グラフ DB を用いた プロジェクト分析の試行と評価

横田 剛典[†1]　有本 泰仁[†1]　宮本 貴之[†1]　青山 幹雄[†2]
古賀 篤[†3]　解田 康起[†3]　渡辺 政彦[†1]

概要： ソフトウェアシステムの大規模化，複雑化等により，開発プロジェクトの状況を正確に理解することが困難になっている．そのため，プロジェクトの可視化の重要性は増々高まっている．本稿では，グラフ DB によるソフトウェア開発プロジェクト管理データの分析方法とその試行経験について報告する．グラフモデルをデータ構造とするグラフ DB は，分析対象のデータ間の関係を辿る構造分析，および分析結果の可視化において従来の RDB（リレーショナルデータベース)に対し優位性を持つ．本取り組みでは，実開発プロジェクトで蓄積された管理データを基に，グラフ DB を用いたプロジェクト分析を試行した．本試行を通し得られたプロジェクト分析におけるグラフ DB 適用の有用性の評価を示す．

キーワード：プロジェクト分析，プロジェクト可視化，グラフ DB，セマンティックグラフモデル

Towards Project Analysis by Using Graph Database

MASANORI YOKOTA[†1]　YASUHITO ARIMOTO[†1]　TAKAYUKI MIYAMOTO[†1]
MIKIO AOYAMA[†2]　ATSUSHI KOGA[†3]　YASUKI TOKITA[†3]
MASAHIKO WATANABE[†1]

1. はじめに

ソフトウェアシステムの大規模化，複雑化等により，開発プロジェクトの状況を正確に理解することが困難になっている．そのため，管理データを収集し，それらのデータを基にプロジェクトを定量的に可視化することで状況を理解し，改善するという実証的アプローチを実践しながら，プロジェクトを遂行していく必要がある．

ソフトウェア開発プロジェクトにおける実証的アプローチの実践を支援するためのツールとして，EPM(Empirical Project Monitor)[8]等が研究開発されている．EPM 等の既存のツールでは，収集データの管理，分析に RDB(リレーショナルデータベース)のシステムが利用されている．

一方，近年では，NoSQL として様々なデータベースが開発されている．著者らは，その中でもグラフ DB[10]に着目し，それを実証的アプローチによるプロジェクト管理のデータ分析へ適用した．

グラフ DB は，NoSQL の DB の 1 種で，データをグラフモデルとして格納し，管理する仕組みを持つ[4,10]．グラフ DB は RDB と比較して，データ間の関連を辿ってデータを検索する処理のパフォーマンスの向上が期待できる．また，データ間のつながりをグラフモデルとして可視化することで，分析結果を俯瞰して捉えることができる．

†1 キャッツ(株) プロダクト事業本部
Product Div., CATS Co., Ltd.
†2 南山大学 理工学部 ソフトウェア工学科
Dep. of Software Engineering, Nanzan University
†3 (株)NTT データ ビジネスソリューション事業本部
Business Solution Sector, NTT DATA Corporation

本取り組みでは，実開発プロジェクトに対し，グラフ DB を用いたプロジェクト管理データの分析を適用した．その結果，以下の 2 つの観点で，グラフ DB を利用することの有用性を評価し，課題を抽出した．

(1) データ取得における有用性
(2) プロジェクト可視化における有用性

本稿の構成は次の通りである．2 章では関連研究について述べる．3 章では本取り組みにおけるグラフモデルの適用方法について述べる．4 章では，グラフ DB を用いたプロジェクト管理データの分析を，実開発プロジェクトに適用した結果を述べる．5 章では分析結果の評価と今後の課題について述べる．6 章を本稿のまとめとする．

2. 関連研究

本章では，関連研究について述べる．

2.1 グラフモデル

グラフ DB で用いられる一般的なグラフモデルとして，エッジラベル付きグラフとプロパティグラフ[10]が挙げられる．広く利用されているエッジラベル付きグラフの例として RDF(Resource Description Framework)がある[11]．RDF を用いたプロジェクト管理データの交換インタフェース仕様として，PROMCODE がある[1]．

これに対し，プロパティグラフでは，ノードとエッジにプロパティと呼ぶデータの集合を付与できることから，エッジラベル付きグラフモデルより表現能力が高い．そのため，本稿では，プロパティグラフを用いる．

プロパティモデルの構成は次の通りである．

(1) プロパティグラフは，ノード，エッジ，プロパティから構成される．
(2) ノードには，プロパティ(キー / 値ペア)が含まれている．
(3) エッジはノード間の関係を表す．
(4) 関係には名前と方向があり，必ず開始ノードと終了ノードがある．
(5) 関係にも，プロパティを含めることができる．

プロパティグラフの例を図 1 に示す．

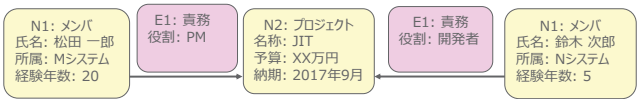

図 1　プロパティグラフの例

Figure 1　An example of property graphs

グラフモデルを用いることで，様々なシステムや実世界の要素間の繋がりを汎用的に表現することが可能となる．

2.2 グラフ DB を用いたプロジェクト分析

OSS(Open Source Software)開発コミュニティの構造を，グラフモデルを用いて表現する方法の提案がある[5]．GitHub 上の複数の OSS コミュニティに対し，提案手法を適用したコミュニティ進化の構造のモデル化が紹介されている．また，モデルを分析し，OSS コミュニティの特性を明らかにする試みが行われている．

また，グラフ DB が Github 上の複雑なエンティティやその関係を辿った検索に有効であると示されている[6]．この研究の成果は次の 3 点である．(1) GitHub に参加している開発者の中で，複数のスキルを持つ開発者を検索する方法の提案(優秀なリソースの検索)，(2) 同一のルートプロジェクトを持つプロジェクト間の不均衡さを測る基準の明確化，(3) 成熟したプロジェクトを検索する方法(OSS コミュニティに初めて参加する開発者の判断支援)．

3. グラフ DB の適用方法

本章では，本取り組みで実施したグラフ DB を用いたプロジェクト分析方法について述べる．

3.1 グラフ DB を用いたデータ分析プロセス

本取り組みにおいて適用したプロジェクト管理データの分析プロセスを図 2 に示す．

各実施項目の内容は以下の通りである：

(1) 分析項目の設定：対象とする分析項目を設定する．

(2) データベースの作成

(a) **モデリング：**分析に使用するデータ項目をノード，エッジ，プロパティの形式のグラフモデルとして定義する．

(b) **データ収集：**構成管理ツールやチケット管理ツール等が提供する API を使用して，必要なデータを収集する．

(c) **収集データの前処理：**収集したデータを加工し，定義したグラフモデルの形式に整形する．

(d) **データベースへ挿入：**収集データから作成したグラフモデルをグラフ DB へ挿入する．

(3) データ分析：設定した分析項目に従って分析し，結果を出力する．

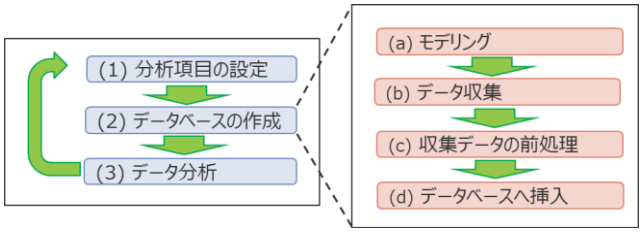

図 2　データ分析プロセス

Figure 2　A process for data analysis

本章の残りでは，実施項目(1)と(2)について述べる．まず，(1)で設定した分析項目について述べ，次に(2)-(a)で定義したグラフモデルの構成要素を述べる．最後に(2)-(b)から(2)-(d)のプロセスを説明する．実施項目(3)については，次章で述べる．

3.2 対象とした分析項目

対象の実開発プロジェクトにおいて，分析対象は結合テスト工程とした．分析項目は，対象期間内でのチケット間の関係，チケットと機能間の関係，チケットと期間の関係とした(表 1)．

表 1　分析項目

Table 1　Analysis items

分析項目	分析内容
チケット関係分析	対象期間内のチケットの関係を可視化し，定性分析を行う．
機能分析	開発対象となった機能とチケットの関係を可視化し，定性分析を行う．
時系列分析	時間の移行による以下の項目の移り変わりを可視化し，定性分析を行う： ・ バグチケット ・ 機能 ・ 機能に関連するソースコードへのコミット履歴

3.3 グラフモデルの構成要素

本取り組みにおける分析対象の開発プロジェクトにおいて，プロジェクト情報は，チケット管理ツールとして Redmine [9]，構成管理ツールとして Subvesion (SVN) [2]を用いて管理していた．これらのツールから取得可能なデータを用いて，グラフモデルを定義した．

図 3 に，各ツールとそこで管理される対象プロジェクトのデータを示す．

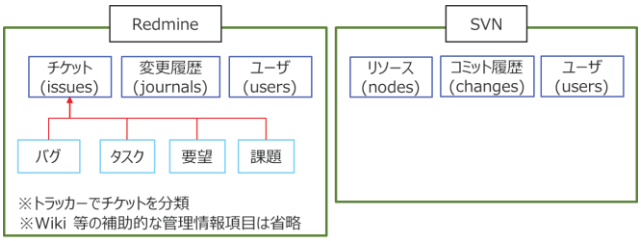

図 3　プロジェクト管理データの構成

Figure 3　Structure of project management data

これらの取得可能なデータからグラフモデルの構成要素となるノード，エッジ，プロパティを定義した．表 2 に定義したノードとその情報源を記す．

表 2　ノード定義

Table 2　Definition of nodes

情報源	ノード
基礎情報	Developer(開発者)
	Function(機能)
Redmine	Ticket(チケット)
	Bug(バグ)
	Task(タスク)
	Requirement(要求)
	Issue(課題)
	ChangeLog(変更履歴)
	ChangeDetail(変更詳細)
SVN	Artifact(ソースコード)
	CommitLog(コミット履歴)
その他	Period(期間)

各ノードの関係をエッジとして定義した．表 3 に，定義したエッジの対応を示す．エッジには開始ノードと終了ノードがある．表 3 の“From”の欄は開始ノードを表し，“To”は終了ノードを表す．

表 3　エッジ定義

Table 3　Definition of edges

エッジ	From	To
ASSIGN_IN	チケット変更履歴	開発者
ASSIGN_OUT	チケット変更履歴	開発者
BROTHER_OF	機能	機能
CHANGED_BY	チケット	チケット変更履歴
CHANGED_ON	チケット変更履歴	時期
CHECK_ON	チケット	時期
CHILD_OF	チケット	チケット
COMMITED_BY	ソースコード	開発者
	コミット履歴	開発者
COMMITED_ON	コミット履歴	時期
COMPOSED_OF	機能	ソースコード
CREATED_ON	チケット	時期
FIRST_ASSIGN_IN_ON	チケット	時期
HAS	チケット変更履歴	チケット変更詳細
LAST_ASSIGN_OUT_ON	チケット	時期
MAINLY_ASSIGN_TO	チケット	開発者
RELATED_TO	チケット	機能
	チケット	コミット履歴
TARGETED_BY	ソースコード	コミット履歴

また，各ノードにはプロパティを定義した．ノードの元データの付加情報を各ノードのプロパティとした．例として，ノードの 1 つであるチケットのプロパティを表 4 に示す．

表 4　チケットのプロパティ定義

Table 4　Properties of tickets

プロパティ	データ型
チケット番号(id)	文字列
題名(subject)	文字列
ステータス(status)	文字列
優先度(priority)	文字列
プロジェクト(project)	文字列
起票者(author)	文字列
起票時刻(created_on)	日付時間
更新時刻(updated_on)	日付時間
説明(description)	文字列
子チケット(children_ids)	文字列のリスト
関連チケット(issue_relations)	文字列のリスト
対象バージョン(fixed_version_id)	文字列
担当者(assigned_to_id)	文字列
開始日(start_date)	文字列
期日(due_date)	文字列
進捗(progress)	数値
予定工数(estimated_hours)	数値
担当者一覧(developer_ids)	文字列のリスト
主担当者(developer_id)	文字列

3.4　プロジェクトデータの収集からグラフ DB へのグラフモデルの挿入

グラフ DB は, Neo Technology 社の Neo4j[7]を利用した．また，データ収集，グラフモデル構成，グラフ DB へのグラフモデルの挿入を実行するためのスクリプトを Python で記述した．

データの収集は，管理ツールで提供されている API を使用した．収集したデータから，Python のデータとしてのグラフモデルを構成し，API を使って Neo4j へグラフモデルを構成した．

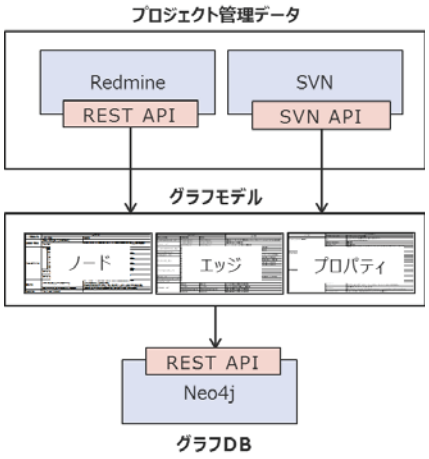

図 4　グラフデータベースの作成

Figure 4　Creating graph database

4. プロジェクト分析への適用

4.1 分析データの概要

分析に用いたデータの概要を表 5 に示す．変更履歴やコミット履歴等の変更に関するノードが多いほか，プロジェクトを経てストックされるリソースであるソースコードの数が多かった．

表 5　分析データ概要

Table 5　Summary of data analyzed

ノード種別	ラベル	ノード数
要求チケット	Ticket, Requirement	56
タスクチケット	Ticket, Task	1,071
課題チケット	Ticket, Issue	312
バグチケット	Ticket, Bug	306
チケット変更履歴	ChangeLog	8,422
チケット変更詳細	ChangeDetail	14,315
ソースコード	Artifact	33,714
コミット履歴	CommitLog	39,005
機能	Function	42
開発者	Developer	149
期間	Period	25

4.2 プロジェクトデータの意味的構造の分析

プロジェクトデータの意味的構造をグラフモデルで可視化し，構造分析を行った．本節では 3.2 節で述べたチケット関係分析，機能分析，時系列分析の 3 種類の分析結果を示し，5 章でその評価を行う．

4.2.1 チケット関係分析

プロジェクトの全チケットの親チケット-子チケット関係をグラフで可視化した結果を図 6 に示す．図 6 を大域的に見ると，タスク，要求，課題，バグそれぞれの子チケットとしてタスクが多数存在し，その親子関係からタスクのクラスタがいくつか形成されている(赤色の点線囲).

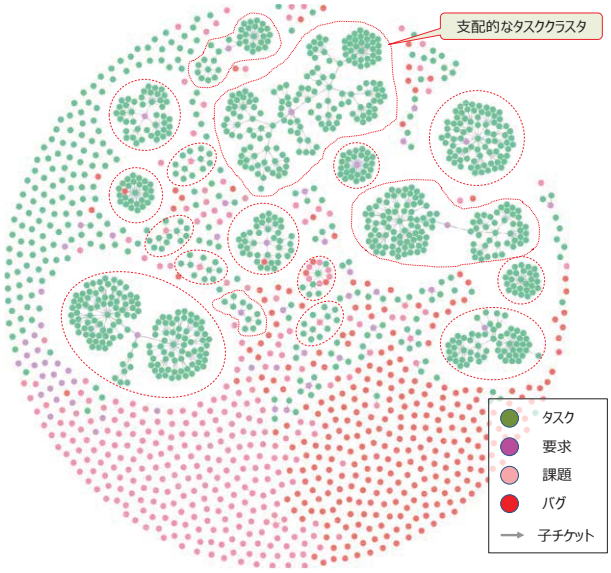

図 6　プロジェクトの全チケットの親子関係

Figure 6　Clustering of parent-child set of tickets

形成されたタスククラスタの内，特定のタスククラスタの局所的な関係に着目したものを図 7 に示す．図 7 を見ると，当クラスタが多段階の親子関係から構成されており，クラスタの最上位の親が要求チケットとなっている．

大域的な視点からプロジェクトにおいて支配的なタスクを把握することができ，局所的な視点からそのタスクの発生要因やグループ階層構造を知ることができる．

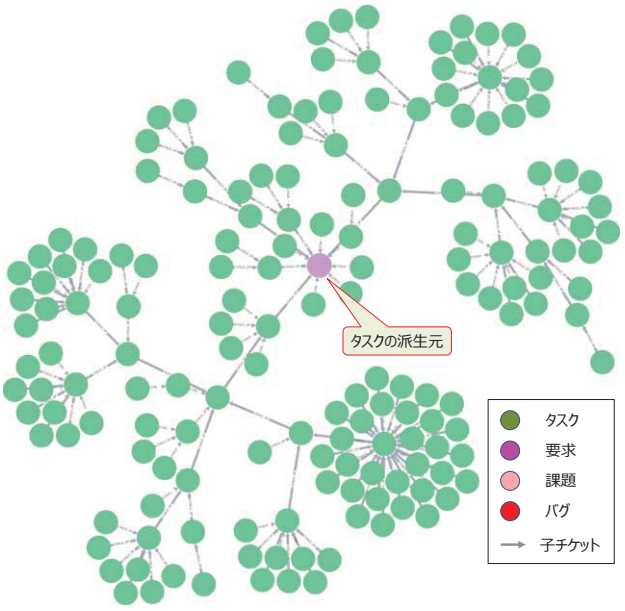

図 7　タスククラスタの局所構造

Figure 7　Local structure of task cluster

4.2.2 機能分析

プロジェクトで開発対象となった機能とチケットの関係を可視化した結果の一部を図 8 に示す．図 8 を見ると，単独の機能やある機能に関連する同列の機能(兄弟機能)を中心としてチケットがクラスタを形成しており，図 6 のチケットが機能毎にグループ化されて表示されている．

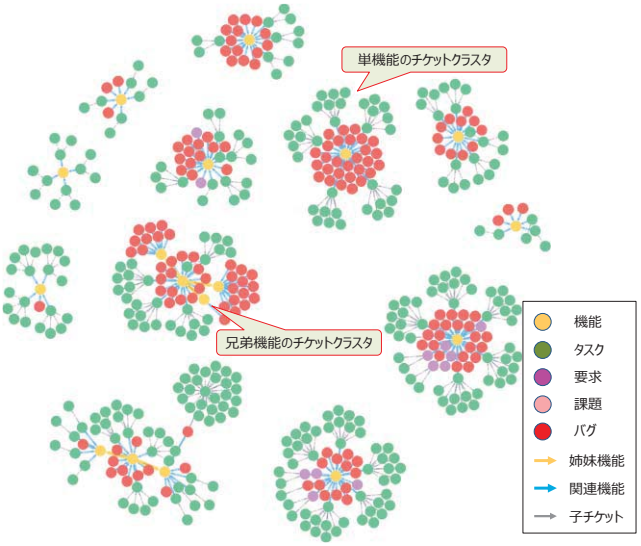

図 8　機能とチケットの関係

Figure 8　Graph structure of functions and tickets

ある機能 F1 周辺の局所構造に着目した抽出したグラフを図 9 に示す．当機能には 5 種類のタスクと独立した多数のバグが関係づけられている．テキストラベル等の情報から回帰テストに関するタスクが多いことが読み取れる．

大域構造の視点からは各機能のバグやタスク，要求，課題の量や比率などの定量的特性を視覚的に把握できる．一方，局所構造の視点からはタスクの親子階層構造やチケット派生関係，タスクの内容情報といった質的データを読みとることができる．

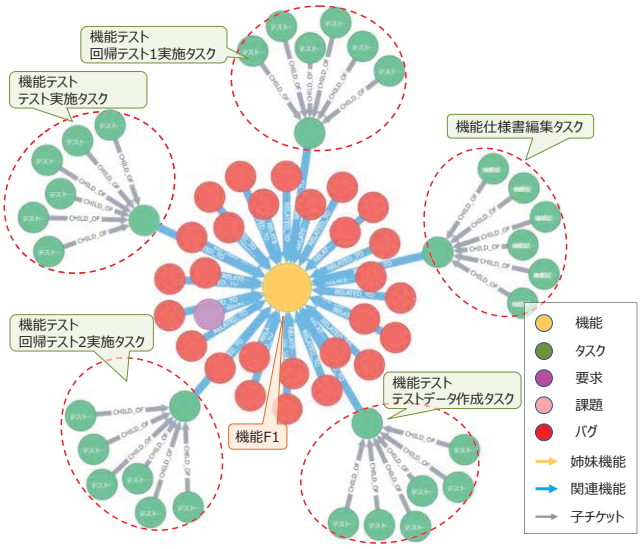

図 9　機能 F1 周辺の局所的な関係
Figure 9　Local structure around the Function F1

4.2.3 時系列分析

(1) バグチケットと期間の関係の分析

バグと期間の関係を可視化した結果を図 10 に示す．

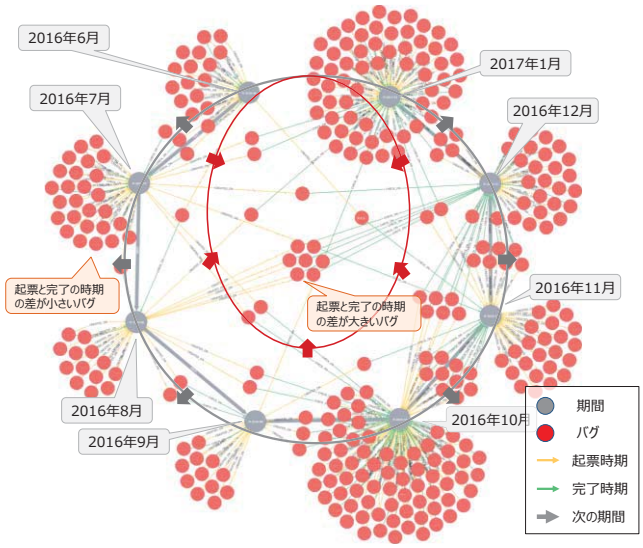

図 10　バグと期間の関係
Figure 10　Relationship between bugs and duration for fixing them

バグチケットと期間が(a)起票時期(黄色線), (b)完了時期(緑色線)の 2 種類の関係があり，起票から修正完了までの期間が長いノードほど中央上方に，短いノードほど周辺に現れている．これによって，チケットの起票数や完了数が多かった時期や，処置期間が長かったチケットを把握できる．

(2) 機能 F1 に関連するチケットと期間の関係の分析

機能 F1 と関連するチケットと期間の関係を可視化した結果を図 11 に示す. 図 9 で見たチケットの情報に期間の情報が付加されており，タスク実施期間とバグ処置期間の時間的前後関係が表現されている．

ノードの内容情報や時間的前後関係から「テスト実施タスクによってバグが起票された」のようなノード要素間の因果関係を読み取ることができる．

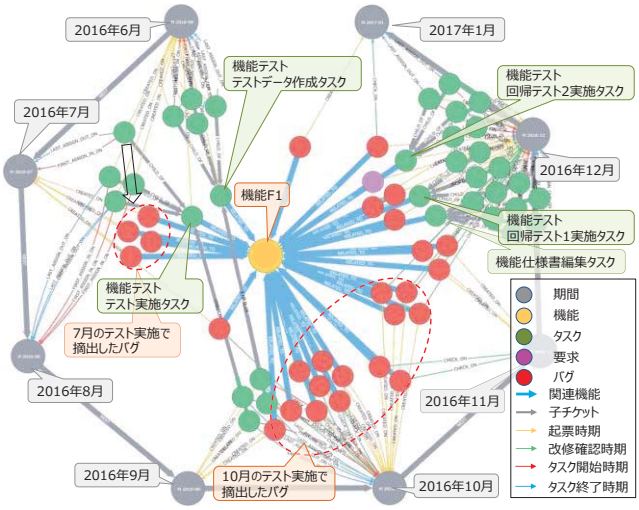

図 11　機能 F1 に関連するチケットと期間の関係
Figure 11　Relationship between ticket and task duration around the Function F1

(3) 機能 F1 に関連するソースコードのコミット履歴分析

機能 F1 に関連するソースコードとコミット履歴，開発者，期間の関係を可視化した結果を図 12 に示す．

コミット履歴が期間やコミット対象のソースコード，コミットした開発者の情報を持っている．図 12 を見ると，コミット数が多いほどエッジの数が多く，長期間にわたってコミットした人やコミットされたソースコードほど中央に現れている．

図 12 からは，機能 F1 に関連するソースコードの内，ソース a が長期間にわたって複数の開発者から多数の変更を加えられていることが把握できる．また，ノード要素間の意味的構造からソース a は開発者 C と開発者 E によって主にコミットされていること，ノード要素の内容情報からソース a の名称やパッケージを読み取ることができる．

プロジェクトの中で，より多く変更が加えられたソースコードを把握することができ，さらに，そのソースの開発者毎のコミット数の内訳やソースの内容などを掘り下げて知ることができる．

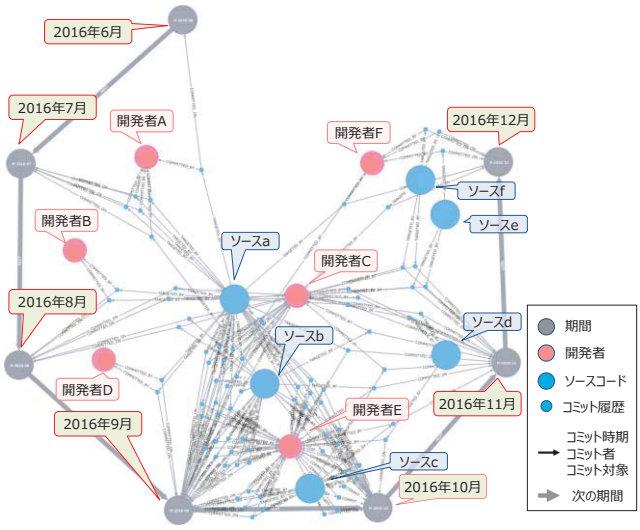

図 12　機能 F1 に関連するコミットと期間の関係
Figure 12　Relationship between commit and task duration around the Function F1

5. 分析結果の評価と課題

5.1 グラフ DB を用いたプロジェクト分析の有用性の評価

5.1.1 データ取得における有用性の評価

グラフ DB を用いてプロジェクト分析に必要なデータを取得する過程で頻繁に利用したデータの取得パターンとそのクエリを表 6 に示す. データ取得のパターンは(1) 検索・フィルタ，(2) グループ分割，(3) 集計 の 3 つに大別することができる. 以下, その 3 つの機能について検証を行う.

(1) 検索・フィルタ

プロジェクト分析を行う上では，機能や期間，開発者等様々な観点から取得するデータの条件を選定して利用する場面がある.

グラフ DB では,「優先度が高いバグの抽出」のようなプロパティ値を利用した条件検索ができる. さらに，表 6 にあるように,「機能 ID が”F001”を構成するソースコードにコミットした開発者の取得」のような関係構造を利用した条件検索を行うことができる. 関係構造を辿った条件検索の応用として，順序構造や階層構造等の再帰構造を持つデータの検索も可能である. 関係を辿った局所的なデータ操作を高効率に行うことができる[10].

RDB でもテーブル結合を行うことによって関係構造を利用したデータの条件検索と同様のデータ取得ができるが, テーブル結合を多数行うにつれて処理効率が落ちることが知られている.「友達の友達」のような再帰構造を持つデータの検索は難しいことも指摘されている[10]. また，テーブル構造が厳格に定められている RDB では Redmine のカスタムプロパティに見られるように，柔軟に付加できる属性を定義する場合には，テーブル構造が意味的構造と乖離した構造となることが多く，データ取得の効率が低化する恐れがある.

したがって，多種類のノードの関係構造を利用した条件検索や，再帰構造を持つデータの検索など，柔軟なデータの検索やフィルタを効率的にできることがグラフ DB の優位点だと言える.

表 6　データの取得パターンとクエリ
Table 6　Data collection patterns and associated queries

分類	項目	例	クエリ	取得データ例
検索・フィルタ	プロパティ値を利用したフィルタ	優先度が高いバグチケットを取得 ([優先度]が"高"の『バグチケット』に限定)	MATCH (バグN:Bug) WHERE バグN.優先度 = "高" RETURN バグN	バグN {バグチケット1} {バグチケット2} {バグチケット3} …
	関係構造を利用したフィルタ	機能に関連するバグチケットを取得 (『機能』と関係を持つ『バグチケット』に限定)	MATCH (機能N:Function)<-[:RELATED_TO]-(バグN:Bug) RETURN バグN	
	関係構造とプロパティ値の両方を利用したフィルタ	機能IDが"F001"の機能に関連するバグチケットを取得 ([id]が"F001"の『機能』と関係をもつ『バグチケット』に限定)	MATCH (機能N:Function)<-[:RELATED_TO]-(バグN:Bug) WHERE 機能N.id = "F001" RETURN バグN	
	多数のノードの関係構造とプロパティ値を利用したフィルタ	機能IDが"F001"の機能の実装者を取得 (『機能』を構成する『ソースコード』に『コミット』した『開発者』に限定) ※グラフDBの3種の関係を利用したデータ取得は RDBでは3回のテーブル結合に相当し、効率が悪化する。	MATCH (機能N:Function{id:"F001"}) -->(:Artifact)-->(:CommitLog)-->(開発者N:Developer) RETURN 開発者N	開発者N {開発者1} {開発者2} {開発者3} …
		上記の内、期間が2016年10～11月のものを取得 (『コミット』した『期間』の[ID]が 201610 以上, 201611 以下)	MATCH (機能N:Function{id:"F001"})-->(:Artifact) -->(コミットN:CommitLog)-->(開発者N:Developer), (コミットN)-->(期間:Period) WHERE 201610<= 期間.id <=201611 RETURN 開発者N	
		バグ改修コミットが行われたソースにコミットした実装者を取得 (『バグチケット』に関連した『コミット』の対象となっている『ソースコード』に『コミット』した『開発者』) ※共通の関係を持つ同種ノードの一階再帰検索。 RDBでは『友達の友達』のような方式でのデータの検索は難しい。	MATCH (バグN:Bug) OPTIONAL MATCH (c1:CommitLog),(c2:CommitLog), (バグN)-->(c1)<--(a:Artifact)-->(c2)--(開発者N:Developer) WHERE NOT c1 = c2 RETURN 開発者N	
	再帰構造の検索	機能IDがF001の機能に関係するチケットを子孫チケットまで取得 (『チケット』の子の『チケット』を再帰的に取得) ※階層構造や順序構造を持つノードの再帰検索。	MATCH (機能N:Function{id:"F001"}) OPTIONAL MATCH (親チケットN:Ticket), (子孫チケットN:Ticket), (機能N)-->(親チケットN)<-[:CHILD_OF*]-(子孫チケットN) RETURN 親チケットN , 子孫チケットN	親チケットN , 子孫チケットN {親チケット1}, {子孫チケット1} {親チケット1}, {子孫チケット2} {親チケット2}, {子孫チケット3}…
グループ分割	プロパティ値を利用したグループ分割	優先度別にバグを取得 (『バグチケット』の[優先度]別にデータを取得)	MATCH (バグN:Bug) RETURN バグN.優先度 AS 優先度, バグN	優先度, バグN "高" , {バグチケット1} "高" , {バグチケット2} …
	関係構造を利用したグループ分割	機能別にバグを取得 (『バグチケット』と関係を持つ『機能』別にデータを取得)	MATCH (機能N:Function)--(バグN:Bug) RETURN 機能N , バグN	機能 , バグN {機能}, {バグチケット1} {機能}, {バグチケット2} …
集計	ノード数の計数	機能ID別にバグ数を取得 (『バグチケット』のノード数を取得)	MATCH (機能N:Function)--(バグN:Bug) RETURN 機能N.id AS 機能ID, count(バグN) AS バグチケット数	機能ID, バグチケット数 "F001", 12 "F002", 15 …
	プロパティ値の集計	機能ID別にバグの処置期間の平均値を取得 (『バグチケット』の[処置期間]の平均値を取得)	MATCH (機能N:Function)--(バグN:Bug) RETURN 機能N.id AS 機能ID, avg(バグN.処置日数) AS 平均処置日数	機能ID, 平均処置時間 "F001", 10.5 "F002", 25.2 …

(2) グループ分割

データを分析するためには，機能別・期間別・開発者別等様々な切り口でデータを分割抽出することが必要となる．

グラフ DB では RDB と同様にデータをグループ分割して取得することができる．グラフ DB は(1)検索・フィルタ と同様に，プロパティ別にグループ分割ができるほか，関係構造を利用したグループ分割ができる．複数のキーでグループ分割することも可能で，例えば親チケットと親チケットの起票期間別にデータをまとめて抽出することができる．

このように，ノードの関係構造を利用した柔軟なデータの分割が効率的に行えることがグラフ DB の優位点だと言える．

(3) 集計

定量分析を行うには，データの個数や和などのデータの集計が必要となる．データ集計は単純な集計結果を表示する目的で用いられるほか，相関分析のような統計分析に必要なレコードテーブルを作成する目的で用いられる．

グラフ DB では RDB と同様にデータを集計するプロシージャが利用できる．グラフ DB で集計する場合，関係を辿ってデータ収集とテーブル作成を行い，その後に集計演算を行うという順序となる．「機能 ID が”F001”を構成するソースコードへのコミット数をソースコード別に集計する」のようなデータ取得をする際にも，必要なデータだけを効率的に収集することができる．関係構造を利用して，様々な観点からデータの集計やレコードテーブルの作成を行うことできる．

RDB ではテーブルが直接集計される．テーブル結合がある場合にはテーブル結合によってテーブル作成を行い，その後に集計するという順序となる．テーブル結合時には基本的にはテーブルのレコード全てが結合対象となるので，上記の集計を行う場合には，機能 ID が”F001”以外のレコードを含めてテーブル結合が行われるため，余分な処理が生じる．

集計用途が固定されており用途に対してスキーマ構造や実行計画を最適化した場合は RDB の方が効率的となるが，多様な集計用途で用いられ，用途が時々刻々と変化する場合にはグラフ DB の方がより効率的であると言える．

5.1.2 プロジェクト可視化における有用性の評価

意味的構造を表現できるグラフモデルを用いたプロジェクトの可視化の有用性を検証する．グラフモデルによる表現はノードのレイアウト方法によって異なる．(1)〜(4)ではグラフモデルによる表現の利点を述べ，(5)，(6)では本評価で用いた可視化ツール Neo4j Browser のレイアウト方法の利点を述べる．Neo4j Browser は力学シミュレーション(Force Simulation)[3]を利用したレイアウト方法を採用しており，関連が多いほど引力が強く，ノードが大きいほど斥力が強いという性質を持っている．

(1) 質的(定性)データと量的(定量)データの同時可視化

ノードやエッジはその数によって量的(定量)データを読み取れるほか，エッジの意味やノードの内容を見ることよって質的(定性)データを読み取ることができる．

この性質はプロジェクトにおける支配的な要素を特定し，その要素の詳細な情報を把握するといった用途において有用だと考えられる．

(2) 多対多の関係の可視化

二次元表では多対多の関係は 2 種類の要素間の関係しか表現できないが，グラフモデルではノードとエッジを用いることによって，多種類の要素間の関係を表現できる．

プロジェクト分析においては，成果物間の対応関係や開発者とタスクの対応関係等の多種多様な関係の情報を扱う必要があり，それらの対応関係を把握するといった用途において有用だと考えられる．

(3) 多元情報の可視化

ノードやエッジの形質(色，大きさ，テキストラベル，等)をノードの種類別に割り当てることによって情報を多元的に可視化できる．例えば，重要度が高いほどノードを大きく，優先度が高いほど色を濃くして要素を表示することで，優先度や重要度別にバグ数を表現できる．

分析対象の要素を多次元の属性別に分類して表現する用途において有用だと考えられる．

(4) 階層構造の可視化

チケットの親子関係や派生関係等から形成される階層構造を可視化することができる．

プロジェクトで扱うデータには，WBS やソフトウェアパッケージ等の階層構造が多くみられる．それらの階層構造を可視化することよって，プロジェクトにおける要素の位置づけを把握する用途において有用だと考えられる．

(5) ネットワーク中心的なノードの把握

中心的な要素が図の中央に現れる性質がありその把握に役立つ．例えば，図 6 ではネットワーク中心的なノードが各クラスタの中心に現れている．図 10 や図 11 では，期間ノードが環状に並べて配置されており，長期間関連を維持するノードが中心に集まっている．

期間や機能など指定した領域の中で中心的な役割を果たしている要素を発見する用途において有用だと考えられる．

(6) 類似関係や前後関係の把握

周辺に似た関係を持つノード同士は近くに配置される性質があり，類似の性質を持つノードを発見することができる．また，図 11 のように期間等の順序を持つノードを順序通りに並べれば，関連するバグやタスク等のノードもそれに応じて配置され，時間的前後関係を把握できる．

時系列の関係を見ることによって因果関係を把握する等の用途で有用だと考えられる．

5.2 今後の課題

(1) グラフモデルの可視化ツールの検討

本評価で用いた Neo4j Browser では，ノードやエッジのプロパティに対応させてノードやエッジの形質を調節することができず，形質をプロパティ別に表示したい場合は別種のノード，エッジとして定義し直す必要がある．また，レイアウト方法は力学シミュレーションによる配置または手動配置のみが利用可能で，階層別配置や環状配置等のレイアウト方法には対応していない．

他の可視化ツールやレイアウト方法を用いた場合を含めて，グラフモデルによるプロジェクト可視化の有用性を評価する必要がある．

(2) 分析対象データの拡張

本評価で分析に用いたデータは Redmine，SVN の 2 種類のツールのデータのみで，SVN の取得対象のリソースをソースコードに絞っていた．設計書やテストケース等の他のリソースを対象にした場合や，Skype の発話データ等，他のツールで運用されるデータを用いた場合に，グラフモデルの構造の可視化によってどのようなプロジェクト分析が可能かを検証する必要がある．

(3) 意味的構造に基づく分析効果の評価

意味的構造の定義内容によってデータの取得効率や，構造の可視化によって把握できる内容が変化する．今後，グラフモデルによるプロジェクト分析の事例を蓄積し，どのような意味的構造を定義することによってデータ取得が効率的にできるか，意味のある構造の可視化ができるか等を評価する必要がある．

6. まとめ

本取り組みでは，実開発プロジェクトの分析に，グラフ DB を適用し，その有用性を評価した．そのために，グラフ DB の適用方法を明らかにし，プロジェクトの管理データの意味的構造を分析した．

分析項目として，プロジェクト内で発行されたチケット間の関係，機能とチケットの関係，期間とチケットの関係を対象とした．

分析結果から，プロジェクト管理データの取得におけるグラフ DB の有用性，およびプロジェクト可視化におけるグラフモデルの有用性が明らかになった．

今後，より多くの事例へグラフ DB を適用する．さらに，グラフモデル可視化ツールの検討，分析対象データの拡張，分析結果に対する新たな評価項目の適用を通して，プロジェクト管理の実証的アプローチへのグラフ DB の適用可能性を研究する．

7. 参考文献

[1] M. Aoyama, et al., Applications of PROMCODE Open Project Management Platform to Large-Scale Multi-Vendor Projects, Proc. of ProMAC 2015, The Society of Project Management, Oct. 2015, pp. 31-36.
[2] Apache Subversion, https://subversion.apache.org.
[3] M. J. Bannister, et al., Force-Directed Graph Drawing Using Social Gravity and Scaling, Proc. of GD 2012, LNCS Vol. 7704, Springer, Sep. 2012, pp. 415-425.
[4] 石坂 登 他，グラフ型データベース入門，リックテレコム，2016.
[5] 加藤 聖也，稲垣 遥太，青山 幹雄，グラフモデルを用いた OSS コミュニティ構造分析方法の提案と評価，情報処理学会ソフトウェア工学研究会，Vol. 2017-SE-195, No. 23, Mar. 2017, pp.1-8.
[6] Z. Luo, X. Mao, and A. Li, An Exploratory Research of GitHub Based on Graph Model, Proc. of FCST 2015, IEEE, Aug. 2015, pp. 97-103.
[7] Neo Technology, Neo4j, 2016, http://neo4j.com/.
[8] 大平 雅雄 他，ソフトウェア開発プロジェクトのリアルタイム管理を目的とした支援システム，電子情報通信学会論文誌 D-I，Vol. J88-D-I, No. 2, 2005, pp. 228-239.
[9] Redmine, http://www.redmine.org.
[10] I. Robinson, et al., Graph Databases, O'Reilly, 2015.
[11] W3C, Resource Description Framework (RDF), 2014, https://www.w3.org/RDF/.