ソフトウェア工学 96−12
（1994．1．21）

サービス仕様の関係に基づいたインクリメンタル開発プロセス導出法

金井 敦, 市川 晴久

NTT ソフトウェア研究所

仕様を複数の部分仕様に分割しその部分仕様を分散してインクリメンタルに開発することにより，ソフトウェアの開発期間を短縮する開発方法論を提案する．本方法論では，部分仕様間の関係をフォーマルに表現し，その関係から開発プロセスを導く．あたえられた関係を満たすように仕様を分割すると，無作為に分割した場合に比較して手戻りの発生や無駄な試験を無くし効率的な開発が可能となる．また，部分仕様間の関係からリソースなどの制約を考慮する前の開発プロセス（弱開発プロセス）を導くルールも与える．本方法論で求めた開発プロセスはウオータフォール型開発プロセスよりも開発期間が短くなることを示すとともに，PBX サービスの場合の例を示す．

Incremental Development Methodology based on Relationships between Service Specifications

Atsushi Kanai and Haruhisa Ichikawa

NTT Software Laboratories

This paper proposes a software development methodology that decreases development time by dividing the requirement specifications into several sub-specifications, which are then distributedly and incrementally developed. The relationships between the sub-specifications are formally expressed, and the development process is derived on the basis of those relationships. Backtracking or useless testing is avoided, and thus software is developed more efficiently when specification division is based on this methodology than with random division of the specifications. Rules are described for dividing the development process from the sub-specification relationships without considering the resources (weak development process). Estimation shows that development using this methodology takes less time than that using the waterfall model, and an example of PBX service is introduced.

1 はじめに

ソフトウェアの開発期間の短縮を目的としてさまざまな開発法が提案されている．独立に設計製造できるモジュールに分割することによる並行開発や要求定義期間の短縮のためのプロトタイピングは従来から試みられている．また，段階的に開発することによりライフサイクルトータルとして信頼性の高いソフトウェアをより短期に提供するための手法としてスパイラルモデル [1] や多くのインクリメンタル開発法 [2, 3, 4] が提案されている．しかし，どのような場合にどのようなインクリメンタル開発法を適用すれば良いか明確な指針がなかった．また，これまでの方法は開発プロセスを完全に決定してから開発を開始するというもので仕様のインクリメンタルな設計手順と開発プロセスの実行関係が明確ではなかった．その中で，仕様決定の順序関係に基づいて開発期間を短縮しようという試みもある [7] が，この方式は特定の条件でのみ使用可能な方式であり一般的な方法ではなかった．

上記の問題点が健在化する例として，PBX のサービスのように例えば，ダイヤル発信，短縮ダイヤル，再ダイヤルといった数百のサービスの集合としてアプリケーションプログラムを開発しなければならない場合がある [?]．この場合，しらみつぶしに無作為的にそれらのサービスを開発していた場合には，上記の従来の方法では，

以下の問題があり効率的な開発ができない．

1. 仕様設計に手戻りが発生
   独立に設計したサービス間で仕様を合わせなければならない場合（使い勝手をそろえるためなどで）に手戻りが大きい．
2. 的確な要員・開発期間見積り配置が不可能
   要員を多く揃えたとしても，上記問題などから開発期間や開発稼働を減らすことはできない．このため適正な要員数や開発に要する期間を見積もることができない．
3. 開発期間の遅延
   数百のサービスをすべて決定するのには時間がかかるため，完全に仕様を決定してから製造に入るウオータフォール型の開発の場合には，開発期間の短縮が難しい．

現状は，経験的にサービスを幾つかの独立と思われる部分に分類し，すこしでも対象を小さくすることにより上記問題の影響を小さくすることが行なわれていた．しかし，この分類は経験に頼っており体系的には行なわれていなかった．

このような場合は各サービス仕様の関係（仕様の重複度合や決定の順序関係）を明確化しそれに基づいて，重複開発を防ぐと共に，個々の仕様の開発順番を旨く配置することにより上記問題点を解決することにより開発時間を短縮し要員の見積もりを的確にすることが可能となる．しかし，この場合に，膨大なサービスの全ての関係を一度に決定するのは要求定義および設計工程が不必要に伸びる危険性がある．そこで，全体の開発順番を確定してから開発を開始するのではなく，徐々に仕様の関係を明確化しながらそれに即して開発プロセスを詳細化し，開発そのものを進めていく方法が有効と考えられる．また，多数のサービスの関係を漠然と定義していたのでは，効率や開発管理の観点から問題があるため，仕様の関係をフォーマルや開発プロセスをフォーマルに記述することが重要であるといわれている [5, 6]．さらに膨大な数のサービス仕様の関係から開発プロセスを自動的に導出する基礎としても，仕様の関係や開発プロセスをフォーマルに記述する必要がある．

本論文では，メッセージシーケンス (MSC) をベースに個々のサービス仕様を設計しそれらの合成として完全なサービスを実現する開発方式の場合 [12] について，上記のような状況で仕様の関係をインクリメンタルに詳細化しながらその仕様から設計，製造，試験の各アクティビティの実行順序を得ることにより効率的な開発を実現する方法を提案する．

2 本開発方式の考え方

2.1 前提とする開発環境

本開発法では仕様を分割し分離して開発するため，従来の言語や開発環境を前提にすると同一ソースプログラム内に各アクティビティの成果を混在してコーディングしなければならない場合が多く，必ずしも有効に仕様が分離できるとは限らない．このため，ここでは，主に通信ソフトウェアを対象としたソフトウェア開発環境 SDE[12] の機能を前提にする．しかし，ツールなどの支援がなくても本開発法が適用できる場合もあるため，開発ツールの存在は本開発法には本質的な問題ではない．

SDE では，プロトコル設計時のシーケンス合成機能として，通常の言語ではサポートされていないメッセージシーケンス（コントロールフロー）を合成する機能が提供されている．この機能は，もともとはプロトコル設計時に個別のメッセージシーケンスを設計しておいて，それらを合成することにより一つの完全なプロトコルとするために使用される機能である．この機能は，一般のプログラム開発にも応用可能であり，本開発法では，通常のサブルーチンのような分割のみでなく，上記のような分割方法をも前提とする．

2.2 概要

ここでは，PBX のアプリケーションプログラムの開発のように，例えば，ダイヤル発信，短縮ダイヤル，再ダイヤルといった多くの個別のサービス仕様を決定しその個別仕様の合成として完全なサービスを開発する場合を前提としている．この場合，個別の仕様が独立でかつソフトウェアの物理的構造も独立ならば，それぞれのサービス仕様を勝手に決定し勝手に開発すれば問題はない．しかし，MMI を同じコンセプトにしたい，優先仕様をベースに次の仕様を決定したいなどの理由でサービス仕様の決定を独立に決定したくない場合がある．また，すでに決定した仕様の一部を再利用して仕様を決定したい場合あるいはその仕様に対する試験を早くやりたい場合などが実際の場面ではある．このような状況は以下に分類することができる．ここでは，短縮ダイヤルといった個別のサービスを個別仕様，個別サービスの集合としての PBX のサービスを全体仕様と呼ぶ．

1. すでに決定されている個別仕様の一部を流用あるいはその個別仕様に依存して新しい個別仕様を決定する場合．
2. 未決定の個別仕様同士をお互いを考慮しながら決定する場合．
3. ある個別仕様が決定されなければ次の個別仕様が決定できない場合．
4. 個別仕様同士が全く独立に決定できる場合．
5. 4 の場合であるが，開発リソースの関係から同時に開発できず，特定の個別仕様を優先して開発したい場合．
6. 個別仕様そのものかあるいは個別仕様の関係そのものが明確になっていない場合．

上記の場合の個別仕様の関係として，1 および 2 の場合を優先決定関係，3 の場合を同時決定関係．4 の場合を独立決定関係，5 の場合を弱い優先関係と呼ぶ．6 の場合を未決定関係と呼ぶ．

本方式では，開発プロセスの導出仮定を図 1 に示すようにモデル化する．まず，ユーザの要求などから個別仕様としてどのようなものがあるかを明確にする．その個別仕様から上に示した個別仕様の関係 (SR) と仕様をインプリメントした場合の物理的な関係 (PR)（プログラムの重なり関係）を明確化する．それに基づいて，アクティビティの実行関係の集合である弱開発プロセス (DP) を導出する．ここで導出された DP はまだ完全に開発に必要な時間的手順レベルにはなっていないため弱開発プロセスと呼ぶ．例えば，独立に実行できるアクティビティは弱開発プロセスでは現実にシリアライズされて実行されるかどうかに関わらず単に独立に実行できるという表現となっている．例えば独立に実行できるアクティビティであっても一度に一つのアクティビティしか実行できない場合は時間的にシリアライズする必要があるといったように，リソースの制約に基づいて最終的な開発プロセス（実開発プロセス）が弱開発プロセスをもとに導出される．本論文では，個別仕様が与えられたところから弱開発プロセスの導出までを対象としている．リソースの制約やソフトウェアの構造などから実開発プロセスを求めるところは様々な戦略が [9, 10, 11] 提案されているが，ここでは直接の対象としない．

上記のモデルは，すべての仕様の関係が一度に与えられる場合であるが，上記モデルの発展型として，図 2 に示すように仕様関係の詳細化を段階的に行ないそれにともない開発プロセスも段階的に導出して実行可能なアクティビティから順次実行していくモデルも実際の局面では有効である．

本方式では，開発プロセスを決定するため，上記に示した個別仕様間の依存関係付けをしておく必要がある．本開発法の標準的な開発手順を以下に示す．ここでは，開発に必要なプロセスが要求定義，設計，製造，試験のアクティビティからなるものとする．

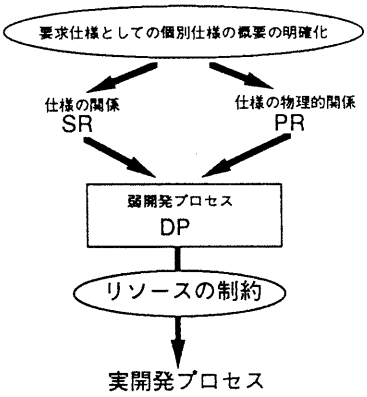


図 1: 開発プロセス導出モデル

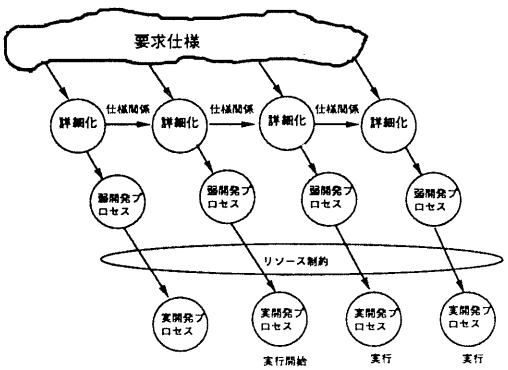


図 2: 段階的開発プロセス導出モデル

決定できる個別仕様（完全なサービスを構成するすべての個別仕様である必要はない）から順に，以下の手順で関係付けを行なう．この手順は個別仕様ごとに並行に行なわれ，実行可能なアクティビティ生まれるたびに，アクティビティの実行を開始し，すべてのアクティビィの実行を終了した時点で開発が終了となる．

本論文では，個別仕様を MSC で表現することを前提としているため，個別仕様に対応する MSC の上で重複があるかないかが物理的重複関係となる．一般には，モジュールの共有などにより物理的重複関係が発生する．

このような開発方法をとることにより，1章で述べた 問題点を以下の様に解決できる．

1. 仕様設計に手戻りが発生
   部分仕様間の関係を明確にしてから開発することにより手戻りの発生をおさえることができる．
2. 的確な要員・開発期間見積り配置が不可能
   部分仕様の関係から開発プロセスを導出するので，アクティビティごとにより細分化された的確な配置が可能となる．
3. 開発期間の遅延
   ウォータフォール型開発のように各工程を完全に終了させてから次工程に進むわけではなく，アクティビティ毎に実行可能なものから実行するので，開発期間をウォータフォール型開発に比較して短くすることができる．

3 準備

3.1 個別仕様関係の表現

個別仕様を s_i とする．全体仕様 Se はすべての個別仕様 s_i の合成＋として表現される．合成とは，シーケンスの合成やサブルーチンの組み込みなどにより二つの仕様を 一つのプログラムにすることを意味する．

$$Se = S_1 + S_2 + ... + S_n$$

定義 3.1

$$s_i \rightarrow s_j$$

s_i が決まらなければ s_j が決まらない．

定義 3.2 優先決定関係

$$s_i \Rightarrow s_j$$

$$(s_i \rightarrow s_j \quad and \quad s_j \underset{not}{\rightarrow} s_i)$$

s_i が決まらなければ s_j が決まらないが，s_j が決まらなくても s_i を決めることができる．

定義 3.3 同時決定関係

$$s_i \approx s_j$$

$$(s_i \rightarrow s_j \quad and \quad s_j \rightarrow s_i)$$

s_i が決まらなければ s_j が決まらず，s_j が決まらなくては s_i を決めることができない．

定義 3.4 独立決定関係

$$s_i \parallel s_j$$

$$(s_i \underset{not}{\rightarrow} s_j and s_j \underset{not}{\rightarrow} s_i)$$

s_i が決まらなくても s_j を決定でき，s_j が決まらなくも s_i を決めることができる．

定義 3.5 弱優先関係

$$s_i \trianglerighteq s_j$$

（$s_i \parallel s_j$ の場合の特殊な関係）s_i が決まらなくても s_j を決定でき，s_j が決まらなくも s_i を決めることができるが，別な理由で（例えば早く試験をしたいなど），もし s_i と s_i が同時に決定できない場合は s_i を優先する．

定義 3.6 未決定関係

$$s_i \Updownarrow s_j$$

仕様関係が明確になっていないため，$\Rightarrow, \approx, \| \trianglerighteq$ のどちらかに決定することができない．

複数の関係をまとめて記述する場合，以下の表現をする．

$r \in R1(R1 = \{\Rightarrow, \trianglerighteq\})$ の場合，$s_i \ r \ (s_j, s_k)$ という表現を許し，$s_i \ r \ s_j \ \ and \ \ s_i \ r \ s_k$ を意味する．

$r \in R2(R2 = \{\|, \approx\})$ の場合，$r(s_i, s_j, s_k)$ という表現を許し，$s_i \ r \ s_j \ \ and \ \ s_i \ r \ s_k$ を意味する．

定理 3.1 $s_i \approx s_j and s_i \Rightarrow s_k$ ならば $s_j \Rightarrow s_k$

3.2 物理的構造の重複関係

以下の演算子を定義する．

定義 3.7

$$s_i \cap s_j (\cap : S \times S \rightarrow P)$$

両仕様のシーケンスの一部が同一である部分のシーケンスの部分を抽出する演算子．ここで，S は個別仕様の集合，P は部分シーケンスの集合である．

定義 3.8

$$\bigcap_{i=1}^{n} s_i$$

または

$$\cap(S_1, S_2, ..., S_n)$$

仕様のシーケンスにすべて共通の部分シーケンスを抽出する演算子．

定義 3.9

$$s_i \ominus s_j = s_i - (s_i \cap s_j)$$

s_i から s_j との共通部分を取り去ったシーケンスである．

定理 3.2

$$(s_i \ominus s_j) + s_j = s_i + s_j$$

3.3 アクティビティの実行順序関係表現

開発アクティビティとして設計，製造，試験があるものとしそれぞれ，$D(s_i)$, $M(s_i)$, $T(s_i)$ と表現する．ここで，s_i はそのアクティビティで対象とする部分仕様である．上記アクティビティを代表して A と表現する．

定義 3.10 優先実行関係

$$A(s_i) \Rightarrow A(s_j)$$

は $A(s_i)$ の次に $A(s_j)$ を実行することを意味する．

定義 3.11 同時実行関係

$$A(s_i) \approx A(s_j)$$

は $A(s_i)$ と $A(s_j)$ が同時に実行されることを意味する．

定義 3.12 独立実行関係

$$A(s_i) \| A(s_j)$$

は $A(s_i)$ と $A(s_j)$ を独立に実行することを意味する．

定義 3.13 弱優先実行関係

$$A(s_i) \trianglerighteq A(s_j)$$

$A(s_i)$ と $A(s_j)$ は独立に実行できるが，独立に実行することができない場合は $A(s_i)$ を優先して実行する．

3.4 弱開発プロセス（DP）への変換ルール

弱開発プロセス（DP）を以下のように定義する．

定義 3.14 弱開発プロセス

$$DP = \{a \ \ r \ \ b | a, b \in A, a \neq b, r \in R\}$$

ここで，R はアクティビティの実行順序関係である．

設計アクティビティへの変換

ルール 3.1

$$if \ \ s_i \Rightarrow s_j \ \ then \ \ D(s_i) \Rightarrow D(s_j)$$

ルール 3.2

$$if \ \ s_i \approx s_j \ \ then \ \ D(s_i) \approx D(s_j)$$

ルール 3.3

$$if \ \ s_i \| s_j \ \ then \ \ D(s_i) \| D(s_j)$$

ルール 3.4

$$if \ \ s_i \Updownarrow s_j \ \ then \ \ D(s_i) \approx D(s_j)$$

ルール 3.5

$$if \ \ s_i \trianglerighteq s_j \ \ then \ \ D(s_i) \trianglerighteq D(s_j)$$

製造アクティビティへの変換

ルール 3.6

$$D(s_i) \Rightarrow M(s_i)$$

ルール 3.7

$$if \ \ s_i \cap s_j = \phi \ \ then \ \ M(s_i) \| M(s_j)$$

ルール 3.8

$$if \ \ s_k \Rightarrow s_i \ \ and \ \ s_k \Rightarrow s_j \ \ and \ \ (s_i \ominus s_k) \cap (s_j \ominus s_k) = \phi$$

$$then \ \ M(s_i) \| M(s_j)$$

ルール 3.9

$$if \ \ s_i \cap s_j \neq \phi \ \ and \ \ D(s_i) \Rightarrow D(s_j) \ \ then \ \ M(s_i) \Rightarrow M(s_j)$$

ルール 3.10

$$if \ \ s_i \cap s_j \neq \phi \ \ and \ \ D(s_i) \approx D(s_j) \ \ then \ \ M(s_i) \approx M(s_j)$$

ルール 3.11

$$if \ \ s_i \cap s_j \neq \phi \ \ and \ \ D(s_i) \| D(s_j) \ \ then \ \ M(s_i) \approx M(s_j)$$

ルール 3.12

$$if \ \ s_i \trianglerighteq s_j \ \ then \ \ M(s_i) \trianglerighteq M(s_j)$$

試験アクティビティへの変換

ルール 3.13

$$M(s_i) \Rightarrow T(s_i)$$

ルール 3.14

$$if \quad s_i \cap s_j = \phi \quad then \quad T(s_i) \,||\, T(s_j)$$

ルール 3.15

$$if \quad s_k \Rightarrow s_i \quad and \quad s_k \Rightarrow s_j \quad and \quad (s_i \ominus s_k) \cap (s_j \ominus s_k) = \phi$$
$$then \quad T(s_i) \,||\, T(s_j)$$

ルール 3.16

$$if \quad s_i \cap s_j \neq \phi \quad and \quad M(s_i) \Rightarrow M(s_j) \quad then \quad T(s_i) \Rightarrow T(s_j)$$

ルール 3.17

$$if \quad s_i \cap s_j \neq \phi \quad and \quad M(s_i) \approx M(s_j) \quad then \quad T(s_i) \approx T(s_j)$$

ルール 3.18

$$if \quad s_i \cap s_j \neq \phi \quad and \quad M(s_i) \,||\, M(s_j) \quad then \quad T(s_i) \approx T(s_j)$$

ルール 3.19

$$if \quad s_i \trianglerighteq s_j \quad then \quad T(s_i) \trianglerighteq T(s_j)$$

ここで，$\approx$ の関係にあるアクティビティは同時に実行しなければならないので，融合したアクティビティとなる．そこで，$A(S_i) \approx A(S_j)$ を $A(S_i, S_j)$ と記述する場合もある．

定理 3.3 $s_i \Rightarrow s_k$ のとき $M(s_k) = M(s_k \ominus s_i)$ および $T(s_k) = T(s_k \ominus s_i)$ が成り立つ．

4 詳細化方法と目標

要求仕様の詳細化が人間によって行なわれ，部分仕様が上記の関係式で表現されそれに基づいて開発プロセスが徐々に完成して行く．この時の詳細化の方法，その目標およびアクティビティへの変換方法を示す．なお，仕様そのものの詳細化はここでは扱わない．

4.1 詳細化の方法と目標

部分仕様間の関係を徐々に明確にし，その関係を用いて開発プロセスを導出する．詳細化する目標は，なるべく $\Updownarrow$ を減らし，$||$ と $\trianglerighteq$ を増やすことである．$||$ と $\trianglerighteq$ を増やす理由は，一般になるべく並行に実行できるアクティビティを増やすことにより開発期間の短縮が可能となるからである．満足するアクティビティが得られたら詳細化を中止する．満足するかどうかの基準は，開発するときのリソース条件やリリース時期などにより異なるので，ここでは議論しない．

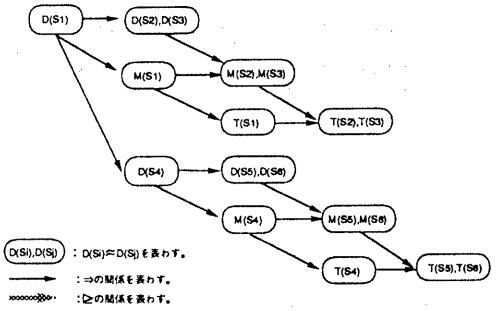
定義 4.1 実行可能アクティビティ
$a' \Rightarrow a(a', a \in A)$ である $\forall a'$ に対して $p(a') = true$ が成立しているとき a は実行可能であると呼ぶ．

ここで，p(a) は a の実行が終了していれば true していなければ fault の値をとる．“アクティビティが実行可能である”とは，そのアクティビティ (a) よりも優先実行関係が高いアクティビティ (a') の実行がすべて終了していることを意味している．

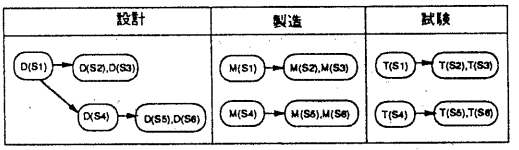
定義 4.2 開発プロセスの完全性
“DP が完全である”とは，最初に実行するアクティビティから順次アクティビティを実行したものとして実行可能なアクティビティをたどることによりすべてのアクティビティの実行を終了したものとできることである．

DP が完全でなくても，実行可能なアクティビティが存在し，そのアクティビティが含む個別仕様関係の詳細化が十分なされていると判断した場合は，アクティビティの実行を開始できる．すべてのアクティビティの実行を終了した場合に DP が終了となる．

DP で表現されたアクティビティは依存関係のある直前のアクティビティがすべて実行されていれば他のアクティビティの状態とは独立に実行可能 (定義 4.1) となる．



(a)本開発法



(b)ウォータフォール型開発

図 3: 開発期間の比較

4.2 詳細化の方法

以下に詳細化の手順を示す．

1. 詳細化にともない，個別仕様の関係 (SR) と物理的構造の重複関係 (PR) を蓄積する．
2. SR と PR に基づいて，アクティビティへの変換ルールを用いてアクティビティの実行順序関係の集合すなわち弱開発プロセス DP に変換する．
3. DP 内の実行可能なアクティビティを抽出し，その詳細化レベルで十分かどうか判断し，十分ならば実行する．1，2，3 を繰り返す．

なお，DP は各アクティビティの関係を断片的に集めただけであるため，機械的には取り扱いが容易であるが，人間にとっては理解が難しく関係の不足などがあっても分かり難い．このため，理解が容易となるように，DP をアクティビティネットワークの形に直して表現する場合がある．

5 ウオータフォール型開発と本開発法

代表的な開発モデルであるウォータフォール型開発との比較を行なう．

5.1 開発期間

ウォータフォール型開発の開発期間を $TW(dp)(dp \in DP)$，本開発法の開発期間を $TS(dp)$ とする．アクティビティ実行のためのリソースが十分にあり，ウオータフォール型開発の各工程では本方式と同一なアクティビティの関係が明確にされたものとする（一般にはアクティビティの関係が明確化されていない場合もあるためこの条件はウォータフォール型に有利な条件である）．

ウオータフォール型開発では，設計／製造／試験と工程を明確に区切って開発するため，開発期間は以下の様になる．

$$\begin{aligned} TW(dp) &= TWD(dp) + TWM(dp) + TWT(dp) \\ &= MAX(T(dpD_1), T(dpD_2), ..., T(dpD_n)) \\ &\quad + MAX(T(dpM_1), T(dpM_2), ..., T(dpM_n)) \\ &\quad + MAX(T(dpT_1), T(dpT_2), ..., T(dpT_n)) \quad (1) \end{aligned}$$

ここで，TWD，TWM，TWT はそれぞれ設計工程，製造工程，試験工程の実行期間，T(p) パス p の実行時間，dpA_i は工程 A の並行に実行できるアクティビティのパスを示す．従って例えば，$MAX(T(dpD_1), T(dpD_2), ..., T(dpD_n))$ は設計工程の中のクリティカルパスの実行期間を意味している．本方式での開発期間は工程の区切りがないため以下の様になる．

$$TS(dp) = MAX(T(dpA_1), T(dpA_2), ..., T(dpA_n)) \quad (2)$$

性質 5.1

$$TS(dp) \leq TW(dp) \quad (3)$$

証明は紙面の関係で省略する．

性質 5.1 に示すように，本方式では必ずウオータフォール型開発以下の開発期間で開発できる．

5.2 プロセスの手戻り性

ウォータフォール型開発では，各工程の途中や最後でレビューなどを行ない各工程を完全に実施し，次工程に移ることが品質作り込みの基本戦略である [13]．

各工程が完全に実施されるということは，この工程内の各アクティビティが完全に実施されたということと等価である．したがって，本方式がウォータフォール型開発と同等な品質を確保できることを示すには以下のことを言えれば良い．

1. すべてのアクティビティが完全に実施される．
2. 1 の際に，ウォータフォール型開発の場合に工程間の後戻りによるロスが生じないと同様に，アクティビティ間でも後戻りが生じない．

1 について，本開発法では，工程毎に，例えば，設計ならばすべての設計が完全であることを確認するわけではなく，アクティビティ毎にその完璧性を確認する．従って，すべてのアクティビティの実行を終了した後にはすべてのアクティビティが完全に実施されている．

2 について，各アクティビティの関係が明確になっているため，例えば，独立決定関係にある設計アクティビティ同士はお互いに無関係であるためお互いの完全実施には無関係に自分を完全に実施できる．このため，工程に区切ることをしなくても，アクティビティ単位でその実行の完全実施を保障すれば，ウォータフォール型開発と同様に後戻りは発生しない．

5.3 試験性

ここでの，試験性とは，仕様を分割して開発することによる試験アクティビティの稼働の増大の程度を意味している．試験性 (T) は以下の様に定義する．

$$T = ES(s)/EW(s) \quad (4)$$

ここで，すべての仕様を一括して試験する場合の試験稼働を $EW(s)$ とし，本開発法の試験稼働を $ES(s)$ とする．

本開発法での試験は部分仕様 (s_i) 毎に行なわれるため，試験稼働は以下に示すように，各部分仕様の試験稼働の和で表される．

$$ES(s) = \sum_{i=1}^{n} ES(S_i) \quad (5)$$

一般的には $T(S_i)$ は純粋に $M(S_i)$ で開発した部分の試験のみでは済まない場合がある．開発の手順とソフトウェアの構造によっては，その S_i と構造的につながりのある S_i よりも前に開発された仕様の部分の再試験が必要になるからである．もし，再試験が必要にならない場合は $T = 1$ つまり $ES(s) = EW(s)$ となる．一般的にインクリメンタルな開発法では，T の値をいかに小さく押えるかが問題となる．仕様分割としては，関数やサブルーチンとしてプログラムを分割しプログラムの実行フロー（コントロールフロー）の中

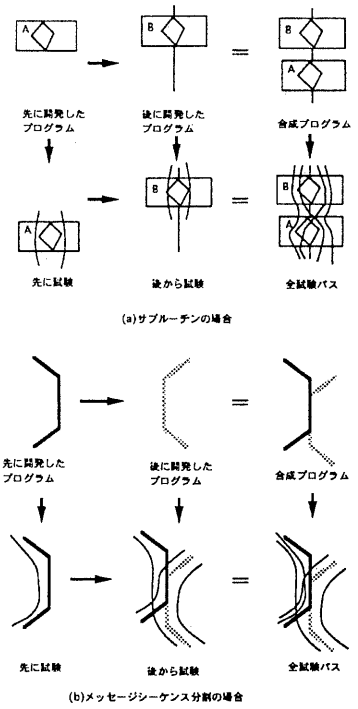


図 4: 分割方法と試験パス

に埋め込む方式が一般的である．この場合は，図 4(a) に示すように以前に開発して試験した部分をもう一度試験しなければならない場合が発生しするため，無計画に試験をすると大幅に試験稼働が大きくなる場合がある．一方，図 4(b) に示すように，SDE でのメッセージシーケンス合成の手法を使った仕様の分割方法では，開発したシーケンスをすべてのパスについて試験するようにすれば無駄な試験は発生しない．以上の考察から，メッセージシーケンスを用いた仕様分割が試験稼働の点からみて有利であると言える．

6 PBX サービスへの適用例

6.1 PBX の仕様例

実際の PBX サービスの内線発呼サービスの一部を例に本開発法を具体的に述べる．開発するサービスは表 1 に示す 1 1 種の個別仕様の集合と決まったものとする．

一般に，ユーザ，ービス開発者あるいは設計者の意思が入るために，たとえ最終仕様がまったく同一であったとしても，仕様の関係については一意に定まるものではない．そこで，ここでは，サービス設計の状況としてありそうな状況を想定して述べる．

6.2 仕様の詳細化と開発プロセスの導出

ステップ 1 最初に，以下に示す関係について決まったものとする．

1. 最も基本的でまず決定してしましたい仕様は信号音呼び出し (S_5) である．これはもっとも基本となる個別仕様であるのでまず最初に仕様を決定したい．
2. 次に決定したい仕様は基本機能でかつ特殊な機能ボタンを使用するホットライン呼出，短縮ダイヤル，再ダイヤル，メ

表 1: 開発するサービスの個別仕様

個別仕様名	個別仕様
S1	コールバック
S2	キャンプオン
S3	ステップコール
S4	相手話中呼出
S5	信号音呼出
S6	音声呼出
S7	おしゃべり広場
S8	ホットライン呼出
S9	短縮ダイヤル発信
S10	再ダイヤル
S11	メモダイヤル

モダイヤル (S_8, S_9, S_{10}, S_{11}) である. この仕様群が参考とする部分仕様は S_5 のみである.

3. 次に決定したい仕様は相手話中時のサービスであるコールバック, キャンプオン, ステップコール, 相手話中呼出 (S_1, S_2, S_3, S_4) である. この部分仕様群も参考とする部分仕様は S_5 のみである.
4. 次に決定したい仕様は特定番号を指定したサービスであるおしゃべり広場 (S_7) である. この部分仕様も参考とする部分仕様は S_5 のみである.
5. 最後に, 捕捉的なサービスである音声呼出 (S_6) を決定する. この部分仕様も参考とする部分仕様は S_5 のみである.

上記を関係式で表現すると以下となる.

$$\begin{aligned} SR \quad = \quad & \{S_5 \Rightarrow ((S_8, S_9, S_{10}, S_{11}), (S_1, S_2, S_3, S_4), S_7, S_6), \\ & \| ((S_9, S_{10}, S_{11}), (S_1, S_2, S_3, S_4), S_7, S_6), \\ & S_5 \trianglerighteq (S_8, S_9, S_{10}, S_{11}) \trianglerighteq (S_1, S_2, S_3, S_4) \trianglerighteq S_7 \trianglerighteq S_6\} \end{aligned} \tag{6}$$

SR と DP を図5に示す.
PR としては, 以下の関係が明確化された.

$$S_5 \cap s_i \neq \phi (i = 1, 2, 3, 4, 6, 7, 8, 9, 10, 11)$$

ステップ2 次に,SR について $\| (S_8, S_9, S_{10}, S_{11})$ の関係と, PR について $(S_i \ominus S_5) \cap (S_j \ominus S_5) = \phi (i \neq j, i = 6..11, j = 6..11)$ を明確化した.
SR と DP を図6に示す.

ステップ3 次に, SR について $\| (S_3, S_4, (S_1, S_2))$,$S_2 \Rightarrow S_1$ の関係を, PR について $(S_3 \ominus S_5) \cap (S_4 \ominus S_5) = \phi, (S_i \ominus S_5) \cap (S_j \ominus S_5) = \phi (i = 1, 2, j = 3, 4), S_1 \cap S_2 \neq \phi$ を明確化した.
SR と DP を図7に示す.
実際の最終的なサービス仕様の全体は図8に示すように, 個別仕様のシーケンスの先頭部分を共有するようなサービスとなった.

7 おわりに

仕様を部分仕様に分割し, その関係を明確化することにより, 開発プロセスを導出する手法をフォーマルに示した. 今後は, SP, PR, DP のフォーマルな検証方法を検討するとともに, 実際の開発への適用を試みる.

謝辞

本研究の機会と有益な御助言を頂いたNTT ソフトウェア研究所, ソフトウェア開発技術研究部, サービスソフトウェア方式研究グループの皆様に深謝致します.

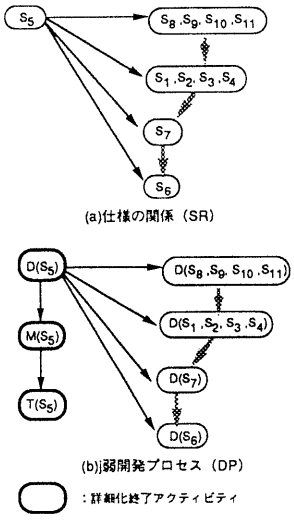


図 5: ステップ1での仕様の関係と弱開発プロセス

参考文献

[1] Barry W. Boehm,:A Spiral Model of Software Development and Enhancemnet, ACM SIGSOFT Software Engineering Notes, Vol.11, No.4, pp.22-42, Aug (1986).

[2] 大野 豊, : ソフトウェア工学の背景と展望, 情報処理, Vol. 28, No.7, pp845-852, July.(1987).

[3] Walker Royce, :TRW's Ada Process Model for Incremental Development of Large Software Systems, Porc. 12th ICSE, pp.2-11 (1990).

[4] D. R. Graham, :Incremental development review of non-monolithic life-cycle development models, information and software technology,

[5] Leon Osterweil, :Software Processes are Software Too, Proc. of 9th ICSE, pp.2-13, Apr.(1987). technology,

[6] Watts S. Humphrey and Marc I. Kellner, : Software Process Modeling: Principles of Entity Process Models, Proc. of 11th, pp331-342(1989).

[7] 金井 敦, 寺内 敦, 山中 顕次郎: 通信ソフトウェアの畳込開発法, 情報処理学会論文誌, Vol.34, No.5, pp.1166-1176, May. (1993).

[8] 荒川 則秦, 山中 顕次郎, 加藤 順: サービス直交性に着目した通信サービスアーキテクチャ, 信学会春季全国大会 (1989年), B-395,3-101 (1989).

[9] 松尾谷 徹, 高田 淳司, 三橋 秀夫: ソフトウェア開発におけるテスト資源と修正資源の配分問題, 電子情報通信学会, 信学技報, R93-3, pp.13-18, May. (1993).

[10] 塚本 幸展, 松本 啓之亮 : ソフトウェア分散開発におけるシステム開発上流工程でのソフトウェア分割方式, 情報処理学会, ソフトウェア工学, 82-2, pp.9-16, Dec. (1991).

[11] 石若 通利, 元治 景朝, 荻原 剛志, 井上 克郎, : 動的詳細化が可能なプロセス記述の表記法とその開発現場への応用について, 情報処理学会, ソフトウェア工学, 88-5, pp.33-39, Nov. (1992).

[12] H. Ichikawa, M. Itoh etc. , :Incremental Specification and Development of Cmmunication Software, IEEE Trans. on Computers. pp.553-561, Vol.40, No.4, Apr. (1991).

[13] 山田 茂, : ソフトウェアマネジメントモデル入門 1,2,3, 完, bit, Vol.22, No.12 (1990), Vol.23, No.1,2,3 (1991).

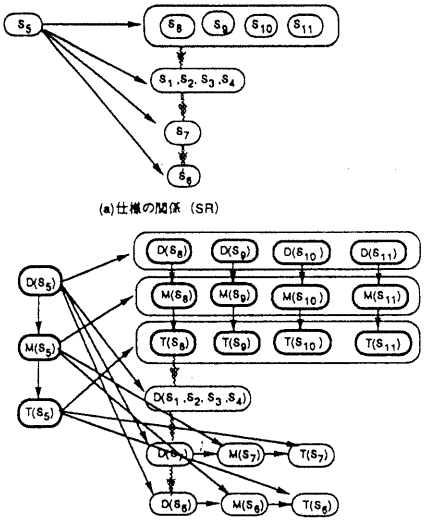


図 6: ステップ 2 での仕様の関係と弱開発プロセス

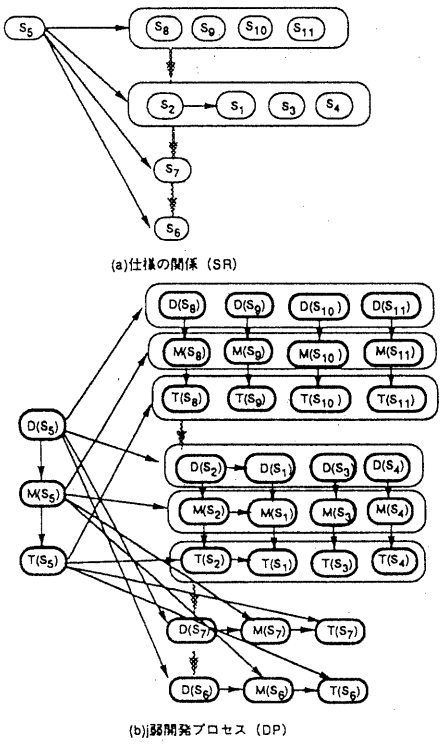


図 7: ステップ 3 での仕様の関係と弱開発プロセス

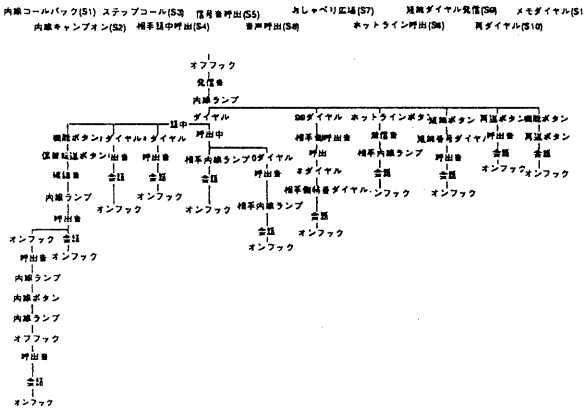


図 8: 例題とした PBX サービス