Vol.2012-SE-175 No.17
2012/3/16

プログラムパッケージ名を用いたソフトウェア保守作業におけるプロジェクト理解支援システム

山田洋明[†]　　櫨山淳雄[†]

ビジネスのグローバルな競争下において、ソフトウェア開発は効率的に行うことが求められている。その中でもソフトウェア保守はソフトウェア開発全体の約 8 割もコストがかかるといわれており、ソフトウェア保守を効率よく進めることにより、ソフトウェア全体を効率よく進められる。そのため本研究ではソフトウェア保守を効率よく進めるためのプロジェクト理解支援システムの開発を行った。このシステムではプログラムパッケージ名に着目をした。プログラムパッケージ名では機能名を書くようにされているため、これを用いることにより仕様書にかかれている機能との関連を見つけ出した。そして設計書と議論の字句解析を行うことによりプログラムと設計書、議論の三つの関連性を見つけ出し、その設計書や議論を開発者に提示することにより、プロジェクト理解を狙った。

A Support System for Helping to Understand a Project in Software Maintenance Using Name of Program Package

HIROAKI YAMADA[†]　ATSUO HAZEYAMA[†]

Under the global competition of business, software development is required to be conducted more efficiently. In particular, software maintenance holds the majority in the costs of software development. Developers must understand a project to conduct software maintenance efficiently. Therefore this study develops a support system for understanding a project to software maintenance. This study proposes to use the name of program package. By using the name, the system extracts associations with parts of requirements specification. Furthermore, based on the cosine similarity using the vector space model between design documents or communication messages and the parts extracted by the name of program package related artifacts are recommended by the system.

1. はじめに

近年、情報技術の進化やユーザからの多様な要望に伴い、ソフトウェア開発は大規模化、複雑化してきている。それに反して開発期間の短縮や高い品質のソフトウェアが求められており[5]、ソフトウェア開発を効率的に進めることが求められている。その中でもソフトウェア保守のコストは、ソフトウェア開発全体の 80%～90%にもおよび、アメリカでは年間 700 億ドルかかるとも言われている[1]。このことからもわかるとおり、ソフトウェア保守を効率的に行うことができれば、ソフトウェア開発全体を効率的に進めることができると考えられる。

ソフトウェア保守では、担当者として任命された開発者は、過去にそのシステム開発に携わっていた人に問い合わせることによって情報を得ることができれば、効率よく保守作業を進めることができるだろう。しかし、過去のシステム開発者が忙しく連絡をとることが困難なこともある。あるいは、そのシステム開発者がいなくなり、連絡すら取れなくなる可能性もあり得る。

ソフトウェア開発では様々な設計書が作成される。その設計書は将来に価値のある情報源となり[2]、効率化、信頼性の向上につながる。これはソフトウェアの新規開発だけでなく、保守の場面においても同じことが言える。しかしソフトウェア開発では有用な情報が多く蓄積されていたとしても、自分が欲しい情報がどこにあるのかがわからないといった問題もある。それに加えそのような情報は単体で存在するのではなく複数の情報が関わって存在しているため、関連する情報を適切に見つけだすことはとても困難である[9]。

このような情報過多の問題を解決するために近年では、ソフトウェア工学においても推薦システムが開発されてきた。今までの推薦システムでは、関連すると推測されるソースコードのみを推薦するもの[9,10]、ソースコードのコミットコメントを解析し関連する設計書を推薦するもの[3]があった。コミットコメントの場合、コミットコメント自体が書かれない場合がある。

そこで本研究では、プログラムのパッケージ名に着目する。パッケージ名には機能名をつけることが標準で決まっており[6]、そのため設計書に書かれた機能名とパッケージ名との関連性を見つけ出すことができると考えられる。これにより通常の開発で書かれたプログラムに付加情報をつけることなくシステムがプログラム以外の設計書と関連づけることが可能である。そして、その関連付けられた設計書や議論を開発者に提示することによりプロジェクト理解の促進を狙う。

[†] 東京学芸大学大学院
Graduate School of Education, Tokyo Gakugei University

Vol.2012-SE-175 No.17
2012/3/16

2. 関連研究

本節では、本研究に関連する関連研究を述べ、その後問題点を提示することにより、本研究で解決すべき問題点を挙げる。

2.1 Ye らの研究

Ye らの研究として CodeBroker というシステムを構築している[10]。CodeBroker は開発者の情報過多の問題や時々刻々と出現する新情報や新機能を理解、適用するために開発された。このシステムは 4 つのエージェントから構成されており、このエージェントがプログラムの doc コメントを解析、類似検索を行い、類似したソースコードに関わる情報の提示を行っている。

このシステムでは類似する Code 情報のみをユーザに提示し、他の成果物は提示しないため、その Code が何を意味しているのか、どのような背景のもと作成されたかという情報まではわからない。そのためその Code を理解するのは難しいと考えられる。

2.2 Cubranic らの研究

Cubranic らの研究として Hipikat というシステムを構築している[3]。Hipikat では OSS 開発においてプロジェクトに入ったばかりの人が早くそのプロジェクトを理解できるように開発された。Hipikat サーバではクライアントから送られてきたユーザ情報と成果物の識別子を 4 つの識別モジュールで解析を行い、解析されたコンポーネントとともに確信度などをユーザに提示する。

このシステムでは Code をコミットする際に、コミットコメントとして手動で識別子を入力しなければならない。このコミットコメントは書くことを忘れられて書かれないことがある。そのためこの識別子も書かれない状況が発生してしまう。そのため成果物同士の関連性が見つけられないといった問題が発生するのではないかと考えられる。

2.3 関連研究のまとめ

Ye らの研究では、Code 情報のみを提示するものであった。そのため提示された Code 情報から Code の使い方などを理解しなければならなかった。そのため本研究では、Code に関連する設計書や、その設計書に関連すると推測される議論を開発者に提示することにより、この問題を解決する。

Cubranic らの研究では、メーリングリストなど様々な情報を提示することによってプロジェクトの理解を狙っていたが、情報を推薦するためにコミットコメントに対して手動で識別子を入力しなければならなかった。そのため本研究では、手動で関連情報を入力することなく、関連情報を推薦できる手法を提案する。

3. アプローチ

本節では、まず本システムが用いる情報の蓄積環境について述べ、その後本研究で用いるプログラムパッケージ名について述べ、最後に提案システムについて述べる。

3.1 情報蓄積環境

本研究で扱う設計書や議論内容の蓄積システムとして、著者らの研究室で提案されたシステム“ShinGiTai”[4]を用いる。このシステムは Web アプリケーションとして開発されており、成果物管理機能や BBS 機能、障害管理機能などのソフトウェア開発を行う際の機能が備えられている。このシステムから設計書と議論内容の収集を行う。

3.2 プログラムパッケージ名

本研究では、効率良くソフトウェア保守を行うために、プログラムパッケージ名に着目し、プログラムパッケージ名から設計書や BBS 等の議論内容を取得することを目指す。本論文ではプログラムパッケージ構造を、ソースコードの構文のことではなく、Java パッケージのことを呼ぶ(図 1)。Java パッケージの名前の付け方は基本的にはドメインを逆にしてつけることが一般的となっている[6]が、今回使う Java パッケージの粒度としてはもっと細かいものを想定している。この場合の Java パッケージは機能をまとめる機能名を Java パッケージ名に付ける (例.図 1 の場合では丸で囲ったパッケージ名”bug”が障害管理機能である)。

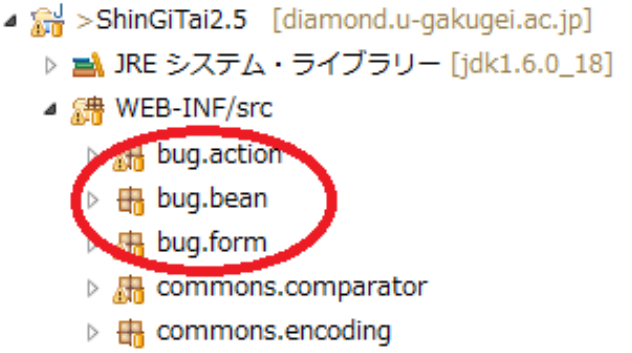

図 1 パッケージ名

3.3 プログラムパッケージ名と機能名との関連性の事前調査

前項で述べたプログラムパッケージ名を用いることによって、パッケージ名と機能との関連付けをすることができるのかの調査結果を示す。

(1)事前調査 1 (パッケージ名から機能名の抽出)

この調査は、あるプロジェクト(行数約 15KLOC で開発された Web アプリケーション)の仕様書と CVS(Concurrent Versions System)に格納されているソースコードを用いた。本調査手順を以下に示す。

I. CVS のソースコードからパッケージ名の抽出(1 番トップのパッケージ名のみの抽出)。
II. Ⅰで抽出したパッケージ名の翻訳を行い日本語に変換(GoogleAPI を用いて翻訳)。
III. 仕様書から機能名の抽出
IV. Ⅱで翻訳を行った翻訳後の単語と機能名に書かれている単語との一致の調査

表 1　事前調査 1 の結果

パッケージ名	翻訳後	機能名	判定
admin	管理者	該当機能なし	×
artifact	アーチファクト	成果物管理	×
assembler	アセンブラ	該当機能なし	×
bug	バグ	障害管理	×
commons	コモンズ	該当機能なし	×
communication	のコミュニケーション	コミュニケーション	○
community	コミュニティ	該当機能なし	×
inspection	検査	インスペクション	×
knowhow	ノウハウ	ノウハウ	○
login	ログイン	該当機能なし	×
minutes	議事録	議事録	○
peer	ピア	ピアレビュー	○
personal	個人	該当機能なし	×
plan	計画	計画立案	○
progress	進捗状況	進捗報告	○
teacher	教師	管理者	×
user	ユーザが	グループ・ユーザ	○

調査結果を表 1 に示す。表中の「パッケージ名」の欄は CVS から抽出したパッケージの名前を示しており、「翻訳後」の欄は GoogleAPI を用いてパッケージ名を英語から日本語に翻訳した結果であり、「機能名」の欄は仕様書から抽出した機能名を示しており(「該当機能なし」はパッケージ名に対応する機能が、仕様書には書かれていないことを意味する)、「判定」の欄は翻訳後の単語と仕様書の機能名とが一致しているかを示している。この判定は単語の大文字小文字を区別せずに、そして完全一致または部分一致しているものが”○“でありそれ以外の場合が×とした。この結果を見ると 17 個のパッケージの中で 7 個のパッケージが翻訳することで、仕様書の機能と関連性を見出すことができた。この結果をパーセンテージで表すと 41%であった。

(2)事前調査 2 (機能名からパッケージ名の抽出)

(1)ではパッケージ名から仕様書の機能名を抽出できるかの調査を行った。ここでは(1)とは逆に機能名からパッケージ名の抽出がどれくらいできるのかの調査を行う。本調査手順を以下に示す。

I. 仕様書から機能名の抽出
II. Ⅰで抽出した機能名の翻訳を行い英語に変換(GoogleAPI を用いて翻訳)。
III. CVS のソースコードからパッケージ名の抽出((1)と同様に 1 番トップのパッケージ名のみの抽出)。
IV. Ⅱで翻訳を行った翻訳後の単語と機能名に書かれている単語との一致の調査

表 2　事前調査 2 の結果

機能名	翻訳後	対応するパッケージ名	判定
グループ・ユーザ	User Groups	user	○
管理者	Administrator	teacher	×
計画立案	Planning	plan	○
進捗報告	Progress Report	progress	○
議事録	Proceedings	minutes	×
成果物管理	Deliverables Management	artifact	×
ピアレビュー	PEER REVIEW	peer	○
インスペクション	Inspection	inspection	○
コミュニケーション	Communication	communication	○
ノウハウ	Know-how	knowhow	○
障害管理	FaultManagement	bug	×

調査結果を表 2 に示す。表中の欄は(1)と同様に、機能名は仕様書から抽出した機能名を示しており、翻訳後の欄では GoogleAPI を用いて機能名を日本語から英語に翻訳した結果であり、パッケージ名は CVS から抽出したパッケージ名の名前を、最後の判定は翻訳後の単語と仕様書の機能名とが一致、不一致しているかを示しているもので

ある。この結果を見ると 11 個の機能の中で 7 個の機能が翻訳することで、パッケージ名との関連性を見出すことができたことがわかった。この結果をパーセンテージで表すと 64%という結果であった。

(3)事前調査結果

表 3 は(1)の結果でパッケージ名から機能名の抽出で判定が“○”であったもの、または(2)の結果で機能名からパッケージ名の抽出で判定が“○”であったものを判定として“○”として作成した。この結果を見ると 17 個の機能の中で 8 つの機能がパッケージ名との関連性を見出すことができたことがわかった。この結果をパーセンテージで表すと 47%である。文献[11]では CVS の 1000 トランザクション中で、あるファイルやクラスの修正に対して、それとともに修正されるべきファイルやクラスの推薦に関する精度が平均で 26%であったと報告している。本研究と対象は異なるが、全体の精度は本研究の方が高いため、パッケージ名と機能名との関連性は高いのではないかと推測できる。この結果より、本研究でパッケージ名を使うこととする。

表 3　事前調査結果

機能名	パッケージ名	判定
グループ・ユーザ	user	○
管理者	teacher	×
計画立案	plan	○
進捗報告	progress	○
議事録	minutes	○
成果物管理	artifact	×
ピアレビュー	peer	○
インスペクション	inspection	○
コミュニケーション	communication	○
ノウハウ	knowhow	○
障害管理	bug	×
該当機能なし	admin	×
該当機能なし	assembler	×
該当機能なし	commons	×
該当機能なし	community	×
該当機能なし	login	×
該当機能なし	personal	×

3.4　システムアーキテクチャ

本システムと開発支援システム ShinGiTai を含んだ、システムアーキテクチャを図 2 に示す。ShinGiTai では成果物管理機能や BBS 機能など、システム開発活動で用いられる機能がある。この機能により登録された情報が DB に保存される。そしてそれと同時に開発者は統合開発環境を用いてプログラム開発を行い、その作成したプログラムを CVS(Concurrent Versions System)に登録する。なお CVS のプロジェクト名と ShinGiTai で登録しているプロジェクト名を同名にしておくことで、CVS に登録されているプログラムが ShinGiTai 中のどのプロジェクトのものなのかを判断する。そしてデータベースに登録された情報と、CVS に登録された情報を用いて本システムがマッチングを行い、ユーザに推薦内容を提示する。

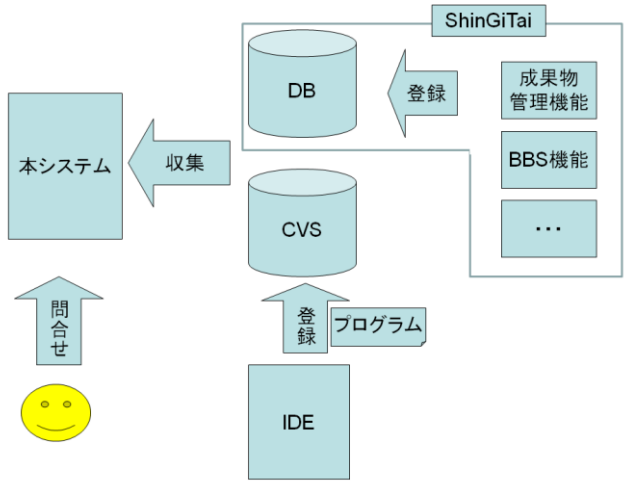

図 2　システムアーキテクチャ

3.5　システムの機能

3.5.1 システムの入出力

本システムの入出力の流れを図 3 に示す。まずユーザはシステムに Login を行う。その後自分が所属するプロジェクトの一覧が表示されるのでそこから探し出したいプロジェクトの選択を行う。次にモードの選択でソースコードから設計書の内容や議論内容を検索したいのか、また設計書から議論内容を検索したいのかの選択を行う。選択した検索モードに基づき内部処理を行い、最後にその結果を一覧として表示する(図

Vol.2012-SE-175 No.17
2012/3/16

4)。一覧表示では、画面上部にパッケージ名と関連があるとされた機能を示し、画面下部に類似していると算出された順に成果物が表示される。成果物には種類、名前、概要、類似度を表示する。種類では BBS や議事録などその成果物の種類を示しており、名前は BBS などのスレッド名、概要はその成果物の中身の一部、類似度は内部処理の結果、パッケージ名とどのくらい類似しているのかを示している。そして名前を選択することにより、機能の詳細と関連する議論の詳細を閲覧することができる。

内部処理の各機能については次に述べる。

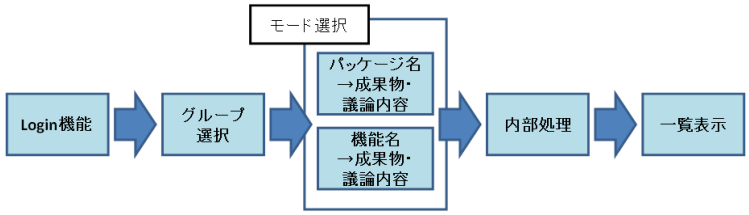

図 3　入出力の流れ


List

関連のあった機能:9. コミュニケーション

検索結果

	種類	名前	概要	類似度
1	BBS	先生へ, ビアレビューの仕様について	今, ビアレビューの仕様についてまとめてい	0.18640065
2	BBS	障害16について	To: Type:	0.15169603
3	BBS	Warファイルあげました.	Warファイルあげました. 発見し	0.12365067
4	BBS	ビアレビュークラス図コメント	To: Type:	0.10844641
5	BBS	パフォーマンスについて重要なお知らせ	To: Type:	0.101563856
6	BBS	先生へ, ビアレビューの仕様について	先生, ビアレビューの仕様について質問です	0.10029589
7	BBS	BBSへの関連付けの成果物	To小林君へ Type:	0.09757332
8	BBS	サーバにWARファイルをデプロイしました	To: Type: 変更点は	0.09134419
9	BBS	心技体プロジェクト第36回	To:皆さんへ	0.083172515

図 4　一覧画面

3.5.2 内部処理(図 5)

ここではシステムの内部処理について記述する。内部処理の大まかな流れは以下のとおりである。

(1)候補機能名抽出
(2)成果物単語抽出
(3)類似検索

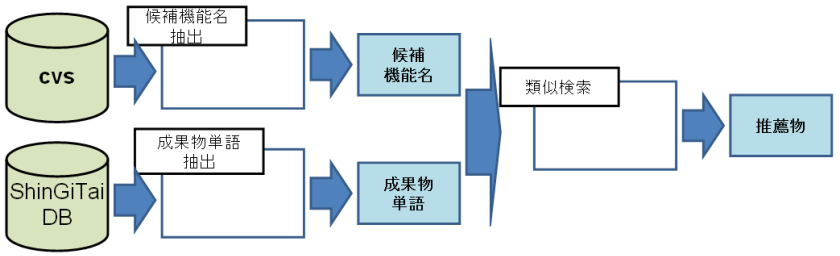

図 5　内部処理

(1)の“候補機能名抽出”では、CVS 中のソースコードからパッケージ名の抽出を行い、そのパッケージ名と仕様書に書かれている機能名と関連があるものを「候補機能名」として抽出する。(2)の“成果物単語抽出”では、ShinGiTaiDB に登録されている成果物と、「候補機能名」とのマッチングを行うために成果物を単語として分け「成果物単語」として抽出を行う。そして最後の(3)の“類似検索”では、「候補機能名」と「成果物単語」との類似検索を行い、関連のあると推測される成果物の抽出を行う。

詳しい内容に入る前にここで用いる用語について説明する。

・候補単語・・パッケージ名およびパッケージ名を翻訳したもの。
・機能名・・・仕様書に記述されている機能名。
・機能名単語・機能名を単語に分けたもの。
・候補機能名・機能名の中でパッケージ名と類似していると判断されたもの。
・説明文・・・仕様書に記述されている機能名を説明している文章。
・説明文単語・説明文を単語に分けたもの。
・成果物単語・設計書の内容や議論内容などを単語に分けたもの。

(1)候補機能名の抽出(図 6)

ここでは開発者によって選択されたパッケージ名から、そのパッケージ名に関連すると推測される「候補機能名」の抽出を行う。

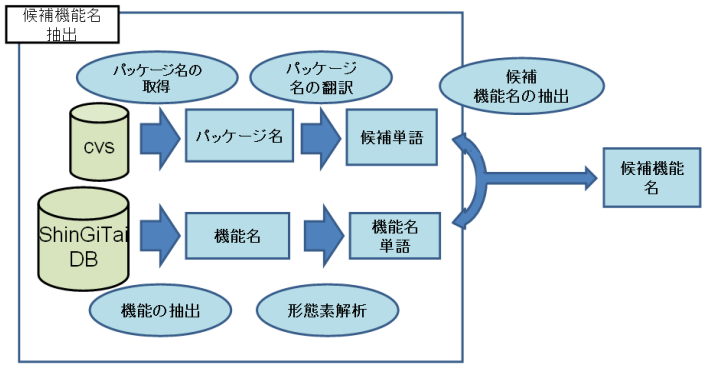

図 6　機能名の抽出

・パッケージ名の取得

本システムは ShinGiTai のデータを用いて Login を行う。そして ShinGiTai に登録されているプロジェクトと CVS に登録されているプロジェクトが同名であるため、システムに Login 後パッケージ名の抽出を行う。これによりパッケージ名の抽出を行う。

・パッケージ名の翻訳

本システムが検索の対象とする設計書は日本語で書かれている。一方開発言語のパッケージ名で用いられる言語は英語であるために、まずパッケージ名の翻訳作業を行い日本語に変換を行う。そしてこの翻訳された単語と翻訳する前の単語を「候補単語」として保持する。

・機能名の抽出

候補単語と仕様書に書かれている「機能名」とのマッチングを行うために、仕様書から「機能名」の抽出を行う。抽出方法はまず章の抽出を行う(図 7 赤枠)。この抽出では行の先頭が数字であるものを章区切りとみなして、その後ろにある文字列を「機能名」として扱う。そしてこの抽出した機能名と次の機能名との間の文字列をその機能の「説明文」(図 7 水色枠)として、「機能名」と一緒に保存する。

・機能名単語の抽出

“パッケージ名の翻訳”によって抽出された「候補単語」と仕様書から抽出した「機能名」とのマッチングを行うために「機能名」を Sen[8]を用いて形態素解析にかける。「機能名」は図 7 のように「コミュニケーション」と単語のみであることもあるが、「障害登録機能」などのように複数の単語から構成されていることもある。そのため「候補単語」と「機能名」とのマッチングを行うために機能名を形態素解析にかけ、「機能名」を単語に分ける。この分けられた単語を「機能名単語」として保持する。

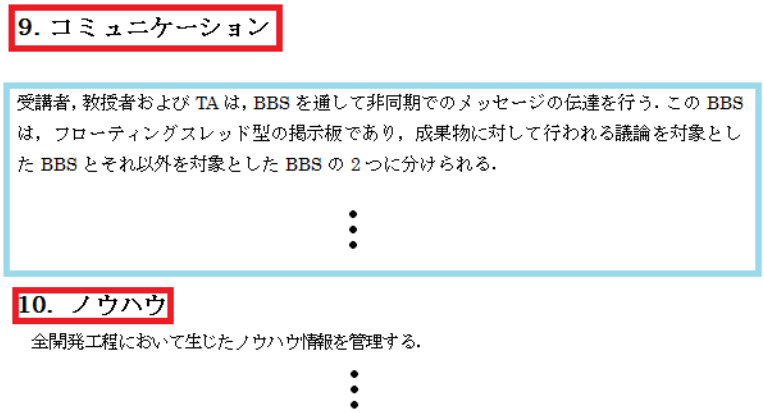

図 7　仕様書例

・候補機能名の抽出

“パッケージ名の翻訳”結果の「候補単語」と仕様書から抽出した「機能名単語」とのマッチングを行う。マッチングを行う前に「機能名単語」の翻訳を行い日本語から英語の「機能名単語」の抽出を行う。この理由としては 3.3 項で述べたとおり、どちらか一方の翻訳をしてマッチングするよりも、マッチング対象の両方を翻訳しマッチングした方が精度が高いからである。

「機能名単語」と「候補単語」とのマッチングを行う。ここでのマッチングは「候補単語」と「機能名単語」とが一致するものを探すだけの簡単なものである。これにより抽出された機能名を「候補機能名」として保持する。

(2)成果物単語の抽出(図 8)

ここでは先ほど述べた「候補機能名」に関連すると推測される設計書の内容や議論内容等の抽出を行うために、プロジェクトに所属する設計書の内容や議論内容等を単語レベルに分割する。

・成果物の取得

本システムの“プロジェクト選択機能”によって選択されたプロジェクトを基に ShinGiTai の DB から設計書の内容や議論内容等の情報を取得する。

・成果物単語の抽出

“候補機能名の抽出”の処理と同様に形態素解析を行い、設計書の内容や議論内容から「成果物単語」の抽出を行う。設計書の内容や議論内容は自然言語で書かれているため、「説明文」と、設計書の内容や議論内容との類似度を測る際に形態素解析を行う。この形態素解析を行った結果に基づき単語の抽出を行い、設計書の内容や議論内容を「成果物単語」として保持する。

なおここでの議論内容は１つの投稿単位ではなく、１つのスレッドを単位として扱う。この理由は、スレッドを１つの単位として扱うことにより、開発者による内容の把握が容易になると考えたからである。

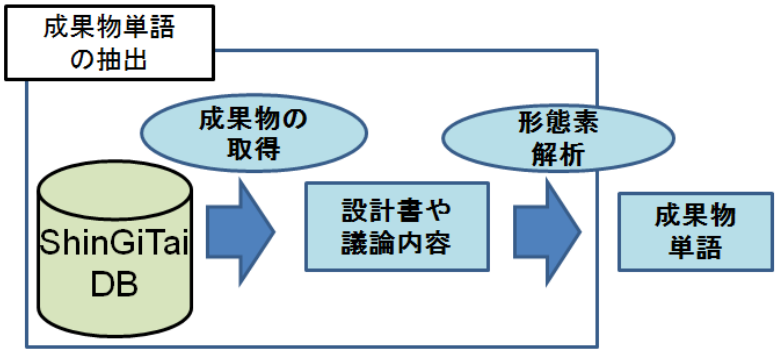

図 8　成果物単語の抽出

(3)「候補機能名」と「成果物単語」との類似検索(図 9)

ここでは、(1)の処理で抽出された「候補機能名」を構成する形態素である「機能名単語」とその「説明文」、そして(2)の処理で抽出された「成果物単語」を用いて「候補機能名」に関連すると推測される、設計書の内容や議論内容等を見つけ出す。

・説明文の形態素解析

類似検索を行うため、まず「説明文」を形態素解析にかけ、単語の抽出を行う。この抽出した単語を「説明文単語」として保持する。

・機能と成果物との類似検索

「成果物単語」と「機能名単語」、「説明文単語」を用い類似検索を行う。検索手法にはコサイン相関を用いる。コサイン相関ではベクトルのなす角の余弦値で表すため、各文書のベクトル化を行う。このベクトル化には文書の重みを用い設計書の内容や議論内容を文書ベクトルとして表現する。ベクトルの要素には文書に用いられる単語の重みを用いる。その重みの算出には TF-IDF(Term Frequency-Inverse Document Frequency)[7]を用いる。これを用い、「成果物単語」と「機能名単語」、「説明文単語」との類似度の算出を行う。

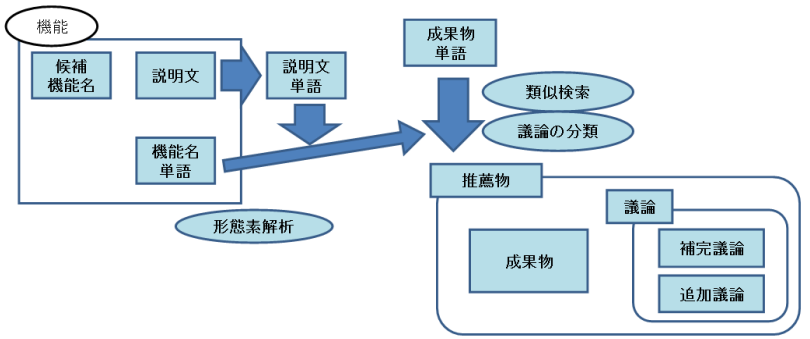

図 9　類似検索

・議論の分類

抽出された議論内容を補完議論と追加議論の 2 種類の議論として分類する。設計書の作成時刻前の議論を補完議論とし、作成後の議論を追加議論とする。この分類により設計書に書かれている内容と、書かれていない内容に区別することができると推測できる。これにより開発効率が上がると推測できる。なおこの議論の分類は“機能と成果物との類似検索”と同時に行う。

4.　既存研究との比較

本節では既存研究と本研究との比較を行い優位性を述べる。比較を表 4 に示す。

表 4　既存研究との比較

ツール	推薦物	コーパス
本研究	議論内容等	パッケージ名
CodeBroker	ソースコード	コメント
Hipikat	メーリングリスト等	識別子

まず CodeBroker は開発の効率を上げるために、開発者に対して現在開発中のソースコードに関連すると推測されるソースコードやソースコード片の提示を行っていた。

しかし、ソースコードのみの提示ではそのソースコードの使い方がわからないといった問題があった。次に Hipikat ではソースコードだけではなく、それに関連するメーリングリストなどを提示することによりこの問題を解決していた。しかしこの研究では、推薦物を推薦する際にソースコードのコミットコメントに意図的に識別子を入力しなければならないといった問題があった。

既存研究の問題点を挙げると以下のようになる

・ソースコードのみの推薦で理解がしづらい

・推薦のために付加情報の入力が必要である

このような問題を解決するために本研究では以下のことを行った。

「ソースコードのみの推薦で理解がしづらい」の問題を解決するために、本研究ではソースコードではなく、そのソースコードに関連する成果物の推薦を行った。これによりそのソースコードの理解を助けることができる。

「推薦のために付加情報の入力が必要である」の問題を解決するために、本研究ではパッケージ名を用いた。パッケージ名はコミットコメントの意図的な入力とは異なり、一般的な開発を行っていれば作成される。

5. おわりに

本研究では、プログラムパッケージ名を用いることにより、ソフトウェア保守に必要となる設計書の内容や議論内容を推薦するシステムの提案を行った。今後の予定としては適用実験を行い本システムの有効性を確認する予定である。そして本研究の今後の課題を以下に述べる。

・単語のゆがみ

類似検索を行う際に、現在では形態素解析を行いコサイン相関を用いて類似度の算出を行っている。しかしなら仕様書などの成果物が作成される際には、似た意味であるが違う言葉が使用されることがある。形態素解析では単語の抽出を行うがその単語の意味は考慮しないためこのような場合では別の単語として抽出を行ってしまう。そのためオントロジーなどを用いて似た意味の単語同士を同じ単語として扱わなければならないといった問題がある。

・類似検索手法の妥当性

類似検索を行う際に、現在は TF-IDF 法を用いている。設計書の内容と議論内容などの成果物との類似検索を行う際の最善の方法であるという検証は行っていない。文書間の類似性を検索する際にパターンマッチングを行うなどさまざまな類似検索方法がある。その中から本研究の設計書と議論内容などの成果物と一番精度が高い類似検索手法を選別しなければならない。

・機能抽出方法

仕様書から機能の抽出を行う際に、現在は章番号の数字を用いて抽出を行っている。しかしすべての仕様書がこのフォーマットであるとは限らない。そのためどんなフォーマットでも文書の解析ができるようにしなければならない。

参考文献

1) A. B. AL-Badareen, M. H. Selamat, M. A. Jabar, J. Din and S. Turaev : The Impact of Software Quality on Maintenance Process, International journal of computers, Vo. l5, pp. 183-190, 2011.

2) A. Ankolekar, K. Sycara, J. Herbsleb, R. Kraut and C. Welty : Supporting Online Problem-Solving Communities with the Semantic Web, Proceedings of the 15th International Conference on World Wide Web, pp. 575-584, 2006.

3) D. Cubranic and G. C. Murphy : Hipikat: Recommending Pertinent Software Development Artifacts, Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), pp. 408-419, 2003.

4) 三浦真人, 小林祐介, 島田和幸, 高橋晃一, 清木進, 櫨山淳雄 : 個人とコミュニティの支援を有するソフトウェア開発グループ演習環境の提案," 情報処理学会研究報告グループウェアとネットワークサービス, pp. 19-24, 2007.

5) 森崎修司：ソフトウェアインスペクションの動向, 情報処理, Vol. 50, No. 5, pp. 377-384, 2009.

6) ORACLE,URL(http://www.oracle.com/technetwork/java/codeconventions-135099.html), 最終閲覧日:2011.07.21.

7) G. Salton and M. J. McGill：Introduction to Modern Information Retrieval, McGraw-Hill, 1983.

8) Sen: http://www.mlab.im.dendai.ac.jp/~yamada/ir/MorphologicalAnalyzer/Sen.html

9) 島田隆次, 市井誠, 早瀬康裕, 松下誠, 井上克郎：開発中のソースコードに基づくソフトウェア部品の自動推薦システム A-SCORE, 情報処理学会論文誌, Vol. 50, No. 12, pp. 3095-3107, 2009.

10) Y. Ye and G. Fischer : Information Delivery in Support of Learning Reusable Software Components on Demand, Proceedings of 2002 International Conference on Intelligent User Interfaces (IUI'02), pp. 159-166, 2002.

11) T. Zimmermann, A. Zeller, P. Weibgerber and S. Diehl : Mining Version Histories to Guide Software Changes, IEEE Transactions on Software Engineering, Vol. 31, No. 6, pp. 429-445, 2005.