Vol.2012-SE-177 No.11
2012/7/20

実行時情報に基づく Web アプリケーション可視化手法

伊藤秀朗† 団野博文† 三部良太†

情報システム保守の効率化への関心は高まっている．本論文では，実行時情報に基づく Web アプリケーションの振る舞いを可視化する手法を提案する．提案手法の特徴は，HTTP GET/POST 送信データ，アプリケーション・プログラムの呼出し，発行した SQL 文の情報(＝実行時情報)を，クライアント PC が送信したリクエストに紐付け可視化することである．提案手法に基づいてプロトタイプを作成し，実システムの保守現場を対象に実証評価を行った結果，システム保守のうち原因調査工程において工数を約 46%削減する効果があることを確認したので報告する．

A Visualization Technique for Web Application Based on its Execution Log

HIDEAKI ITO† HIROFUMI DANNO†
RYOTA MIBE†

Recently, software maintenance activity becomes more important for information systems. This paper describes a visualization technique for web application behavior based on its execution log. A feature of our visualization technique is to link the execution log to HTTP requests sent by client PCs. The execution log includes HTTP GET/POST methods, application program call history, SQL statements executed by application programs. We conducted a case study with applying a prototype tool to maintenance activity of a real information system. As a result of evaluation, the proposed approach reduced about 46% of problem analysis task efforts.

1. はじめに

近年，情報システムの導入によって業務の効率化が図られている．一例に，製造業・流通業などにおいて，調達・生産・流通・販売といったサプライ・チェーン全体の最適化を実現する SCM（Supply Chain Management）システムがある．

情報システムは，そのトラブルや企業内外環境の変化に対応する為，継続的な保守を必要とする．しかし，情報システムの保守費用がしばしば多額であることが問題に挙げられる．IT 予算の約 6 割が保守費用に充てられているという報告もある[1]．このような背景から，企業情報システム保守の効率化への関心は高まっている．

JUAS によれば，システム保守の各工程の工数比率の分布は，その形状から「ふたこぶラクダ」モデルと呼ばれる[2]．これは，システム保守工程（原因調査，修正案作成，修正，テスト，受入・移行）のうち，原因調査工程とテスト工程に要する工数の割合が他工程に比べ多いことを指している．

システムの保守において障害の原因調査は重要なタスクの 1 つであり，プログラムの動作を把握し原因箇所の切り分けを行う．一例として，Web サーバのログ，業務プログラムが出力するログ，DB サーバのログ等から時間近接を頼りに関連するログを抜き出し紐付け，Web アプリケーションの動作を可視化する方法がある（図 1-1）．しかし，Web アプリケーション・システムでは複数 PC による同時アクセスが頻発し，時間近接だけでは関連するログを紐付けることすら困難である．また，前述のログは一般に膨大であり，人手による関連ログの紐付けは地道で手間の掛る作業でもある．

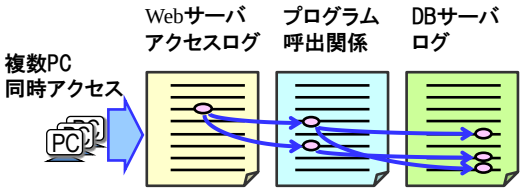

図 1-1 実行時の動作の把握手段の一例

本論文では，システム保守における原因調査に要する工数軽減を目的とし，実行時情報に基づいて Web アプリケーションの振舞いを可視化する手法を提案する．提案手法に基づいてプロトタイプを作成し，実システムの保守現場において実証実験を行った．その結果，システム保守の前半工程，特に原因調査工程において工数を約 46%削減する効果を確認した．

2. Web アプリケーション可視化手法

2.1 提案手法のコンセプト

Web アプリケーションの動作を可視化するには，Web3 階層の各処理を紐付けるキーが重要である．そこで，クライアント PC から送信されたリクエストをキーとして実行時情報の各データを紐付け Web アプリケーションの振舞いを可視化する手法を提案する．ここで，実行時情報は，リクエスト・メッセージ・データ（HTTP の場合，URL，

† （株）日立製作所 横浜研究所
Hitachi, Ltd. Yokohama Research Laboratory

GET/POST データ，Cookie など)，プログラム呼び出し関係データ（呼び出し元/先のメソッド名やそれが属するクラスの完全修飾クラス名など)，DB アクセスデータ（発行された SQL 文など）から成る．図 2-1 を用いて提案手法のアウトプットイメージを説明する．クライアント PC の画面操作による画面遷移などによって Web3 階層毎にログ出力される(図 2-1 の点線枠)．提案手法では，画面１から画面２に遷移する際に送信されたリクエスト(req001)に関連する Web3 階層毎のログのみ(リクエスト・メッセージ・データ１，プログラム呼出関係データ１，DB アクセスデータ１)を抽出することができる．

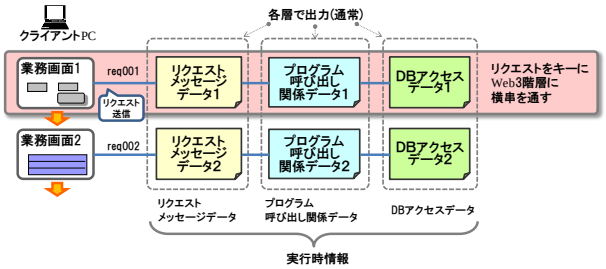

図 2-1　提案手法のアウトプットイメージ

また，提案手法では稼動中の Web アプリケーション・システムを前提とし，上述の実行時情報を，ミドルウェアであるアプリケーション基盤にて取得する．つまり，Web アプリケーションのソースコードを改変せずに上述の実行時情報を取得する．提案手法をソフトウェアのレイヤで実現する場合，アプリケーション毎にリクエスト ID を持ち回るためのソースコード改変が伴う．

提案手法の概要を以下に示す；

(1) アプリケーション基盤のサーブレット・コンテナにて，受信したリクエスト毎に識別子(＝リクエスト ID)を付与し，リクエスト ID とともにリクエスト・メッセージ・データ(HTTP GET/POST 送信データ, URL など)を出力する，
(2) サーブレット・コンテナは，リクエストと１対１に対応するスレッドのローカル領域にリクエスト ID を格納する，
(3) プログラム実行基盤(JavaVM 等)においてスレッドのローカル領域のリクエスト ID とともにプログラムの呼び出し関係データ(呼び出し元/先メソッド名など)を出力する，
(4) アプリケーション基盤が有する JDBC API の実装コンポーネントにおいて，スレッドのローカル領域のリクエスト ID とともに DB アクセスデータ(発行した SQL 文など)を出力する．

提案手法により，リクエスト ID を基にして業務画面とそれに関連する DB のテーブルとを洗い出すことができ，さらに DB のテーブルからプログラムへと遡ることもできる．そのため，Web3 階層に対して横断的に Web アプリケーション・プログラムの動作を把握することができ，システム保守担当者による動作把握や不具合調査が容易になることが期待できる．

2.2 提案手法の実装

提案手法を，弊社アプリケーション実行基盤製品 uCosminexus Application Server(以下では，Application Server と表記)にプロトタイプ実装した．以下では，提案手法を実装した箇所(図 2-2 参照)をまとめて APTracer と呼称する．

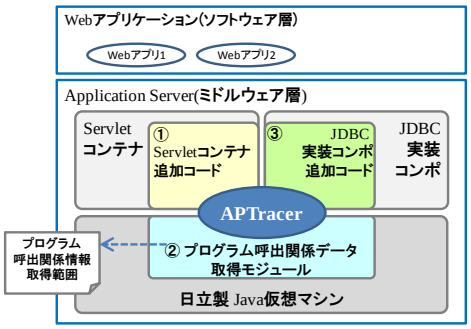

図 2-2 提案手法の実装箇所

2.2.1 APTracer による実行時情報の取得

提案手法を実装した箇所(①Servlet コンテナ追加コード ②プログラム呼出関係データ取得モジュール③JDBC 実装コンポ追加コード)について説明する．まず，①は，サーバに到着したリクエストに対して，次の３処理を実行する；

A) リクエストと１対１に対応するスレッドのローカル領域にリクエスト ID を格納する，
B) リクエスト・メッセージ・データを取得する，
C) リクエスト ID と共に処理 B で取得したリクエスト・メッセージ・データをファイル等に出力する．

Servlet コンテナはサーバに到着したリクエストに応じてコンテキストに処理を手渡す役割を担っているため，上記３処理をコンテキストにリクエストが渡る直前に実行できる．なお，処理 A では，標準クラスライブラリとして提供されている java.lang.ThreadLocal クラスを利用した．処理 B では，javax.servlet.http.HttpServletRequest インタフェースの getter を用いて，URL，HTTP GET/POST データなどのリクエスト・メッセージ・データを取得している．次に②では，Bytecode Instrumentation 機能a)を利用し，呼び出し元／先のメソッド名等を取得してスレッドのローカル領域のリクエスト ID と共にファイル等に出力するような処理を実行するバイトコードをメソッドの先頭と終了命令直前に埋め込む．これにより，ソースコードを改変することなく，実行時にプログラムの呼び出し関係データを取得できる．また，

a) Bytecode Instrumentation 機能は，JavaSE5 より導入され，クラスロード時のバイトコード操作によって，あたかも Java プログラムを書き換えたかのような動作をさせることができる．

バイトコードを埋め込む対象範囲を指定することもできる(例えば, 図 2-2 のようにプログラム呼出関係情報取得範囲ファイルに取得したいパッケージ／クラス名を指定する). 最後に③では，Application Server の JDBC 実装コンポーネントに対し，DB アクセスデータを取得し，スレッドのローカル領域のリクエスト ID と共にファイル等に出力する処理を追加した．対象とした実装クラスの JDBC API は以下である；

- javax.sql.DataSource,
- javax.sql.Connection,
- javax.sql.Statement,
- javax.sql.PreparedStatement,
- javax.sql.CallableStatement,
- javax.sql.ResultSet,
- javax.sql.ResultSetMetadata.

例えば，javax.sql.Statement#execute(sql: String)の引数は，発行された SQL 文そのものであり, 引数値を取得する処理を追加している．

また，APTracer では，取得する実行時情報は，APTracer に対して実行時情報の取得開始・終了を指示したユーザに由来するもののみであり，他ユーザのクライアント PC からのリクエストに関係する実行時情報は取得しないよう制限している．例えば，システム保守担当者 a は不具合再現のため不具合のあった業務を遂行してリクエスト A，B，C を業務システムに順に送信し，システム保守担当者 b は別作業で業務システムにリクエスト D を送信した場合を考える（図 2-3 参照）．システム保守担当者 a が APTracer に対して実行時情報の取得を指示した期間(この識別子を以下では“テスト ID”と呼称)に送信されたリクエスト B の実行時情報のみ(図 2-3 中の(2)～(4))を取得する．また，システム保守担当者 a が実行時情報の取得を指示した期間内にシステム保守担当者 b もリクエスト D を送信しているが，APTracer はリクエスト D の実行時情報は取得しない．これによって, 実行時情報の参照時にリクエスト ID ではなく，テスト ID をキーに検索することができる．また，取得すべき実行時情報の量を低減することにもなる．

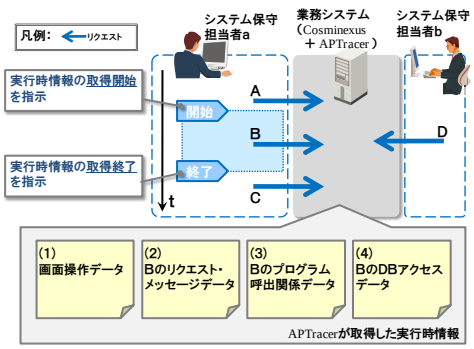

図 2-3　APTracer が取得する実行時情報

なお, APTracer は, ブラウザ上の操作情報(マウスの軌跡, テキストボックスの入力値とその入力順序など)を取得する機能と連携し，システム保守担当者のブラウザ上の操作情報をテスト ID と紐付ける．

2.2.2 APTracer で取得した実行時情報の参照

テスト結果参照アプリケーションによって，APTracer が取得した実行時情報を参照する．図 2-4 は，テスト結果参照アプリケーションの画面遷移を示している．実行時情報の各データを参照するには,「記録一覧」画面に表示されたテスト ID 一覧(他ユーザによるテスト ID も表示)より目的のテスト ID を選択する．選択後，記録詳細画面に遷移し「記録詳細一覧表」が表示される．「記録詳細一覧表」の各列は，ブラウザ操作情報，リクエスト・メッセージ・データ，プログラム呼び出し関係データ，DB アクセスデータに対応する．また,「記録詳細一覧表」の各行はサーバに送信されたリクエストに対応する．図 2-4 の例では，５つのリクエストがサーバに送信されたことを表している．

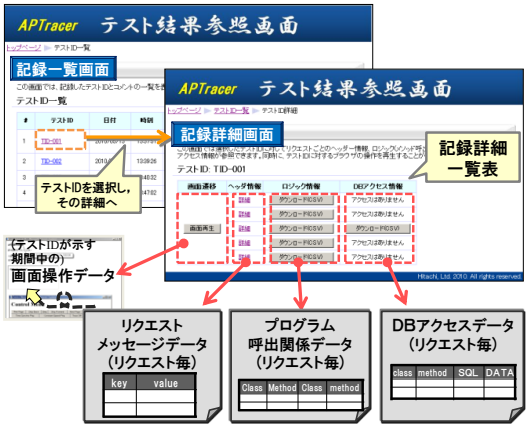

図 2-4　テスト結果参照アプリケーションの画面遷移

3. 実証実験

Application Server 上で実稼動する Web アプリケーション・システム(以下，システム X)とそれを保守する IT 部門を対象に，提案手法の実証実験を実施した．実施期間は，事前調査（Phase 0: 2009 年 12 月～2010 年 1 月）と APTracer 適用期間（Phase 1: 2010 年 7 月末～2010 年 9 月）に分けて実施した．Phase 0 では APTracer を適用する前のシステム X の保守現場の実態を把握し，Phase 1 と比較した．

3.1 APTracer の導入先

システム X には，本番機／検証機／開発機があるが，本実証実験では開発機にのみ APTracer を導入した．開発機は，不具合の再現，プログラムの修正／機能追加時にシステム保守担当者が利用する．但し，開発機上でシステム X を動作させるためにはシステム保守担当者が予め適切な DB のデータを用意する必要がある（検証機は，受入テストが実施されるため，本番機の DB とほぼ同期が取られた環境である）．

3.2 APTracer 適用現場の情報収集

APTracer を利用者の観点で評価するため，システム X の保守担当者が，案件毎の工数（工程別）や APTracer の試用評価などを「案件別工程別工数記入シート」に記入した．また，案件進捗を管理している運用担当者が，案件番号，案件の規模などの情報を「案件別情報シート」に記入した．

案件別工程別工数記入シートは Excel ファイル（図 3-1 参照）で用意し，システム保守担当者は以下項目の数値を記入する；

案件番号：保守案件を識別する番号，
案件種別：「保守」「機能追加」のどちらか，
コメント：案件の簡単な説明，APTracer の試用感想等，
工数：工程別に，案件に要した 1 日あたりの時間，
試用評価：工程別に APTracer の試用の評価（以下参照），
　1：現状把握に利用した
　2：他の担当者との情報連携に利用
　3：成果物作成に利用
　4：その他（上記 1～3 以外で利用）
　8：APTracer を利用できなかった
　9：APTracer を利用しなかった

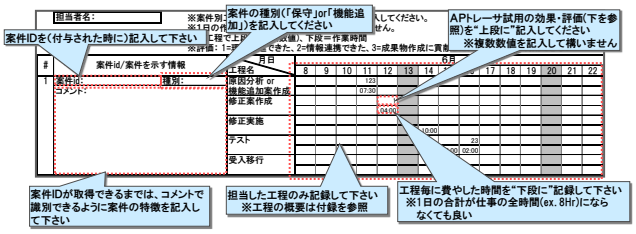

図 3-1　案件別工程別工数記入シートの例

一方，案件情報シートは Excel ファイル（図 3-2 参照）で用意し，システム X の運用担当者は，案件の全工程が終わり次第，その案件に関する以下項目の数値を記入する；

案件 ID：保守案件を識別する番号，
案件種別：「保守」「機能追加」のどちらか，
見積工数：案件の見積工数，
実績工数：案件の実績工数，
対象規模：案件で修正対象クラスの総ステップ数，
修正規模：案件で修正を行った箇所のステップ数，
テストケース数：実施したテストケース数，
B 票へのリンク：案件に関連するドキュメントの ID やファイル名を記入．

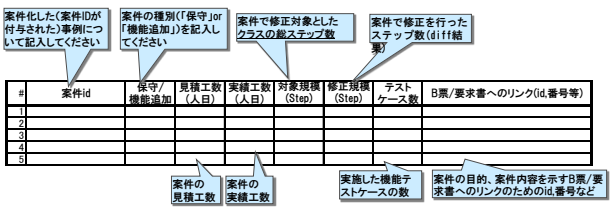

図 3-2　案件情報シートの例

なお，Phase 0 においても，案件別工程別工数記入シート及び案件別情報シートを利用して APTracer を適用していないシステム X の保守現場の情報を収集した．

3.3 結果

案件別工程別工数記入シートについて，Phase 0 では案件 140 件分，Phase 1 では案件 83 件分の情報を収集した．さらに，案件情報シートについて，Phase 0 では案件３９件分，Phase 1 では案件 52 件分の情報を収集した．

案件別工程別工数記入シート及び案件情報シートを通じて得られた情報をうち，APTracer の効果検証の評価対象とした案件の数を表 3-1 にまとめる．

表 3-1　Phase 0 及び Phase 1 における評価対象案件数

		原因分析	修正案作成	修正実施	テスト	受入移行
事前調査（案件総数140件）	案件情報有	39件	39件	39件	39件	39件
	工数数値有	13件	6件	13件	12件	10件
APトレーサ適用（案件総数83件）	定性評価有	23件	47件	60件	56件	18件
	案件情報有	52件	52件	52件	52件	52件
	工数数値有	10件	25件	37件	43件	11件

以下では，Phase 0，Phase 1 において評価対象とした案件の選別の方法を説明する．

3.3.1 Phase 0 において評価対象とした案件

Phase 0 において案件別工程別工数記入シートから収集した案件 140 件分のうち，案件情報シートに各種情報が記載された案件は 39 件であった（図 3-3 では，案件Ａと案件Ｂが相当）．残りの 101 件（＝140 件－39 件）は，事前調査期間内で完了しなかった案件と考えることができる（図 3-3 では，案件Ｃ，案件Ｄが相当）．

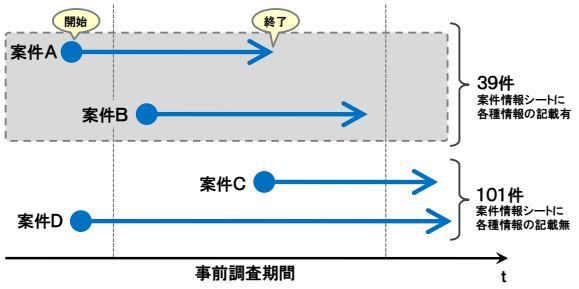

図 3-3　評価対象とする案件

上記で述べた案件 39 件について，案件別工程別工数記入シートに工数数値が記入されていた案件数を各工程（原因調査，修正案作成，修正実施，テスト，受入移行の 5 工程）で集計した．この結果，原因調査工程では 13 件，修正案作成工程では 6 件，修正実施工程では 13 件，テスト工程では 12 件，受入移行工程では 10 件であった．ここで，案件別工程別工数記入シートに工数数値が記入されている案件数が工程毎に異なる理由は 3 つ考えられる．1 つは，図 3-3 の案件Ａの場合には工程前半の工数数値が案件別工程別工数記入シートに反映されなかったためである．もう 1 つは，

工程が省略された（もしくは前後の工程のどちらかに片寄された）ために工数数値が案件別工程別工数記入シートに反映されなかったためである．最後は，案件別工程別工数記入シートへの工数数値の記入忘れである．

3.3.2 Phase 1 において，評価対象とした案件

Phase 1 において，案件別工程別工数記入シートから収集できた案件 83 件分のうち APTracer の定性的な評価（案件別工程別工数記入シートの「試用評価」項目）が記入されている案件を工程別に集計すると，原因調査工程では 23 件，修正案作成工程では 47 件，修正実施工程では 60 件，テスト工程では 56 件，受入移行工程では 18 件であった．

一方，事前調査の場合と同様に，案件別工程別工数記入シートから収集できた案件 83 件分のうち案件情報シートに各種情報が記載された案件は 52 件であった．前記案件 52 件のうち案件別工程別工数記入シートに工数数値が記入されていた案件数を各工程で集計すると，原因調査工程では 10 件，修正案作成工程では 25 件，修正実施工程では 43 件，テスト工程では 44 件，受入移行工程では 11 件であった．

4. 評価

4.1 APTracer の定性評価

案件別工程別工数記入シートに記入された「試用評価」項目を集計する．工程毎に収集した試用評価を，「利用された」（=試用評価の 1，2，3，4，8）と「利用されなかった」（=試用評価の 9）で振り分けることで，工程毎に APTracer の利用具合をみる．さらに，「利用された」に振り分けられた試用評価を，「貢献できた」（=試用評価の 1，2，3，4）と「貢献できず」（=試用評価の 8）に振り分けることで，APTracer の貢献具合をみる．

APTracer の定性的な評価が記入されている案件を工程別に集計すると，原因調査工程では 23 件，修正案作成工程では 47 件，修正実施工程では 60 件，テスト工程では 56 件，受入移行工程では 18 件であった．これらに対し評価軸（利用された／利用されず，及び利用された案件のうち貢献できた／貢献できず）で分類した結果を表 4-1 にまとめる．貢献できた／できずの行には，案件数のほか，ヒアリングで得られた代表的な（貢献できた／できなかった）理由を載せている．ここで，利用された案件数と利用されなかった案件数の和が対象案件と等しくならない工程があるのは，その工程が数日に渡り，ある日は APTracer を利用したが，別の日は利用しなかった場合があったためと考えられる．

表 4-1　APTracer の定性的評価

	原因分析	修正案作成	修正実施	テスト	受入移行
対象案件	23件	47件	60件	56件	18件
利用された	11件	10件	1件	13件	1件
貢献できた	7件 動的SQL文の可視化	5件 プログラム動作の確認	1件 修正対象のクラスの絞り込み/確認	0件 －	0件 －
貢献できず	4件 欲しい情報が取得できず	5件 変更仕様書作成に利用できず	0件 －	13件 テストエビデンス作成に利用できず	1件
利用されず	15件	39件	59件	45件	17件

表 4-1 の各工程の，APTracer が利用された／利用されなかった案件数を比較すると，システム保守全般で APTracer が利用される機会が少なかったことわかる．考えられる原因の 1 つは，開発機向けのテストデータの準備が大変であるために，開発機に導入された APTracer を利用する機会を逃してしまったことである．

次に，システム保守の前半（原因調査～修正実施）においては，APTracer の効果あったことがわかる．原因調査工程では，利用された 11 件の案件のうち 7 件で貢献できたと評価された．システム保守担当者に評価内容をヒアリングしたところ，従来ならばソースコードを追いながら担当者の頭の中で組み立てなければならない動的 SQL 文を可視化できる点を高く評価されたことがわかった．また，修正案作成工程では，利用された 10 件の案件のうち 5 件で貢献できたと評価された．ヒアリングによれば，ベテランのシステム保守担当者であっても業務プログラム内のロジックを再確認するために，APTracer で取得したメソッドの呼び出し関係を参照していたことが分かった．

4.2 APTracer の定量評価

APTracer を利用した／利用しなかった場合の，工程の作業工数の比較する．しかし，案件別工程別工数記入シートに記入された「工数」項目を，APTracer を利用した／利用しなかった場合で比較することはできない．なぜなら，案件別工程別工数記入シートに記入された工数は，案件の特性（規模，複雑さなど），担当者の技術力，業務システムに対する知見などの要因の影響を受けた数値である．したがって，上記の要因を取り除いて正規化した工数を対象にする必要がある．

そこで，案件別工程別工数記入シートに記入された「工数」を案件情報シートに記入された「修正対象規模」項目で除算した「単位工数」を導入する（下式）．ここで，作業工数は，修正対象規模に比例すると仮定している．

$$\text{単位工数} := \frac{\text{ある案件の工程に要した作業時間（時間）}}{\text{修正対象規模（1000 step）}}$$

システム X の保守にここ数年来携わっているシステム保守担当者が多いことから，担当者の技術力およびシステム X に対する知見は，システム保守担当者によって大差ないと仮定する．この仮定より，案件別工程別工数記入シートに

記入された作業時間は，担当者の技術力およびシステム X に対する知見に依存しない．さらに，案件の複雑さについては，バラツキはあるものの極端に簡単／複雑な案件はないものとする．このとき，単位工数を案件に対して平均した値（＝平均単位工数）は，案件の複雑さによるバラツキを平準化した作業時間とみなすことができる．従って，平均単位工数は，案件・担当者に依存する要因を取り除いた作業工数とみなすことができる．

Phase 0 に収集した案件の中で，平均単位工数の算出対象となった案件は，原因調査工程では 13 件，修正案作成工程では 6 件，修正実施工程では 13 件，テスト工程では 12 件，受入移行工程では 10 件である（表 3-1 の 2 行目を参照）．前記案件に対して，工程毎に平均単位工数を算出した結果を図 4-1 に載せる（中程度の濃さの棒グラフ）．修正案作成工程及びテスト工程で平均単位工数が高く，「ふたこぶラクダ」の性質が表れていることを確認できる．

Phase 1 においては，APTracer を利用した案件と利用しなかった案件を対象に，工程別に平均単位工数を算出する．平均単位工数算出対象となった案件のうち APTracer を利用した案件数は，原因調査工程では 4 件，修正案作成工程では 4 件，修正実施工程では 1 件，テスト工程では 12 件，受入移行工程では 0 件であった．一方，APTracer を利用しなかった案件数は，原因調査工程では 6 件，修正案作成工程では 21 件，修正実施工程では 36 件，テスト工程では 31 件，受入移行工程では 11 件であった．前記案件に対して，工程毎に平均単位工数を算出した結果を図 4-1 に載せる．Phase 1 において，APTracer が利用されなかった場合には，Phase 0 と同様に，原因調査工程及びテスト工程における平均単位工数が高くなっており，「ふたこぶラクダ」の性質が表れていることが確認できる．

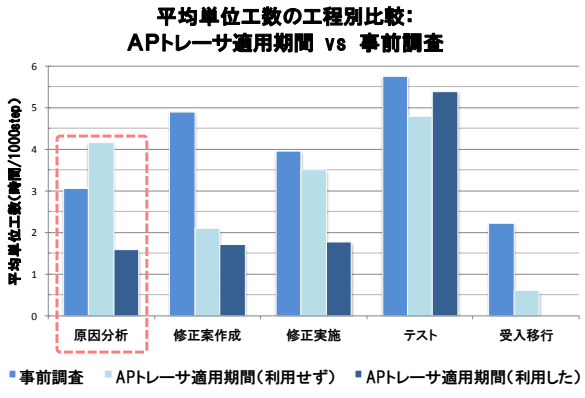

図 4-1　APTracer の定量的評価

各工程について，Phase 0，Phase 1（APTracer を利用した／利用せず）の 3 ケースにおける平均単位工数の比較を行う．原因調査工程について，Phase 0 では平均単位工数が 3.0（時間／kstep），APTracer を利用しなかった場合では 4.2（時間／kstep）であるにも関わらず，APTracer を利用した場合では 1.6（時間／kstep）に抑えられている．Phase0 の平均単位工数と比較すると APTracer の利用によって平均単位工数が 46.6%削減されたと分かる．定性評価（表 4-1）から分かるように原因調査工程は APTracer が貢献できた工程であったことからも，APTracer の効果はあったと言える．修正案作成工程について，Phase 0（4.9 時間／kstep）と APTracer を利用した場合（1.7 時間／kstep）を比較すると大幅な工数削減が見られるが，APTracer を利用しなかった場合（2.1 時間／kstep）と比べると工数の削減は見られない．従って，今回の APTracer の実証実験だけでは結論を導くことができない．修正実施工程について，Phase 0 では平均単位工数が 3.9（時間／kstep），APTracer を利用しなかった場合では 3.5（時間／kstep），利用した場合では 1.8（時間／kstep）となった．しかし，APTracer の利用した場合では平均単位工数の算出対象となった案件数が 1 件のみであったため，今回の APTracer の実証実験からは一概に APTracer による工数の軽減があったと言うことができない．テスト工程では，Phase 0 では平均単位工数が 5.7（時間／kstep），APTracer を利用しなかった場合では 4.8（時間／kstep），APTracer を利用した場合でも 5.4（時間／kstep）となり，APTracer の利用による工数の軽減が見られない．これは，テスト工程では APTracer は貢献できなかったという定性評価の裏づけにもなっている．受入移行工程では，APTracer の利用が無かった為評価することができなかった．

5. 関連研究・製品

プログラムの動的解析ツールとしてプロファイラがある．プロファイラは，プログラム実行時のボトルネック箇所やメモリリーク等の発見を支援する．製品には，JVMTI(Java Virtual Machine Tool Interface)[3]を利用した JProbe[4]，JProfiler[5]や Bytecode Instrumentation 機能を利用した CA Introscope[6]，ENdoSnipe[7]などがある．プロファイラは，プログラム処理のログを詳細に取得し集約するため，システム運用時のプログラム性能監視には適していると言える．しかし，一連のプログラム処理を抽出するためのキー情報が埋没する/無い為に，システム保守におけるプログラム動作把握や不具合の原因調査には不向きである．特に，Web アプリケーション・システムでは，クライアント PC－サーバ間通信(例えば HTTP 通信等)の単位でプログラムの動作を把握することが求められるが，既存のプロファイラ製品では上記通信とプログラム処理とを紐付けることは難しい．

提案手法では，Servlet コンテナにて生成したリクエスト ID(＝クライアント PC－サーバ間の HTTP 通信の識別子)を，スレッドのローカル領域に格納し持ち回ることでクライアント PC－サーバ間通信とプログラム処理を紐付ける．

6. まとめ

本論文では，実行時情報に基づいた Web アプリケーションの振る舞いを可視化する手法を提案した．具体的には，クライアント PC から送信されたリクエストをキーに実行時情報の各データ(リクエスト・メッセージ・データ，プログラム呼出関係データ，DB アクセスデータ)を紐付け，Web3 階層を跨いだ一連の処理(HTTP GET/POST データ通信，業務アプリケーション・プログラムの呼出し，DB データの参照／格納など)を可視化する．また，提案手法は，既存の Web アプリケーション・プログラムのソースコードへの影響を考慮し，ミドルウェア(AP 基盤など)で実行時情報の各データを取得する．提案手法のプロトタイプ(APTracer)を試作し，実システムの保守現場を対象に APTracer の実証評価を行った．その結果，Web アプリケーション・システムの不具合の原因調査工程に要する工数を軽減する効果(APTracer を利用しない場合に比べ, 平均単位工数を約 46%削減)を確認した．特に，Web アプリケーション・プログラムによって発行された動的 SQL 文を可視化できる点が有効であることが分かった．

今後の課題は，以下の 2 点である；

(1) APTracer の性能評価
APTracer によるオーバーヘッドなど, Web アプリケーション・システムに影響を評価検証する必要がある．

(2) APTracer で取得した実行時情報の表示機能の強化
APTracer で取得した実行時情報の各データをグラフィカルに描画する機能の追加である．

参考文献

1) JUAS; 第 17 回企業 IT 動向調査 2011; 2011.
2) 増井和也，弘中茂樹，馬場辰男，松永真；～ISO14764 による～ソフトウェア保守開発；ソフト・リサーチ・センター.
3) Oracle; JVMTI リファレンス;
http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/jvmti/jvmti.html .
4) GrapeCity, Inc.; JProbe 概要;
http://www.grapecity.com/tools/products/jprobe82 .
5) ej-technologies; JProfiler;
http://www.ej-technologies.com/products/jprofiler/overview.html .
6) CA Technologies; CA Introscope;
http://www.ca.com/jp/products/detail/ca-wily-introscope/benefits.aspx .
7) Acroquest Technology Co., Ltd.; ENdoSnipe;
http://www.endosnipe.com/ .