特 別 研 究 報 告 書

題 目

仕事同士の関連性の可視化

指導教員

報 告 者

檀上 未来

岡山大学工学部 情報工学科

平成 25 年 2 月 7 日 提出

要約

近年，ソフトウェア開発の現場ではアジャイル開発が広まっている．アジャイル開発は開発するソフトウェアの仕様や設計の変更が頻繁に起こることを想定している．アジャイル開発では，仕様や設計の変更に伴って頻繁に新しい仕事が発生する．仕事を単に並べた仕事一覧を使用して，開発メンバ全員で情報を共有する．

このような仕事一覧を用いたアジャイル開発では，3 つの問題が存在する．仕事に対する認識に差が生じること，仕事の依存関係が把握できないこと，および仕事内容が重複することである．これらの問題が原因で仕事同士が持つ関連性の把握が困難になり，ソフトウェア開発の進捗に遅れが生じる．

そこで本論文では，仕事同士の関連性の可視化を考案した．仕事同士の関連性として，集約関係，依存関係，および重複関係の 3 種類を想定し，可視化にはグラフを用いた．仕事同士の関連性を可視化した仕事一覧を用いて仕事の情報を共有することにより，仕事同士の関連性の把握が容易になる．

アジャイル開発と相性の良いシステムとして，仕事の単位としてチケットを用いるチケット管理システムをとりあげた．また，仕事同士の関連性の可視化を実現するソフトウェアとして，チケット管理システムを持つプロジェクト管理ソフトウェア Redmine を例にとった．本研究でのチケットの情報と仕事同士の関連性の対応を定義した．最後に，実際のソフトウェア開発で作成されたチケットを使用してグラフを作成し，仕事同士の関連性が可視化できていることを確認した．

目次

図 目 次

表 目 次

第 1 章

はじめに

近年，ソフトウェア開発の現場ではアジャイル開発 [1] が広まっている．アジャイル開発は，発生した仕事を単に並べた仕事一覧を使用して，開発メンバ全員で仕事一覧を共有する開発法である．このような仕事一覧を用いる際に，以下の 3 つの問題が発生する場合がある．

(1) 開発メンバごとの仕事一覧に対する認識に差が生じる

(2) 仕事の担当者が，自身の仕事に他の仕事が依存していることを把握できない

(3) 開発メンバ間で仕事内容が重複する

これらの問題発生を事前に防ぐために，アジャイル開発では開発メンバ間の対話や顧客との協調をプラクティスの重要な部分として位置づけている [2]．これによって上記の問題を解決することができる．しかしながら，アジャイル開発の普及と共に，開発に携わる人数が増加してきた．また，従来はウォーターフォール開発を採用していたチームがアジャイル開発を取り入れ，不慣れな開発メンバがいる状態でアジャイル開発を進める場合がある．このため，関連性の把握が開発メンバ間の対話や顧客との協調では補えなくなってきた．

また，オープンソース開発によってソーシャルコーディングが注目されている [3]．ここでは，不特定多数のユーザや開発者からの改善提案やコードの提供 (プルリクエスト) がある．これらの関係を把握することは困難であるといえる．

これらの問題は，仕事同士が関連性を持っているにも関わらず，関連性の把握が困難なことが原因である．関連性の情報は，従来アジャイル開発で用いていた仕事一覧を見るだけでは参照できないため，把握に時間がかかる．

この問題の対処として，本研究では仕事同士の関連性の可視化を提案する．仕事同士の関連性とは，仕事が持つ他の仕事との関連性を指す．可視化の方式としてグラフを使用する．仕事同士の関連性の可視化によって，仕事同士の関連性の把握を支援する．

第 2 章

ソフトウェア開発における仕事の発生

2.1　アジャイル開発の普及

近年，ソフトウェア開発の現場ではアジャイル開発が広まっている．アジャイル開発の開発工程を図 2.1 に示す．アジャイル開発は，反復と呼ばれる開発期間をくり返す．1 回の反復で計画，設計，実装，テスト，およびリリースを行う．アジャイル開発では，開発対象を多数の小さな機能に分割し，1 回の反復で 1 つないし複数の機能を開発する．開発を計画した機能が反復内に十分に実現できない場合でも，反復の終了時には機能追加された新しいバージョンをリリースする．この際，新しいバージョンが顧客の要求を満たしているかどうか確認し，顧客の要求と差異がある場合は仕様を見直し，次の反復に入る．このようにして，アジャイル開発は仕様の変更に柔軟に対応できる．

2.2　アジャイル開発法スクラム

アジャイル開発を採用する開発手法は複数存在する．ここではアジャイル開発の例として，スクラム [1] を挙げる．スクラムでは，チームを組んで開発を行う．チームのメンバが受け持つ 3 つの役割を以下に示す．

(1) チームメンバ

チームメンバの人数は 5 人から 9 人が適当とされる．チームメンバは実装とテストができることが前提となる．

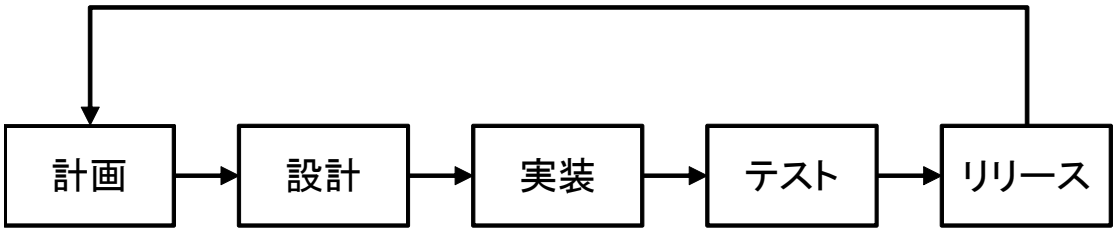

図　2.1　アジャイル開発の開発工程

(2) スクラムマスタ

主にチーム内外の関係調整と外部妨害の対処を行う．また，以下で示すデイリースクラムの進行役である．

(3) プロダクトオーナ

製品の総責任者であり，顧客の要望の代弁者である．以下で示すプロダクトバックログに，顧客の要望に優先順位をつけたものを反映させる．

スクラムで使用される 6 つの用語を以下に示す．

(1) スプリント

スクラムにおける開発期間の単位である．2 週間から 1ヶ月の短い期間が推奨されている．短期間でくり返し製品の仕様や設計を見直し，顧客の要望との差異を減らすことが目的である．

(2) プロダクトバックログ

製品に要求される機能とその優先度をリスト化したものである．主にスプリントの最後に優先順位を見直し，次のスプリントに入る．

(3) スプリントゴール

スプリントの最初に決める，スプリント中に実装するべき目標である．

(4) スプリントバックログ

スプリントゴールをより詳細なタスクに分解したものである．

(5) タスクリスト

スプリントバックログをチームメンバに割り当てるために，スプリントバックログをさらに細かく分けたタスクのリストである．1 つのタスクは 4〜16 時間で仕上げられる程度の粒度にすることが推奨される．

(6) デイリースクラム

スプリントの間，毎日決まった場所および時刻で開催するミーティングを指す．時間の目安は 15 分程度である．ここでは以下の 3 点をスクラムマスタの質問に答える形で発表し合う．

(A) 前回のスクラム以降何を行ったか

(B) 仕事の障害となっていることはあるか

(C) 次回のスクラムまでに何を行うか

スクラムの開発の流れを図 2.2 を用いて説明する．

(1) プロダクトバックログを定める．

(2) プロダクトバックログからスプリントで実装するべきスプリントゴールを選択する．

(3) スプリントゴールをより詳細なタスクに分解したスプリントバックログを作成する．

(4) スプリントバックログをタスクリストに分割する．

(5) タスクリストをチームメンバに割り当てる．

(6) スプリントの間，毎日デイリースクラムを開催する．

(7) スプリントの最後にスプリントレビューミーティングを開き，スプリントで作成した新機能のデモを行う．このため，スプリントの最後は製品が動作可能な状態でなければならない．このとき，製品はスプリントゴールが実装されていることが想定される．

(8) 次回のスプリントに備えてプロダクトバックログの内容と優先度の見直しを行う．

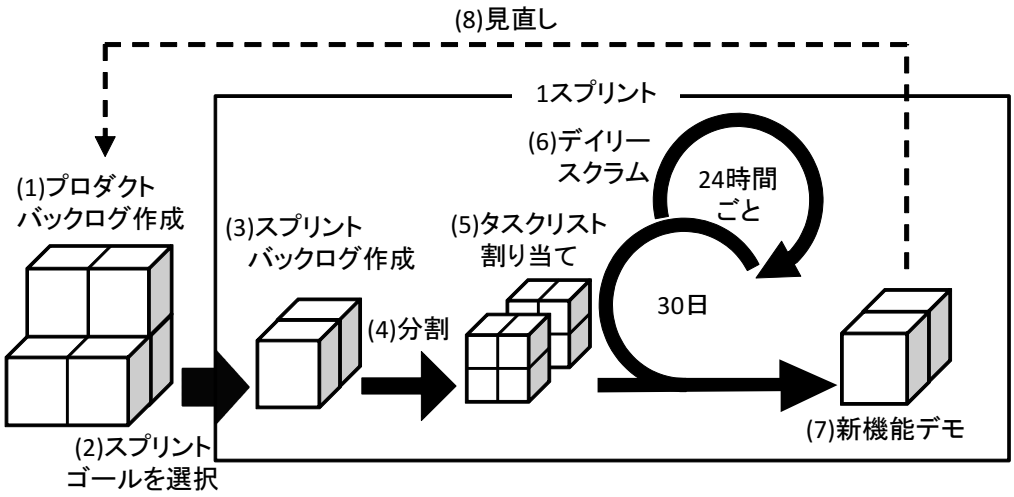

図 2.2 スクラムの反復

2.3 アジャイル開発における仕事の発生

アジャイル開発では，開発するソフトウェアの仕様や設計の変更が頻繁に起こることを想定している．ソフトウェアの仕様や設計の変更に伴い，新しい仕事が発生する．

例として，スクラムでの新しい仕事の発生について図 2.2 を用いて説明する．スクラムでは，スプリントの最後にスプリントレビューミーティングを開き，(7) で示すようにスプリントで作成した新機能のデモを行う．このとき，スプリントで作成した新機能が顧客の要求を満たしていない場合がある．この場合，(1) で作成された仕様や設計を記述するプロダクトバックログを (8) で示すように見直し，書き換える．これによって，スプリントごとに仕様や設計が変更される．また，プロダクトバックログを書き換えることで，プロダクトバックログから選択されるスプリントバックログとスプリントバックログを分割して作成されるタスクリストが新しく作成される．このように，新しいスプリントが始まるたびに新しい仕事が発生する．スプリントは 2 週間から 1ヶ月で新しくなるため，頻繁に新しい仕事が発生しているといえる．

第 3 章

アジャイル開発における問題点

3.1　アジャイル開発での仕事一覧

アジャイル開発では，仕事一覧を作成し開発メンバ全員が仕事一覧の情報を共有できる状態にする．アジャイル開発ではホワイトボードに付箋を貼って仕事一覧を管理する場合が多い．ホワイトボードで仕事一覧を共有する例を図 3.1 に示す．この例では，仕事を進捗別に「ToDo」,「Doing」，および「DONE」に分類している．図 3.1 に示す仕事一覧を簡略化したものを図 3.2 に示す．ただし，ここでは進捗による分類は行っていない．仕事一覧の枠の中の四角で囲んだ部分を 1 つの仕事とする．図 3.2 は,「システム A 実装」,「バグ 1 解決」,「機能 a 修正」,「バグ 1 修正」,「バグ 2 修正」，および「機能 b 実装」の 6 つの仕事を含む仕事一覧を表している．仕事の粒度はそれぞれ異なる場合がある．仕事には担当者が存在し，担当者は複数の仕事を担当する場合がある．図 3.2 のような仕事一覧を用いたアジャイル開発では，以下の 3 つの問題が発生する．

(1) 開発メンバごとの仕事一覧に対する認識に差が生じる

(2) 仕事の担当者が，自身の仕事に他の仕事が依存していることを把握できない

(3) 開発メンバ間で仕事内容が重複する

これらの問題に対して，次節で詳しく述べる．

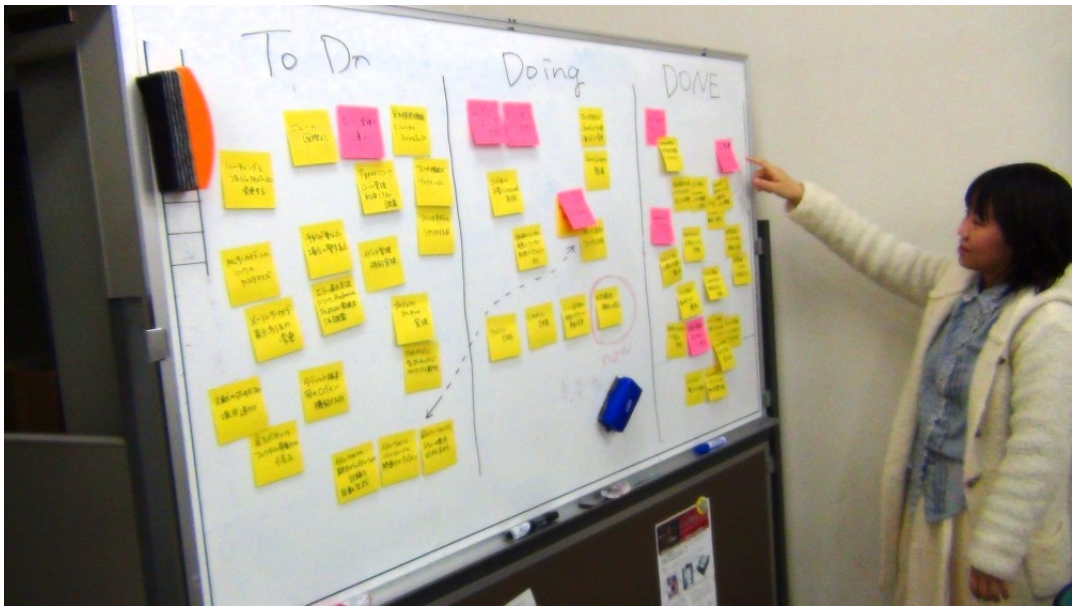

図　3.1　ホワイトボードでの仕事一覧の管理

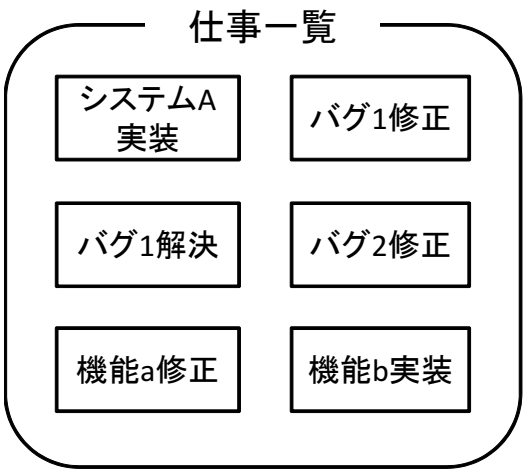

図　3.2　仕事一覧の例

3.2　仕事一覧を用いた開発における問題点

3.2.1　仕事に対する認識の相違

開発メンバごとの仕事に対する認識に差が生じる場合がある．例として仕事の依頼を挙げる．仕事の依頼の手順と，依頼時の仕事に対する認識の差を図 3.3 に示す．依頼者が被依頼者に機能 a 修正を依頼する手順を以下に示す.

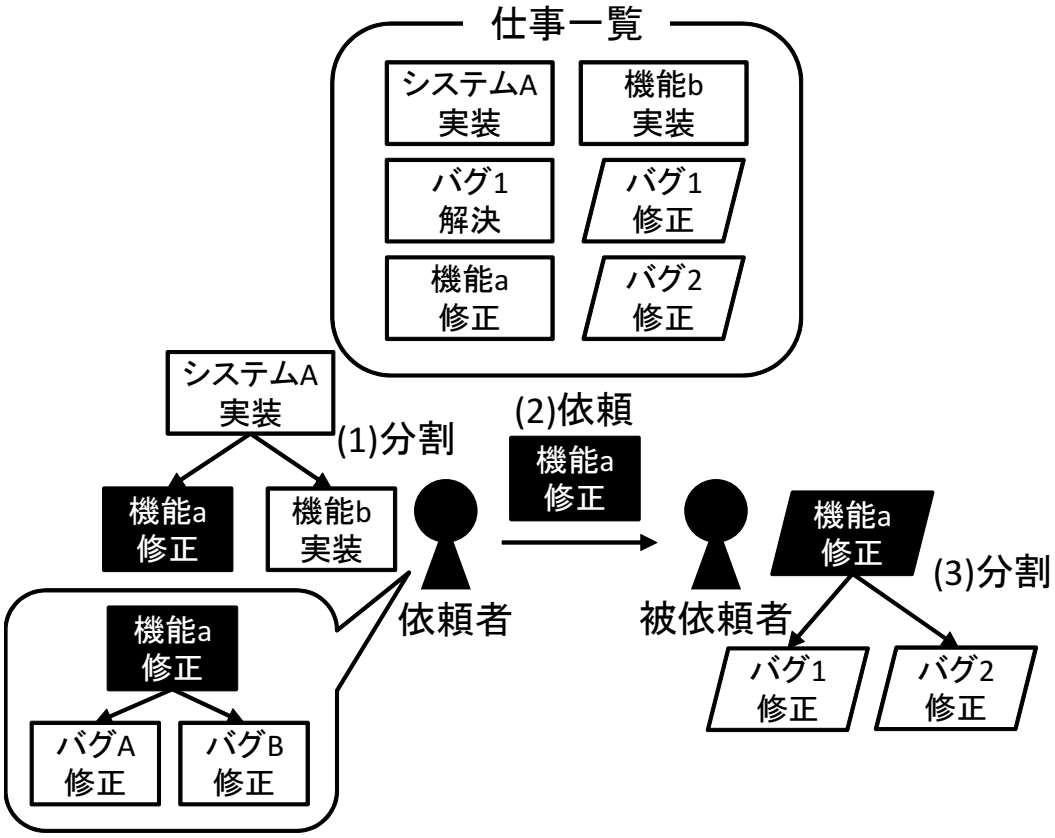

図　3.3　仕事の依頼と仕事に対する認識の差

(1) 依頼者がシステム A 実装を機能 a 修正と機能 b 実装に分割

(2) 依頼者が被依頼者に機能 a 修正を依頼

(3) 被依頼者が依頼された機能 a 修正をバグ 1 修正とバグ 2 修正に分割

まず，被依頼者の認識に対する問題を述べる．仕事一覧を見ることで被依頼者はシステム A 実装の存在を把握できる．しかし，機能 a 修正がシステム A 実装を分割して作成されたことは把握できない．このため，被依頼者は依頼された機能 a 修正がシステム A 実装の一部であることを把握できず，依頼者の意図通りに機能 a 修正を実行できない可能性がある．また一般に，担当する仕事の目的が不明な場合，作業効率は低下する [4][5]．このため，依頼者の意図通りに機能 a 修正を実行できないだけでなく，バグ 1 修正とバグ 2 修正を処理するためにかかる時間も増加する可能性がある．

次に，依頼者の認識に対する問題を述べる．手順 (2) で依頼した際に，被依頼者の認識に対する問題で述べたとおり，依頼者の想定する機能 a 修正の内容を被依頼者が正しく受け取

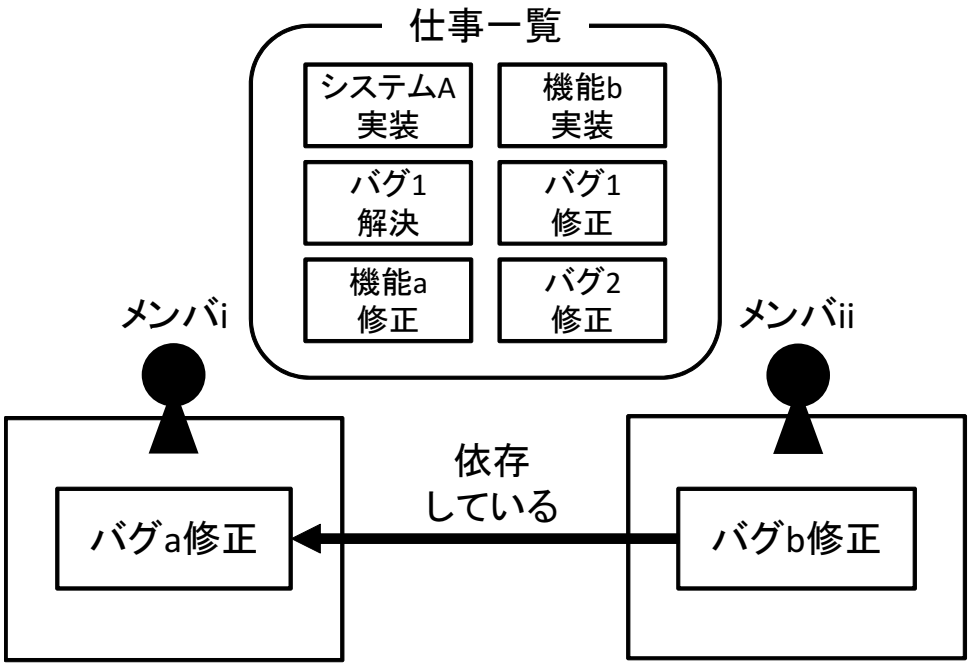

図　3.4　依存している仕事

れない場合がある．この場合，依頼者の意図通りに機能 a 修正を実行できない．依頼者の想定する機能 a 修正の内容は四角で囲まれたバグ 1 修正とバグ 2 修正である．また，被依頼者が実行した機能 a 修正の内容は菱形で囲まれたバグ 1 修正とバグ 2 修正である．一般に，依頼者が想定した内容と実際の内容が異なる場合，依頼した仕事の内容は見直しが必要である．このとき，仕事一覧を見ることで依頼者はバグ 1 修正とバグ 2 修正の存在を把握できる．しかし，これらの仕事が機能 a 修正を分割して作成されたことは把握できない．このため，依頼者は実行された機能 a 修正の内容と，想定した機能 a 修正の内容との差異に気づかない．依頼者は機能 a 修正の内容の見直しを提案することなく開発を進めてしまう場合がある．

開発メンバごとの依頼された仕事に対する認識の差により，これら 2 つの問題が発生する．

3.2.2　仕事の依存

仕事の担当者が，自身の仕事に他のメンバの仕事が依存していることを把握できず，他のメンバの進捗に悪影響を及ぼす場合がある．図 3.4 はメンバ ii の仕事がメンバ i の仕事に依存している場合を示す．この場合，バグ 2 修正にとりかかるにはバグ 1 修正の終了が必要である．仕事一覧を見ただけでは，メンバ i はバグ 2 修正がバグ 1 修正に依存していることを把握できない．このため，バグ 1 修正が後回しにされる可能性がある．バグ 1 修正が後回し

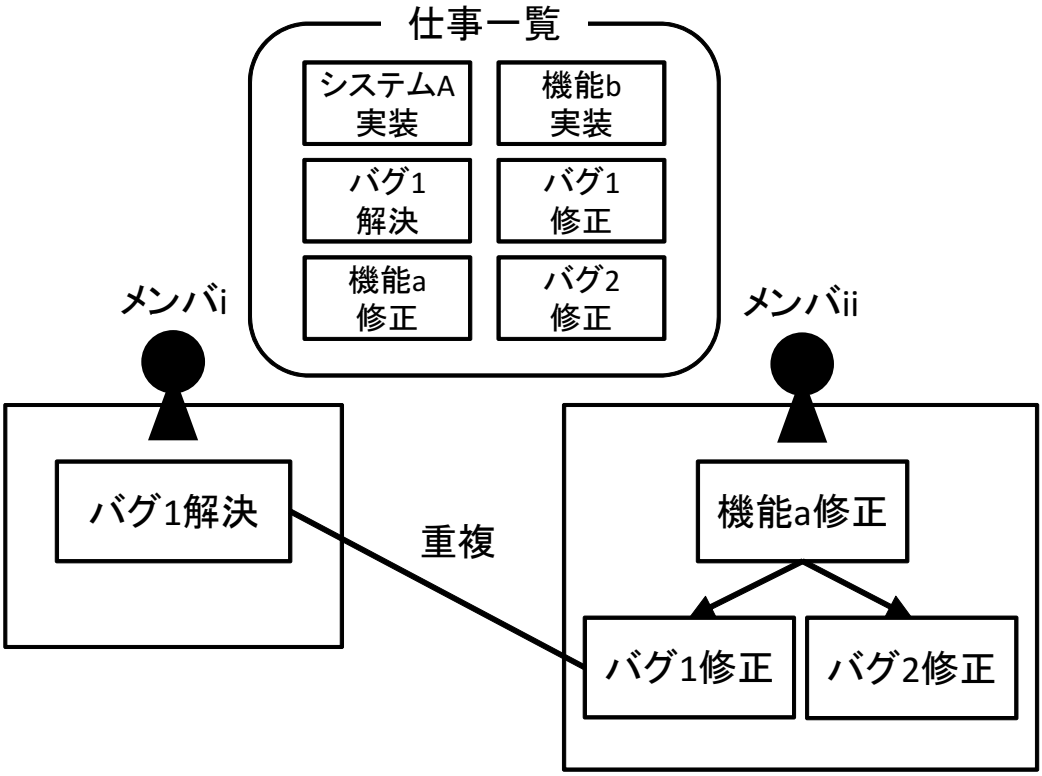

図 3.5 重複している仕事

にされた場合，メンバ ii はバグ 2 修正に取り掛かれず，開発は停止状態になる．このように仕事の担当メンバだけでなく他のメンバの進捗も遅れるため，開発全体の進行の妨げになる場合がある．

3.2.3　仕事内容の重複

他のメンバが既に対処している仕事にもう一度対処してしまい，開発メンバ同士で仕事内容が重複してしまう場合がある．図 3.5 はメンバ i の仕事とメンバ ii の仕事が重複する場合を示す．メンバ ii の担当する機能 a 修正を分割するとバグ 1 修正とバグ 2 修正になる．メンバ ii が作成したバグ 1 修正とメンバ i が既に担当しているバグ 1 解決は同じ内容を指す．しかし，メンバ ii は同じ内容であることに気づかずにバグ 1 修正を実行する可能性がある．メンバ i とメンバ ii が別々の環境で修正し開発期間の最後に統合すると，バグ 1 を修正するという処理が競合する．競合の修正には時間がかかるため，開発全体の進行の妨げになる．また，メンバ i とメンバ ii が同じ作業を実行すること自体効率が悪い．

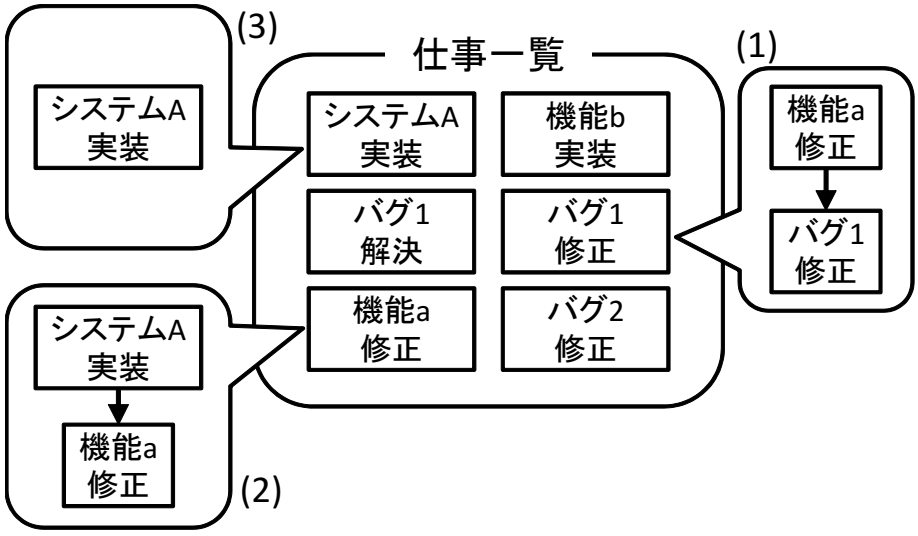

図　3.6　別の仕事との関連性の把握手順

3.3　問題点の原因

3.2 節で挙げた問題は，ソフトウェア開発の進捗に悪影響を与える．これらの問題の原因は，仕事同士が関連性を持っているにも関わらず，図 3.2 のような仕事一覧からでは関連性の把握が困難な点である．図 3.2 のような仕事一覧を用いたアジャイル開発では，関連性の情報は開発メンバの頭の中や仕事を管理する紙面上，およびデータベース上に記録される．例えば，仕事を管理する紙面とはホワイトボードに付箋を貼りつけたものや書類が挙げられる．開発メンバが少ない状況では，開発メンバ間でコミュニケーションをとり，記録された関連性の情報を把握する．これによって開発メンバ全員で開発の現状の認識を統一する．関連性の情報を把握する手順を図 3.6 に示す．吹き出しの情報は，仕事とその集約先の仕事を表している．図 3.6 で示すように，開発メンバはバグ 1 修正の元の仕事を以下のような手順で把握する．

(1) バグ 1 修正は機能 a 修正を分割して作成されたことを確認

(2) 機能 a 修正はシステム A 実装を分割して作成されたことを確認

(3) システム A 実装は別の仕事を分割して作成されたわけではないことを確認

この手順の結果，バグ 1 修正の元の仕事はシステム A 実装であることが把握できる．このように，仕事の構成を把握するために，複数の手順を踏む必要がある．しかし，このような方

法で情報を把握することには限界があり，情報が見落とされる場合がある．また，大人数の開発では開発メンバ間のコミュニケーションで現状の認識を統一するのは困難である．

第 4 章

対処

4.1　仕事同士の関連性の可視化

仕事がもつ他の仕事との関連性を**仕事同士の関連性**と呼ぶ．本研究では，3 章で述べた問題点に対処するため，仕事同士の関連性を可視化する．仕事同士の関連性として 3 種類を想定している．図 4.1 にソフトウェア開発の仕事一覧の例を示す．これは，図 3.2 で示した仕事一覧の例に 3 種類の仕事同士の関連性の情報を付加し，可視化したものである．図 4.1 で可視化した 3 種類の仕事同士の関連性を以下に示す.

(1) 集約関係

ある仕事を分割した場合，分割された仕事は元の仕事に集約される．分割された仕事が全て終了すると，元の仕事が終了したことになる．この仕事同士の関連性を集約関係と呼ぶ．例えば，システム A 実装をバグ 1 解決，機能 a 修正，および機能 b 実装に分割する．この場合，バグ 1 解決，機能 a 修正，および機能 b 実装はシステム A 実装に集約される．また，システム A 実装を集約先，バグ 1 解決，機能 a 修正，および機能 b 実装を分割先と呼ぶ.

(2) 依存関係

仕事 A にとりかかるために別の仕事 B の終了が必要な場合，仕事 B は仕事 A に依存している．この仕事同士の関連性を依存関係と呼ぶ．例えば，バグ 2 修正にとりかかるためにバグ 1 修正の終了が必要な場合，バグ 2 修正はバグ 1 修正に依存している．

(3) 重複関係

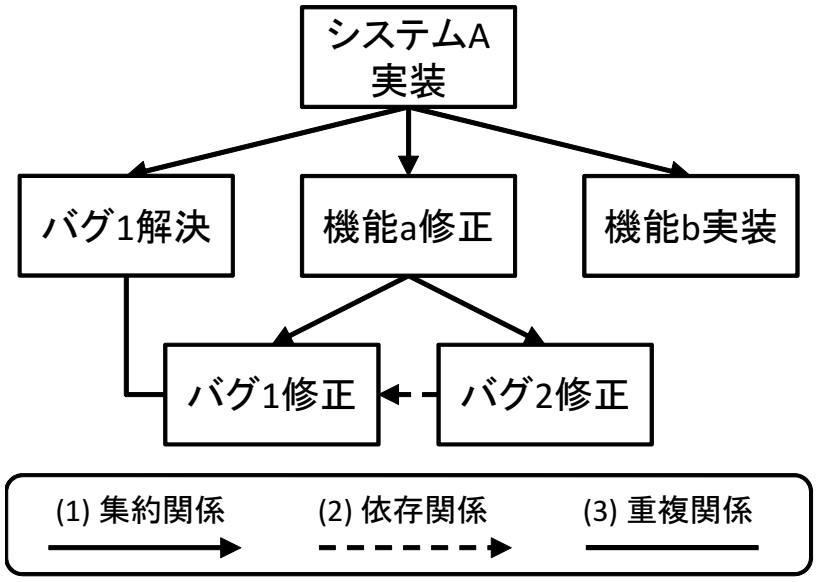

図　4.1　仕事同士の関連性を可視化した仕事一覧

仕事 A と仕事 B の内容が同様である場合，仕事 A と仕事 B は重複している．この仕事同士の関連性を重複関係と呼ぶ．この関連性は方向を持たない．例えば，バグ 1 解決とバグ 1 修正は，仕事の名前は異なるが同じ仕事内容を指す．この場合バグ 1 解決とバグ 1 修正は重複している．

これらの関連性は任意の仕事間で発生し，関連する 2 つの仕事の選定方法に制限はない．ただしループ構造はないものとする．ループ構造とは，例えば，システム A 実装からシステム A 実装自身へ仕事同士の関連性を持つような構造を指す．

図 4.1 のように仕事同士の関連性を可視化することで，3.2.1 項，3.2.2 項，および 3.2.3 項で挙げた問題に対処する．

3.2.1 項では，仕事に対する認識の差の発生を問題に挙げた．仕事の依頼において依頼者は，仕事同士の関連性を可視化した仕事一覧を見ることで，依頼者が担当していない仕事の持つ関連性を把握できる．同様に，被依頼者は担当していない仕事の持つ仕事同士の関連性を把握できる．これによって，依頼された仕事に対する認識が統一できる．

3.2.2 項では，仕事同士の依存関係を把握できないことを問題に挙げた．仕事同士の関連性を可視化した仕事一覧を見ると，他のメンバの仕事が担当する仕事に依存することに気づける．これによって，依存関係を持つ仕事に先にとりかかり，開発を進行できる．

3.2.3 項では，メンバ間で仕事内容が重複することを問題に挙げた．仕事同士の関連性を可視化した仕事一覧を見ると，重複する仕事が存在することに気づける．これによって，重

複する処理の発生を抑えられる．

4.2　可視化の方式

仕事同士の関連性のうち，集約関係と依存関係は方向を持つ．また，仕事同士の関連性を可視化したものは閉路をもつ場合がある．これらの特徴を満たすため，グラフを使って可視化する．グラフは，頂点を表すノードと，ノード間の連結関係を表すエッジで構成される．図 4.1 は仕事同士の関連性をグラフで表したものである．

各仕事はノードで表現し，仕事同士の関連性はエッジで表現する．また，方向を持つ関係性と方向を持たない関係性が混在するため，図 4.1 のように有向と無向のエッジが混在したグラフになる．

第 5 章

設計

5.1 ソフトウェア開発における仕事同士の関連性の実現

5.1.1 チケット管理システムの導入

2.1 節で述べたように，アジャイル開発では頻繁に新しい仕事が発生する．このため，図 3.1 のように，付箋で仕事を管理する方法には限界がある．これは，ホワイトボードの面積には物理的な制限があるためである．この問題を解消するために，チケット管理システムの導入を考える．チケットとは，問題や課題，見積もり工数や実行工数，および関連するソースなどの情報を集約したデータである．チケット管理システムでは，チケットをソフトウェア上で管理する．チケット管理システムにより，仕事を管理できる量の制限がなくなる．

5.1.2 チケット駆動開発

チケット管理システムは，チケット駆動開発と呼ばれる仕事の管理方法を採用する．チケット駆動開発が持つ 2 つの規則 [6] を以下に述べる．

(1) 仕事をチケットで管理する

開発メンバは，チケットに仕事の内容，担当者，進捗情報，および作業履歴を残す．チケット一覧は，仕事をチケットの形で一覧にしたものである．チケット一覧を見ることで，開発メンバは作業状況を共有できる．このとき作成されるチケット一覧は，本論文では図 3.2 に相当する．

(2) 仕事を始める前に必ずチケットを作成する

仕事を始める前に，必ずチケットを作成する．これによって，全ての仕事とチケットが対応付けられ，チケット一覧を仕事の履歴として利用できる．また，ソフトウェア開発に必要な作業の把握にもれがなくなる．

これらの規則を適用することで，チケット一覧を仕事一覧，および仕事の履歴として利用できる．このため，開発メンバはチケット一覧を見るだけで自身の担当の仕事を把握できる．

5.1.3 チケット管理システムとアジャイル開発

チケット管理システムとアジャイル開発は相性が良い．チケット駆動開発は仕事の管理方法であり，この方法を採用した仕事の管理システムがチケット管理システムである．また，アジャイル開発は開発工程の方針である．チケット管理システムとアジャイル開発が相性が良いことを示す 3 つの理由 [6] を以下に述べる．

(1) チケット内容の変更に強い

アジャイル開発は，開発するソフトウェアの仕様や設計の変更が頻繁に起こることを想定しているため，当初登録したチケットの内容が変更される場合が多い．また，ソフトウェアの仕様の変更により，チケットの枚数が増加する場合もある．チケット管理システムでは，開発メンバがチケットの情報を簡単に変更できる．また，開発メンバがチケットを自由に作成することができる．このため，チケットの変更や増加に対して柔軟に対応できる．

(2) リリースバージョンの変更に強い

アジャイル開発では，顧客との話し合いでリリースバージョンが含むチケットを変更する場合がある．チケット管理システムでは，開発メンバがチケットのリリースバージョンを他の反復期間に簡単に変更できる．このため，チケットが対応する反復期間の変更に柔軟に対応できる．

(3) 開発工程の変更に強い

アジャイル開発では，計画，設計，実装，テスト，およびリリースという開発工程を 1 回の反復としてくり返す．しかし，次回の反復の内容を検討した結果，開発工程が変更になる場合がある．例えば，製品の完成に必要な機能の設計と実装が全て終了した場合がある．この場合，次の反復ではテストを主体とした開発が行われる．チケット管

理システムは，開発工程ではなくチケットを中心に開発を計画するため，開発の計画が開発工程に依存することがない．このため，各反復の開発工程の変更によらず，開発を進めることができる．

5.2 プロジェクト管理ソフトウェアにおける仕事同士の関連性

4 章で述べた仕事同士の関連性をグラフ構造で表示する．これにより，仕事同士の関連性の可視化を実現する．本研究では実現するアプリケーションとして，Web ベースのプロジェクト管理ソフトウェアとして Redmine[7] を例にとる．

Redmine はバグ管理システムをもとにしたプロジェクト管理ソフトウェアである．Redmine はチケットやロードマップなどのプロジェクト管理機能が優れているため，5.1.3 項で述べた使用法を適用できる．

Redmine ではチケットを用いて開発を管理する．本研究では，チケットと仕事が対応しているものとする．仕事同士の関連性の可視化に使用するチケットの項目を以下に述べる．

(1) チケット ID

チケットが持つ ID である．チケットの識別に使用する正整数である．

(2) サブジェクト

チケットのタイトルである．チケットが示す仕事を端的に表現した文字列を持つ．

(3) 親チケット

チケットが発生するもととなったチケットの情報である．仕事同士の関連性での集約関係における集約先を示す．

(4) 子チケット

チケットがもとになって発生したチケットの情報である．仕事同士の関連性での集約関係における分割先を示す．

(5) 関連するチケット

チケットが相互に関連することを示す．仕事同士の関連性のうち，依存関係と重複関係を示す．

表 5.1 仕事同士の関連性とチケットの項目の対応

要素	グラフの要素	チケットの項目
仕事	ノード	チケット ID，サブジェクト
仕事同士の関連性	エッジ	親チケット，子チケット，関連するチケット

これらの Redmine のチケットが持つ情報をグラフ内に表示する．仕事同士の関連性の可視化で示す要素とグラフの要素，および Redmine のチケットの項目の対応関係を表 5.1 に示す．ノードは仕事を表して，チケットの項目のうちチケット ID とサブジェクトを表示する．また，エッジは仕事同士の関連性を表し，チケットの項目のうち親チケット，子チケット，および関連するチケットを利用して可視化する．

5.3 仕事同士の関連性の可視化の実現

5.3.1 プロジェクト管理ソフトウェア上での可視化

Redmine 上で図 4.1 に示すようなグラフによる仕事同士の関連性の可視化を行う．グラフの作成には Graphviz[8] を用いる．Graphviz は，オープンソースのグラフ作成ソフトであり，DOT 言語で書かれたグラフ表現を画像に変換する．Redmine 上に作成したグラフを表示するページを作成する．

5.3.2 仕事同士の関連性の付与手順

実際のプロジェクト上にある仕事に対して，仕事同士の関連性の可視化を行う．表 5.2 は，ソフトウェア開発における実際のチケットと，チケットが持つ仕事同士の関連性を示したものである．これらのチケットは，著者が所属する研究グループのソフトウェア開発において作成されたチケットの一部である．表中の番号は，それぞれのチケットに割り振られたチケット ID を指す．著者が所属する研究グループの開発メンバは 7 人である．チケット駆動開発を行っており，チケット管理システムとして Redmine を使用している．また，一部にアジャイル開発の手法を取り入れている．ソフトウェアの開発工程のうち，計画から開発までを 2 週間で反復し，4 回目の反復でソフトウェアのリリースを行う．

本研究では表 5.2 に示すチケットに仕事同士の関連性の可視化を適用する．チケットに仕事同士の関連性を付与する手順を以下に示す．

表 5.2 チケットが持つ仕事同士の関連性の一覧

チケット ID	親チケット ID	子チケット ID	関連するチケット ID
362			396
396		399	362
398			406,460
399	396		416
406			398
416		460	399
425		466	
434			466
460	416		398,466
461		462	
462	461		464
464			462,466
463		491	
466	425		434,460,464
491	463		

(1) キーワードの抽出

チケットのサブジェクトとディスクリプション (チケットの詳細な説明)，およびコメント履歴から，キーワードを抽出する．サブジェクトとディスクリプションに複数回出現する単語やグループ内で開発するシステムの名称，および実在するアプリケーションやプログラミング言語を優先的に抽出する．また，チケットのディスクリプションとコメント履歴に他のチケット ID が明記されていた場合，明記されたチケット ID をキーワードとして抽出する．この際，1 つのチケットから抽出するキーワードの数は 3 つ以内とする．

チケットの内容の例を表 5.3 に示す．このような内容のチケットの場合，キーワードとして以下の 3 つが抽出される．

(A) Rails

実在するアプリケーションの名称である．

(B) ♯460

表 5.3 チケットの内容の一例

チケットの持つ項目	項目が持つ情報
チケット ID	500
サブジェクト	Rails3.2.9 に対応
ディスクリプション	文書管理システムで使用する Rails のバージョンを 3.0.9 から 3.2.9 に変更する． Rails のバージョンを 3.2.9 に変更して，♯460 で 行った作業と同様の作業を行う．
コメント履歴	作業会で，テストの通過を確認した． 現在，JavaScript を使用する以下の箇所は動作しない． (1) フィルタ (文書，名簿，メールアーカイブ) (2) 文献の添付ファイルの表示とアップロード

チケットのディスクリプションに明記された他のチケット ID である．

(C) JavaScript

実在するプログラミング言語の名称である．

(2) 仕事同士の関連性の付与抽出されたキーワードをもとに，チケットに仕事同士の関連性を付与する．関連するチケット，および親チケットと子チケットの情報を付与する方法について以下に述べる．

(A) 関連するチケット

同様のキーワードを持つチケット同士は関連性を持つ．また，キーワードとして他のチケット ID を持つチケットは，チケット自身とキーワードになっているチケットとの間で関連性を持つ．関連性を持つと考えられるチケット同士を，お互いの関連するチケットとする．

(B) 親チケットと子チケット

グループ内で開発するシステムの名称をキーワードとして持つチケットのうち，作成日時が一番古いものを親チケットとする．この親チケットよりも作成日時が新しいもので，同様のシステムの名称をキーワードとして持つチケットを子チケットとする．

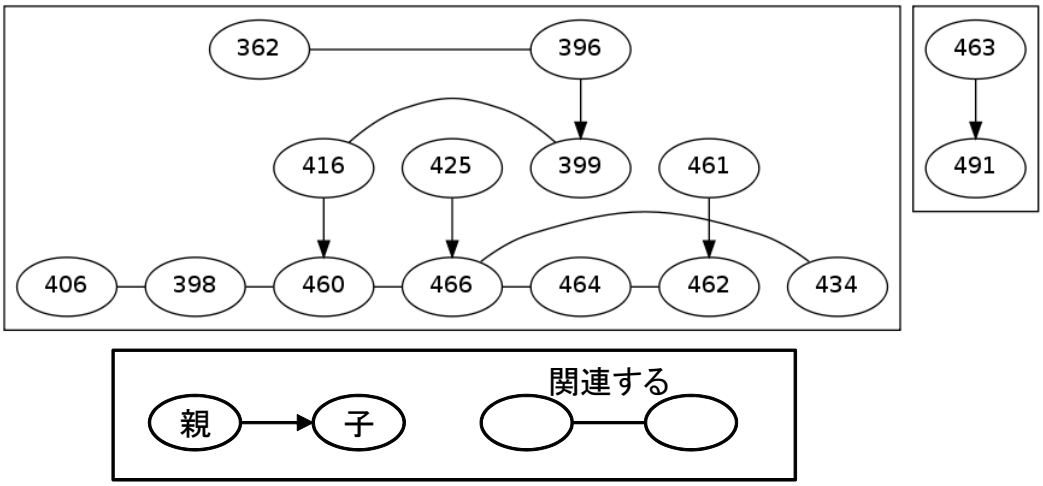

図　5.1　仕事同士の関連性を持つチケットのグラフ

5.3.3　グラフの提示方法

表 5.2 をグラフにより可視化した結果を図 5.1 に示す．仕事同士の関連性の可視化を適用したグラフには，表 5.1 に示すようにノードにチケットのサブジェクトを表示する必要がある．しかし，このグラフのノードにチケットの内容を示すサブジェクトを記述すると，ノードの大きさが不揃いになる．また，サブジェクトが長いものに全てのノードの大きさを揃えると，グラフ自体が大きくなり，仕事同士の関連性の把握が困難になると考えられる．

これに対して，ノードにチケット ID を示し，マウスオーバ時にサブジェクトを表示することで対処する．図 5.2 はチケット ID362 のノードにマウスオーバした際の表示を拡大して示したものである．これによって，チケットの大まかな内容を確認できる．

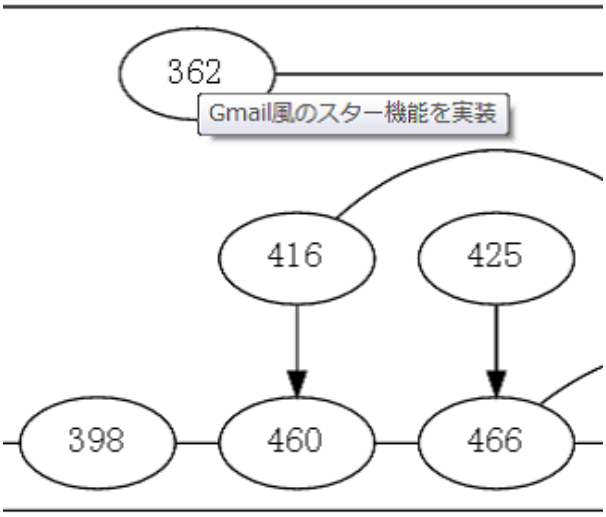

図　5.2　マウスオーバ時の表示

第 6 章

おわりに

作業一覧を用いたアジャイル開発における問題として以下の 3 点を述べた．

(1) 開発メンバごとの仕事一覧に対する認識に差が生じる

(2) 仕事の担当者が，自身の仕事に他の仕事が依存していることを把握できない

(3) 開発メンバ間で仕事内容が重複する

これらの問題への対処として，仕事同士の関連性の可視化を提案した．仕事同士の関連性として，集約関係，依存関係，および重複関係の 3 種類を想定し，可視化にはグラフを用いた．また，アジャイル開発とチケット管理システムの関係について述べ，仕事同士の関連性を実現するソフトウェアとしてプロジェクト管理ソフトウェアである Redmine を例にとった．Redmine のチケットにおける仕事同士の関連性の扱い方について述べた．さらに，Redmine 上で実際のソフトウェア開発で作成されたチケットを使用して，仕事同士の関連性の可視化を行った．仕事同士の関連性の付与手順と，仕事同士の関連性が可視化されたグラフにおける情報の提示方法について述べた．この際，グラフは Graphviz を用いて作成した．

残された課題として，仕事同士の関連性の付与の自動化とその実装がある．

謝辞

本研究を進めるにあたり，懇切丁寧なご指導をしていただきました乃村能成准教授に心より感謝の意を表します．また，研究活動において，数々のご指導やご助言を与えていただいた谷口秀夫教授，山内利宏准教授，後藤佑介助教に心から感謝申し上げます．

また，日頃の研究活動において，お世話になりました研究室の皆様に感謝いたします．

最後に，本研究を行うにあたり，経済的，精神的な支えとなった家族に感謝いたします．

参考文献

[1] Schwaber, K. and Beedle, M.: Agile Software Development with Scrum, Prentice Hall, (2002). スクラムエバンジェリストグループ，長瀬嘉秀，今野睦 (訳) : アジャイルソフトウェア開発スクラム，ピアソン桐原，(2003).

[2] Beck, K., Beedle, M., Van Bennekum, et al. : Manifesto for Agile Software Development, Agile Alliance(online), available from 〈www.agilemanifesto.org〉(accessed 2013-02-07).

[3] GitHub Inc.: GitHub(online), available from 〈https:/github.com〉(accessed 2013-02-07).

[4] ハイラム・W・スミス (著)，黄木信，ジェームス・スキナー (訳) : TQ 心の安らぎを発見する時間管理の探求 : 成功を約束する「10」の法則，pp．83，キングベアー出版，(1999).

[5] デビッド・アレン (著)，田口元 (訳):ストレスフリーの仕事術 : 仕事と人生をコントロールする 52 の法則，pp．88，二見書房，(2006).

[6] 小川明彦，阪井誠 : Redmine によるタスクマネジメント実践技法，pp．164-172，翔泳社 (2010).

[7] ファーエンドテクノロジー株式会社 : Redmine.JP，ファーエンドテクノロジー株式会社（オンライン），入手先〈www.redmine.org〉（参照 2013-02-04）.

[8] Perry, D.: Graphviz - Graph Visualization Software(online), available from ⟨www.graphviz.org/Home.php⟩ (accessed 2013-02-04).