ソフトウェア工学 135－13
（２００１．１１．２２）

ユースケースを使ったユーザのための進捗管理

吉田　享子　　永田　守男
平成国際大学　　慶應義塾大学

これまでのソフトウェア開発の進捗管理は、開発者のために、開発者の視点でなされたものであった。特に実装工程では、モジュールまたはクラスのコーディング・テストによって進捗報告がなされている。しかし、これらのものでユーザが進捗度を理解することは難しい。オブジェクト指向では、ユーザにも開発者にも理解できるユースケースによって、工程の一貫性を保つことが唱えられている。本研究では、実装工程の進捗管理において、ユースケースを使って、開発者の作業を増加させることなく、ユーザが理解できる進捗管理システムを提案する。分析時にユースケースから認識したクラス・メソッドを、実装時のクラス・メソッドと関係づけることでこれを実現した。

A Progress Control System for End-User Using Use Cases

Kyoko Yoshida　　Morio Nagata
Heisei International University　　Keio University

Progress control for the software development project has been done for the developers with their points of view. Especially in implementation phase, the progress reports of coding and testing have been made using modules or classes. However, it is not easy for users to understand those reports. In object oriented method, it is advocated to keep the consistency of the process by using use cases that are understandable for users and the developers. This research proposes a progress control system using use cases for implementation phase, which is easy to catch for user and doesn't increase the developer's burdens. It was realized by linking classes and methods in implementation phase with them in analysis phase.

１．はじめに

ソフトウェアの開発におけるプロジェクトマネージメントは成功例よりも失敗例の多さで語られる。開発スタート時に計画されたコスト、スケジュール、品質の目標を満たすことのできるプロジェクトは少ない。米国においてもソフトウェアプロジェクトの３分の１から３分の２が、スケジュールと予算を超過すると言われている（[1]）。開発途中でスケージュールの遅れに直面したプロジェクトは、工期と費用の再見積もりがされるが、それもまた予定通りにはいかないという場合が多い。現在でも多くのプロジェクトが当初の予想以

上の労力と時間をかけざるを得ない状態に陥っている。

工期の決定は、顧客やユーザの要望や、ソフトウェアを開発する会社の都合によるものであるが、これらのステークホルダーと、実際にソフトウェアを開発する技術者との開発工程における理解の齟齬にも原因の一端があると思われる。開発が始まると、技術者は目前のソフトウェア開発にのめりこまざるを得ない状況に陥る。

一方、実際にシステム開発を発注する顧客（ここでは、ユーザと同じとする）には、特にシステム分析以降の開発工程はブラックボックスになってしまう。ユーザにも分かる方法でソフトウェア工程を可視化して、ソフトウェア作成の難しさを正確に理解してもらう必要がある。また、開発チームはユーザに対してすべての工程において進捗状況を説明する責任がある。本研究では、この点に留意し、ユーザと開発者が共有できる進捗管理システムの開発を提案する。

2．背景と目的

2．1　今までの進捗管理の問題点

ソフトウェア開発における進捗管理は、ユーザの要求仕様に基づいたシステムを、納期通りに、予算内で作成することを目標とし、この目的を実現・達成するためにソフトウェアの開発工程を管理することである。これは、今までは主としてシステムを開発する立場からの進捗管理であり、ユーザの立場からの視点はほとんど含まれていない。

実際に、システムの分析・設計・実装・テストと流れる開発工程の中で、特に設計以降の工程においては、開発の進捗の実質的主導権は開発者のものになる。ユーザに進捗状況の説明があっても、構造化開発ではモジュールや関数ごとの、オブジェクト指向の場合ではクラス単位の進捗説明がなされる。ユーザには、モジュールやクラス単位の報告では、実際の工程の進み具合を理解することは難しい。ユーザは、不安感をもちつつ開発工程を見守るしかすべがない。実際にどの程度までソフトウェアは完成しているのかを把握するためには、システムの要求仕様と設計・実装・テストの工程を仕様と結び付けたユーザが理解できる進捗管理が必要である。

また、開発工程の途中で工程が遅れることはよくあることである。この場合、技術者を増員したり、作業時間を増やしたりして対応するが、ユーザへの報告は後回しになることが多い。特に実装段階では、開発者の視点で見ると、クラスやメソッドはひとつひとつ同じ重さを持つソフトウェアであり、ひとつでも欠けるとシステム自体が動かないと考えてしまう。しかしユーザから見た場合、運用で回避できる仕様もある。分析の段階では、非常にまれにしか起こらないケースと頻繁に起こるケースを、同じレベルで要求することも多い。この場合、必要な部分と後回しにできる部分との切り分けを再度行う必要が生じる。この時点で、ユーザと開発者の視点の違いを解消し、スムーズな話し合いを助けるソフトウェアが必要になる。

一方、オブジェクト指向開発においては、反復的でインクリメンタルな開発が提唱されている。これは、ウォーターフォール型よりも一層進捗管理が複雑になる面がある。特に反復的な開発では、遅れだすとユーザも開発者も先が見えない不安を抱えることになる。現在の遅れがその後の作業に与える影響を測るのが難しく、精神的な余裕もなくなる。優先度の高い仕様が積み残されて後回しにされたり、リスクが背後に隠される危険性が、工程が明確に区分されているウォーターフォール型よりも大きい。遅れの原因と対応については、開発者だけではなくユーザも関与して以降の対策を講じる必要がある。

2．2　研究の全体像

ソフトウェア開発工程の分析過程では、ユーザ

と開発者はユースケース(シナリオ)を使用して、システムが提供しなければならない機能を確認する。設計工程に入ると、コンピュータ上のアーキテクチャの問題や実装するための技術的な部分が大きくなるため、その作業内容はユーザの理解が届かないものになる。開発者は設計工程以降に各工程に対応した単位で開発をする。たとえば実装段階では、クラスやメソッドの単位で作業をとらえるようになる。しかし、ユーザの理解はユースケースに留まったままである。

この隔たりを埋めるためには、ユースケースとクラスやメソッドを関係付け、実装段階でクラスやメソッドからユースケースに逆に翻訳して、ユーザに表示することが有効であると考えた。今回は、ユーザと開発者が共通に理解できる要求分析過程での成果物であるユースケースを使用し、開発者には今までと同じクラスやメソッドで、ユーザにはユースケースで進捗管理を扱えるシステムを提案する。

3．提案システムの概要

3．1　システムの前提

ユースケース自体は、オブジェクト指向にも構造化技法においても使用できるが、今回は、オブジェクト指向での開発を前提としている。特にソフトウェア開発工程の中では最も工数がかかり、開発の遅れに大きく影響を与える実装およびテスト工程の進捗管理を扱った。

UML による統一ソフトウェア開発プロセスでは、反復的でインクリメンタルな開発が唱えられている。今回は、その１サイクル目の実装とテストの工程を考え、反復的でインクリメンタルな開発に及ぼす影響については、この研究の延長として考える。規模としては、数ヶ月・数人で作業するソフトウェア開発を考えている。また、開発者が行っている進捗管理の視点を変えてユーザ用に作成することで、開発者に追加の作業を課すことは避けた。

3．2　システムの流れ

提案するシステムは、開発工程のすべてにおいてユーザにはユースケースを単位として、開発者には工程に対応した単位で進捗管理を行うシステムである。システムは、次の流れ（図１）になる。

(1) ユーザと開発者はユースケースを作成する
(2) システムが、ユースケースの文章からシーケンス図候補を自動生成する
(3) ユーザと開発者は、シーケンス図を使って文章タイプごとにクラスとメソッドをあてはめる
(4) 開発者は実装のためのクラスの汎化、分解、合体、メソッドの追加を行う
(5) 開発者はメソッドごとの見積もり工数を入力する
(6) 開発者はメソッドごとの進捗を入力する
(7) ユーザ用（ユースケース）と開発者用（クラス・メソッド）に対して、それぞれに分かりやすい進捗資料を提示する

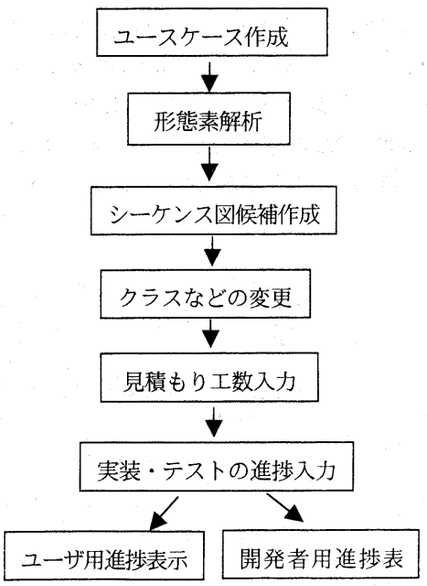

図１　システムの流れ

ユーザと開発者は、ユースケースを作成しながら、要求仕様を確認する。今回は、ユースケースは単文で作成し、詳細なものを作成する。

ここでユーザは、ユースケースの開発優先度も入力できる。このユースケースを解析して、システムはシーケンス図を自動生成する。これを基に、ユーザと開発者はシステム分析を更に進め、1つのユースケース文に対して、2つのクラスまたはオブジェクトとその間のメソッド（メッセージ、イベント）を確定し、結びつける。ここで作成されるのは、ユーザが問題領域を分析する過程で認識できるクラスである。

設計から実装工程で、開発者はこのクラスからアーキテクチャやプログラミング上の問題を考慮した実装のクラスを設計する必要がある。そのためにクラス設計の画面を使って、メソッドの追加や、クラスからスーパークラスを抽出する汎化、合体、分解を行って、実際に実装するクラスとメソッドを設計する。実装用のクラスの設計終了後、開発者は、メソッドごとに工数を見積もって入力する。

実装工程に入ったあとは、一定期間ごとに、メソッドごとのコーディングとテストの進捗状況を入力する。進捗のパーセンテージには、データ収集の簡素化のため限られた進捗（例えば、30%、70%、90%、100%）のみ採用することが多い。進捗測定に絶対的な客観性を持たせるという観点から0%と100%（終了か否か）の2値のみ採用する場合もある [2]。今回は、作成が完全に終了したことを確認するため0%か100%を使用する。ここから自動的にユーザ用のユースケースを単位とした進捗と、開発者用の進捗を表示する。

4．ユースケースの解析

今回使用したユースケースは、Craig Larmanが提唱しているもの[3]で、図2の形式である。

ユースケースの主たる部分はイベントを順序だてて記述したものである。最初の文章では、このユースケースの事前条件を記述する。次からは、「アクタの行動」と「システムの応答」が、番号を付けて順番に記述される。「アクタの行動」においてはアクタが主語となり、“システムに対するアクション”と“他のアクタへのアクション”が記述される。「システムの応答」においてはシステムが主語となり、“他のシステムプロセスに対するアクション”と“アクタへのアクション”が記述される。また、アクタは、ユースケースの「アクタ：」部分に記述されているものとする。

ユースケースのテキストファイルを入力として、茶筌[4]を通して形態素分析を行った。その結果からイベントの典型的な順序の「アクタの行動」と「システムの応答」の文章をイベントの順番に従っ

ユースケース：	ユースケース名	
アクタ：	アクタのリスト	
目的：	ユースケースの目的	
概要：	要約情報	
分類：	1．プライマリ,セカンダリ,またはオプショナル 2．本質的または現実的	
クロスリファレンス：	関連するユースケースとシステムの機能	
イベントの典型的な順序		
アクタの行動		システムの応答
番号を付けたアクタの行動		番号を付けたシステムの応答の説明
イベントの別の順序 行番号：例外を記述		

図2　拡張書式の形式

て解析した。

イベントの典型的な順序は自然言語で記述されるが、上のような記述内容のため文章としては図2のパターンに集約される。ここでは、制御に関する文章は対象としていない。実例（[3]など）を調べた結果もほとんどこのタイプに分類された。このタイプに含まれないものについては、文章自体が意味を明確に伝えていない場合が多く、仕様書としても曖昧さを残しているものと考えられた。各文章には1対1でクラスまたはオブジェクトとメソッドを対応させた。これらは候補として作成されたもので、ユーザと開発者が再度検討を加え、変更する。

	文章タイプ	対応シーケンス図
アクタ	AがCをαする	A Cをαする
	AがCをEにαする	A　E Cをαする
システム	BからCをαする	B　C αする
	CをEにαする	C　E αする

図3　文章タイプと対応するシーケンス図

5．クラスの変更とユースケースへの進捗の対応付け

シーケンス図からクラスへのつながりは次のように考えた。一般にクラスに送信されているすべてのメッセージが、そのクラスで定義されるべきメソッドの大部分と考えられる[3]。ただし、クラスからアクタへのメソッド（メッセージ）はクラスで責任を負うべきメソッドとした。この前提で、シーケンス図からクラスとメソッドを定義し、ユースケース文と関係付ける。このクラスをここでは、Userクラスと呼ぶ。設計でのクラスと分けて型（タイプ）と呼ぶ文献もある。

分析段階以降に行われるクラスの再構築については、メソッドの追加に加え、今回は以下の変換に対応させた。この変更後のクラスを、実装クラスとする。ここでの変更を以下のように整理した（図4参照）

(1)複数クラスからスーパークラスの作成
(2)複数クラスを合体して、1つのクラスを作成
(3)クラスの複数クラスへの分解
(4)クラスのメソッドから複数クラスのメソッドへの分解
(5)クラスのメソッドの追加

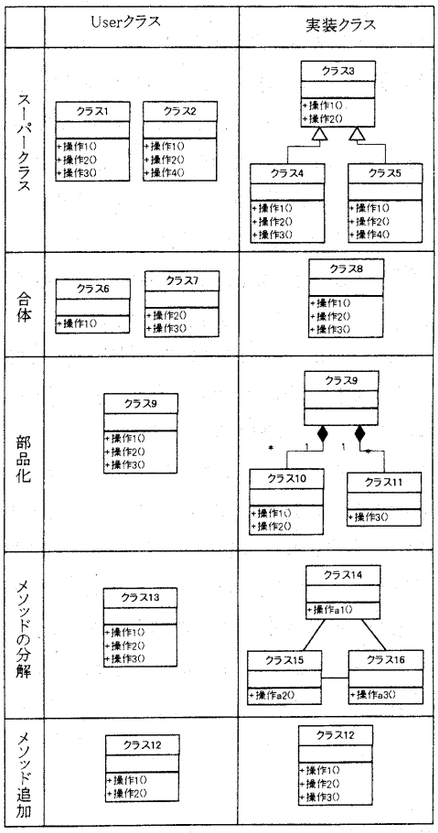

図4　クラスの変更

6．進捗管理のための出力

進捗状況を測るための出力としては、以下のものを作成する。

開発者用

(1)クラス単位でのコーディング進捗状況

(2)クラス単位でのテスト進捗状況

ユーザ用

(1)ユースケース文に変換されたコーディングとテストの進捗状況（図6）

(2) ユースケース単位のコーディング進捗状況のグラフ

(3) ユースケース単位のテスト進捗状況のグラフ

User クラスから実装クラスへの進捗計算の変換アルゴリズム

ユースケースのひとつの文に、ひとつの User クラスが対応する。また、ひとつの User クラスのメソッドは、実装クラスのメソッドが複数組合わされたものになる。実装クラスへの変更においてメソッドの分解が行われた場合は、変更前の User クラスのメソッドが、実装クラスのメソッドに分解されることになる。この場合は、変更後の複数のメソッドによって実装が分担される。よって、それぞれのメソッドを構成する割合（ここでは、構成割合という）が必要になる。また、進捗を管理する上で、実装クラスのメソッドごとの工数見積、進捗実績が必要になる。

工数見積もりは、開発者が実装クラスを決めた時点で入力する。進捗実績は、実装とテストの段階で開発者が入力する。今回は、0%か100%を使用したため、進捗実績は0か1の値をとる。1つのユースケース文に対して図5のような構成のデータをコーディングとテストそれぞれに対して作成する。

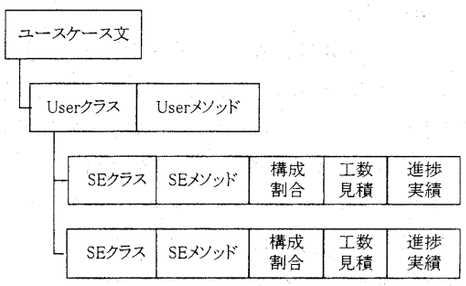

図5 UserクラスとSEクラスの関係

User クラスの、コーディングやテストの進捗計算は次の式で求められる。

$$\text{UC文の進捗}(\%) = \frac{\Sigma(\text{工数見積} \times \text{進捗実績} \times \text{構成割合})}{\Sigma(\text{工数見積} \times \text{構成割合})} \times 100$$

ここで求めた値を時系列に従って、ユーザに分かりやすい形で表示する。この例を図6に示す。

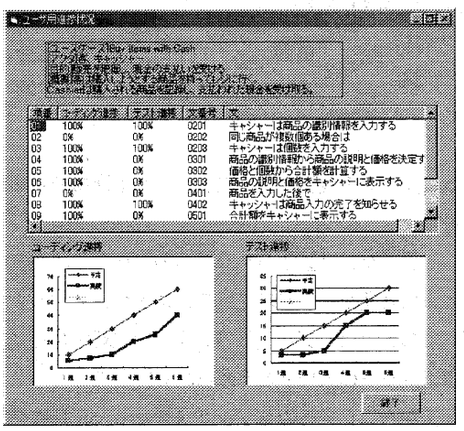

図6 ユーザ用画面

7．試作システムについて

ユースケースから文章のタイプ分けをしてクラスとメソッドの候補を挙げ、シーケンス図を作成した。その上で、ユースケースの文章とクラスとメソッドを結び付け、実装工程までのクラス変更にも対応した。実装工程の進捗を開発者にはメソッドレベルで入力してもらい、ユーザにはユースケースレベルで各種の進捗状況を表示した。実装工程の進捗を開発者に余分な工数をかけることなく、ユーザ用に可視化した。作成した図の例を図7に示す。

クラスの変更については、実際にはクラスの合体や分解のあとスーパークラスの作成が行われたり、スーパークラスのスーパークラスを作成したりする。今回のシステムでは、変更を複雑に組み合わせることは考慮していない。より実用レベルに近づけるためには、クラスの汎化・合体・分解等を自由に行えるようにする必要がある。また今回は、多相性についても扱っていない。一方 GUI の面でも、クラス図を見ながら変更を行えるような仕組みを加えるとより使いやすいシステムになると思われる。

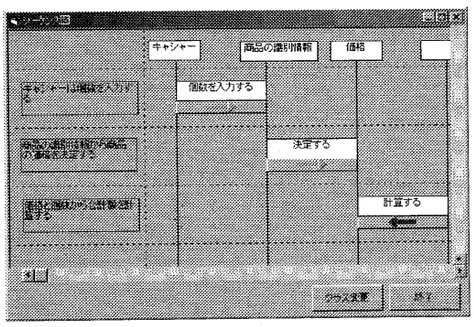

図7　シーケンス図画面

8．結論

オブジェクト指向が提唱している反復的でインクリメンタルな開発は、一方で進捗管理が難しい面をもつ。このシステムは今後、反復的でインクリメンタルな開発でも使用できるように改善したい。また、複数のユースケースは include や extend の関係で結ばれるものもある。これらのユースケースへの応用も今後の課題である。

一方、ユースケースの優先度を利用して、開発の順番をコントロールすることが薦められている（[5]）。ユーザにユースケースを使用して優先度を入力できるようにして、実装工程の途中でもユーザの希望を取り込むことができるようにしたい。

また、ユースケースでシーケンス図を作成する方法は、いろいろと研究されている（[6]）。本研究で提案した方法の最初の部分でこれらの成果を利用することによって、より正確なユースケース図を自動的に作成することが可能になる。そうなれば、クラスなどを変更する手間を軽減できることが期待される。

オブジェクト指向では、UML による統一ソフトウェア開発プロセスでもユースケース駆動が言われており、ユースケースの各開発工程に果たす役割は大きい。各工程でのユースケースの利用について今後も研究していく予定である。

9．参考文献

[1] Steve McConnell : Software Project Survival Guide． 1998（日本語訳）アルテア・ジャパン訳「ソフトウェアプロジェクト　サバイバルガイド」日経 BP ソフトプレス 1998

[2] Project Management Institute: A Guide to the Project Management Body of Knowledge 1996（日本語訳）「プロジェクトマネジメントの基本知識体系」 財団法人　エンジニアリング振興協会訳 1997

[3] Craig Larman : Applying UML and Patterns An Introduction to Object-Oriented Analysis and Design. Prentice Hall, 1998（日本語訳）今野睦＋依田智夫監訳　依田光江訳 「実践 UML　パターンによるオブジェクト指向 ガイド」プレンティスホール、

1999
[4] 奈良先端科学技術大学院大学情報科学研究科 日本語形態素解析システム「茶筌」 http://chasen.aist-nara.ac.jp/index.html.ja
[5]Ivar Jacobson,Grady Booch, James Rumbaug : The Unified Software Development Process. 1999 (日本語訳) 藤井拓監修 日本ラショナルソフトウェア株式会社訳 「UML による統一ソフトウェア開発プロセス」翔泳社 2000
[6] 矢後友和、原田実 「日本語要求仕様文章からオブジェクト指向による動的モデルを生成するCAMEO/D の開発」 情報処理学会第 62 回全国大会特別トラック(4)講演論文集,特 4-95-96 (2001.3)
[7] Paul Harmon Brian Sawyer : UML for Visual Basic 6.0 Developers. 1999 (日本語訳) オージス総研監訳 「VBプログラマのための UML」翔泳社 1999
[8] Walker Royce : Software Project Management : A unified Framework Addison Wesley LongMan. 1998 (日本語訳) 日本ラショナルソフトウェア株式会社監訳 「ソフトウェアプロジェクト管理」アジソン・ウェスレイ・パブリシャーズ・ジャパン 1999
[9] Philippe Kruchten : The Rational Unified Process. 1999 (日本語訳) 藤井拓監訳 日本ラショナルソフトウェア訳 「ラショナル統一プロセス入門」ピアソン・エデュケーション 1999