アジャイル開発における適切なイテレーション期間のシミュレーションによる推定

塩浜龍志[†1] 坂本一憲[†1] 久保秋真[†2]
鷲崎弘宜[†1] 深澤良彰[†1]

アジャイル開発とは, 顧客参加を促しながら, 開発期間を短い単位であるイテレーションに区切って反復的に開発を進め, プラクティスと呼ばれる実践が推奨される項目・定石群を用いることで, 迅速に開発を行うことが出来る開発プロセスの総称である. イテレーション期間はアジャイル開発の進め方を定める重要な要素の一つである. しかし, その決定は根拠のない定性的な判断によって行われることが多く, 不適切なイテレーション期間で開発を行ったプロジェクトの失敗事例も報告されている. そこで我々はプロジェクトごとの適切なイテレーション期間を推定することにより, アジャイル開発の適用を支援する手法を提案した. 本論文の貢献は, 特定のプロジェクト要件における適切なイテレーション期間を推定しアジャイル開発の適用を支援をしたこと, 様々なプロジェクト要件によるシミュレーションを通し, プロジェクト要件と適切なイテレーション期間の関係について定量的に分析したことの２点である.

Estimating the appropriate iteration term of agile development using simulation

Ryushi Shiohama, Kazunori Sakamoto,
Shin Kuboaki, Hironori Washizaki
and Yoshiaki Fukazawa

Agile development is the software development methodology. it divide development term to short term that called iteration and uses body of knowledge about software development that called practice. Iteration length is important to agile development, but now there is no evidence to decide it because of lack of past experiences. So we proposes a new method to estimate appropriate iteration term of agile development by conducting the simulation to help apply the real projects. Our contributions are estimating appropriate term for each projects and analysis the relevant between iteration term and project constraints.

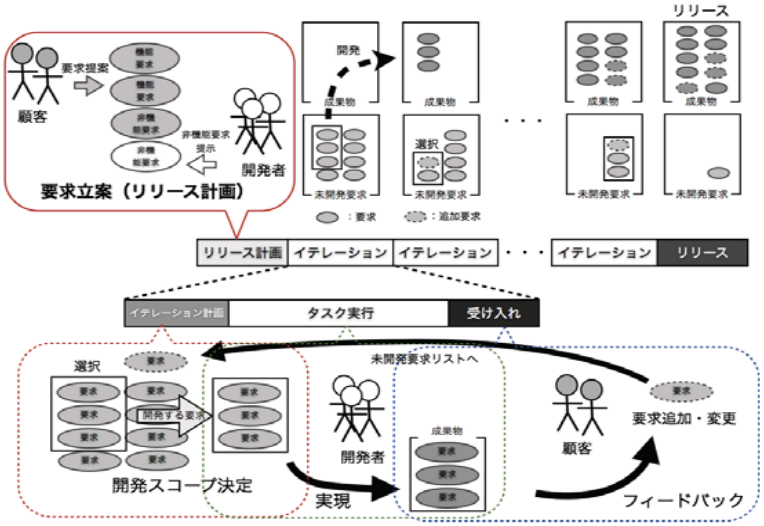

図 1 アジャイル開発モデル図

1. はじめに

アジャイル開発とは, 顧客参加を促しながら, 開発期間を短い単位であるイテレーションに区切って反復的に開発を進め, プラクティスと呼ばれる実践が推奨される項目・定石群を用いることで, 迅速に開発を行うことが出来る開発プロセスの総称である 5).

アジャイル開発の特徴を解説するため, 現在最も多く使用されているアジャイルモデルである 5), eXtreme Programming8)(以降, XP) と Scrum9) の共通項を抽出した開発形態を図 1 に示す. XP や Scrum では, イテレーションごとに要求の一部を実現し成果物をあげ, 成果物に対し顧客からのフィードバックを受け, フィードバックを活かし次期イテレーションを行う, というイテレーティブな開発手法をとる.

†1 早稲田大学
Waseda University
†2 株式会社アフレル
Afrel Co.,Ltd.

各イテレーションはイテレーション計画, タスク実行, 受け入れから構成される 3). 「イテレーション計画」では未実現要求リストからイテレーション内で実現すべき要求の範囲（スコープ）を決定し, 「タスク実行」でそれら要求を実現する. 「受け入れ」はイテレーション終了時に行い, イテレーション内で実現された要求は顧客からのフィードバックにより受け入れ, 変更, 追加が行う. 受け入れられた要求は成果物となり, 変更された要求, 追加された要求は未実現要求リストに戻る. 以降, 我々はこの図の開発形態をアジャイル開発の一般的な開発形態として扱う.

ソフトウェア開発において, 全てのプロジェクトに適する開発プロセスは存在しないと言われている 12). 開発の効率化を行うためには, 適した形に開発手法のパラメータを変更するなど, プロジェクトの条件に合わせ柔軟に対応する必要があり, 特にアジャイル開発ではイテレーション期間や選択するプラクティスなど変更し得るパラメータが多数存在する. 我々はその中でイテレーション期間に注目した.

アジャイル開発においてイテレーション期間は, 開発の粒度を定め, 開発にリズムを与え, 要求変更へ対応するタイミングを定めるなど, プロジェクトの成否に大きな影響を与える要因の一つである 7). しかし, イテレーション期間の決定方法に関する具体的な文献は少なく, XP で用いられる 1 週間や, Scrum で用いられる 1 ヶ月と言った経験に基づく参考値は存在するが, プロジェクト条件と適切なイテレーション期間の関係について言及するものはない. そのため, 現在イテレーション期間の決定はプロジェクトマネージャの定性的な判断により行われている. しかし定性的な判断は, 過去の事例や文献の不足から難しく, 不適切なイテレーション期間がアジャイル開発の大きな失敗要因となっている 6). 例えば, 長すぎるイテレーション期間で開発した場合, 顧客によるフィードバックを受ける機会が減り, 要求変更への迅速かつ柔軟な対応というアジャイル開発の特徴が失われる. また, イテレーションの開発スコープが肥大化することで複雑さが増し開発効率の低下も考えられる. 短すぎるイテレーション期間で開発した場合, イテレーション回数が増え, 計画や受け入れなどが冗長に行われるため, コスト面でのオーバーヘッドが増加する. また, タスク粒度を不必要に細かくする必要があり, 開発の遅れや複雑さの増加につながる.

アジャイル開発の既存の研究では, ウォーターフォール開発との比較や, その有用性を検証するものは多数存在する 11)8), 一方で, アジャイル開発の適用支援を目的とした研究は, 要求優先順位付けの方法についてシミュレーションモデルを作成し, 効率の良い優先順位付け方法について述べた 2), 情報のやり取りに注目しペトリネットを用いたモデルによりプロセスパターンの適用方法について言及した 4), ペアプログラミングとリファクタリングというプラクティスをシステムダイナミックモデルを用いシミュレートした 13) などが存在するが少なく, さらにイテレーション期間について言及するものはない.

そこで我々はアジャイル開発におけるプロジェクトごとの適切なイテレーション期間を推定することにより, アジャイル開発の適用を支援する手法を提案する.

本論文の主な貢献は以下の 2 点である.

- 特定のプロジェクト要件における適切なイテレーション期間を推定し, アジャイル開発のプロジェクトへの適用を支援する手法を提案する.
- 様々なプロジェクト要件によるシミュレーションを通し, プロジェクト要件と適切なイテレーション期間の関係について定量的に分析する.

本論文の構成を以下に示す. 2 節では提案手法について解説, 3 節でその結果について評価, 考察を行う. 4 節で本論文をまとめる.

2. シミュレーションによるイテレーション期間推定

我々はシミュレーションを用いプロジェクトごとの適切なイテレーション期間を推定し, アジャイル開発の適用を支援する手法を提案する. 提案手法では, アジャイル開発の一般的な開発形態の特徴を抽出しモデル化を行った. 本節ではシミュレーションを用いた理由, モデル化の方法, 振る舞いについて解説する.

2.1 シミュレーションを用いる理由

開発プロセスの実態を明らかにするには, 実存のプロジェクトの状態や結果を観察し普遍的な法則を見つけ出す方法や, 一般的な事象をモデル化したシミュレータを用いそれらから現実の事象を推定する方法などが存在する. 前者は, 人為的な誤差が含まれることや, 同じ条件による事例が存在しないことから, 正確な分析を行うために膨大な数のデータが必要となる. 一方, 後者は開発プロセスのそれぞれ既知の部分をモデル化し, それらを組み合わせシミュレータを作成することでデータ不足を補う形での定量的な分析が可能となる. そこで我々は後者の方法を採ることに決めた. ただし, 後者の方法においては, 分析結果の有効性はモデル化した範囲にとどまり, 現実に適合することの検証が望ましい. 本論文では, 4 節にて事例研究を用いた検証を行っている.

2.2 モデル入出力

本モデルは, 入力としてプロジェクトの特徴を十分抽出し得る 5 つのプロジェクト条件 1) を受け, 結果としてプロセスの相対的な評価を行うための値を出力する. まず, 入力として与える 5 つのプロジェクト条件を表 1 に示し, 以下で詳説する.

表 1　入力

名前		取り得る値	単位
プロジェクト制約	開発期間	1 以上の整数値	日
	変動性	0, 5, 10, 15, 20, 25, 30	%
	複雑性	25, 50, 75	%
開発者		0.25(初級者), 1(中級者), 2.5(上級者)	
要求	必要工数	1 以上の整数値	人日
	重要度	0〜1	

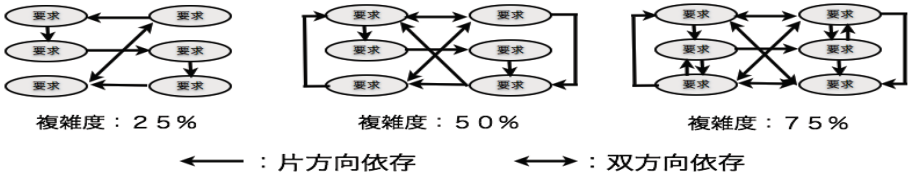

図 2　複雑性による依存関係の違い

開発期間　プロジェクトに携わる作業日数を表す. ただし我々は, 提案手法の範囲として初期の要求立案（図 1 における「リリース計画」）以降を仮定しているため, 最初のイテレーションが開始する時点をプロジェクト開始日と仮定する.

変動性　変動性は開発期間内で要求変更が起こる確率を表す. 要求変更とは要求の追加，変更，削除を示す. 要求変更は, 成果物を顧客が使用した際など顧客の理解度上昇により得られる変更と，市場の変化などの外的要因により起こる変更の二種類が存在する. 我々はこれらの要求変更をそれぞれ異なる方法でモデル化しているが，変動性のパラメータはどちらの要求変更に対しても影響する. なお，変動性は過去の実績とプロジェクトの新規性, 分野, 初期の要求の具体性などから推測して入力する.

複雑性　要求同士の依存関係が発生する確率を表す. 依存関係は要求の実現, 変更時の統合のタスクを考慮する際に影響を与える. 依存関係はその性質上, 片方向と双方向の二種類が存在し, 双方向の依存関係は片方向のものよりも強い影響を与えるものとする. なお, 複雑性は過去の実績とプロジェクトの分野, 規模などから推測して入力する. 複雑性と依存関係の関係を図 2 に示す.

開発者　開発者の能力を表す. 我々は開発者の能力を, 初級, 中級, 上級に大別することとした. 開発者の開発能力の差については 14)15) において報告されており, 本手法では, 現在の統合開発環境の進化や, オブジェクト指向, テスティングシステムなどを考慮した, 15) の初級者と上級者で 10 倍程度, 中級者と上級者で 2.5 倍程度の差があるという報告を参考とした.

要求：重要度　要求全体に対し, それぞれの要求の重要度を 0 1 の相対的な値で重み付けしたものを表す. 例えば, 在庫管理システムを考えた際に「在庫検索機能」, 「商品登録機能」, 「検索結果ソート機能」, 「入力サポート機能」などの要求があるとする. その際, 「在庫検索機能」や「商品登録機能」はシステムの根幹の要求であり最も重要度も高いと考えられるため 1 を与え, 「検索結果ソート機能」のように根幹ではないがシステムとして不可欠な要求には 0.6, 「入力サポート機能」に関しては付加的な要求であるため 0.2 などという形で入力する.

要求：必要工数　各要求について実現に必要と想定される工数を表す. 本手法においては, 上述の中級の開発者 1 人が 1 日で開発可能な量を 1 人日として, 何人日という単位で入力する.

次に出力結果について述べる. 出力結果は上述の通り, 相対的な比較を行うために我々が設定した値である価値とコストから構成される. また, 結果の考察などで用いる際, 我々は単純にコストパフォーマンスを表す価値/コストだけでなく, コスト, 価値, 価値/コストの 3 つの値を示す. これは, プロジェクトによって重視される評価軸が異なるため, それらを考慮の上評価を行うことを可能とするためである.

価値　プロジェクトの進行度合いを相対的に比較するための値を表す. 具体的には, 成果物に含まれる要求の重要度を合計したもので表す.

コスト　コストはその開発にかかったのべ時間を表す. 材料費や設備費などの付加的コストはイテレーション期間の変化に対して固定であると想定されるため考慮しない.

2.3　シミュレーションのモデル化方法.

図 1 の開発形態を元に統計的なデータや文献などを参考にモデル化した。シミュレータのモデル図を図 3 に示す.

(1)　「イテレーション計画」では,「要求優先順位付け」「スコープ決定」「タスク分割」を行う.

(a)　「要求優先順位付け」は要求の重要度と依存関係に基づき行う. 具体的には, 要求の価値に基づき降順に並び替えを行い, 次に優先順位の高い要求が依存関係を持つ要求の順位を上げる. この方法はアジャイル開発の, 価値駆動開発というプラクティスおよびイテレーション毎に動く部分機能を開発するというプラクティスから抽出した.

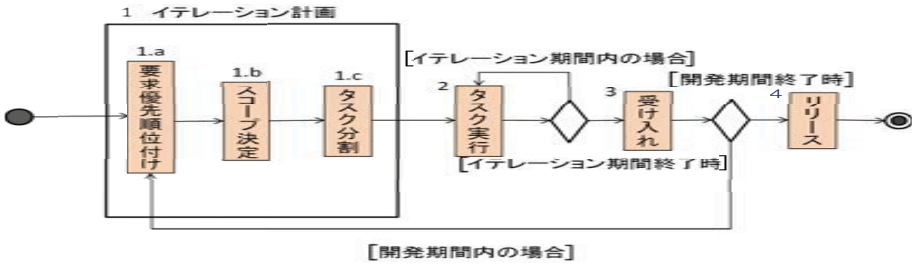

図 3　シミュレータモデル図

(b)　「スコープ決定」の方法には, 全体の要求に対し残りのイテレーション数から当該イテレーションで開発すべき要求数をスコープとするスコープボックスな方法と, イテレーションの長さや開発者数などから開発可能と見積もられる要求数をスコープとするタイムボックスな方法が存在する. 一般的にアジャイル開発ではタイムボックスが用いられることが多いこと. また, スコープを適切な大きさに保つことにより開発効率があがる 11) という報告を考慮して後者を選択した.

(c)　「タスク分割」では, 要求の必要工数から要求のタスク分割を行う. 本手法ではタスク分割の方法による影響を減らすため, 適当であると報告されているタスクの大きさ 5) である 0.5〜2 人日で固定し分割するものとした. また, 要求自体の実現に必要なタスクのみではなく、この際に依存関係から統合のタスクも考慮される. 統合のタスクは具体的には既実現の要求リストにある依存関係を持つ要求の数で決定される.

「イテレーション計画」の時間は, イテレーション期間とスコープの大きさに比例する形をとるよう, 便宜上以下の方法で決定するものとした.

pt：イテレーション計画時間 (hour), it：イテレーション期間 (day), ss：スコープの大きさ (一人辺りの見積もり工数)

pt = it * 2/5 + ss * 0.1

(2)　「タスク実行」では, 「イテレーション計画」で生成されたタスクを実現する. タスク実行の状態遷移図を図 4 に示す. 開発者はそれぞれが empty, assigned の二状態を持ち, タスクは todo, doing, done の三状態を持つ. 各単位時間毎に状態更新を図 4 の条件に従い行う. タスクへの assign の順序は明示的には指定していない. これは, タスク粒度を揃えているため、この部分での差異は誤差と考えられるため.

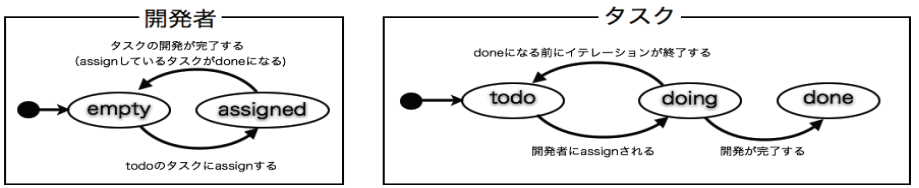

図 4　タスク実行状態遷移図

単位時間 (1 時間) 当たりの各開発者の開発量を以下の式で表す. 開発者能力: Da

開発量: Dc(i) = Da(i) * (0.5　2.0 の乱数)　乱数は状況による開発効率の差を表現する. 1 日の開発時間を 8 時間とし, 「タスク実行」が終了した時点で done 状態のタスクが, そのイテレーションで開発されたタスクとなる. 開発されたタスクはそれぞれ要求に戻され, 「受け入れ」段階では要求単位で扱われる. 全てのタスクが開発されていない要求は, 次期イテレーションへの持ち越しとなる 5).

(3)　「受け入れ」では, タスク実行で開発完了した各要求について変更または受け入れが行われる. これらはプロジェクトの変動性に基づいて確率的に発生する. ただし, 16) で示される要求充足モデルに従い, プロジェクトの収束に比例し, 変更の発生確率も逓減する. 変更された要求は変更作業および統合作業のタスクが追加され, 未完成の要求リストに戻される. 変更されなかった要求は成果物として受け入れが行われる. 次に, 成果物に対するフィードバックとして要求の追加が行われる. 追加要求も, 変更同様に変動性および要求充足モデルを考慮し発生確率が決定される. また追加要求に関しては別途, 市場の変化による要求追加も考えられる. この値は変動性にのみ基づき発生する.

(4)　「リリース」では, イテレーションを開発期間終了まで繰り返し, 終了時点での価値とコストを出力する. 本手法ではリリース時点で実現されていない要求が存在することを許容している. これはアジャイル開発のプラクティスであるタイムボックスという考え方に起因している. 本手法ではイテレーション期間を整数の連続値として与えているため, 最後のイテレーションにおいて日数が余るなどの問題が生じる. 本手法では, 各シミュレーションにおいて結果がばらつくことを防ぐため, 余りの日数がイテレーション期間の半数を超える場合は, 最後のイテレーションだけ規定の日数未満

として, 半数を超えない場合は, 最後のイテレーションと直前のイテレーションをまとめる, という方法で固定した.

2.4　本手法の活用方法

本手法は, アジャイル開発の実プロジェクトへの適用支援を目的としており, 利用者としてはイテレーション期間決定を行うプロジェクトマネージャを想定している. 主な利用方法として, プロジェクト開始前に使用しイテレーション期間決定の際の指標として役立てることができる. また, プロジェクト開始時に予測し得なかった事態の発生に伴いイテレーション期間を開発途中に変更する必要性が生じた場合について当該時点までのプロジェクトの特性および新たな状況を考慮してシミュレーションし現状からの適切なイテレーション期間の推定にも役立てられると考えられる. さらに, プロジェクト終了後に振り返りとして使用し, 推定された適切なイテレーション期間と実際の期間との比較する方法が考えられる.

3.　シミュレーション結果と考察

我々は, モデルの妥当性を示すために複数の開発事例を取り上げシミュレートし, 開発後の振り返りや開発結果より定性的に推定される適切なイテレーション期間との比較を行った. また, モデルへの入力値を変化させることで様々な状況を仮想的に生み出し, その結果からプロジェクト要件と適切なイテレーション期間の関係を分析した. 誤差によるばらつきを抑えるため 1000 回シミュレートした結果の平均値を示している.

3.1　事 例 研 究

（株）永和システムマネジメントの XP 実践レポートの事例（以降：事例１）について考察する. 事例１は, アジャイル開発の評価するために擬似的に作成されたプロジェクトで, 詳細なプロジェクト要件に加えて, イテレーション期間の長さ, 開発後の振り返りなどについても公開している. 事例１では, 開発期間１ヶ月（２０日間）, イテレーション期間は５日間, 開発者は入社一年目の新人プログラマ１名, 入社３年程度の中堅のプログラマ２名, 開発経験豊富なプログラマ２名の計５名. 要求は７つのベースとなるストーリーが初期に定義され, 要求のうちいくつかについては, 顧客からのフィードバックを受けることで徐々に具体的な要件が定まったと報告されている. なお, 事例１では事後のシミュレーションであるため, 複雑性はストーリー名から依存関係を推測し, 変動性は開発中における要求変更, それぞれの要求の工数は開発にかかった工数から逆算する形で推測した. 要求の重要度は述べられていなかったため, ストーリー名からの推測によって抽出した. これらから抽出した入力値を表２に示し, その シミュレーション結果を図５に示す. 事例１では評価の際に重視される

表 2　事例報告１における入力値

名前	値
開発期間	20
変動性	10%
複雑性	50%
開発チーム	0.25, 1, 1, 2.5, 2.5（５人）
要求（工数）	10, 10, 15, 10, 10, 10, 5 人日
要求（重要度）	0.3, 0.6, 1, 0.3, 0.6, 0.6, 1

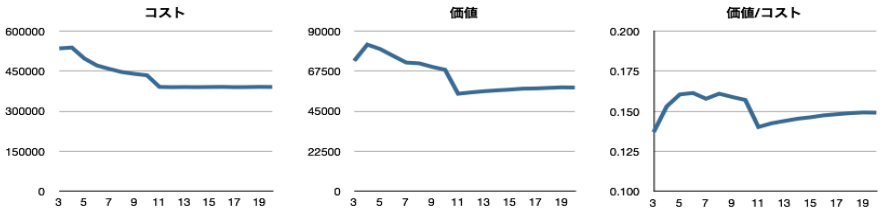

図 5　事例１シミュレーション結果（横軸：イテレーション期間, 縦軸：価値, コスト, 価値 / コスト）

評価軸が特に明記されていないため, コストパフォーマンスを表す, 価値/コストによる評価を行う. 図５の価値/コストを参照すると本手法の結果としては, ５日～９日のイテレーション期間において, 値が高いことが見て取れる. 実際のプロジェクトと比較すると, 事例１は成功をおさめたという報告がなされているが, 事後に行われた開発者の振り返りでは「イテレーション期間が少し短かった」という意見が出ている. この意見は本手法の５日～９日程度のイテレーション期間が適切であるという推定と一致するため, 本手法がこの事例について妥当に機能したと考えられる.

3.2　プロジェクト要件と適切なイテレーション期間の関係

プロジェクト要件の中で特に変動性と複雑性に関して, それらを変化させた時に適切なイテレーション期間がどのように変化するかについてシミュレーションを行い, その結果について考察する. 開発期間：６０日, 開発者：５人（0.25, 1, 1, 1, 2.5）, 要求数：３０という値を固定し, 変動性および複雑性のパラメータを変化させる事で仮想上で様々な状況を生み出し, それぞれにおける適切なイテレーション期間の変化を分析した. その結果を得られたデータを図６に示す.　図８は各複雑性において, 変動性を変化させた時にその適切なイテレーション期間がどのように推移したかを表している.

まずそれぞれのグラフを見ると複雑性の大きさに関わらず, 変動性が高いプロジェクトに

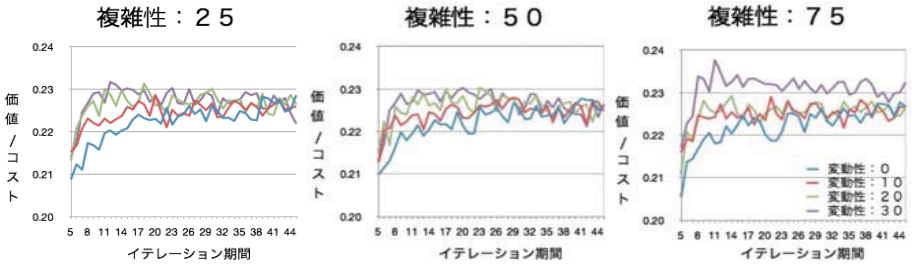

図 6　プロジェクト要件と適切なイテレーション期間の関係

おいては短いイテレーション期間が適切であり, 変動性が低い場合は長いイテレーション期間が適切であるという結果が示されている. この結果は定性的に言われている見解と一致する. 次に３つのグラフを比較することで複雑性について考察する. まず, 複雑性が高いものは比較的に, 長いイテレーションにおいて高い値をとっている. これは, 短いイテレーション期間では要求間の依存関係が多い場合に, それらを別々に開発しなければならない状況が発生し, 統合のコストが増えてしまうことに起因すると推測される. 複雑性とイテレーション期間の関係について定性的に述べられている文献は少ないため, 新たな関係性を導きだすことに成功したと考えられる.

4. おわりに

本稿ではアジャイル開発の実プロジェクトへの適用支援を目的とし, アジャイル開発の特にイテレーション期間に注目したシミュレーションを行った.

本手法の貢献は, 実プロジェクトをシミュレートすることでプロジェクトごとの適切なイテレーション期間を推定し, 事例不足などから適用範囲が限られているアジャイル開発の適用支援を行ったこと. また, モデル上でパラメータを変化させることで様々な状況を仮想的に生み出し, それらをシミュレートすることで, プロジェクト制約である変動性, 複雑性とイテレーション期間の関係を定量的に分析したことである.

今後の展望として, 様々なプロジェクト制約における有効性を検証するために, クラス図やアクティビティ図などを入力とし, プロジェクト条件からの入力値抽出を容易にすることが挙げられる. また, 非アジャイル開発で多用されるファンクションポイント法といった開発プロセス・プロジェクトの見積もりや測定を, 本手法に組み入れる可能性も検討したい.

参　考　文　献

1) 独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター ：非ウォーターフォール型開発に関する調査　調査報告書, 2009 情財第 0507 号
2) Dan Port, Alexy Olkov：”Using Simulation to Investigate Requirements Prioritization Strategies.”, 2008 Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering
3) 天野勝：アジャイル開発の進め方, http://www.slideshare.net/esmsec/ss-5656398
4) 服部哲, 落水浩一郎：確率ペトリネットによる組織パターンの検証, コンピュータソフトウェア Vol. 23（2006), No. 1
5) Craig Larman：初めてのアジャイル開発，日系 BP 社 (2004), 越智典子訳
6) 池上俊也 ：成功するアジャイル, 失敗するアジャイル, 日経 SYSTEMS(2010/12/21), http://itpro.nikkeibp.co.jp/article/COLUMN/20101215/355245/
7) 設楽秀輔：アジャイル トランスペアレンシー ～アジャイル開発における透明性の確保について～, http://gihyo.jp/dev/serial/01/agile-transparency/0002, 2009/11/10
8) Pekka Abrahamsson, Michele Marchesi and Giancarlo Succi ：Extreme Programming and Agile Processes in Software Engineering: 7th International Conference, XP 2006, Oulu, Finland, June 17-22, 2006, Proceedings , Springer July 26, 2006
9) Ken Schwabe and Mike Beedle ：Agile Software Development with Scrum, Prentice Hall October 21, 2001
10) Boehm, B, and Papaccio, P. 1988：”Understanding and Controlling Software Costs.”, IEEE Transactions on Software Engineering, Oct. 1988.
11) Jones, C. 2000.：”Software Assessments, Benchmarks, and Best Practices.”,Addison-Wesley.
12) 玉井哲雄：ソフトウェア工学の基礎, 岩波書店（2004）
13) Lan Cao, Balasubramaniam Ramesh, and Tarek Abdel-Hamid. 2010. Modeling dynamics in agile software development. ACM Trans. Manage. Inf. Syst. 1, 1, Article 5 (December 2010), 26 pages.
14) H. Sackman, W. J. Erikson, E. E. Grant：“Exploratory experimental studies comparing online and offline programming performance”,Communications of the ACM Volume 11 , Issue 1 1968
15) Tom DeMarco, Timothy Lister：”Peopleware”, 日経 BP 社 (2001)
16) 中谷多哉子, 近藤城史, 他：要求の依存関係に基づいた要求充足度の観測と管理に関する調査研究, SSR 平成 21 年度研究報告