により、今までの類似パターンによる反復的なテスト実施を「効率的」に行え、時間や手間が掛かっていた作業をテストツールで解決する事で、時間／人／作業が大幅に縮小される為、「生産性の向上」にも繋がる。

テストツールを使用することでテストフェーズに関しては解決できるが、開発プロセス全体にわたっての解決が図れていない。第Ⅲ章ではツールでは解決できない分析／設計フェーズで発生する問題を洗い出し、この解決方法について述べていく。

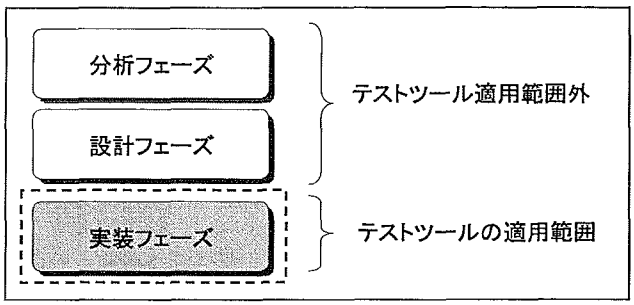
第Ⅲ章　ツールで解決できない問題（ＵＭＬ／ＭＤＡの活用）

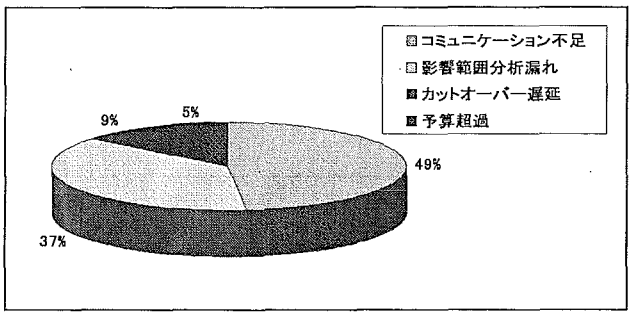
第Ⅱ章では、ツール（テストツールや資源管理ツール等）の効率的活用による生産性向上とテストの効率化について述べてきた。
この章では、テストツールでは解決することのできない分析／設計フェーズで発生する問題を洗い出し、その解決方法について考察していく。

１．テストツールで解決できない問題

前章で述べたテストツール（WinRunner等）は、テストフェーズで有用となるツールであり、分析／設計フェーズで発生する問題を解決することはできない。

分析/設計フェーズで発生する問題をアンケートから考察すると、分析フェーズの問題では「コミュニケーション不足」が４９％、設計フェーズでは「影響範囲分析漏れ」が３７％となっており、このことから、「コミュニケーション不足」と「影響範囲分析漏れ」が分析／設計フェーズで発生する問題と考えられる。





（１）　コミュニケーション不足

コミュニケーション不足を解決するためにはどのような手段が有用か、これはユーザーのビジネス要件を明確にするコミュニケーションツールが必要であると考えられる。

一般に複数の人間が協力して何かを実現しようとする場合には設計図が必要となる。建築であれば建築設計図、電化製品であれば回路図がある。これらの設計図をコミュニケーションツールとして使用すれば、言語の違いをこえたコミュニケーションが可能となる。

そこで、コミュニケーションツールとしてのUML（Unified Modeling Language）に着目し、UMLのメリットとデメリットを洗い出し、その有用性について考察していく。

（２）　影響範囲分析漏れ

オブジェクト指向開発においては、機能拡張等による変更に対し、その変更が及ぼす影響範囲を分析するための手法及び、ツールが存在しない。そのため、ほとんどが担当者判断の影響分析となり、リリース後に影響範囲分析漏れを起因とする障害が発生することが

ある。そこで、モデルを中心としたアプリケーション開発アプローチであるＭＤＡ（Model Driven Architecture）に着目し、その有用性について考察していく。

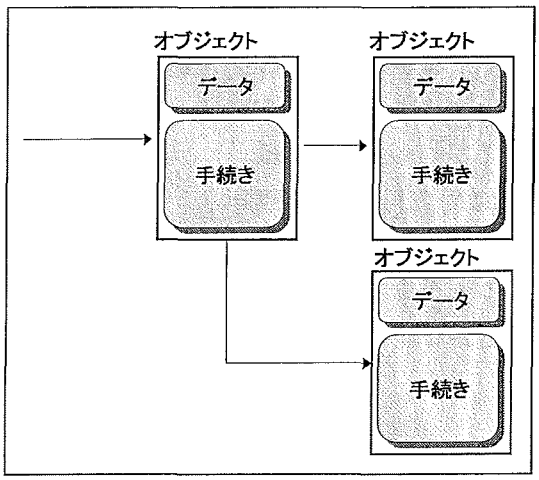
２．ＵＭＬ／ＭＤＡを活用した問題解決

（１） ＵＭＬ／ＭＤＡの背景にあるオブジェクト指向開発とモデル

ＵＭＬ／ＭＤＡを考察する前に、その背景にあるオブジェクト指向について述べる。

ａ．オブジェクト指向開発

オブジェクトとは、関連するデータと振る舞いについてまとめたものであり、これらオブジェクトを組み合わせることによりソフトウェアを構築することをオブジェクト指向開発と言う。オブジェクト指向とは、ソフトウェアの設計や開発における考え方の一つである。



例えば、テレビを例にオブジェクトを説明すると、メーカー問わず全てのテレビが共通で、電源ボタンを押せば電源がＯＮになり、チャンネルボタンを押せばチャンネルがかわる。これは、操作者がテレビ内部でどのような回路が動作しているかを理解する必要はなく、ただテレビを操作するための手順を理解していればよい。すなわち、テレビというオブジェクトは、自身を動作されるための振る舞いを知っており、内部動作の詳細は隠蔽している。これらオブジェクトを組み合わせたものがオブジェクト指向開発である。

ｂ．モデル

オブジェクト指向では、さまざまな視点からオブジェクトを捕らえる。

- ■ どのような構造のオブジェクトか
- ■ オブジェクト間でどのようなメッセージが交わされているか

これらは言葉だけでは伝えづらいので、図（ダイアグラム）を使って表すことになり、そうした図のことをモデルという。

モデルを用いることで、技術者間でより正確にオブジェクトに関する情報や考えを交換することができるようになる。例えば、特定の建物や場所の位置など、皆で共通できる情報を地図（モデル）で表すことにより、言葉では伝えづらい部分を明確にすることができる。

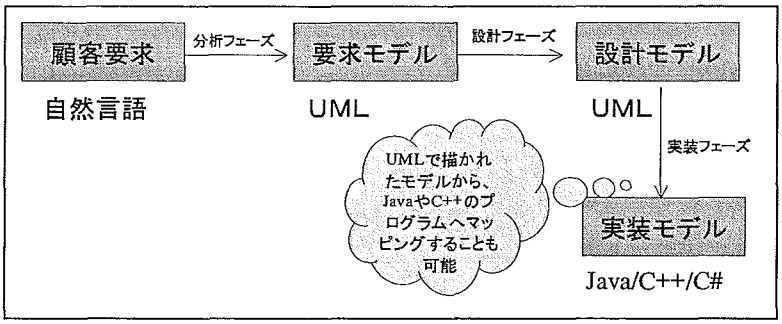
（2） コミュニケーションツールとしてのUML

a. UMLとは

UMLとは「Unified Modeling Language（統一モデリング言語）」の略であり、オブジェクト指向開発の分析・設計においてシステムをモデル化する際の記法（図法）を規定した言語である。UMLは言葉より図を多用した記法のため、UMLで作成した設計図はビジュアル的な設計図となる。

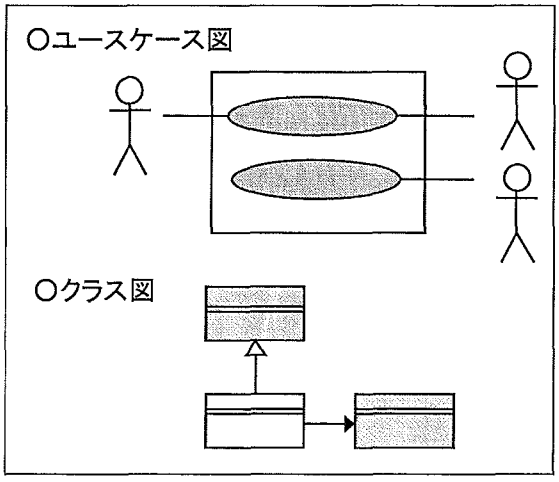
b. UMLの適用範囲

UMLの適用範囲は、分析フェーズから設計フェーズまでである。使用するツールによっては、UMLで作成された設計モデルを、Java や C++のプログラムにマップすることが可能である。



・ 分析フェーズ

分析フェーズでは、顧客要求（自然言語）から要求モデルを作成する。
要求モデルでは、システムのサービスを表現するための「ユースケース図」や、どのような概念があるのかを把握するための「クラス図」が作成される。



・ 設計フェーズ

設計フェーズでは、要求モデルをインプットに、システムの構造をあらわすための「クラス図」や、オブジェクトの振る舞いをあらわすための「シーケンス図（コラボレーション図）」を作成する。

c. UMLのメリットと問題点

○UMLの使用メリット

UMLは、分析フェーズから設計フェーズ及び実装フェーズまで、終始一貫して使用可能であり、UMLで作成した設計図はビジュアル的な設計図となる。このUMLで作成した設計図をコミュニケーションツールとして使用すれば、ユーザー及びシステム部門の枠を越えたコミュニケーションが可能となる。

このことから、UMLはコミュニケーションツールとして有用であると考えられる。

○UMLの問題点

UMLはコミュニケーションツールとして有用であるが、分析段階ではシステム化の範囲を分割しにくいため、一つのモデルが膨大で複雑になりやすい。また、設計モデルから実装への落とし込みに標準がなく、現状において、実際に使用できるのは分析／設計フェーズのみであり、実装モデルの作成は一からとなる。

（3） MDAを活用した影響分析

MDAとは「Model Driven Architecture」の略であり、モデル駆動型アーキテクチャと訳される。

MDAはOMG（Object Management Group）が策定する「モデルを中心に開発を進めるソフトウェア開発手法」であり、そのベースにはUMLが使用される。

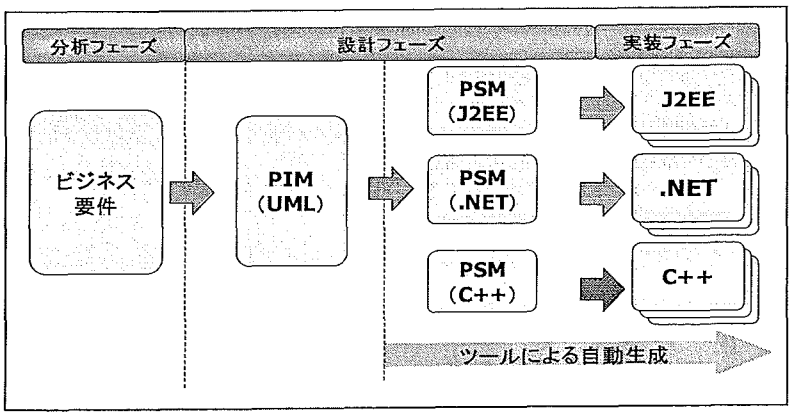
a. MDAの特徴

最大の特徴は、設計情報であるモデルをプラットフォームに依存しないモデル（PIM：Platform independent Model）とプラットフォームに依存したモデル（PSM：Platform Specific Model）に分離するところにある。

■ PIM (Platform Independent Model)
プラットフォームに依存しないモデルである。
プラットフォームとは、CPUやOS、ミドルウェア、言語などを表す。

■ PSM (Platform Specific Model)
プラットフォームに依存したモデルである。
よって、実際に開発対象となるプラットフォームと1対1に対応する。

開発者はビジネス要件からプラットフォームに依存しないモデル（PIM）を作成する。それ以降のフェーズであるPSMの作成及び、実装フェーズについては、ツールにより自動で行われる。つまり、開発者はPIMの作成に専念すればよい。



b. 影響分析の現状と、ＭＤＡを活用した影響分析

○影響分析の現状

アンケートの通り、８０％以上が影響分析の実施方法が標準化されておらず、担当者判断による影響分析となっている。そのため、現状の影響分析では、担当者判断で、プログラムベースでの影響分析となり、業務視点での影響分析が行われず、漏れが発生しやすい。

■影響分析の実施方法は標準化されているか

標準化されていない ⇨ ８６％

■影響分析の実施方法

担当者判断 ⇨ ８１％

○ＭＤＡを活用した影響分析

ＭＤＡにおいては、ＭＤＡの特徴で説明したとおり、開発担当者はＰＩＭの作成のみであり、以降の作業はツールにより自動で行われる。つまり、ＭＤＡを活用した影響分析では、このＰＩＭで作成したモデルをベースに行えばよく、業務視点での影響分析が可能となる。

モデルでの影響分析は変更に伴う影響をビジュアル的に特定できるため、影響分析漏れが発生しにくいと考えられる。

４．まとめ

ＵＭＬ／ＭＤＡの活用で述べた通り、分析／設計フェーズで発生する問題を解決するためには、ＵＭＬ及びＭＤＡの活用が有用であると考えられる。

（１） コミュニケーション不足

ＵＭＬで作成した設計図はビジュアル的な設計図となるため、これを活用することによりユーザーとシステム部門の枠を越えたコミュニケーションが可能となる。

（２） 影響範囲分析漏れ

ＭＤＡを活用することで、担当者判断による影響分析ではなく、業務的視点での影響分析が可能となる。ＭＤＡはＵＭＬをベースとしているため、変更に伴う影響をビジュアル的に特定できる。

第Ⅳ章　Ｗｅｂアプリケーションのテストの現状と課題

第Ⅳ章ではＷｅｂアプリのテストの現状と課題について、保険会社各社からのアンケート結果を元に現状と課題を分析し、それらの解決方法について考察していくこととする。

１．Ｗｅｂアプリケーションのテストの現状について

アンケートの結果から、全体の８２％の保険会社で「テストの現状に満足していない」という結果が明らかとなった。
多くの保険会社が問題点や課題を抱えたまま、ソフトウェア開発・アプリケーションテストを実施しているという現状が浮かび上がってきている。
以下に主な問題点・課題を挙げる。

- テスト計画・戦略が整備されていない
- テストの基準・評価が明確ではない
- テスターにスキルの差があり、人依存になっている
- 優秀な人材は開発チームに取られてしまう
- テストケースの洗い出しができない・テストにかける時間がない
- テストツールが難しい。専門家が必要
- テストツールのスクリプトの作成・保守が大変
- テスト範囲、影響範囲の把握漏れが多い

当研究チームでは、これらの問題点・課題を「テスト体制・意識の問題」・「人・リソース・スケジュールの問題」・「テストツール・開発手法の問題」の大きく３つに分類し、それぞれの問題を解決するための提言を行うこととする。

２．「テスト体制・意識の問題」について

（１）Ｗｅｂアプリ開発におけるテスト体制・意識の問題
「テスト体制・意識の問題」には以下のものが該当すると考える。

- テスト計画・戦略が整備されていない
- テストの基準・評価方法が明確ではない
- ユーザー部門の参加意識が乏しい
- テスト環境が整備されていない
- 部門間の連携・情報共有が不足している

Ｗｅｂアプリ開発では、第Ｉ章で述べたとおり短期開発となるケースが多く、テスト計画作成のための工数確保が困難であると推測できる。また、オープン系開発の特徴でもあるが、開発･テストにおいて個人依存の度合いが高く、テスト基準･評価方法が標準化されていないことも問題視されている。従来からの課題でもある「ユーザー部門の開発･テストへの参加意識が乏しい」といった意見も根強く残っており、こういった問題の解決を求められている。

（2）「テスト体制・意識の問題」を解決するための提言

では、「テスト体制・意識の問題」を解決するにはどうすればよいのだろうか。
当研究チームでは５項目に及ぶ改善案を提言する。

- ■ テスト基準・テスト範囲・ゴールを明確にする
- ■ プロのテストエンジニアを育成できる組織・体制作りを進める
- ■ ユーザー部門も参加したプロのテストチームを構成する
- ■ テストチームにも権限・予算を付与する
- ■ テストも重要なプロジェクトとの意識改革が必要

これらの提言はすべて「テスト･テスターの地位向上」というゴールを目標としている。テストは「品質保証」という重要な作業だが、ソフトウェア開発においては軽視されるケースが散見される。例えば、「開発者とテスト担当者が同じ」･「独立したテスト部門はあるが経験の浅い開発者を配置している」･「ＩＴ技術者は配置しているが、保険業務に精通した担当者が不在」などである。

「テスト･テスターの地位向上」を推進し、「プロのテスト集団」を構成することで、より精度の高いテストが実現できる。また、開発チームも本来の開発作業に集中することができるので、「生産性の向上」も実現できると考える。

なお、テストチームを根付かせるために、教育・ローテーション・キャリアパスの明示など、人事面のサポートの必要性も合わせて提言したい。

３.「人・リソース・スケジュールの問題」について

（1）テストにおける人・リソース・スケジュールの問題

「人・リソース・スケジュールの問題」には以下のものが該当すると考える。

- ■ 優秀な人材は開発チームに取られてしまう
- ■ テスターのスキルに差があり、人依存のテストになってしまう
- ■ テストケースの洗い出しができない
- ■ テストにかける時間がない・スケジュールが厳しい

アプリケーション開発において、納期の遵守が重要な命題だが、スケジュール通り順調に開発が進むケースは少ない。開発スケジュールに遅れが発生すると、多くの場合はテスト期間として確保している日程を利用し、遅れを吸収するケースが多いと思われる。

また、テスト工数を見積もる際も、テスト項目作成・事前準備期間などが考慮されていないままテスト計画が立案され、非現実的な見積もりとなってしまうケースもあるではないだろうか。特にテスト担当者のスキル・経験が十分ではない場合にこういった問題が発生する可能性が高い。

このような「リソース・スケジュールの問題」の改善案について検討した。

（2）「人・リソース・スケジュールの問題」解決のための提言

では、「人・リソース・スケジュールの問題」を解決するにはどうすればよいのだろうか。当研究チームでは３項目に及ぶ改善案を提言する。

- テストを１つのプロジェクトとして計画・見積もり・実施する
- テスト計画もきちんと策定し詳細化すること
- 優秀な人材をテストプロジェクトに投入しモデルプランを作成する
  ※そのノウハウを元に手法・プロセスの標準モデルを作成する

我々は、テストを開発プロジェクトのタスクの一つとして考えるのではなく、開発プロジェクトと平行する「テストプロジェクト」として独立させ、１つのプロジェクトとして、計画・見積もり・実施を行うことを提言する。

すべてのプロジェクトで実施することが理想だが、システム部門の規模によっては、難しい場合もあるだろう。そういった場合は、中規模プロジェクトでモデルケースとして実施しモデルプランの策定を行い、次回以降のプロジェクトで活用するだけでも効果があると考える。

なお、モデルプラン策定においては、次のような点に留意する必要があると考える。

- 設計・詳細見積り・人員配置など、開発プロジェクト同様の手順を実施する
- 第３者の視点でテストが実施できるような人員配置をする
- 責任範囲を明確にし、必要なリソース・人員を確保する
- テストツール･自動化を積極的に活用する
- テスト粒度を細かくし、手戻りによる再テストの影響を最小限にする
- 開発の進捗に合わせた、柔軟なスケジュール調整を可能とする

４．「テストツール・開発手法の問題」について

（１）テストツール・開発手法の問題

第Ⅱ章でテストツールの紹介をしたが、実際に導入・利用している保険会社は少ないことが、アンケートの結果で明らかになった。下記にその結果を示す。

○「テストツールの導入・利用状況」

有償テストツールの導入・利用状況だが、「導入・利用している」（アンケート：３５％）と利用率は低い。無償テストツールになると、「導入・利用している」（アンケート：１６％）とさらに利用率は下がる。

無償ツールの場合、サポート・セキュリティの課題があり導入が難しい面もあるが、テストツール自体が保険会社にあまり浸透していないことがわかる。

○「テストツールの必要について」

テストツールの必要性についてだが、「必要」（アンケート：８２％）と肯定的な回答が多数を占めている。つまり、必要性は感じているがなかなか導入に踏み切れないのが現状のようだ。

（２）テストツールが導入できない理由

アンケートの結果より、テストツールの必要性は感じているが、導入できていない実情が明らかになってきた。以下に導入できない代表的な意見・理由をまとめてみた。

- 価格が高く、容易に導入できない
- 機能不足、独自環境では動かない。製品仕様が合わない
- 操作方法が難しく熟知できない。専門家が必要
- 費用対効果が期待できない
- テストツール（スクリプト）の作成・保守が大変

このように、コスト・操作性・機能などに問題を感じていることがわかる。

テストツールは、ＣＲＭ・ＥＲＰパッケージなどと比較するとコストの割に成果がわかりにくい面がある。また、どのようなテストにも対応すべく機能が “天こ盛り” となっているため、かえって利用者に高いスキルを要求する場合もある。

（３）テストツールを活用するには

では、テストツールを導入・活用するにはどうすればよいのだろうか。

以下にテストツール導入にあたり、留意すべき点についてまとめてみた。

■ 導入方法・使用方法・運用方法の方針は事前に決めておく
■ ツールが得意なこと、不得意なことは最初に理解しておく
■ コストを考え、有償・無償ツールを使い分ける
■ スペシャリストを育成する。開発チームの片手間では難しい
■ 会社・グループで共有化できるものを選定する（資産・知識の共有）
■ テスト全体を見渡し、再利用性の高いスクリプトの作成を心がける

テスト計画段階から、これらの留意点を考慮しておけば、テストツールは十分活用できると考える。また、再利用性の高いシナリオ・スクリプトを作成／共有することで、次回以降のテスト工数の削減も可能となる。

つまり、「テストツールの特性を理解する」・「適材適所で活用」・「繰り返し使う」ことが定着しテストツールをうまく活用できれば、高い費用対効果が見込め、生産性の向上にも繋がると考える。

また、そのレポート・分析機能は非常に充実しており（特に有償ツール）、テスト成果物の作成、ボトルネックの検出などでも活躍が期待できる。

（4）テストツールの活用で期待できる効果

テストツールで期待できる効果を具体的に以下にまとめてみた。

- 繰り返し利用することでシナリオ・スクリプトが充実
  ⇒ 高い再利用性を実現(費用対効果が高い)
- 回帰テストの自動化・手順の標準化・スキルの平均化
  ⇒ 人依存のテストからの解放・スキルの均一化
- ボトルネックの発見
  ⇒ 人による負荷テストにかかる時間・コストの軽減
- テスト担当者のスキルに依存しない高品質なテスト
  ⇒ テスト要員の単価削減
- 豊富なドキュメント
  ⇒ テスト成果物作成工数の軽減、ドキュメントの標準化

これらの効果が期待できると考え、当研究チームではテストツールの積極的な活用が、生産性の向上・効率化に貢献できるとの結論に達した。

第Ⅴ章 結論 ― テストの効率化と生産性の向上を図るには

最後に、テストの効率化と生産性の向上を図るためのポイントを簡単にまとめてみた。

1．テストツールの積極的な活用

第Ⅱ/Ⅳ章でも述べたが、テストツールの活用はテストの効率化に非常に効果的だと考える。特に保険会社のシステムにおいては、新規開発よりも保守・メンテナンスの比重が高いことが特徴である。つまり、稼動中のアプリケーションに修正を加えるケースが多く、回帰テストの重要性が非常に高い。このような作業にぜひ活用していくべきだと考える。

2．ＵＭＬの積極的な活用

ＵＭＬはデファクトスタンダードとなりつつあり、ユーザー部門とのコミュニケーションツールとしてだけではなく、オフシュア開発においても活躍が期待できる。また、将来的にはＭＤＡなどとの連動も期待できるため積極的な活用を提言したい。

3．テスト・テスターの地位向上

最近では就職情報誌などでも「テスター」の募集が増えてきている。今後アプリケーションがさらに複雑化・巨大化することが予想され、テスターの経験・スキルは非常に重要なものとなると想像できる。そういった人材の確保・育成をするためにもテスト・テスターの地位向上・専門職化は不可欠と考える。

4．テスト計画の見直し

テストは製造業で例えると「品質管理・品質保証」という重要な役割であることを再認識し、テスト計画・スケジュール・人員配置の重要性を再考すべきだと考える。

5．組織・体制の見直し

前述の提言と重なる部分もあるが、テスト・テスターに地位の見直しを図るにあたり、組織・体制の見直しも視野に入れておく必要があると考える。また、キャリアパスの提示・役職・ジョブローテーション・教育など、人事面でのサポートも重要だと考える。

おわりに

保険業界は規制緩和により、ますます商品が非常に多様化・複雑化することが予想され、いまやシステム対応なしでは保険料計算すらできない時代に突入している。
また、「銀行窓販」に代表されるように、いろいろなチャネルへの進出がますます盛んになることも予想され、それぞれのニーズに合った新商品・新特約開発も激化することだろう。
このような状況の中、システム開発のスピードアップ・開発効率化・精度の向上は、経営戦略上非常に重要な課題となってきており、ますますシステム部門への期待や重圧は増すことだろう。

今回、当研究チームでは「Ｗｅｂアプリケーション開発のテストの効率化と生産性の向上」というテーマで、主にテストツールと開発手法の研究を中心に進めてきた。
研究当初、ＸＰ(eXtreme Programming)などに代表される「アジャイル開発」や「テストツールの活用」などで生産性向上やテスト効率化が図れると期待していた。
ところが、研究を進めていくうちに「アジャイル開発」は、保険会社システム部門の業務の中心となりつつある、「３ヶ月程度の短期開発・メンテナンス」に不向きということが判明した。
また、保険会社の多くはいまだ「メインフレーム」を中心に据えたシステム構成となっており、「ウォーターフォール開発モデル」がフィットする開発案件も多い。

我々は、最新の開発手法を取り入れるだけではさまざまな問題は解決できないと判断し、「ツールの活用」・「テストの地位向上」・「組織・体制の見直し」などを提言した。
なかでも「組織・体制の見直し」などは非常に重要な課題だと考える。
最近話題となっている「２００７年問題」で、今までシステム部門を支えてきた熟練エンジニアの定年を迎え始める。彼らは保険業務に精通しており、テストに関するノウハウも非常に豊富である。若手エンジニアがこういった知識を吸収する場として、「テストプロジェクト」を立ち上げることは非常に有意義であり、面白い試みだと思うがいかがだろうか。

最後にこの論文を作成するにあたり、アンケートにご協力いただいた各社の皆様、
ならびにサポートいただいた関係各位に深くお礼申し上げます。

＜参考文献・商標＞

・@IT-アットマーク・アイティー http://www.atmarkit.co.jp/

・日経BP社 ソフトウェア開発データ白書2005

平成18年2月20日　印刷
平成18年2月20日　発行

発　行　所　社団法人　日本アクチュアリー会
東京都中央区晴海1−8−10
晴海アイランド トリトンスクエア
オフィスタワーX 2階
電　話（5548）6033

発　行　者　日　笠　克　巳

印　刷　所　株式会社　サ　ン　ワ
電　話（3265）1816