Vol.2012-SE-176 No.11
Vol.2012-EMB-25 No.11
2012/5/22

ソフトウェア開発におけるトレーサビリティ確保のための開発環境の検討

渥美 紀寿[1,a)]　小林 隆志[1,b)]　高田 広章[1,c)]

概要：ソフトウェアの品質保証においてトレーサビリティを確保することが強く求められている．トレーサビリティを確保するためには様々な開発支援ツールを連携する必要がある．しかし，開発拠点がグローバル化している場合にそれぞれの拠点で同様の環境を構築する必要があるなどの原因により，ライフサイクル全体を通したトレーサビリティを確保するための環境を構築することが困難である．本稿ではソフトウェア開発ライフサイクル全体において統合的にトレーサビリティを管理するために，既存ツールとの連携でき，トレーサビリティ情報を容易に活用できる開発環境について議論する．さらに，これを実現することを目的としたオープン・トレーサビリティ・ツールプラットフォーム TERAS を紹介する．

キーワード：トレーサビリティ，開発環境，ツールプラットフォーム

A Study on a Development Environment for Software Traceability Management

NORITOSHI ATSUMI[1,a)]　TAKASHI KOBAYASHI[1,b)]　HIROAKI TAKADA[1,c)]

Abstract: Software traceability management is becoming an increasingly important facet of software quality assurance. However, it is difficult to maintain full lifecycle traceability because all developers must use the same complex environment which combined various development tools. In this paper, we discuss a development environment to manage the traceability across the full application lifecycle, which can collaborate with existing various development tools and utilize effortlessly to the traceability information. Moreover, we introduce Open Traceability Tool Platform TERAS to realize it.

Keywords: traceability, software development environment, tool platform

1. はじめに

ソフトウェアの品質保証のためにソフトウェア開発における各種情報間のトレーサビリティを確保することが強く求められている．例えば，どの要求項目がどのように実装されどのようなテストが行われたか，不具合に対してどのような修正が行われたかなどを正確に追跡し提示することが必要である．安全性の高いソフトウェア開発が求められる車載ソフトウェア開発では，自動車分野の機能安全規格である ISO26262 の中にも，これらのトレーサビリティを確保することが指示されている．また，各種トレーサビリティ確保できていることによって，類似の不具合が発生した場合の修正支援や，不具合の多いモジュールの確認，類似機能の分析など，ソフトウェア開発に有用な情報を分析することが可能になる．

トレーサビリティを確保するためには，ソフトウェア開発のライフサイクル全般に亘って各バージョンの個々の成果物とそれに関連する情報を関連付ける必要がある．ソフトウェア開発において様々な成果物や情報は作業管理，変更管理，構成管理，要求管理，テスト管理, リリース管理,

[1] 名古屋大学大学院 情報科学研究科
Graduate School of Information Science, Nagoya University
[a)] atsumi@nagoya-u.jp
[b)] tkobaya@is.nagoya-u.ac.jp
[c)] hiro@ertl.jp

課題管理，障害管理などにおいて管理される．これらの管理はこれまで開発工程ごとにそれぞれが別々に行っており，それぞれの管理下にあるリソースは他の管理下のリソースと関連が多く存在する．

いくつかの情報間の関係はすでに自動的に保存され活用できる環境が整っている．たとえば，設計モデルからソースコードを自動生成するツールを用いて設計モデルとソースコードの関連を保持したり，Trac や Redmine などの課題管理ツールを用いて課題とそれに対する作業結果との関連を保持したりすることが可能である．しかし，これらのツールではトレーサビリティの関連付けの対象が限定的であり，ソフトウェア開発のライフサイクル全体に渡るトレーサビリティの管理ができない．また，これらを各管理のためのツールは複数存在し，プロジェクトを構成する異なる開発チームにおいてそれぞれ別々のツールを利用していることも少なくない．

この問題を解決するために ALM (Application Lifecycle Management) の考え方が登場した．ALM とはアプリケーションソフトウェアを開発・運用するに当たり，そのライフサイクル全体を総合的に管理することでソフトウェアの品質や開発生産性，変化への対応力などを向上させようという考え方のことである [1,2].

この ALM を実現するツールプラットフォームとして，一般社団法人 TERAS [3] では開発拠点がグローバル化する組込みソフトウェア開発の全ライフサイクルを支援し，実装から設計中心のソフトウェア開発に移行し，全体システムとしての安全性・信頼性を確保するためのオープンツールプラットフォームの構築を目指している [4]．著者らは学術会員として TERAS に参加しており，TERAS 技術委員会において，TERAS ツールプラットフォームの仕様について議論している [5]．本稿ではソフトウェア開発のライフサイクル全体に渡るトレーサビリティを統合的に管理するための要件について議論し，オープンソースツールを活用したチケット駆動開発におけるトレーサビリティ管理について述べる．さらに，TERAS 技術委員会において検討しているソフトウェア開発環境についても紹介する．

2. 関連研究

トレーサビリティに関する研究は，開発成果物間のトレーサビリティを自動的に復元することを目的とした研究がほとんどであり，そのトレーサビリティを管理する方法やトレーサビリティを活用する方法についての研究があまりされていない．

ソフトウェア開発の各段階において，トレーサビリティを自動的に復元する手法について紹介する．文献 [6] では，MDA において作成される複数のモデル間およびソースコードとの間のトレーサビリティを汎用的に利用できるよう eclipse の EMF に基づいた支援ツールを提案している．

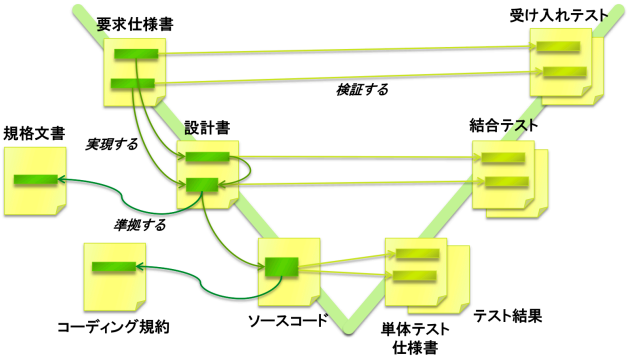

図 1 V 字モデルにおけるトレーサビリティ

文献 [7] では，組込みソフトウェアにおけるソースコードとテストケースとの間のトレーサビリティを自動的に導出し，それを可視化するツールについて紹介している．文献 [8] では，版管理システムと課題追跡システムからデータマイニング手法を用いてソースコードとバグとの間のトレーサビリティを自動的に導出する手法を提案している．文献 [9–11] では，ドキュメントを適当な単位に分割し，分割された単位中の単語とソースコード中のコメントや識別子名との類似度をコサイン類似度によって求めることによってドキュメントとソースコードとのトレーサビリティを自動的に導出する手法を提案している．文献 [12] では，B-splines カーブによって類似度を評価することによってドキュメントとソースコードとのトレーサビリティを自動的に導出する手法を提案している．

3. ソフトウェア開発におけるトレーサビリティ管理

3.1 トレーサビリティの定義

本稿で取り扱うトレーサビリティを定義する．

ソフトウェア開発において必要となるトレーサビリティは，単に要求とその実現箇所との関係だけではなく，多くの種類の関係が含まれる．図 1 は，V 字型プロセスにおいて，必要となるトレーサビリティを示したものである．トレーサビリティは，開発工程間の「実現する」関係 (要求-設計-実装)，開発成果物とテストなどの検証項目との間の「検証する」関係 (要件定義-運用テスト，設計-シナリオ・結合テスト，実装-単体テスト)，変更前後の対応関係などを追跡可能にし，その関係間における技術選択や対応が必要となる根拠を付随する情報として記録されるものである．

要求を網羅していること，要求が適切に実現されていることを容易に確認できるようにするために以下のことを可能にする必要がある．

- 要求を基にソースコードに至るまでの段階的な変化を追跡可能
- 品質を確認するために行った作業およびその結果を追跡可能

- 変更要求に対してその要因とそれに対する対応方法およびその結果を追跡可能
- 不具合に対してその原因と解決方法およびその結果を追跡可能

要求仕様や設計モデルなどの成果物には複数の項目が 1 つの文書として記述されており，上記を追跡可能にするためには細分化された個々の項目間の関連付けが必要となる．本稿では，これらの関連する細分化された項目間の関係をトレーサビリティと定義する．細分化された項目間の関係をトレーサビリティとして定義することにより，同一の成果物内における関連もトレーサビリティとして管理することが可能となる．

3.2 ALM (Application Lifecycle Management) プラットフォーム

ソフトウェア開発において様々な成果物や情報は作業管理，変更管理，構成管理，要求管理，テスト管理, リリース管理，課題管理，障害管理などにおいて管理される．これらの管理はこれまで開発工程ごとにそれぞれが別々に行っており，それぞれの管理下にあるリソースは他の管理下のリソースと関連が多く存在する．

ALM プラットフォームは，これらのツールの管理する情報関を関連付けることを可能とするものである．ALM プラットフォームを中心として，アプリケーションライフサイクル全体に渡った管理をすることによって，各管理システムを跨いで情報を参照することが可能となる．このような ALM プラットフォームを実現した例として，Microsoft 社の Visual Studio Team Foundation Server(TFS)[*1]や，IBM 社の Jazz プラットフォーム[*2]がある．Jazz プラットフォームでは，要素技術として Web を利用しており，各情報に対して Web ブラウザで閲覧することが可能である．開発ライフサイクルデータにアクセスするための仕様が Open Services for Lifecycle Collaboration (OSLC) [13] として標準化が進められている．

OSLC では，開発成果物の情報に関するデータモデルが定められており，すべて REST API を通じて Web ブラウザ等からアクセスすることが可能である．そのため OSLC に準拠したツール間においてトレーサビリティを確保することが容易となる．

3.3 トレーサビリティ管理のための既存ツールの問題点

既存の ALM プラットフォームを用いることによって，トレーサビリティを確保するための各要素間の関連付けは容易になる一方，これらのツールがサポートしていない過去の資産や，他のツールの情報は管理対象とできない．現在の開発スタイルは，オフショア開発や分散開発が進んでおり，発注先すべてにおいて同様の環境を構築しなければ，トレーサビリティを確保することができないという点には問題が残る．また，これらのツールで関連付けられた関連付け情報に対する API などが提供されていないことが多く，トレーサビリティを利用した拡張ツールを ALM ツールベンダ以外が開発することが困難である．

OSLC では変更要求を中心としたトレーサビリティ管理が可能であり，ソフトウェア開発中の作業は変更要求に基づいて行われ，成果物を登録する際に変更要求と関連付ける．OSLC では想定される利用シナリオにおいて関連が必要だと判断されたリソース間には関連付けることができるようにそれぞれの管理対象物を表現するリソースに対して仕様が定められている．そのため全てのリソースに対して関連付けられる訳ではなく，また新しい関係を定義しようとした場合，リソースのデータモデルを変更することとなり，多くの範囲に影響が出ることとなる．

本稿では，上記問題を解決するために，ソフトウェア開発で用いられる多種多様なツール間で利用される情報間の関係を，柔軟に定義・管理できる開発環境に関して議論する．あらゆる要素間に柔軟に関連付けを定義できるように，個々のリソース管理とは別にトレーサビリティ情報を格納するモデルを提案する．

4. オープンソースツールを利用したチケット駆動開発におけるトレーサビリティ管理

本節では第一著者が関わっているオープンソースコミュニティ codefirst [14] におけるチケット駆動開発環境について述べる．チケット駆動開発ではプロジェクトにおけるすべての作業をチケットという単位で管理を行い，そのチケットを消化していくことで開発を進める．このようにすることで作業漏れや進捗状況の確認を開発者間で共有することができる．

4.1 利用するオープンソースツールと連携方法

codefirst では (1) 変更履歴は残っているが変更意図がわからない，(2) 開発者間のコミュニケーション不足，(3) ビルドできない状態になってもなかなか気付かない，(4) ドキュメントの修正忘れなどの問題を起こさないようにするために Redmine，Git，AsakusaSatellite (codefirst で開発したコミュニケーションツール)，Jenkins，Sphinx を連携させて利用している．

(1) Redmine [*3]
　全ての要件，バグ，タスクをチケットとして管理

(2) Git [*4]
　commit メッセージに Redmine のチケット番号を付与

[*1] http://www.microsoft.com/japan/visualstudio/products/2010-editions/team-foundation-server/overview
[*2] http://www-06.ibm.com/software/jp/jazz/
[*3] http://redmine.jp/
[*4] http://git-scm.com/

することにより，Redmine のチケットと関連付けることが可能

(3) AsakusaSatellie [*5]
開発者向けのコミュニケーションツールで，ドラッグ & ドロップによるファイルのアップロード，URL で入力された画像の展開，ソースコードのシンタックスハイライト，検索が可能

(4) Jenkins [*6]
継続インテグレーションシステムであり，ソフトウェアのビルドやテスト，外部で起動するジョブの監視が可能．また，版管理システムと連携可能で，ビルドやテストを行ったプロジェクトの版と関連付けが可能

(5) Sphinx [*7]
テキストデータからドキュメントを生成するツールであり，HTML，ePub，PDF など様々な形式で出力可能である．テキストデータであるため，差分管理可能であり，ソースコードと同様に管理可能

単にこれらのツールを利用しただけでは十分なトレーサビリティを確保することができないため，これらのツール間の連携をするために以下のツールを利用している．

- Redmine と Git の連携
  Redime と連携するために Git-Hooks を利用しており，ブランチ名が id/番号 となっている場合，commit メッセージに自動的に refs #番号 を挿入され，Redmine との連携が可能であり，チケット番号挿入忘れなどによるトレーサビリティ欠如を防いでいる．
- Redmine とレビュー結果の連携
  Code Review Plugin を利用し，行単位でレビュー結果を記録可能である．レビュー結果の実態は Redmine のチケットであり，別途チケットを作成する必要がない．また，そのチケットはその行の更新者が修正担当者に設定される．
- プロジェクト進行状況の確認
  Redmine のチケットサマリプラグインを利用しており，チケットの消化状況を視覚的に把握することが可能である．
- Redmine と AsakusaSatellite の連携
  Redmine のチケット番号を入力することで自動的にチケットへのリンクに変換される，メッセージをチケットの内容に取り込むための仕組みが用意されており，Redmine との連携機能がある．
- Jenkins と Asakusasatellite の連携
  Git の中央リポジトリが更新されるたびに，ビルド，自動テスト，メトリクスの計測，テスト環境へのデプロイなどを実行し，その結果を AsakusaSatellite に通知する

[*5] http://www.codefirst.org/AsakusaSatellite/
[*6] http://jenkins-ci.org/
[*7] http://sphinx-users.jp/

4.2 ツール連携によるチケット駆動開発支援

これらの環境を利用して codefirst では以下の流れで開発を進めている．

(1) チケットの選択
Redmine のチケット一覧から，作業チケットを選択し，担当者を自分にする．

(2) チケットの対応
選択したチケットのチケット番号のブランチを作成し，開発者の環境で修正作業を繰り返す．この時，コミットメッセージには自動的にチケット番号が付与される．

```
# ブランチ作成
$ git checkout -b id/10
# コミット
$ git commit -a -m "foo"
$ git commit -a -m "bar"
# 中央リポジトリへのマージ
$ git checkout master
$ git merge id/10
```

(3) 中央リポジトリに登録
チケットの対応が完了したら，master ブランチにマージし，中央リポジトリに変更を登録する．

```
# master ブランチへのマージ
$ git checkout master
$ git merge id/10
# 中央リポジトリへの登録
$ git push
```

(4) Jenkins による自動ビルドとテストの実行
中央リポジトリに登録されると，自動的に Jenkins がビルドおよびテストを実行する．それぞれ完了すると Asakusasatellite にその結果が通知され，開発者にその結果が公開される．

(5) コードレビュー
機能が完成し，ビルドとテストが成功したら，他の開発者がそのコードをレビューし，問題があればその結果を登録し，問題なければチケットをクローズする．

codefirst ではこのようにオープンソースのツールを活用し，ツール間の連携をすることによってトレーサビリティを確保している．

4.3 本方式の問題点

本方式により，チケットと変更差分，チケットと議論内容，テストとソースコードの版の関連付けが可能であるが，

3.1 節で述べた (1) 要求を網羅していること，(2) 要求に対して適切に実装されたことを確認することが容易ではない．

その理由は以下の通りである．

(1) 要求事項に何があるかはチケットの種類をどのように設定するかに依存している．また，適切にチケットの種類を設定していても，変更セットとの対応しか取れないため，バイナリデータに対してどの項目が変更されたかが不明確である．

(2) 何をテストするかはテスト設計者に依存しており，テストが成功しても十分な確認ができているとは限らない．また，要求事項とテスト項目との対応関係を確保するためには，テストに関するチケットとそれに対応する要求事項を関連付ける必要があるが，これは自動化されていないため対応関係を取得できるとは限らない．

また，それぞれの関連付けに関する情報は Redmine や Jenkins などのツールが内部で保持しており，その情報を参照するためにはそれぞれのツールの API を利用しなければならない．ソフトウェア開発のライフサイクル全体に渡るトレーサビリティの情報を利用するためには，ツールごとにそれぞれの情報を参照するためのツールを開発する必要があり，実現するためのコストが高い．

これらを解決するために，成果物単位ではなく，成果物中の任意の範囲間における関連付けを可能にし，それらの情報を統一的な方法でアクセス可能にすることが必要となる．

5. トレーサビリティ確保のための開発環境の実現にむけて

5.1 開発環境に必要となる要件

管理対象の情報

トレーサビリティを確保し，効果的な ALM を実現するためには，ソフトウェア開発の全ライフサイクルにおけるあらゆる成果物や情報を対象とする必要がある．ソフトウェア開発の全ライフサイクルにおけるあらゆる文書には，要求仕様書，分析モデル，設計仕様書，設計モデル，ソースコード，テスト仕様書，テストコードなどの開発成果物以外に，開発に関わる会議議事録，レビュー結果，コーディングルールなどを含む．また，開発環境は，これらの全て情報を管理し，各種の機能がアクセスするための API を提供する必要がある．

トレーサビリティ情報

各成果物中の任意の要素間に対してトレーサビリティ情報を付与でき，それを活用できる必要がある．トレーサビリティ情報には 3.1 節で述べたように多くの種類があるため，トレーサビリティリンクの型を設定しどのようなリンクであるのかを保存する必要がある．また，版管理にある成果物については，版が更新されるたびにトレーサビリティ情報が不明とならないよう，該当箇所に変更がなかった場合は，古い版と同一の構成要素とみなす機能が必要である．

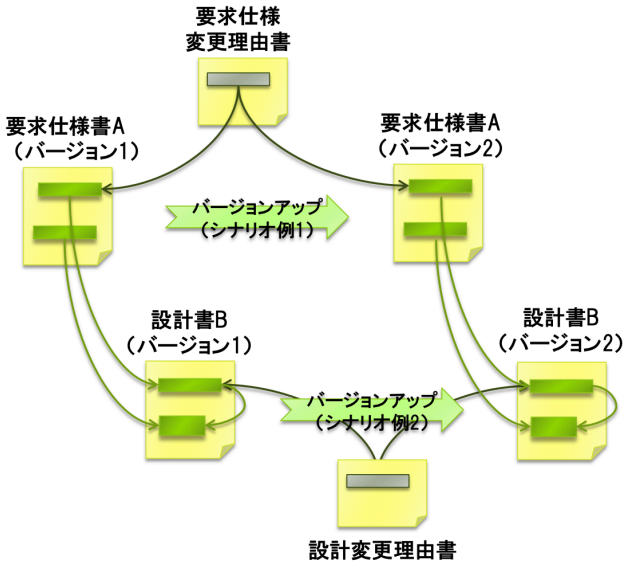

図 2 トレーサビリティ情報の例

管理方法

多くの既存ツールと連携できることが必要であり，そのためには各ツールが管理している成果物に関するデータモデルを変更することなく，トレーサビリティを管理できる必要がある．そのためにトレーサビリティに関する情報を独立して管理する必要がある．トレーサビリティの情報は，(1) どの成果物のどの範囲と，(2) どの成果物のどの範囲が，(3) どのようなトレーサビリティリンクで接続されているかを管理する．

5.2 トレーサビリティ情報の利用シナリオ

本節ではトレーサビリティ情報がどのように作成・利用されるか，シナリオを例に用いて説明する．図 2 に以下で説明するシナリオのイメージを示す．

まず，前提として，既存製品の要求仕様書 A(バージョン 1) と，それに対応する分析モデルや設計書 B(バージョン 1) があり，その間のトレーサビリティ (要求事項と実現方法の間のリンク) が管理されているものとする．また，バージョン 1 において要求仕様書 A 中の要件 (機能要件/非機能要件) は全て設計書 B とのトレーサビリティが確保されているとする．

シナリオとして以下を考える．

(STEP1) 新製品開発のために，要求仕様変更理由書を作成し，要求仕様項目を修正・追加を検討する．

(STEP2) 修正する要求仕様項目に関係する要求仕様項目を探し，その影響を検討する．

(STEP3) 要求仕様書 A をバージョン 2 に改訂する．

(STEP4) 修正された要求仕様項目に関連のある設計項目

を探す．設計の変更方針を検討し，設計変更理由書を作成する．

(STEP5) 改訂された要求仕様書 A(バージョン 2) に対応して，設計書 B をバージョン 2 に改訂する．

(STEP6) 要求仕様書 A 中のすべての要求が設計書 B でカバーされていることを確認する．

(STEP7) 設計書 B 中のすべての記述要素に対して，その根拠が要求仕様書 A に記述されていることを確認する．

バージョン 1 の要求仕様書 A と設計書 B のトレーサビリティが管理されているため，(STEP2) や (STEP4) において，修正を検討している仕様項目に関連する仕様項目や設計項目のリストをトレーサービリティリポジトリから取得することができる．上記シナリオでは仕様と設計の関係のみであるが，例えば実装箇所のソースコードやテストケース等とのトレーサビリティを利用することで，要求仕様の変更に対する波及解析が可能となる．

(STEP3) や (STEP5) でバージョン 2 に改定する際に，バージョン 1 と同じ項目に関しては，ツールプラットフォーム側が同一の項目であることを検知しトレーサビリティ情報を継承する．このように実現することで，人手でのトレーサビリティ情報管理を最小限にすることができる．修正が加えられた項目に関してはそれぞれ新しい項目として管理し，既存のトレーサビリティ情報を用いるか，新しく作成するかを開発者が判断する．

(STEP1) や (STEP4) のように，変更仕様書を作成して変更が行われる場合は，変更仕様と各バージョンとの間にもトレーサビリティ情報を作成する．このような情報を保持することで，変更の理由づけに容易に参照することが可能となる．ある要求仕様項目が変化したバージョンを見つけその変更仕様書を確認することで，仕様項目の変更理由を容易に検索する事ができる．

(STEP6) や (STEP7) では，設計書 B は，要求仕様書 A での仕様項目を過不足なく実現しているかの網羅性解析を行っている．トレーサビリティ情報を活用することで要件に対する実現の抜け漏れを防ぐことが可能となる．上記シナリオでは，(STEP2) で要件が追加され対応する設計がなされなかった場合，(STEP6) において，その要件を発見することができる．また逆向きの網羅性を確認することで，設計上必要となる仕様の漏れを検出することも可能である．

5.3 Open Traceability Platform TERAS

本節では，一般社団法人 TERAS [3] が目指している開発環境の概要について紹介する．TERAS に関する詳細については TERAS の Web ページで公開されている資料[*8]を参照されたい．

TERAS では，開発拠点がグローバル化する組込みソフトウェア開発の全ライフサイクルを支援し，実装から設計中心のソフトウェア開発に移行し，全体システムとしての安全性・信頼性を確保するためのオープンツールプラットフォームの構築を目指している．

図 3 は TERAS のシステム構成である．TERAS では以下のコンセプトに基づいた開発環境の構築を進めている [15].

(1) 開発現場で既に浸透している変更管理や構成管理については，そのまま使えるようにする

(2) 要件管理ツールの導入には，既存資産を活用できるようにする

(3) WEB を利用したツールチェーンだけではグローバルサプライチェーン構築は困難なため，トレーサビリティを HUB，オープンインターフェースをスポークとしたアーキテクチャとして構築する

TERAS ツールプラットフォームでは，各種成果物に関する情報を保存することができ，トレーサビリティに関する情報を管理するための基本機能を提供する．各種成果物とそれらの間のトレーサビリティ情報に REST インターフェースを用いてアクセスするプラグインを作成することによって，従来複数のツールで管理していたような ALM を実現するツールを容易に構築することが可能となる．

また，多くのツールとの連携をすることが可能であり．Open Service for Lifecycle Collaboration [13], Open Data Protocol [16]，Google Data API [17] などのオープン仕様のアダプタを提供することによって，これらに準拠したツールとの連携を可能としている．さらに，既存資産を利用できるよう Trac，Subversion など広く一般的に利用されているツールに対するアダプタを提供する予定である．

6. おわりに

本稿では本稿ではソフトウェア開発ライフサイクル全 体において統合的にトレーサビリティを管理するための方法について議論した．

最初にチケット駆動開発においてトレーサビリティを確保するために，オープンソースの各種ツールを用いて実践している事例について，codefirst での取り組みについて紹介した．この事例では成果物間のトレーサビリティをある程度確保することが可能であるが，成果物内の要素に対してトレーサビリティを確保することができない．また，ライフサイクル全般に渡るトレーサビリティ情報を容易に取得できないなどの問題がある．

これらのことを踏まえ，要件の網羅性や実装の妥当性を確認するためのトレーサビリティの確保において，開発環境で必要となる要件について検討し，トレーサビリティ情報の活用方法についてシナリオを用いて説明した．また，オープン・トレーサビリティ・プラットフォーム TERAS について紹介した．

TERAS が提案する開発環境が実現すると，様々な情報

[*8] TERAS 講演資料 (http://www.teras.or.jp/?p=303)

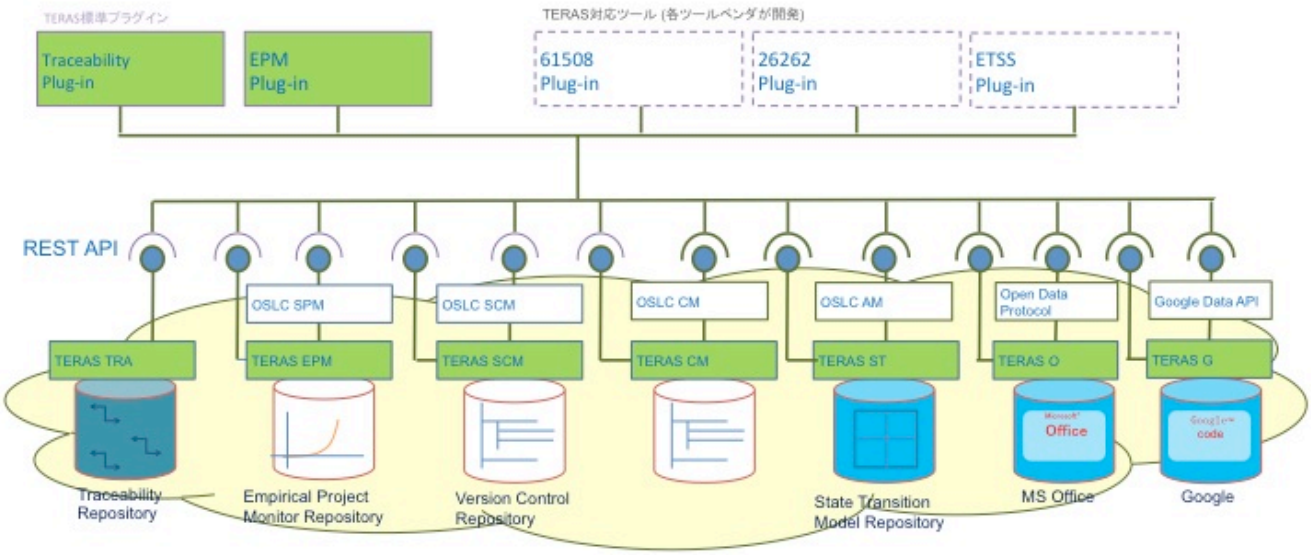

図 3 TERAS のシステム構成 (イメージ図) (第 1 回 TERAS 成果報告会 高田広章発表資料 [5]) より引用

間でトレーサビリティリンクを張ることが可能となり，トレーサビリティを利用した開発支援に関する研究に貢献できる．また，現在研究が進められているトレーサビリティリンクの自動生成手法を利用することによって，手動でトレーサビリティリンクを生成する手間を軽減することが可能になる．

今後はより多くのシナリオを検討し，各種オープンソースツールとの連携を検討した上でトレーサビリティ情報の確報方法や利用インタフェースについての詳細を定める必要がある．

参考文献

[1] Lacheiner, H. and Ramler, R.: Application Lifecycle Management as Infrastructure for Software Process Improvement and Evolution: Experience and Insights from Industry, *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 286 –293 (2011).

[2] Kim, J. A., Choi, S. Y. and Hwang, S. M.: Process & Evidence Enable to Automate ALM (Application Lifecycle Management), *Proceedings of the Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops (ISPAW)*, pp. 348 –351 (2011).

[3] TERAS: http://www.teras.or.jp/.

[4] 穴田啓樹，渡辺政彦，高田広章：一般社団法人 TERAS の紹介 (前編) –安心・安全・快適な社会のためにドキュメントのトレーサビリティを目指す-，*SEC journal*, Vol. 7, No. 4, pp. 183–187 (2012).

[5] 高田広章：今年度技術検討成果の総括，第 1 回 TERAS 成果報告会 発表資料 (2012). http://www.teras.or.jp/?p=303.

[6] Boronat, A., Carsí, J. and Ramos, I.: Automatic Support for Traceability in a Generic Model Management Framework, *Model Driven Architecture - Foundations and Applications*, pp. 316–330 (2005).

[7] Wiederseiner, C., Garousi, V. and Smith, M.: Tool Support for Automated Traceability of Test/Code Artifacts in Embedded Software Systems, *Proceedings of the IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 1109 –1117 (2011).

[8] Corley, C. S., Kraft, N. A., Etzkorn, L. H. and Lukins, S. K.: Recovering traceability links between source code and fixed bugs via patch analysis, *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 31–37 (2011).

[9] Chen, X., Hosking, J. and Grundy, J.: A combination approach for enhancing automated traceability, *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pp. 912 –915 (2011).

[10] Wang, X., Lai, G. and Liu, C.: Recovering Relationships between Documentation and Source Code based on the Characteristics of Software Engineering, *Electronic Notes in Theoretical Computer Science*, Vol. 243, pp. 121–137 (2009).

[11] 宮下　学，海谷治彦，海尻賢二：ソフトウェアプロジェクトにおける traceability の確立 -実プロジェクトへの適応-，電子情報通信学会 技術研究報告，Vol. 111, No. 481, pp. 127–132 (2012).

[12] Capobianco, G., De Lucia, A., Oliveto, R., Panichella, A. and Panichella, S.: Traceability Recovery Using Numerical Analysis, *Proceedings of the 16th Working Conference on Reverse Engineering*, pp. 195 –204 (2009).

[13] OSLC: Open Service for Lifecycle Collaboration, http://open-services.net/.

[14] codefirst: http://codefirst.org/ (2012).

[15] 渡辺正彦：ツールアーキテクチャ構想，第 1 回 TERAS 成果報告会 発表資料 (2012). http://www.teras.or.jp/?p=303.

[16] Microsoft: Open Data Protocol, http://www.odata.org/.

[17] Google: Google Data API, http://code.google.com/apis/gdata/.