情報処理学会論文誌 Vol.56 No.6 1463–1470 (June 2015)

HTTPリクエストをキーとしたWeb3階層ログの紐付けによるWebアプリケーション・システムのトレース手法の提案

伊藤 秀朗[1,a)] 団野 博文[1] 三部 良太[1] 指野 篤司[2]

受付日 2014年9月3日, 採録日 2015年3月4日

概要：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，システム保守プロセスにおける問題分析および修正分析アクティビティに要する工数軽減を目的としたAPTracerを提案する．APTracerとは，Webアプリケーション・システムにおいて Web3 階層にまたがった一連の処理をトレースする手法であって，クライアントPCから送信されるリクエスト・メッセージをキーに Web3 階層の各層で出力されるログ情報の紐付けを特徴とする．実システムの保守現場を対象に評価実験を行った結果，問題分析および修正分析アクティビティ（特に前半の問題分析）の工数を約 6 割削減する効果があることを確認した．

キーワード：保守工数削減，不具合原因分析，Web 階層ログ

A Trace Analysis for Web Application Systems by Linking Execution Logs from Web 3-tiers via a HTTP Request

Hideaki Ito[1,a)] Hirofumi Danno[1] Ryota Mibe[1] Atsushi Sashino[2]

Received: September 3, 2014, Accepted: March 4, 2015

Abstract: Recently, software maintenance process becomes more important for information systems. This paper describes *APTracer* in order to reduce work-time of the problem and modification analysis activity in software maintenance process defined by ISO14764. *APTracer* is a technique to trace a series of processes over web 3-tiers architecture of web application systems. APTracer links execution log output in each tier to a request, for example HTTP request, sent by client PCs. As a result of evaluation, the proposed approach reduced about 60% work-time of the problem and modification analysis activity.

Keywords: reduction of software maintenance work-time, cause analysis for web application system failure, execution log from web3-tiers

1. はじめに

近年，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．情報システムの導入によって企業の業務効率化が図られているが，情報システムの不具合，企業内外環境の変化に対応するためのシステム保守費用が多額であることが問題にしばしばあげられる．企業の 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]．

システム保守のプロセスは，ISO14764 によれば，6 つのアクティビティで構成される（**表 1**）．ここで，問題分析および修正分析，修正の実施，保守レビューおよび受け入れの 3 アクティビティは保守案件 1 件単位に繰り返され，その保守案件のいくつかをまとめて移行アクティビティが実施される．

システム保守のプロセスのうち，問題分析および修正分析アクティビティを迅速化する技術が求められている．システム保守の各アクティビティ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 [2] と呼ばれ，問題分析お

[1] 株式会社日立製作所研究開発グループシステムイノベーションセンタ
Hitachi Ltd., Research & Development Group, Center for Technology Innovation, Yokohama, Kanagawa 244–0817, Japan
[2] 株式会社日立製作所情報・通信システム社 IT プラットフォーム事業本部
Hitachi Ltd., Information & Telecommunication Systems Company, IT Platform Division Group, Yokohama, Kanagawa 244–0817, Japan
[a)] hideaki.ito.cw@hitachi.com

表 1 ISO14764 によるシステム保守プロセスの定義
Table 1 Description of software maintenance processes in ISO 14764.

保守プロセスのアクティビティ	アクティビティの概要
プロセス実装	保守プロセスを実施する前に行う準備(保守計画及び保守手続きの策定，修正依頼手続き及び問題報告手続きの策定，構成管理の開始).
問題分析および修正分析	修正依頼または問題報告の分析，問題の再現または検証，修正実施に向けた修正案の用意などを行う．本アクティビティは修正依頼または問題報告をトリガに繰り返し実施.
修正の実施	保守対象の分析，ソフトウェアの修正およびテストを実施.
保守レビューおよび受け入れ	修正されたソフトウェアおよび修正テストの結果を元に修正されたシステムの完全性を確認.
移行	保守案件を纏めて本番適用するための作業(移行計画の策定，利用者への通知，利用者への教育，移行完了の通知，移行後影響の評価，保守案件の成果物を構成管理下に保管)を実施.
破棄	有用でなくなったソフトウェアを破棄.

よび修正分析アクティビティと修正の実施アクティビティ（特に後半のテスト）に要する工数の割合がほかに比べ多いことを指している．特に，問題分析および修正分析アクティビティはシステム保守プロセス特有のものであり，また長年の保守によりスパゲッティ化したソースコードにもかかわらずその現状把握に十分な時間をかけることが難しく結局その場しのぎの修正にならざるを得ない実態がある．結果的にシステム保守の悪循環に陥ってしまう．

企業情報システムの多くは Web3 階層（画面，プログラム，データベース）やクライアント PC からの同時アクセスなどの特性を持つ Web アプリケーション・システム（以下，WAS）であるが，この特性を考慮した WAS の振舞いを可視化する手法が求められている．不具合発生などの問題調査においては，（クライアント PC 上の）画面からどのような入力があり，それによってプログラムがどのように動作し，DB に対して SQL が発行されたのか，各層を紐付けてはじめて現状を把握したことになる．しかし，実際には各層の紐付けを考慮しないでログ設計された場合がほとんどである．Web サーバのアクセスログ，アプリケーション出力のログ，DB サーバのログなど各層のログは独立して出力されるために，各層のログ間を紐付けるキー情報もなく，クライアント PC からの同時アクセスが発生するにもかかわらず不確かな時間近接による紐付けをせざるを得ないのが現状である（図 1）．Web3 階層別に WAS 開発を分業化することによる開発生産性向上のメリットがある一方で，開発時に Web3 階層にまたがった処理をトレースするためのログ出力にまで考慮されることはめったにない．そのため，不具合発生時の問題調査における上記の現状には，情報システムの運用保守を担う企業 IT 部門の多くが直面している．

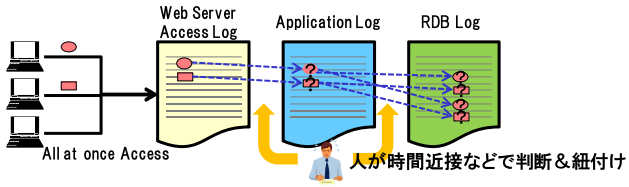

図 1 実行時の動作の把握手段の一例
Fig. 1 A common method to understand how the target application worked when a software failure occurred.

本論文では，システム保守における問題分析および修正分析アクティビティに要する工数軽減を目的に，APTracer を提案する．APTracer とは，各層のログの紐付けを考慮した設計でない WAS であっても，クライアント PC から送信されるリクエスト・メッセージをキーに Web3 階層の各層で出力するログ情報を紐付けることで，Web3 階層にまたがった各層の処理をトレースする手法である．APTracer を弊社アプリケーション実行基盤製品 uCosminexus [*1] Application Server にプロトタイプ実装し，評価実験を行った．その結果，問題分析および修正分析アクティビティ，特にその前半の問題分析における工数を約 6 割削減する効果を確認した．

本論文の構成を述べる．2 章では APTracer を詳述し，3 章では評価実験を述べて，4 章で評価実験を考察する．最後に今後の課題と展望を述べる．

2. APTracer

APTracer が対象とするシステムは，クライアント PC がインターネット/イントラネットを介してアプリケーションにアクセスする Web アプリケーション・システム（WAS）である．また，WAS は，それで扱うデータの永続化のために DB サーバに接続していてもよい．

APTracer の要件は，以下 3 点である；(1) Web3 階層ごとの処理ログを紐付けること，(2) 同時アクセス発生による処理ログの識別不可を防ぐこと，(3) アプリケーション・ソースコードを改変しないことである．(1)，(2) は WAS の不具合時の原因調査を複雑化している主要因への対応であり，(3) は既存アプリケーションに直接手を加えない手段を採用することで APTracer の導入障壁を軽減するためである．

2.1 APTracer のコンセプト

Web3 階層にまたがった各層の処理をトレースするには，Web3 階層の各処理を紐付けるキーが重要である．APTracer では，クライアント PC から送信されたリクエスト・メッセージをキー（= リクエスト ID）にする．そして，Web3 階層の各処理のログにリクエスト ID を付与して出力する（図 2 参照）．ここで，Web3 階層の各処理の

[*1] Cosminexus は，株式会社日立製作所の登録商標.

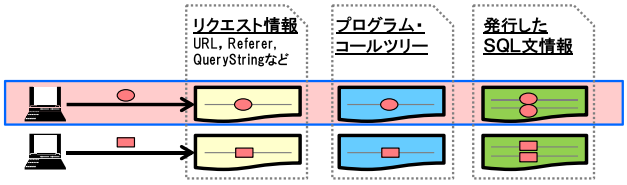

図 2 APTracer のコンセプト
Fig. 2 Overview of APTracer.

ログには，順に，リクエスト・メッセージ情報（HTTP の場合，リクエスト URL，Cookie，リクエスト元 IP アドレス，POST 送信の場合は POST データなど），プログラム・コール・ツリー情報（呼び出し元メソッド名と呼び出し先メソッドの対など），DB アクセス情報（発行された SQL 文など）である．

APTracer により Web3 階層の各層の処理のログにはリクエスト ID が付与されるため，それを紐付けることで Web3 階層を跨いでいたとしてもトレースすることが可能である．また，同時アクセスが発生したとしてもリクエスト ID で Web3 階層の処理のログを識別することが可能である．

また，上述した，Web3 階層の各層の処理のログにリクエスト ID を付与して出力する処理を，Web アプリケーション・サーバ（以下では，単に AP サーバと呼ぶ）で実現する．これによって，APTracer を利用するには，アプリケーションをその AP サーバに配置するだけで良いようにしている．

まず，AP サーバの一般的な役割を述べる．AP サーバは，それに到着したリクエスト・メッセージに応じてコンテキスト[*2]に処理を手渡す役割を担う Servlet コンテナ，JDBC API を実装した JDBC 実装コンポ，そして実行環境として Java[*3]仮想マシン（= JVM）で構成される．リクエスト・メッセージが AP サーバに到着すると，Servlet コンテナはそれに応じてコンテキストに処理を手渡し，コンテキスト内ではアプリケーション・プログラムに従って処理が進められる．ここで，アプリケーション・プログラムによっては JDBC API を利用した DB 問合せが行われるが，このとき AP サーバの JDBC 実装コンポがその処理を担う．**図 3** では JVM のメモリ上にロードされた Java クラスのインスタンス（図中の球形，上半分はコンテキスト内のアプリケーション・プログラムに相当）とその間でやりとりされるメッセージ（図中の矢印）が表されているが，特に実線の矢印に従って処理が進む．

次に，APTracer の実現方法を述べる．図 3 の実線矢印に従った一連の処理の中でいずれのタイミングでもアクセス可能な領域（以降では共通領域と呼ぶ）が重要である．そして，その共通領域へのリクエスト ID の格納/取出し処理を (1) Servlet コンテナ，(2) アプリケーション・プログラム，そして (3) JDBC 実装コンポに追加する．以下では，それぞれの追加処理を順に説明する．なお，それぞれの追加処理は，図 3 では点線矢印で表されている．

(1) Servlet コンテナの追加処理ではそれが受信したリクエスト・メッセージを一意に識別するリクエスト ID を生成する処理（図 3 の処理 2）と，それを共通領域に格納する処理（図 3 の処理 3）と，リクエスト ID とともにリクエスト・メッセージ情報（javax.servlet.http.HttpServletRequest インタフェースの getter で取得可能なものなど）を出力する処理（図 3 の処理 4）をこの順で実行する．

(2) アプリケーション・プログラムへの追加処理では，呼び出し元/先のメソッド名などを取得して共通領域のリクエスト ID とともにファイルなどに出力する処理（図 3 の処理 7，8）を追加する．Bytecode Instrumentation 機能[*4]を利用して上述の処理に相当するソースコードをバイトコードとして Java クラスのロード時に埋め込むことで，アプリケーション・ソースコードを改変することなく上述の処理を実現することができる．

(3) JDBC 実装コンポへの追加処理では，JDBC API の実装クラスに対して，共通領域のリクエスト ID とともに DB アクセス情報を出力する処理（図 3 の処理 11，12）を追加する．

APTracer によって，IT 部門の運用・保守担当者は，業務画面（= リクエスト URL）からリクエスト ID をキーにして関連する DB テーブルや発行された SQL 文をたどることができ，さらに SQL 文を発行したプログラムへと遡ることもできる．これに IT 部門の運用・保守担当者の WAS の不具合原因調査などへの生産性向上の効果を期待できる．

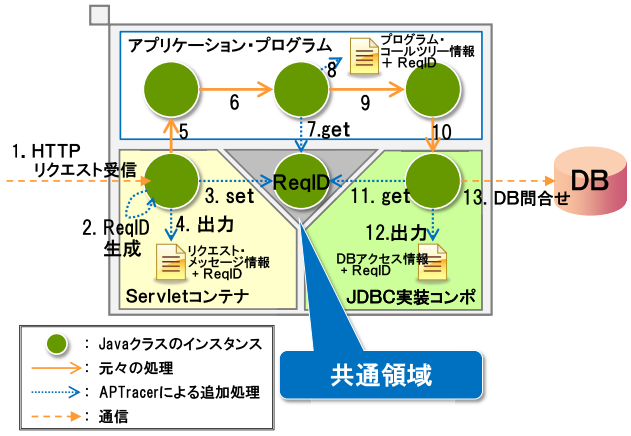

図 3 Web3 階層の処理のログの取得処理
Fig. 3 Acquisition process for execution logs from a web 3-tier application.

*2 AP サーバに配置されたアプリケーションの実態．
*3 Java は，Oracle Corporation およびその子会社，関連会社の米国およびその他の国における登録商標．
*4 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．

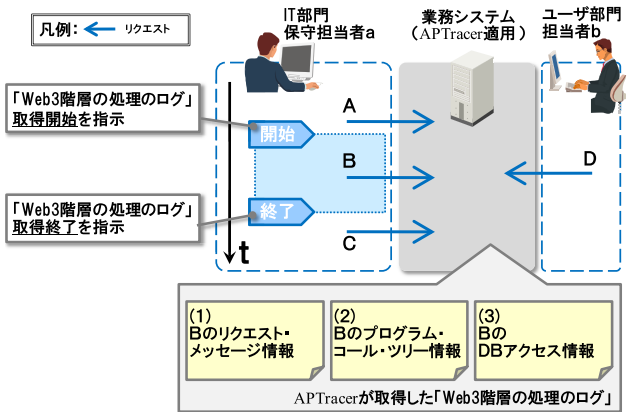

図 4 APTracer のユースケース例
Fig. 4 A use case of APTracer.

2.2 APTracer のユースケース

本論文では，WAS のユーザが指示した期間の，そのユーザのクライアント PC から送信されたリクエストに関係する Web3 階層の処理のログのみ取得するユースケースを検討する．最も単純なものに，WAS のすべてのリクエスト・メッセージに対して Web3 階層の処理のログを出力するものがあるが，APTracer が WAS の性能やリソースを圧迫する懸念があったため，ここでは検討を見送った．

本論文で取り上げるユースケースの例を示す（**図 4** 参照）．

IT 部門保守担当者 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，ユーザ部門担当者 b は同時期に業務システムにリクエスト D を送信した場合を考える．IT 部門保守担当者 a が Web3 階層の処理のログの取得を指示した期間（「テスト ID」で識別）に送信されたリクエスト B の Web3 階層の処理のログのみを取得する．このとき，ユーザ部門担当者 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 Web3 階層の処理のログは取得しない．

このユースケースでは，Web3 階層の処理のログの参照時にリクエスト ID ではなく，テスト ID を元にして検索することができる．また，取得すべき Web3 階層の処理のログの量を低減するなど，WAS への影響を必要最低限にすることができる．

また，テスト ID を元にして，APTracer で取得した Web3 階層の処理のログを参照する「テスト結果参照アプリケーション」もあわせて開発した（**図 5**）．

記録一覧画面（図 5 の上）に表示されたテスト ID の一覧より選択し，記録詳細画面（図 5 の下）にて APTracer で取得した Web3 階層の処理のログを，階層（記録詳細一覧表の列）ごとに，リクエスト単位（記録詳細一覧表の行）で参照する．図 5 で選択されたテスト ID では，業務システムに対してリクエスト・メッセージが 5 回送信され，3 回目のリクエストでは DB へのアクセスがあったことが分かる．

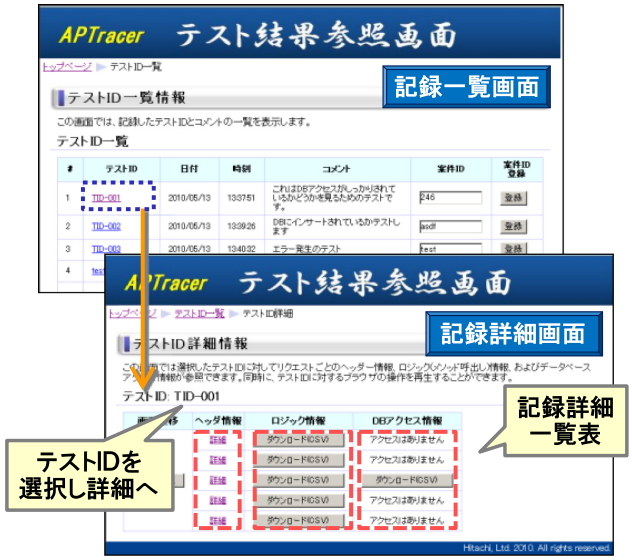

図 5 テスト結果参照アプリケーションの画面遷移
Fig. 5 Screenshot of our application for referring execution logs from a web 3-tier application.

2.3 APTracer の実装

弊社アプリケーション実行基盤製品 uCosminexus Application Server に，APTracer をプロトタイプ実装した．

共通領域は，Java 標準クラスライブラリとして提供されている java.lang.ThreadLocal クラスを用いて実現した．共通領域の実装方法は ThreadLocal を用いる方法に限らず，（図 3 の示した）一連の処理内で取得するリクエスト ID が同一となるよう制御すれば，共通領域としてファイルを採用することも可能であると考える．

JDBC 実装コンポでは次の JDBC API を対象に APTracer の処理を追加した；

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

なお，javax.sql.Statement#execute（sql: String）の引数は発行された SQL 文そのものであり，引数値も取得する実装した．

APTracer の実装対象として弊社アプリケーション実行基盤 uCosminexus Application Server を採用したが，それ特有の機能を利用しないで APTracer を実現しているため，他の AP サーバでも APTracer を実現することは可能と考える．

3. 評価実験

APTracer のシステム保守プロセスへの工数削減効果を検証するため，AP サーバ上で実稼働する Web アプリケー

表 2 案件別工程別工数記入シートの項目一覧
Table 2 Items for task record worksheet of a software maintenance programmer.

#	項目名	記入内容
1	案件番号	保守案件を識別する番号
2	案件種別	「保守」「機能追加」のどちらか
3	コメント	案件の簡単な説明，APTracer の試用感想等
4	作業時間	工程別に，案件に要した 1 日あたりの時間
5	試用評価	工程別に APTracer を以下で評価； 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．

ション・システム（以下，システム S）とそれを運用・保守する企業内 IT 部門組織を対象に APTracer の評価実験（約 1 カ月間）を行った．

3.1 実験対象

システム S には，本番機/検証機/開発機があるが，本実験では開発機に APTracer を導入した．開発機は，不具合の再現，プログラムの修正，機能追加の際に保守担当者によって利用される環境である．

3.2 実験方法

本実験を実施するにあたり「案件別工程別工数記入シート」と「案件別情報シート」を用意した．案件別工程別工数記入シートは，保守案件ごとに各工程の工数情報を取得するためのものであり，保守案件の各工程での作業時間を記入する（**表 2**）．また，案件別工程別工数記入シートには，APTracer の試用評価を記入する項目も設けている．次に，案件別情報シートは，保守案件の工数補正に使用する情報を取得するためのものであり，保守案件ごとに案件番号，案件の規模などの情報を記入する（**表 3**）．いずれのシートも Excel[*5] ファイルとして提供した（**図 6**，**図 7** 参照）．

3.3 実験結果

案件別工程別工数記入シートから保守案件番号ごとに各工程の作業時間（単位は時間）と APTracer 試用評価を集計した（83 件分，**図 8** はその一部抜粋）．ここで，工程の作業時間の記載がなかったもの（図 8 では “-（ハイフン）” で表している）はその工程の作業は発生しなかったと見なし，集計対象にしなかった．これは，作業が発生した場合の作業時間を工程ごとに集計するためである．また，各工程が複数日に亘りある工程の APTracer 試用評価 x は

[*5] Excel は，米国 Microsoft 社の商標または登録商標．

表 3 案件情報シートの項目一覧
Table 3 Items for software maintenance request information sheet.

#	項目名	記入内容
1	案件 ID	保守案件を識別する番号
2	案件種別	「保守」「機能追加」のどちらか
3	見積工数	案件の見積工数
4	実績工数	案件の実績工数
5	対象規模	案件で修正対象クラスの総ステップ数
6	修正規模	案件で修正を行った箇所のステップ数
7	テストケース数	実施したテストケース数
8	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

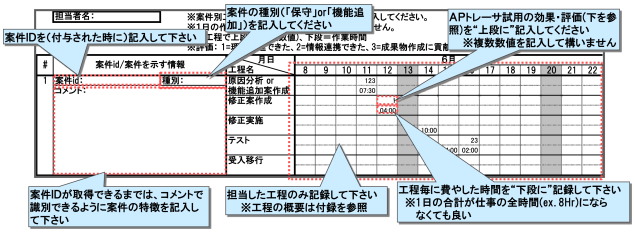

図 6 案件別工程別工数記入シートの例
Fig. 6 Sample of the task record worksheet.

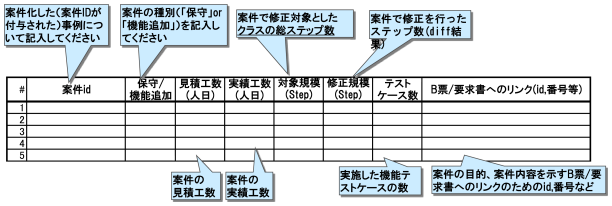

図 7 案件情報シートの例
Fig. 7 Sample of the software maintenance request information sheet.

ISO14764によるシステム保守プロセス定義		問題分析および修正分析アクティビティ		修正の実施アクティビティ		保守レビューおよび受入アクティビティ
案件番号		問題分析	修正分析	修正実施	テスト	受入移行
841	試用評価	1	-	9(3)	9(3)	-
	作業時間(h)	4.0	0.0	17.0	10.5	0.0
1033	試用評価	-	9	9	9	-
	作業時間(h)	0.0	6.0	15.5	9.0	0.0
1039	試用評価	8	8	-	-	9
	作業時間(h)	4.0	4.0	0.0	0.0	1.0
1076	試用評価	1(2)	1,9	9(3)	9(5)	-
	作業時間(h)	6.5	10.5	18.0	23.5	0.0
1098	試用評価	8	8	9	9(2)	-
	作業時間(h)	8.0	4.0	8.0	11.0	0.0
1129	試用評価	-	9	9	9	-
	作業時間(h)	0.0	3.0	2.0	1.0	0.0

図 8 案件別工程別工数記入シートの集計結果（一部）
Fig. 8 Excerpt of aggregation result of the task record worksheets filled in by software maintenance programmers in this experiment.

複数回（$= n$ 回，$n > 1$ の整数）出現する場合があったため，$x(n)$ と略記した．また，APTracer 試用評価の記入がなかったものは “-（ハイフン）” で表している．

また，案件情報シートから実験期間中に完了した保守案件番号ごとに対象規模（単位はステップ数）などのデータを得た（52 件分）．

4. 考察

4.1 定量評価

まず，APTracer を定量評価する．まず，案件別工程別工数記入シートに記入された「作業時間」を正規化するため，作業工数は修正対象規模に比例するとの仮定のもと，案件別工程別工数記入シートに記入された「作業時間」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，システム S の保守担当者の多くは数年来のベテランであり，案件別工程別工数記入シートに記入された作業時間は，システム S に対する知見，担当者の技術力にはよらないものとしている．

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

APTracer を利用した案件と利用しなかった案件に対して，単位工数を平均した値（= 平均単位工数）を工程別に比較した（図 9）．なお，平均単位工数算出対象となった案件数は，APTracer を利用した場合では，問題分析工程 4 件，修正分析工程 4 件，修正実施工程 1 件，テスト工程 12 件，受入移行工程 0 件であった．また，APTracer を利用しなかった場合では，問題分析工程 6 件，修正分析工程 20 件，修正実施工程 35 件，テスト工程 30 件，受入移行工程 11 件であった．

図 9 において着目すべきは，問題分析工程である．平均単位工数は，APTracer を利用しなかった場合では 4.2（時間/kstep）であるにもかかわらず，APTracer を利用した場合では 1.6（時間/kstep）に抑えられている（= 工数の約 6 割削減）．一方，それ以降の工程では，APTracer による工数削減効果を確認することはできなかった．

本実験では APTracer を利用した/利用しなかった案件数がともに少なく，APTracer の利用した場合と利用しなかった場合の平均単位工数の有意差の有無については，議論の余地がある点には注意されたい．また，ある工程での工数（時間）と修正対象規模（kstep）がおおむね比例する前提のもと前述の平均単位工数を採用したが，その妥当性についても議論の余地がある．以上に関しては，APTracer の定量的な評価に関する今後の課題としたい．

4.2 定性評価

案件別工程別工数記入シートから収集した APTracer の試用評価をもとに APTracer を定性評価した（表 4）．案件別工程別工数記入シートから収集できた案件 83 件分のうち試用評価が記入されている案件を工程別に集計すると，問題分析工程では 23 件，修正分析工程では 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．これらを対象に「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）で分類し，さらに，「利用された」に分類されたものを，「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）に分類した．

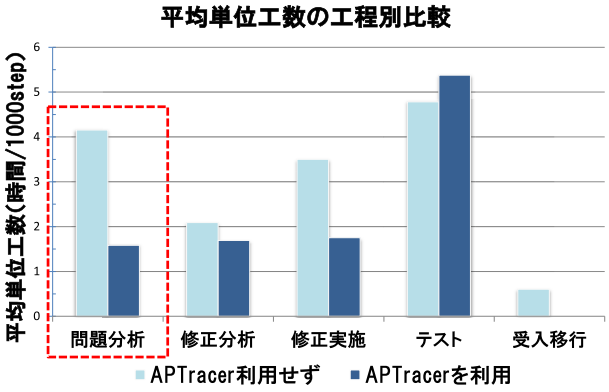

図 9 APTracer 利用時と非利用時の平均単位工数の比較
Fig. 9 Comparison of averaged man-hour of steps in software maintenance process between using APTracer and not.

表 4 APTracer の試用評価
Table 4 Summary of questionnaire: How often APTracer made a contribution to software maintenance programmers?.

	問題分析	修正分析	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 ー	0件 ー
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 ー	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

※ 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，その工程が複数日に亘ったために試用評価の種類が複数出現したため．

システム保守の前半（問題分析～修正実施）に着目すると，APTracer が効果的に利用されたことが分かる．これは，前節の定量評価結果を支持するものである．また，システム保守担当者へのヒアリングを通じて，下記が分かった；

- 問題分析工程では，特に動的 SQL 文をすぐに確認でき，ソースコードを遡ることで問題箇所を特定しやすかった点．従来ならばソースコードを順に追いながら担当者の頭の中で動的 SQL 文を組み立てなければならなかった．
- 修正分析工程では，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点．

5. 関連研究

Ruby on Rails [3] は，Web3 階層にまたがった処理をトレースすることを考慮した Web アプリケーションフレームワークの 1 つである．これを利用する開発者は Web3 階層

のログの紐付けを意識することなく，Web3 階層にまたがる処理をトレース可能なログの出力を実現できる．しかし，既存の企業情報システムの多くは Ruby on Rails のような Web アプリケーションフレームワークで開発されなかった Web アプリケーション・システムである．APTracer は，このような Web アプリケーション・システムを対象にして，後付けであっても Web3 階層にまたがる処理をトレースする手法を提供している点で有用と考える．

APTracer は，サーバサイドのプログラムを対象とした Web アプリケーション・システムのトレース手法であるが，近年は，クライアントサイドのプログラムとサーバサイドのプログラムの両者にまたがるトレース手法の研究がさかんである．たとえば，Matthijssen らは，Ajax アプリケーションにおける，クライアント PC 上の Javascript 呼び出しのログ情報とサーバ上の JSP や Java プログラムの呼び出しのログ情報を紐付ける手法の提案とそのツール FireDitective を開発した [4], [5]．クライアント PC 側とサーバ側のログ情報の取得にはそれぞれ Firefox[*6]のアドオンと Java VM Tool Interface（JVMTI）を利用しており，既存の Ajax アプリケーションにコードを新たに埋め込むことなくそれを実現している．

また，Ishio らは，Java プログラムの呼び出しのログ情報から UML[*7]のシーケンス図を自動で生成するツール Amida を開発した [6]．Java プログラムの呼び出しのログ情報の取得には JVMTI を利用している．Amida は Java プログラムによる処理のトレースに特化しそれを UML シーケンス図で可視化することで Java プログラムの振舞いの理解促進を図る一方で，Web アプリケーションのようにあるセッション内で実行される一連の Java プログラムを識別し紐付けるまでには及んでいない．

6. まとめ

本論文では，Web3 階層のログの紐付けを考慮した設計でない WAS である場合にも Web3 階層にまたがった各層の処理をトレースする手法 APTracer を提案した．その特徴は，クライアント PC から送信されるリクエスト・メッセージをキーに Web3 階層の各層で出力するログ情報（= リクエスト・メッセージ情報，プログラム・コール・ツリー情報，DB アクセス情報）を紐付けることと，アプリケーション・プログラムを改変しないで前述を実現することである．

弊社アプリケーション実行基盤製品 uCosminexus Application Server に APTracer をプロトタイプ実装し，実稼働する WAS（ただし，開発機上）とそれを運用・保守する企業内 IT 部門組織を対象に APTracer の評価実験を行った．その結果，問題分析における工数を約 6 割削減する効果を確認した．

今後は，以下のことに取り組む；

(1) APTracer の性能影響軽減
本報告では触れなかったが，評価実験向けにプロトタイプ実装した APTracer の，AP サーバへの影響は，無視できるほどのものではなく，エンドユーザの利便性を低下させていた可能性がある．APTracer の実用性向上にむけ，APTracer の性能負荷低減の仕掛けを検討していく．

(2) APTracer で取得した Web3 階層の処理のログの可視化方式の強化
評価実験の限られた時間の都合上，APTracer で取得した Web3 階層の処理のログの可視化手段として，それを単に参照する「テスト結果参照アプリケーション」しか開発することができなかった．Web3 階層の処理のログをグラフィカルに図示するなど，それの可視化方式を充実させていく．

[*6] Firefox は，米国 Mozilla Foundation の商標または登録商標．
[*7] UML は，Object Management Group, Inc. の米国およびその他の国における登録商標または商標．

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 (2007).
[3] Ruby on Rails, available from ⟨http://rubyonrails.org/⟩.
[4] Matthijssen, N., Zaidman, A., Storey, M., Bull, I. and Deursen, A.: Connecting Traces: Understanding Client-Server Interactions in Ajax Applications, *18th IEEE Inrternational Conference on Program Comprehension*, pp.216–225 (2010).
[5] Zaidman, A., Matthijssen, N., Storey, M. and Deursen, A.: Understanding Ajax applications by connecting client and server-side execution traces, Empirical Software Engineering, Vol.18, pp.181–218 (2013).
[6] Ishio, T., Watanabe, Y. and Inoue, K.: AMIDA: A Sequence Diagram Extraction Toolkit Supporting Automatic Phase Detection, *Proc. 30th International Conference on Software Engineering*, pp.969–970 (2008).

伊藤 秀朗

2006 年東京工業大学卒業．2008 年東京工業大学大学院理工学研究科物性物理学専攻修士課程修了．同年株式会社日立製作所入社．一貫してシステム生産性・品質向上技術の研究を行ってきた．現在，研究開発グループシステムイノベーションセンタにて，アプリケーションライフサイクルの研究開発に従事．

団野 博文 （正会員）

1970 年大阪府立成城工業高校卒業．同年株式会社日立製作所入社．これまで，ソフトウェア工学全般を研究対象とし，EA，SOA，MDA 等業務分析・設計技術からソフトウェア自動生成技術まで幅広く研究を行ってきた．現在，研究開発グループシステムイノベーションセンタシステム生産性研究部にて，後進の指導・育成に従事．

三部 良太 （正会員）

1990 年電気通信大学卒業．1992 年東京工業大学大学院情報理工学科計算工学専攻修正課程修了．同年株式会社日立製作所入社．一貫してシステム生産性・品質向上技術の研究を行ってきた．現在，研究開発グループシステムイノベーションセンタにて，モデルベース開発，形式手法，アプリケーションライフサイクル等の研究開発に従事．

指野 篤司

1994 年慶應義塾大学卒業．1996 年慶應義塾大学大学院理工学研究科計算機科学専攻修士課程修了．同年株式会社日立製作所入社．これまで，ミドルウェアの開発やマネージドサービスの運用を行ってきた．現在，情報・通信システム社 IT プラットフォーム事業本部サービスビジネス推進部にて，サービスビジネスの企画に従事．