通信ソフトウェアの畳込開発法

金井 敦† 寺内 敦† 山中顕次郎†

通信ソフトウェアの開発期間短縮を目的とした畳込開発法を提案する．この開発法では，サービス仕様を複数の順序関係のある部分仕様に分割し，それぞれの開発工程（ステップ）を次のステップと重ねて段階的に開発する．ウォータフォール型開発に比較して，要求定義および設計工程の期間が製造および試験工程とほぼ同じ場合に開発期間短縮効果が最大になり，分割数を増やすことにより理論的には半分まで開発期間を短縮できることを示す．この開発法を音声蓄積サービスのアプリケーションソフトウェア（音声蓄積サービス AP）開発に適用し，正常仕様，準正常仕様，異常仕様に分割して開発した場合，ウォータフォール型開発に比べて開発期間がほぼ 16% 短縮可能と推定できる．また，音声蓄積サービス AP の場合は準正常および異常系シーケンスを部品化することにより開発期間短縮が可能となることおよびウォータフォール型の開発法に比較してピーク稼働が大きくなることを考察する．

Folding Development Methodology for Communications Software

ATSUSHI KANAI,† ATSUSHI TERAUCHI† and KENJIROH YAMANAKA†

This paper proposes a folding development methodology for communications software. In this methodology, the requirement specification is divided into several sub-specifications which are developed simultaneously in order to decrease development time. The methodology is generally most effective when the time for the specification definition and design phase is almost the same as the time for the later phases, and the development time can be theoretically decreased to a half. The development time is estimated to be decreased to about 84% compared with development based on a waterfall model, for a voice storage application. In this case, the specification is divided into ordinary, exception and error sub-specification. Furthermore, the estimation shows that the development time for a voice storage application is further decreased using reusable sequences for exception and error, and that the peak of the manpower curve for development based on the proposed methodology is higher than that for development based on the waterfall model.

1. はじめに

ソフトウェアの開発期間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列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)~4)]が提案されている．

従来方式では，仕様を小さく分割したり，仕様を確定するために試行を繰り返したりすることにより開発期間短縮や品質向上を目指していた．しかし，独立したモジュールという形で仕様を分割できない場合や，試行錯誤的ではない場合には開発期間短縮の効果は期待できない．本論文では，このような状況での開発期間の短縮を目的として，すべての仕様が決定されるまで開発を待たずに，段階的に決定する分割された仕様（部分仕様）を部分仕様ごとに開発しそれぞれの開発工程の一部を時間的に重ね合わせて実行することにより開発期間を短縮する開発方法を提案する．これは，段階的に決定される仕様に合わせて効率良く開発を進める開発法とみることができる．また，サービス仕様を段階的に決定できる部分仕様に分割しておき，それに対して畳込開発法を適用することにより，従来の開発法では短縮できなかった開発期間を積極的に短縮することが可能となる．このような畳込開発法は一見プロトタイピング等と類似しているように見えるが，これら手法と畳込開発法は競合しない．例えば，開発プロ

† 日本電信電話株式会社 NTT ソフトウェア研究所ソフトウェア開発技術研究部
Software Engineering Laboratory, NTT Software Laboratories, Nippon Telegraph and Telephone Corporation

セスの中で試行が入る nested development[5] のループの内部の開発法としても，ライフサイクル全体の開発法としても畳込開発法は他の開発法と競合なしに用いることができる．

以下，第2章では，畳込開発法の考え方および手順を述べ，第3章で畳込開発法の開発期間短縮効果と適用領域を一般的に評価する．第4章では，音声蓄積サービス（伝言ダイヤルや応答代行等）のアプリケーションソフトウェア（音声蓄積サービス AP）開発の場合について畳込開発の例を説明する．第5章では，音声蓄積サービスを例に期間短縮効果を評価する．最後に，第6章では，部品化による自動化の可能性および畳込開発法の稼働変化について考察を加える．

2. 畳込開発法

2.1 考え方

畳込開発法では部分仕様に対し，パイプライン処理的に多重に開発を行う．以後，部分仕様に対して行う一連の部分開発を“ステップ”と呼ぶ．それぞれのステップでは通常のウォータフォール型開発と同様に要求定義，設計，製造，試験を行う．**図1**に畳込開発法の開発プロセスを示す．同図に示すように，各ステップでは小型のウォータフォール型開発が繰り返されるが，ある時期には要求定義／設計と製造／試験が並行に行われる．以後，要求定義および設計を前工程，製造および試験を後工程と呼ぶ．工程という用語は，畳込開発法では，各ステップ内での要求定義，設計，製造，試験といった開発作業の流れを示すものとし，ステップと区別して用いる．

各ステップでは前工程では前ステップで行われた前工程の結果を，後工程では前ステップで行われた後工程の結果とそのステップの前工程の結果を使用するため，各ステップの前工程および後工程は以下の条件が整うと開始できる．

1. 各ステップの前工程はその前のステップの前工程が終了している場合に開始できる（ステップ1を除く）．
2. 各ステップの後工程はそのステップの前工程とその前のステップの後工程が終了している場合に開始できる．

なお，ここで考える試験とは，単体／結合／総合試験等と呼ばれている開発部門で担当する試験に相当するものとする．導入側で行われる安定化試験等は本方式の対象としていないためここでは考えない．以後，本論文では，要求定義から試験までの期間を開発期間と呼ぶ．

仕様を部分仕様に分割し，畳込開発法を適用するためには部分仕様が以下の関係を満たす必要がある．

1. 先に決定される部分仕様が決まらなければ後に決定される部分仕様は決まらない．
2. 先に決定される部分仕様は，後から決定される部分仕様が存在しなくても決定することができる．
3. 後に決定される部分仕様の変更は前に決定される部分仕様に影響を与えない．

通常のモジュール分割による並列開発は，お互いに独立な部分仕様の開発と見ることができ，モジュール間のインタフェースが決定されていれば独立に開発することができる．しかし，畳込開発法の場合は，上記関係が成り立つような部分仕様間に従属関係が存在する場合が対象となり，同一モジュールに対しても適用可能である．

一方，通信処理のアプリケーションプログラムの仕様は，以下に示すプロセスで決定される場合が多い．

1. サービスの目的を達成するために直接に必要な仕様（正常仕様）を決める．

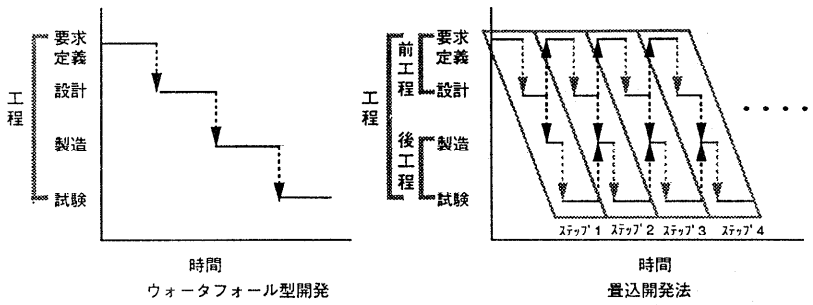


図1 畳込開発法とウォータフォール型の開発プロセス
Fig. 1 Development processes of folding development methodology and waterfall model.

2. 正常仕様の目的達成のために付随して発生する仕様（準正常仕様）を決める．
3. システムのアーキテクチャや条件を考慮しながらシステム異常時の仕様(異常仕様)を決める．

このため，通信ソフトウェアアプリケーションの畳込開発はこの流れにそった部分仕様でステップ分割を行うと自然であり，上記の部分仕様が満たすべき条件を満たしている．そこで，本論文では，畳込開発法の各ステップで開発される部分仕様として，正常仕様，準正常仕様，異常仕様を対応させて音声蓄積サービスAP を畳込開発する具体例について主に議論する．

一般に，畳込開発法は，ある仕様が部分仕様に分割／製造でき各部分仕様の開発において分割損を補う開発支援環境があれば，各ステップで開発される部分仕様の内容には依存しないで開発期間短縮効果をもたらす．畳込開発法に適したソフトウェア開発環境は，以下の性質を満たす機能を持つことが要求される．

1. 各ステップでなるべく独立にプログラムを製造および管理でき，各ステップのプログラムを統合できること（部分独立性）．
2. 複数ステップへ分割したことによる分割損がなるべく小さいこと（小分割損）．
3. 各ステップで製造されたプログラムがそのステップで走行確認できること（部分走行性）．

これらの条件を満たす開発環境のひとつとして，通信ソフトウェア用の開発環境SDE☆が提案され開発されている[6),7)]．本論文では，SDE を開発環境として想定するが，一般には上記性質を持つ開発環境ならばどのような開発環境でも使用できる．

2.2 開発手順

ここでは，畳込開発法の一般的手順について述べる．

i ステップでの前工程では，1から $i-1$ ステップで決定されている部分仕様に基づいて，部分仕様の決定と設計を行う．この場合，検討範囲を判断する必要があるため，各ステップで対象とする範囲をあらかじめ厳密に決定しておく必要がある．

i ステップでの後工程では，このステップの設計工程で設計された結果に基づいて，1から $i-1$ ステップで開発されたプログラムに i ステップでのプログラムを加える．SDE を利用した場合の手順と一般的な環境を利用した手順の比較を**図2**に示す．一般的な開発環境では，同一ソースプログラム内に各ステップで開発したプログラムを混在してコーディングしなければならないため，部分仕様に対応するプログラムのみを独立に管理できず部分仕様とプログラムの対応がとり難い．さらに，$i-1$ ステップで開発したプログラムが開発中の i ステップのソースプログラムと混在するため，1から $i-1$ ステップのプログラム部分の修正が i ステップのコーディング中に発生すると，開発中のプログラムに対して前のステップの修正が入るため開発効率が悪くなる．このような悪影響を防ぐため部分独立性が必要となる．各ステップで独立に開発したプログラムを再利用し，一つのプログラムに結合できる機能があれば，上記問題点は解決できる．以下では，SDE のこの機能を利用し部分独立性を向上させる製造法について述べる．

SDE はメッセージシーケンスからプロセス仕様(SDL☆ 表現[8)])を生成し，そのプロセス仕様からプログラムを生成する．一般に，一つのサービスは複数のメッセージシーケンスの総体から構成される．このため，複数のメッセージシーケンスからプロセス仕様を生成するために，個々のシーケンスをプロセス仕様に組み込む機能が必要である．この機能をここでは，合成と呼ぶ．SDE では，シーケンスを言語 SAL☆☆[6),7)]で表現しているためメッセージシーケンスを記述することがプログラミングに相当する．シーケンスを具体的に言語で記述した実体をプログラムと呼ぶが，以後特に誤解のない場合にはプログラムという意味でシーケンスという言葉を用いる場合がある．

各ステップでは自ステップの部分仕様に対応するメッセージシーケンスの集合を記述する．i ステップでは，$i-1$ ステップまでに開発されたシーケンスの一部を切り取りそれに対して，i ステップで製造対象となったシーケンスを付加する．合成機能を利用しているため，基本的には，i ステップで必要となる部分に至るまでの $i-1$ ステップまでに開発されたシーケンスを再利用して，それに i ステップでのプログラムを追加し，全体を合成することによって開発を進める．既開発シーケンスを再利用する場合に，単なるコピーでは，$i-1$ ステップまでに開発されたソースプログラムに変更があるとコピーしたすべての部分を変更しなければならず保守性に問題が生じる．そこで，それ以降のステップで使用する部分はマクロ化（サブルー

☆ System Design Environment

☆ Specification and Description Language
☆☆ Service Addition Language

チン化）し，それ以降のステップではそのマクロを参照するようにする．この場合，機能を抽象化し階層的にプログラミングする目的で使用される通常のマクロ化と異なる使用法であるため，上記目的でマクロ化すると非常に理解し難いプログラムとなる．SDE では，指定したポイント以前のシーケンスを自動的にマクロ化する機能があるため，この機能を使用することにより，そのステップでは直接マクロ表現せずにプログラミングでき，それ以降のステップでは，自動生成されたマクロを参照することによりそれ以前のステップで開発されたプログラムを使用することができる．これにより製造時の分割損を小さくできる．

各ステップでは，そのステップで開発した部分に関する試験のみを行う．ただし，i ステップで開発したプログラムの試験のためには，$i-1$ ステップまでに開発した部分も必要となるため，1から i ステップまでのプログラムを合成したロードモジュールに対して，i ステップに関する試験をする必要がある．この場合，i ステップ目が最終ステップでない場合には1から i ステップまでの仕様は完成したものではなく動作を記述していない処理があるためそのままでは走行しない．SDE では，プログラムをジェネレートする時に自動的に未定義受信に対する必要な処理（エラー処理）を生成するため未設計の信号に対しては特に考慮しなくてもジェネレートされたプログラムを実行することができる．部分走行性はこのように各ステップでそのステップで開発したプログラムを試験するために必要である．

3. 開発期間の評価

3.1 開発期間定式化

通信ソフトウェア開発の場合の具体的ステップとして，正常系，準正常系，異常系の3ステップに分けることを想定しているが，一般には，開発するソフトウェアやプロジェクトの性質によりより多くのステップを設定できる場合や分割する対象が異なる場合があ

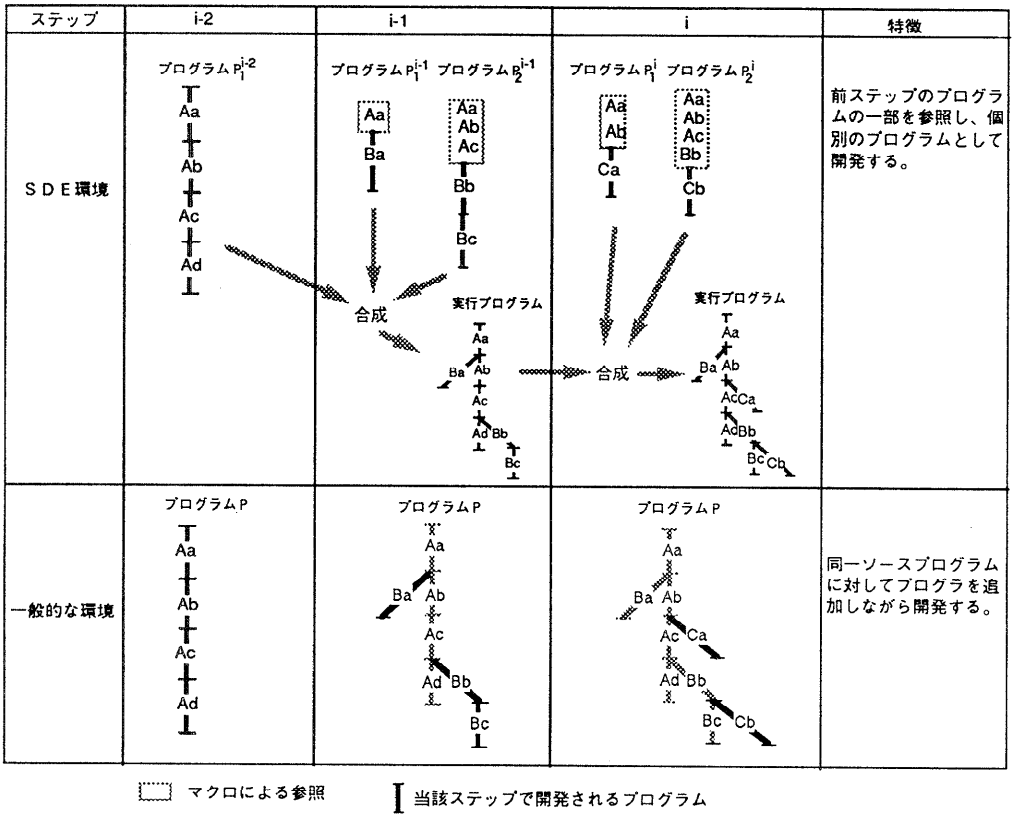


図 2 SDE と一般的な環境の手順比較
Fig. 2 Comparison of development methods between SDE and a usual environment.

る．このため，ここではステップ数は任意なものとして考える．

ウォータフォール型開発の要求定義，設計，製造，試験期間をそれぞれ r, d, m, t とする．また，畳込開発法でのステップ i の要求定義，設計，製造，試験期間をそれぞれ r_i, d_i, m_i, t_i とする．畳込開発法の開発期間短縮効果の目安として，ウォータフォール型開発の開発期間を1とした場合の相対開発期間 E（(畳込開発法の開発期間)/(ウォータフォール型開発の開発期間)）を用いる．ウォータフォール型開発の開発期間と畳込開発法の各ステップの開発期間の総和が等しいとし分割ステップ数を n とすると，E は次式で与えられる．

$$E=(r_1+d_1+\sum_{i=2}^{n}(max(r_i+d_i, m_{i-1}+t_{i-1}))+m_n+t_n)/(r+d+m+t) \quad (1)$$
$$(n>=2)$$

開発期間の総和が等しいので，以下の等式が成り立つ．

$$r+d+m+t=\sum_{i=1}^{n}(r_i+d_i+m_i+t_i) \quad (2)$$

一般的傾向を見るために各ステップの期間はすべて同一という仮定をおいて式1を評価する．計算にあたり，ウォータフォール型開発の全開発期間を C，ウォータフォール型開発の全開発期間に占める前工程期間の比率を α とする $((r+d):(m+t)=\alpha:(1-\alpha))$．

$$r+d=n(r_1+d_1)=n(r_2+d_2)=n(r_3+d_3)\cdots\cdots=n(r_n+d_n)=\alpha C \quad (3)$$

$$m+t=n(m_1+t_1)=n(m_2+t_2)=n(m_3+t_3)\cdots\cdots=n(m_n+t_n)=(1-\alpha)C \quad (4)$$

- $0<\alpha<0.5$ の場合

$$E=(r_1+d_1+(1-\alpha)C)/C=(n-(n-1)\alpha)/n \quad (5)$$

- $0.5<\alpha<1$ の場合

$$E=(r+d+(1-\alpha)C/n)/C=(1+(n-1)\alpha)/n \quad (6)$$

3.2 適用領域

図3は横軸を α として4ステップ($n=4$)の場合について相対開発期間(E)を示したものである．さらに，α が0.5の場合の相対開発(E)をステップ数 n を横軸にして示したものが**図4**である．

各ステップの期間はすべて同一という仮定があるため実際の様々なバリエーションに対して厳密には判断できないが，畳込開発法の適用領域の概略が図3および図4から推定できる．前工程が全体開発期間の半分程度をしめるプロジェクトで最も期間短縮効果が大きく分割ステップ数が4の場合ウォータフォール型開発の62.5%となる．したがって，前工程の期間が極端に長い場合やその逆の場合は本方式の効果が薄くなる．しかし，通常の開発では前工程の期間が全開発の半分程度になる場合が多い[9]ので，大部分の場合かなり効果が大きくなるものと思われる．また，図4から，分

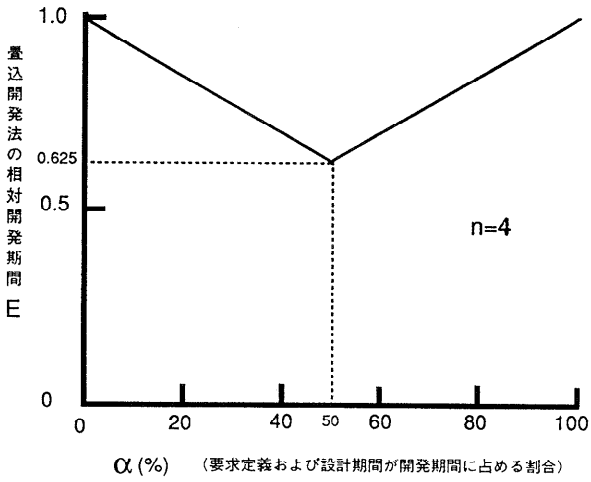


図3 開発期間短縮効果（1）
Fig. 3 Effect on shortening the development time (1).

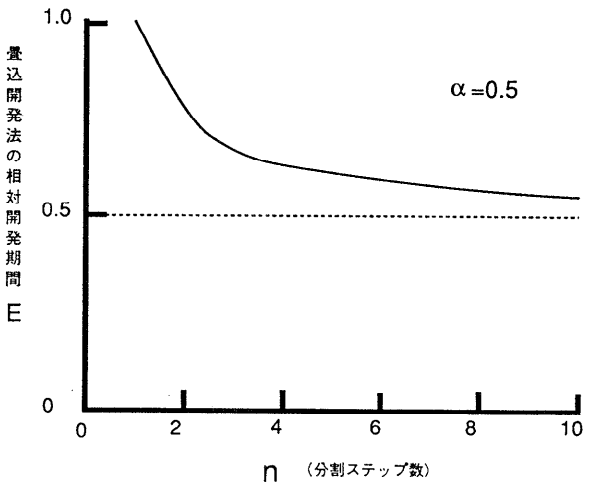


図4 開発期間短縮効果（2）
Fig. 4 Effect on shortening the development time (2).

割数を多くすればするほど期間短縮効果は大きく，相対開発期間 E は 0.5 に漸近する．しかし，分割数を5以上にしても期間短縮効果がほとんど大きくならないことがわかる．これは，無理をしてステップ数を多くしても期間短縮効果はそれほど大きくならないことを示している．

4. 音声蓄積サービスへの適用

ここでは，具体的に音声蓄積サービス AP 開発の場合に畳込開発法を適用した例について述べる．

音声蓄積サービス AP を対象とした場合の各ステップで開発する部分仕様の定義を以下に示す．ここで，音声蓄積サービスとは電話を利用した音声メール，音声掲示版あるいは留守番電話などの音声蓄積処理技術を電話サービスに応用したサービスである．

正常仕様　サービスの目的を達成するために直接に必要な処理である．付随的に発生するエラー処理等は対象としない．システムからの入力促進要求に誤りなくエンドユーザが入力（プッシュボタン（PB）入力）した場合である．

準正常仕様　正常仕様の目的達成のために付随して発生するエンドユーザの想定される誤り入力に対して対処する処理である．例えば，サービスからの入力促進に合致しない入力が来た場合（誤入力，タイムアウトやユーザ側からの切断がこれに含まれる）が考えられる．

異常仕様　システムの異常等が発生した場合の処理である．

ここでは，上記の順番でステップを構成する．以後，それぞれ正常系ステップ，準正常系ステップ，異常系ステップと呼ぶ．SDE 環境での開発の流れを図5

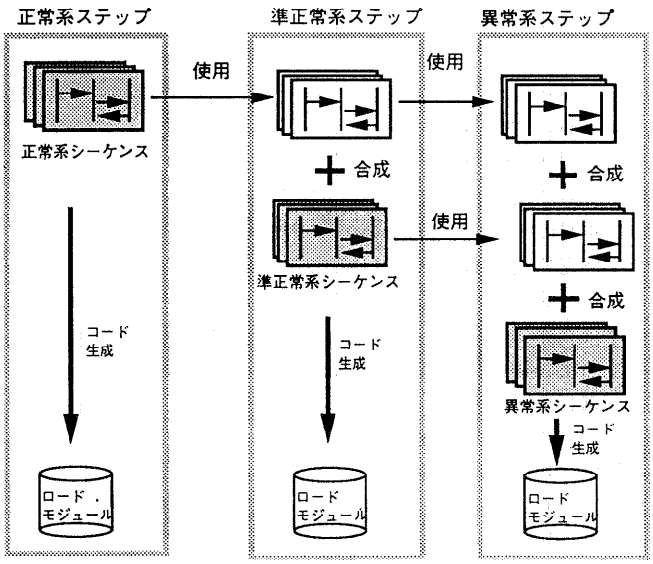


図 5　開発の流れ
Fig. 5　Development flow.

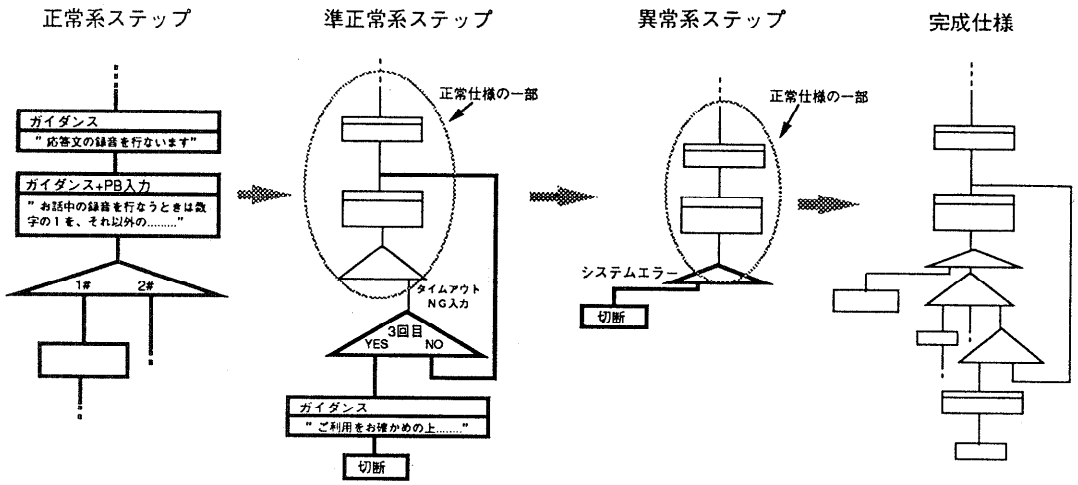


図 6　各ステップで開発される仕様の例
Fig. 6　An example of specification developed in each step.

に，各ステップで開発するシーケンスの例を**図6**に示す．正常系ステップではサービス実現上の中心となる正常仕様を実現する．正常仕様の対象となっていないユーザの誤入力処理やシステム異常処理については，標準のエラー処理が自動的に生成されるため（通常は無視あるいは終了処理を行う），ここで作成された正常系シーケンスは正常仕様のみの確認のために走行可能である．

準正常系ステップでは準正常系シーケンスを付け加える部分に至るまでの正常系シーケンスを切り取り（図6の細字のシーケンス，実際にはマクロ化される），それに準正常系シーケンスを続けて記述する．切り取られる正常系シーケンスの一部は準正常系シーケンス記述前にマクロとして切り出しておく必要があるため，切り取るポイントは正常系シーケンス記述時にあらかじめ指定しておく．また，異常系ステップで使用する準正常シーケンスについても同様にこのポイントを指定する必要がある．作成した準正常系シーケンスを正常系シーケンスと合成する．合成することにより，正常系および準正常系の機能が両立したプロセス実現仕様が得られる．このステップでは，異常系処理が完成していないが，ユーザから見て最終製品と変わらない使い勝手が実現される．

異常系ステップでは，準正常系ステップと同様な手順でファイルオープンエラー等のシステム異常処理シーケンスを作成し正常系および準正常系シーケンスと合成する．この結果，完成したロードモジュールが得られる．

5. 音声蓄積サービスの場合の評価

5.1 評価方法

音声蓄積サービスについて畳込開発法に従って記述実験を行うことにより評価した．記述実験の範囲は製造工程部分であるため，以下の仮定のもとに，全体の開発期間の評価を行った．

仮定1　各ステップの開発量と各ステップの開発期間は比例する．

仮定2　各ステップの各工程が占める割合はすべてのステップで同一である．

仮定3　各工程の期間比は経験値からほぼ要求定義：設計：製造：試験＝3：2：1：4とする．

仮定4　開発量の尺度としてプロセス仕様の状態数を用いる．

ここで用いた言語は前述したシーケンスを記する言語SALであるため，ソースプログラム上の異なる場所で記述されたシーケンスが，同一の状態に合成される場合がある．従って，ソースプログラム上で記述された量（ライン数等）は多重にカウントされる部分があるため，一般には開発した機能の量を反映していない．このため，一般の言語（手続型言語）のように開発した機能の量を単純にソースプログラムの量で判断できない．そこで，仮定4ではソースプログラムの記述スタイルや合成の影響を受けず，より直接開発した機能に比例すると思われるプロセス仕様（SDL）の状態数を開発量の目安とする．

5.2 期間短縮効果

各ステップでの開発量を**表1**に示す．合成を行うために，例えば準正常系シーケンスの前半に正常系のシーケンスを繋げて記述する必要があるが，“参照した部分を含めた状態数”はその繋ぐ正常系シーケンスまでも含めてそのステップで必要とされる全体の状態数である．したがって，この状態数は既開発部分も含まれているためそのステップの実際の開発量はこの値より必ず小さくなる．“新たに開発された状態数”はシーケンスが合成され実行ファイルとなった状態での各ステップでの増加状態数である．これは，純粋にそのステップで開発されるべき開発量とみて良い．ただし，実際には，例えば記述する準正常系シーケンスの前に正常系シーケンスを繋げるという作業が入るので，繋げるシーケンスを前ステップのプログラムからうまく再利用しないと，そのステップでの開発量は“新たに開発された状態数”より大きくなる．これは工程を分割したことによる分割損である．

以上の考察から，**図7**に示すように，領域1は分割損が存在する部分であり，領域2は部品化や自動合成等により単純に開発するのに比較してより少ない稼働で同一機能のソフトウェアが開発できる部分である．領域1の上は分割損が非常に大きい場合でありマクロ化支援や記述法を工夫することにより，領域1の下方

表1 各ステップでの開発量
Table 1 The size of program developed in each step.

	正常系ステップ	準正常系ステップ	異常系ステップ	合計
新たに開発された状態数	309	91	176	576
参照した部分を含めた状態数	309	345	551	1,205
新たに開発された状態数の比（最終生産物を1とする）	0.54	0.16	0.30	1

(領域2に近いところ) まで開発量を減少させることができる．分割損による開発期間延長効果が十分小さい範囲でないと畳込開発法が有効とはならないため，領域1の下方が畳込開発法として有効な領域であるといえる．領域1と領域2の境界が分割損が存在しない部分である．仮定1から，表1の状態数の比がそのまま各ステップの開発期間の比となり，正常系：準正常系：異常系=54：16：30 となる．仮定2および3のもとで，式(1)を用いて評価すると，ウォータフォール型開発の開発期間に比べて 16% の期間短縮が達成される．

6. 考　　察

6.1 部品化効果

音声蓄積サービスAPにおける，準正常系および異常系ステップの自動化の可能性について考察する．仕様書上トータル 1241 のガイダンス再生を行う音声蓄積サービスの代表的な4つのサービスについて，準正常系および異常系シーケンスのすべての場合を調査しパターンを抽出した．各パターンとその出現度数を図8に示す．この図からわかるように，出現するパターンの種類が限られていることから，準正常および異常系のシーケンスは類時なものが多いことが分かる．この類時性に着目し，このパターンに具体的なガイダンス等を埋め込んだ部品データベースを用意するかパターンにガイダンス等を指定して自動的に埋め込む機構を用意することにより，通常の音声蓄積系サービスを開発する場合では，準正常／異常系処理についてほとん

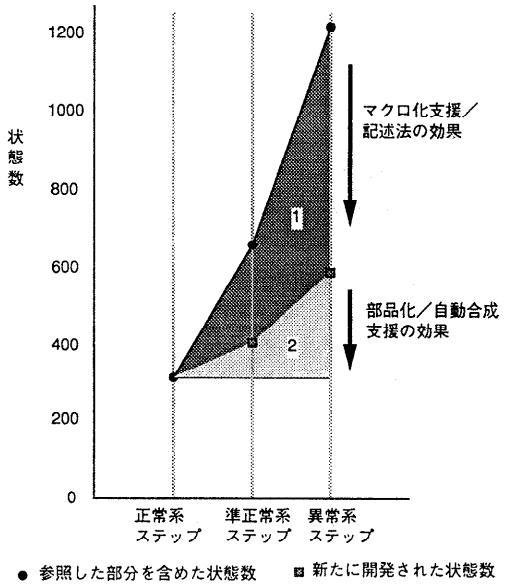


図 7 ステップと開発量（状態数）の関係
Fig. 7 Relationship between steps and the size of developed programs.

パターン番号	1	2	3	4
パターン	ガイダンス 切断	ガイダンス ホームポジション	ガイダンス ガイダンス 正常系へ	ガイダンス ガイダンス ホームポジションへ
出現数	729	2	17	6

パターン番号	5	6	7
パターン	ガイダンス PB入力 ガイダンス 正常系へ リトライ ガイダンス 切断 ホームポジションへ	この部分が準正常 録音 時間経過 ガイダンス	ガイダンス ガイダン ガイダン 切断
出現数	82	8	2

図 8 音声蓄積系サービスで表れる準正常および異常処理パターンと出現数
Fig. 8 Exception/error patterns and the number of occurrences in voice storage services.

ど新規開発する必要がなくなると想定される．アプリケーションドメインを絞り，十分に部品が整った環境では，ほとんど正常仕様のみの記述で高信頼なソフトウェアを容易に開発できる．これは特に，テストマーケット用のサービス開発等，ユーザ要求を早期に実現しフィードバックをかけなければならないが，運用に耐える機能を備えたソフトウェアが必要な場合に有効な手法である．図7では，2の領域の下の方まで開発量が削減される可能性があることがわかる．十分に部品が整った理想的な環境では，開発期間は $r_1+d_1+m_1+t_1$ のみとなり，開発期間は42% 短縮されると予想される．

6.2 開発稼働

稼働の時間変化について考察する．ここでは，COCOMO モデル[9]の中で Putnam モデル[10),11)]を修正して提案されている Organic-Mode の稼働変化曲線をベースとする．このモデルは要求定義を除く設計から試験までの稼働の時間変化がレイリー (Rayleigh) 曲線の一部に従って変化するとするモデルである．プロダクト設計から試験までの総稼働と期間を与えると一意に定まる曲線である．

上記 COCOMO モデルで示されている稼働曲線ではプロダクト設計の開始点を0としており，要求定義は稼働曲線の対象となっていない．畳込開発法の稼働を考察する場合要求定義部分も対象とした方が分かりやすいため，上記曲線をプロダクト設計以前にも適用し要求定義開始時点を稼働0の時点とする．従って，ここでは，要求定義開始から試験終了までを開発期間 T_d と定義する．このようにした場合，稼働のピークを $0.6T_d$ 付近にすると曲線の形は COCOMO モデルで示されている稼働変化と良く一致する．この曲線を**図9**に示す．この場合，要求定義の稼働は実際より若干少なくなると思われるが，ここでは，稼働変化の概観を摑むことを目的とするため特に問題としない．また，ここでは，保守を考慮していないため T_d 以降の稼働は0としている．

考察にあたり以下を仮定する．

仮定1　ステップ分割による稼働の分割損は発生しない．つまり，ウォータフォール型開発の場合と畳込開発法の開発稼働の総和は等しいものとする．

仮定2　保守稼働は比較の対象としない（保守稼働は畳込開発法とウォータフォールモデルで同等であるため）．

仮定3　畳込開発法の各ステップの稼働変化は小型のウォータフォール型開発の稼働変化とする．

仮定4　各ステップの総稼働はそのステップの期間に比例する．

畳込開発法の開発期間が最も短縮される場合の例（理想例）と前章で取り上げた音声蓄積サービス AP

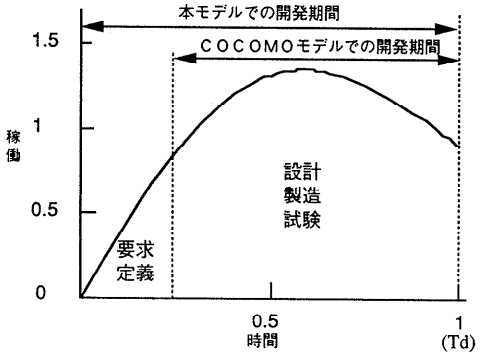


図 9　ベースとする稼働変化モデル
Fig. 9　The supposed labor distribution model.

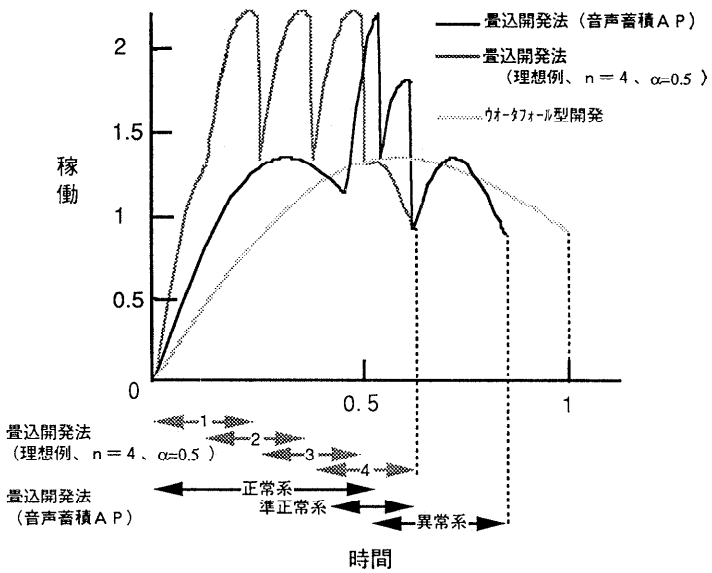


図 10　畳込開発法における稼働の時間変化
Fig. 10　The labor distribution of folding development methodology.

の場合を検討する.

理想例として以下の場合を用いる.

- 分割ステップ数は4とし,各ステップはすべて等期間に分割される.
- 各工程の期間比を前工程:後工程=1:1(α=0.5)とする.

理想例と音声蓄積サービスAPの畳込開発およびウォータフォール型開発の稼働変化を**図10**に示す.畳込開発法で急激に稼働が変化しているのは,ベースとしたモデルがT_dのところで急激に0になるためである.実際には保守的作業がはいるため急に0になることはなく比較的なだらかに0に向かう.このため,それぞれのステップの稼働のピークが他の部分と重なり合い,図10よりも稼働のリップルが平滑化されると思われる.この図から分かるように,理想例,音声蓄積サービスAP両者ともピーク稼働が増加しているが,理想例の場合は平滑化を考慮すると稼働の時間変化は矩形に近くなると思われる.従って,ピーク稼働を多くかけられる場合はその稼働を比較的一定に維持すれば良いことがわかる.しかし,音声蓄積APの場合は稼働の時間変化も大きい.従って,一般に畳込開発法ではそのAPの特性をとらえ必要稼働や時間変化を見極めて稼働計画をする必要がある.

7. おわりに

本論文では,サービス仕様を部分仕様に分割し,それぞれの開発工程を重ね合わせることにより開発期間を短縮する畳込開発法を提案し,一般的効果と適用領域を明確にし,開発期間短縮効果を音声蓄積サービスを例に評価した.また,部品化効果および稼働変化についても考察した.

音声蓄積サービスを通信ソフトウェアの具体的例として,正常仕様,準正常仕様,異常仕様の部分仕様に分割したが,この分割基準はより広いアプリケーションドメインにも適用可能であると思われる.一般には,畳込開発法が適用可能となる仕様分割法を見つけるのは難しい作業であるため,畳込開発法が簡単に適用可能になるためには,仕様分割事例が様々なアプリケーションドメインで蓄積される必要があろう.

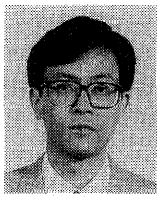
畳込開発法は,従来の開発法や開発モデルとは競合しないので,従来の方法と併用することにより開発期間をさらに短縮することが可能となる.また,開発環境としてSDEを例に開発手順を示したが,従来の開発環境でも支援ツールの開発や開発手順の工夫をすることにより畳込開発法が適用可能になると思われる.今後は,部分仕様の関係をより体系化し従来開発法と組み合わせることにより最適な開発法を導出する方法について検討するとともに実開発に適用していく予定である.

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,市川サービスソフトウェア方式研究グループリーダならびに,関係者皆様に深謝いたします.

参考文献

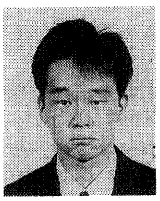
1) Boehm, B. W.: A Spiral Model of Software Development and Enhancement, *ACM SIGSOFT Software Engineering Notes*, Vol. 11, No. 4, pp. 22-42 (1986).
2) 大野 豊:ソフトウェア工学の背景と展望,情報処理, Vol. 28, No. 7, pp. 845-852 (1987).
3) Walker, R.: TRW's Ada Process Model for Incremental Development of Large Software Systems, *Proc. 12th ICSE*, pp. 2-11 (1990).
4) Graham, D. R.: Incremental Development Review of Nonmonolithic Life-cycle Development Models, *Information and Software Technology*, Vol. 31, No. 1, pp. 7-20 (1989).
5) Giddings, R. V.: Accomodating Uncertainty in Software Design, *CACM*, pp. 428-434 (May 1984).
6) Ichikawa, H., Itoh, M. et al.: Incremental Specification and Development of Communication Software, *IEEE Trans. Comput.*, Vol. 40, No. 4, pp. 553-561 (1991).
7) Ichikawa, H., Itoh, M. and Shibasaki, M.: Protocol-Oriented Service Specifications and Teir Transformation into CCITT Specification and Description Language, IECE, *Trans. of the IECE of Japan*, Vol. E 69, No. 4, pp. 524-535 (1986).
8) CCITT Recomendations Z100 to Z104.
9) Boehm, B. W.: *Software Engineering Economics*, Prentice-Hall (1981).
10) Putnam, L. H.: A General Empirical Solution to the Macro Software Sizing and Estimating Problem, *IEEE Trans. Softw. Eng.*, Vol. SE-4, No. 4, pp. 345-361 (1978).
11) Putnam, L. H.: Measurement Data to Support Sizing, Estimating and Control of the Software Life Cycle, *COMPCON*, pp. 352-352F (1978).

(平成4年7月8日受付)
(平成5年1月18日採録)



金井　敦（正会員）

昭和 55 年東北大学工学部通信工学科卒業．昭和 57 年同大学院工学研究科情報工学教室修士課程修了．同年日本電信電話公社横須賀電気通信研究所入所．以来，ソフトウェア移植工数モデル，マルチターゲットクロスコンパイラ，分散開発環境，通信ソフトウェア開発法の研究に従事．現在，NTT ソフトウェア研究所，ソフトウェア開発技術研究部主任研究員．電子情報通信学会，IEEE 各会員．



寺内　敦

昭和 43 年生．平成 3 年大阪大学基礎工学部制御工学科卒業．同年日本電信電話(株)に入社．現在，NTT ソフトウェア研究所に所属．通信ソフトウェアの設計法に関する研究に従事．プログラム合成，知識表現などに興味を持つ．電子情報通信学会会員．

山中顕次郎

昭和 38 年生．昭和 61 年筑波大学第三学群情報学類卒業．昭和 63 年同大学修士課程修了．同年日本電信電話(株)入社．以来，NTT ソフトウェア研究所において，プロトコル合成，検証，FDT の研究に従事．ACM，電子情報通信学会各会員．