ソフトウエア工学 34－5
（1984. 2. 9）

ソフトウェア開発と保守作業の形態の同質性について

落水 浩一郎　今泉 恵美子
（静岡大学 工学部 情報工学科）

1．はじめに

人の作ったプログラムを理解し，変更するのは実に骨のおれる仕事であり，完全に理解するのは不可能ですらある。一方，自分の作ったプログラムを再理解し，変更するのはさほど困難な仕事ではない。プログラム理解のためにもっともよい方法は，作った人にわからない点を聞くことである。自分の作ったプログラムがよく理解できる理由は，プログラムを作ろうと決めた時からソースコードができあがるまでに，決めてきたことと，その間の関係がよくわかっているからである。人の作ったプログラムを理解しようと思ったら，理解する人は，そのような過程を再現する必要がある。このとき，誰もがとは言わないまでも，これから説明しようとすることについての理解者の知識をあまり仮定せずに，たいていの人にはわかってもらえるような文章や説明を書くことはむずかしい。一方，何か質問されたとき，何を聞かれたかを正しく理解することができれば，その質問に正しく答えることができる。「作った人にわからない点を聞く」のが，一番よく理解できるのは，説明者が積極的に，相手は何の説明を要求しているのか（すなわち，何がわからないのか）を理解してくれる点にある。プログラムの保守に関して障害となっている上記2つの問題に対して，解決への一つの方向を考察するのが本小論の目的である。

上記の前半の問題は，ソフトウェア開発過程の定式化とその記録法に関する問題であり，プログラムを作る過程で何がどのように決定されていくのかを明らかにすると共に，その表現手段を与える必要がある（図1①）。後半の問題は，記録法（図1②）とその検索法（図1③）の問題である。ところで，図1において，②の文書量が圧倒的であることと，作成者が何らかの理由で存在しない状況を前提とするとき，計算機支援の必要性がでてくる。このとき，情報の流れは，①→②③→④から，①→⑤→⑦→⑥→⑦→⑧→④とかわることになる。図1のような支援システムの実現可能性に関して，①の内容さえ定義できれば，⑦を⑥に写し，マンマシンインタフェース部の表現法を定める（⑤，⑧）ことはできる。⑦そのものの実現（人とシステムの会話）は困難である。しかし，理解者が，提示される事実をもとに，①の筋道を組み立てうるように⑥を編成し，⑧を検索システムとして実現することは，それなりに効果がある。いずれにしても，その成否は①にかかっており，次章以降に1つの接近法を示す。

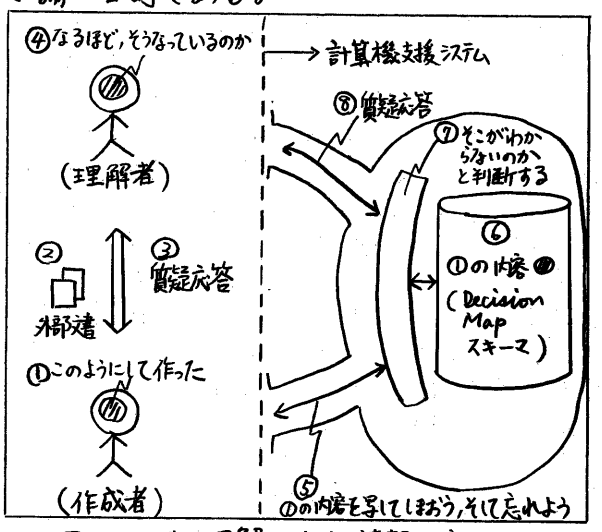

図1 プログラム理解に関する情報の流れ

2．ソフトウェア開発過程の定式化にむけて --- Decision-Map 法

文献(1)の13章の例題(表1)を用いて，図1①に対応する，我々の方法論 Decision-Map (決定地図) 法を説明する。

2.1 要求定義と関連

要求定義とは，現実世界に存在する情報処理に関わる問題の本質(needs)を，現実世界に関わる人々が様々の観点(利益とコスト，管理と運営，作業形態)から現象的に表現してくるものを素材としてつきとめ，それに対する解決法(requirements)を計算機システムという道具を使って与えることである。要求定義以前には活動の主体であった，人間の行動や利用するデータが計算機ソフトウェアとして実現され，人間はシステムのオペレータとなるので，その活動性のみを高め無用の混乱をひきおこさないためには，現行の運用状況で不変に保つべきもの(例えば，データ入力のタイミング)を併せて認識する必要がある。要求定義は，「誰が何処にいて，どういう仕事をしているか，それは何のためか，何故そのような手順になるのか」という問かけから始まり，「誰が何処にいて，どういう仕事をすることになるのか，その時，どのような機能，性能をもつ計算機システムを利用することになるのか」を定義しておわる。Decision-Map 法では，～部の定式化を最初の記述対象とする。この記述体系を RMAP (Map for requirements definition，要求定義地図) と呼ぶ。RMAP の記述法自体は W. Kent による情報システムの概念モデル(2)を基礎にして構築する。

ラインプリンタ（行印字機）があるものとして，その制御は，つぎの行の左端を“いま印刷できる位置”に定めるコマンド“NLCR (New Line Carriage Return, 復帰改行)”といま印刷できる位置に整数のパラメタ n の値で指定される文字を印刷し，印刷した位置のすぐ右の位置を新たにいま印刷できる位置に定めるコマンド“PRSYM (n)”の2つによっています．(いまの議論では，限りなく長い行も許されるものとします．)また“space (空白)”と“mark (印)”という2つの特定の n の値を用います．“PRSYM (space)”により，いま印刷できる位置を空白のままにし，“PRSYM (mark)”によって，たとえば星印のようなあるみえる文字を印刷するものとします．

さらに，整数の引数をもつ整数値関数が2つあって，

$0 \leq i < 1000$ に対して

$0 \leq fx(i) < 100, \quad 0 \leq fy(i) < 50$

を満たすものとします．

さて，y 軸として上から下に49から0まで番号をふった50行を印刷するプログラムを作るものとします．ただし，各行は，x 軸として，左から右に0から99まで番号をふるものとします．そして1000個（一致した場合にはそれ以下）の位置には，

$0 \leq i < 1000$ なる i に対して，

$x = fx(i)$ かつ $y = fy(i)$

を満たせば，印を印刷します．それ以外の位置は，空白のままです．換言すれば，1つの曲線が離散的なパラメタの表現で与えられていて，ラインプリンタをプロッタとして使いたいのです．

表1 問題の定義（文献(1)より引用）

図2において，repository は「情報を静的な意味で保持するもの」，interface は「user/processor 間での情報表現の媒体変換を行うもの」，processor は「interface を通じて流れこむ symbol 列をうけとり，repository 中の情報に対応させ，その内部の情報を変更したり，検索しながら，再び，interface を通じて流れだす symbol 列をつくりだす」。

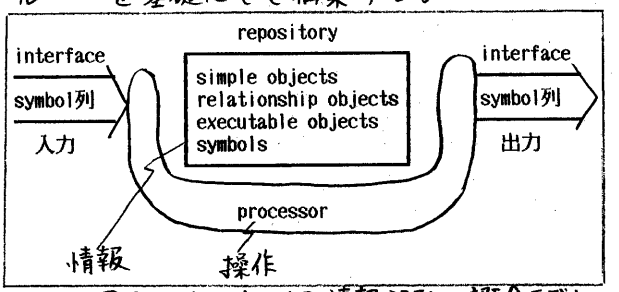

図2 W. Kent による情報システムの概念モデル

repository は4種の object (representative) からなる。symbol は「interface を通過できる唯一の object であり，他の object とつながれて，名前，記述，表現等の役割を果たす」，executable object は「processor が実施する制約，暗黙の定義，導出の表現，ある種の存在テスト，等価テストの具象物をあらわす」，simple object は「実体以外の何ものでもない」，relationship object は「情報の実質のないてであり，4つの object を結びつけるもの」である。彼の概念モデルの特徴は，relationship (関連) の取り扱い方にあり，現実世界の「情報とその操作」の定義には最もふさわしいと，

我々は判断する。すなわち，2つ以上のものがあり，それが結合したときに，通常の意味での情報が生まれるとし，その結びつきの根拠を関連型，ある結びつきに対して，結びつくものが果たす役割をrole，ある役割をもって結びつきに参加できるものの範囲をdomainとして「情報」の概念を定式化した(表2)。

表1の問題をRMAPで表現しなおすと図3のようになる。図3において，"□"と"○"はsimple objectであり，"□"と"○"の間には，"□"は"○"の型であるという関連がある*ことを"◎"のように表わしている。"⬭"は関連型object，"⟷"は関連実例object(リンク)，"⟿{×××}"はexecutable object，下線をひいた文章や記号はsymbolを表わしている。

関連型：ROLE (DOMAIN) ；・・・；ROLE (DOMAIN)

表2　関連型の表記法

表1中の静的情報を図3(イ)のように定義したとき，その操作の定義は図3(ロ)のようになり，(本例では存在しないが)interfaceを流れるsymbol列のユーザ側の表現(入出力仕様)を与えれば，要求仕様定義は完結する。

RMAPによって，表1の問題は，「設計上の意志決定と対応して議論できるように，十分に基本的な定義単位に分解された」ことになる。

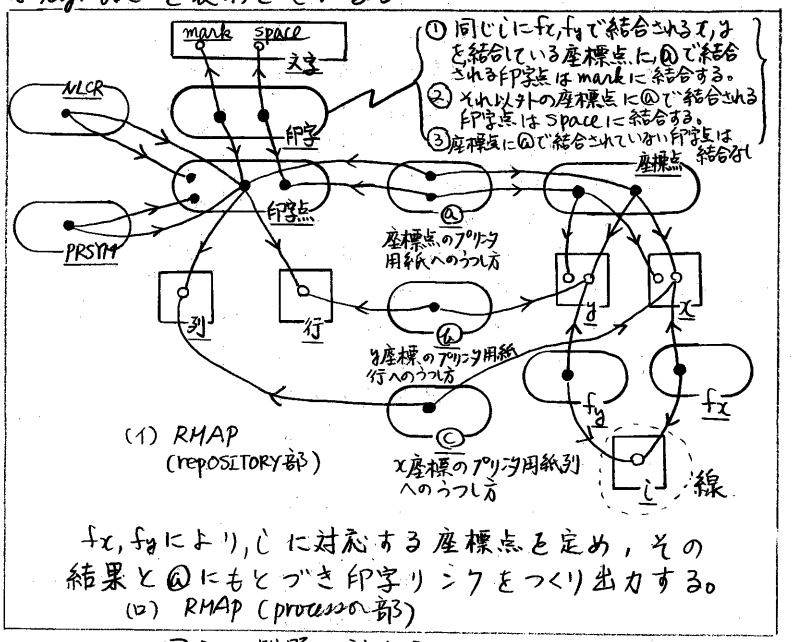

図3　例題に対するRMAP

図3の関連について説明をする。(1)関連型NLCR：現在位置(印字点)；実行後の位置(印字点)およびPRSYM：現在位置(印字点)；実行後の位置(印字点)はラインプリンタの動作特性を定義している。(2)関連型印字点：たて(行)，よこ(列)はラインプリンタ用紙を定義している。(3)関連型印字：印刷位置(印字点)；印刷される記号(文字)はラインプリンタの動作結果(出力)を定義している。(4)関連型座標点：x軸(x)；y軸(y)は2次元の座標空間を定義している。(5)関連型@：座標上の点(座標点)，用紙上の点(印字点)は座標空間とプリンタ用紙の対応を定義している。以上の関連型群は座標空間とプリンタ用紙を図4のように対応させていることになる。座標空間上の曲線とプリンタ用紙上の曲線は，executable objectの①，それ以外の印刷部分と座標空間の対応は②，印字に無関係な点間の対応は③によって規定されている。座標空間内の"曲線"の情報はLの実例の集合である。RMAPは，要求定義者の持つ，問題解決への視点，設計への要請を「関連」という地図上

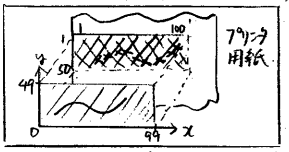

図4 座標空間と用紙の対応

* 「型である」という関連も，本来は他の関連と同一の記法で書くべきであるが，見た目のわかりやすさのために，図3のような便宜的記法を使った。

の記法を用いて明示することができる。

2.2 設計と仮想機械

1章で述べた問題は，プログラム設計に関しては，E.W. Dijkstra の structured programming(1) によって既に解決されている。設計とは，要求仕様の情報を利用しつつ，ソースコードをつくりあげるに必要十分な情報を得るまで，一つ一つ実現のための意思決定を積み上げていく過程である。structured programming においては，設計過程における一つの意思決定は，図5に示す仮想機械（決定の型）として表現される。

良い決定の連鎖をつくりあげるための手段が stepwise refinement であり，Dijkstra の方法の1つの核心である。

プログラムは正しく動作するように，効率がよいように，後の修正に耐えうるように作られなければならないということを目標におき，人間の特性（量に弱い），プログラムの理解要因の分析に基き，設計上の意思決定が，できあがりつつあるプログラム（ソースコード）にどのように反映されるかを確認しつつ書き残し（図5），それによって次に決定すべき範囲を定めていく方法を定式化した。図5は，開発時には作成者の頭の中にひらめき，ソースに変換され，そして書き残されない傾向があるが，これこそ，作成者のみが知っており，再現に苦労する構造である。その意味で，図5に示す一群の仮想機械を，Decision-Map 法ではそのまま採用し，DMAP（Map for design decisions，設計地図）と呼ぶ。このとき，設計とは，「RMAP から DMAP への変換である」と定義できる。記録の立場からは，DMAP が生みだされる過程（変換過程）も書き残す必要があり，文献(1)においては説明の形で記述された内容を Decision-Map 法では，以下のように formal に書き残すことができる。図3(a)に述べた要求仕様の機能表現（静的情報の動的操作特性）をまず分析する。このとき，しをもとに印字リンクをつくりこれを出力することが要求されていることを確認できる。このとき，まず，しをもとに f_x, f_y を計算し，曲線に対応する座標点を定めておき，この結果と，②の対応をもとにして executable object の制約に基づいて，印字リンクを生成することになる。一方，ラインプリンタの動作特性と，f_x, f_y の計算順序が一致しないことがわかり，印字リンクは「すべて生成したあと，出力すべきである」という結論に達する。

```
COMPFIRST (1)
begin
    draw: {build; print};
    var image;
    instr build (image), print (image)
end
```

```
CLEARFIRST (2)
begin
    build: {clear; setmarks};
    instr clear (image), setmarks (image)
end
```

```
ISCANNER (3)
begin integer i;
    setmarks: {i := 0; while i < 1000 do {add mark; i plus 1}};
    instr add mark (i, image)
end
```

```
COMPPOS (4)
begin integer x, y;
    add mark: {x := fx(i); y := fy(i); mark pos};
    instr mark pos (x, y, image)
end
```

```
LINER (5)
begin integer j;
    image: {array line [0:49]};
    print: {j := 49; while j ≥ 0 do
                {lineprint (line [j]); j minus 1}};
    clear: {j := 49; while j ≥ 0 do
                {lineclear (line [j]); j minus 1}};
    mark pos: {linemark (line [y])};
    type line;
    instr lineprint (line), lineclear (line), linemark (x, line)
end
```

```
LONGREP (6)
begin integer k;
    line: {integer array sym [0:99]};
    lineprint: {k := 0; while k < 100 do
                {PRSYM (sym [k]); k plus 1}; NLCR};
    lineclear: {k := 0; while k < 100 do
                {sym [k] := space; k plus 1}};
    linemark: {sym [x] := mark}
end
```

図5 例題のDMAP（文献(1)より引用）

この結果，主記憶上に印字リンクを貯えておくための記憶領域 image の必要性が認識され設計上の最初の決定として記録される（COMPFIRST，図5(1)）。次の決定は印字リンクの生成に関する制約から生みだされる。すなわち，図3の関連型印字に関する executable object の内容を検討すると以下のような結論がえられる。「しから直接計算されるのはのに基づく印字リンクである。②に基づく印字リンクと有効に認識するためには，あらかじめすべての印字リンクを特定の状態にしておくと都合がよい」（CLEARFIRST，図5(2)）。次にiの走査の順番が定められる（ISCANNER，図5(3)）。さらに，座標空間上の位置が定められる（COMPOS，図5(4)）。ここまでの決定は，image のどのような詳細化にも依存しないように作られており，image を詳細化した一群のプログラム family に共通の特性となる。次に，image が本来有している構造の詳細に注目しつつ，行毎に印刷が進行していくのを考えあわせて，image は {array line[0：49]} と詳細化される。

その結果，以前の決定 print，clear，mark pos が一斉に詳細化される。この決定は LINER（図5.(5)）として記録される。配列の添字に座標軸の目盛り，ループの制御変数にプリンタ用紙の行番号が，関連型㋑を利用して与えられる。同様に1つの行の構造を詳細化することにより（LONGREP），設計上の意志決定はすべて完了する。

2.3 仮想機械からソースコードへの変換

仮想機械からソースユードへの変換は，仮想機械中の名前の展開による（図6）。

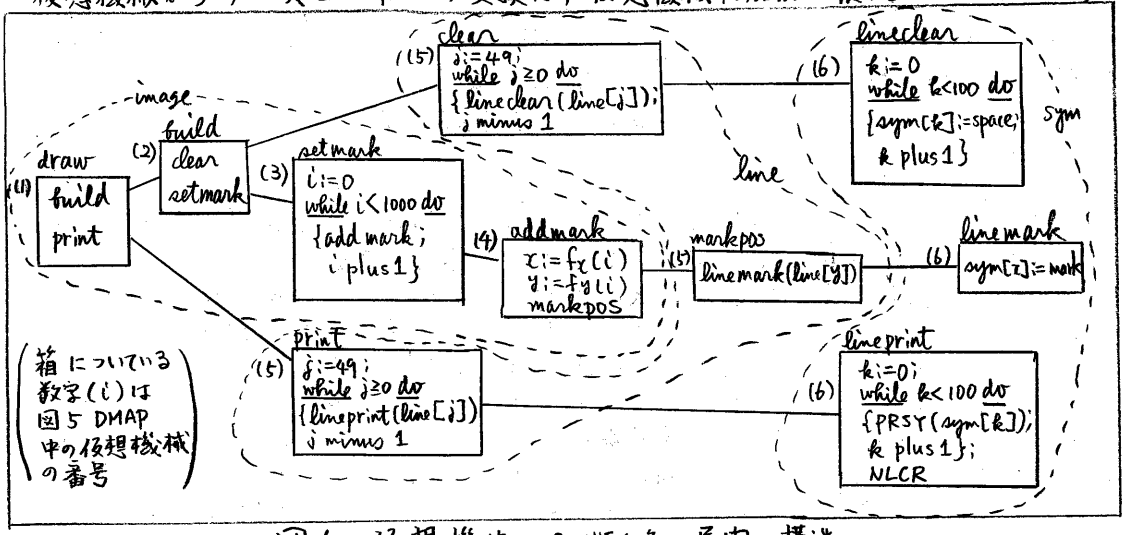

図6　仮想機中のプログラム名の展開の構造

図6を展開し（図7(1)），併せて，image,line を展開し，高級言語の syntax 要素にあわせて簡単な手直しをおこなえば，ソースユードが得られる。展開の際，ソースリストとしてのまとまり（コンパイル単位）は考慮する必要がある。

3. 保守とは

開発と保守の背後にある基本的な構造は，2章で述べてきた，「決定の構造」である。開発とは「新しく決めていくこと」であり，保守とは「決定を変更（再決定）することである。開発時にも決定の変更はなされるので，開発と保守の違いは，既に決められているものと，新しく決めるものの量的な差にすぎない。図7は Dijkstra による，開発時の変更の例であるが，本質的には，保守時の変更

とまったくかわらない。

すなわち，ユーザが図4レベルの概念で，動作状況に関する改善要求を出したとき(図7①)，RMAPを利用して，対応する情報とその操作部分を特定する。この場合，印字リンクのつくり方に関する制約 executable objects に関する変更要求であることを見つける。つぎに，RMAPからDMAPへの変換過程の記録を利用しつつ，DMAP中のLONGREPの特定にいたる(②)。図5と図6の関係より，対応するソースコード部を特定しておき(③)，LONGREPにかわる新しい意志決定SHORTREPをつくり(④)，対応するソースコードを入れかえる(⑤)。

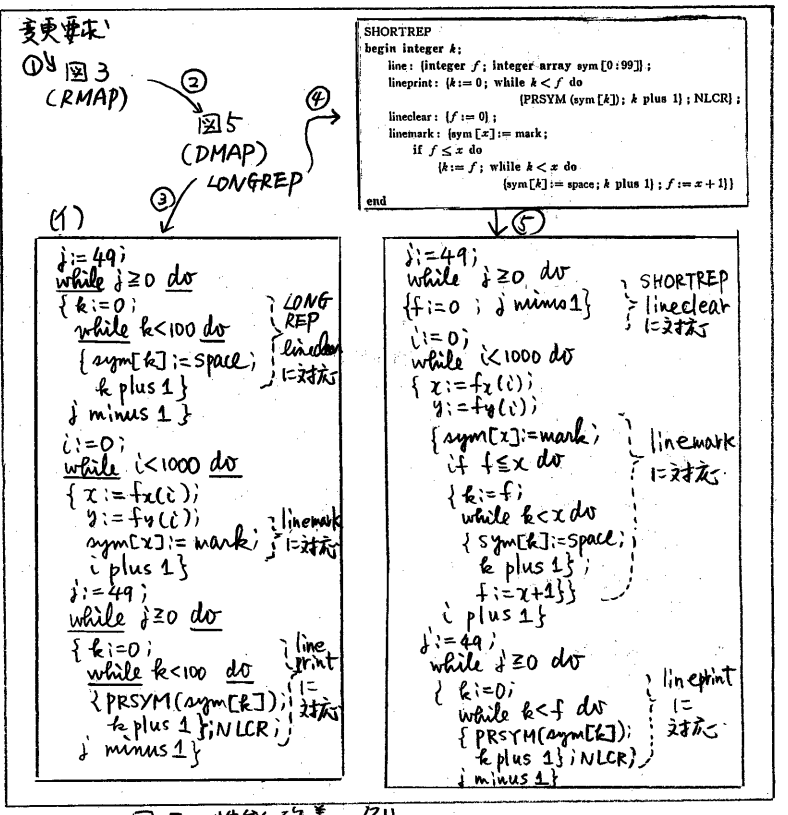

図7 性能改善の例

4. おわりに

ソフトウェアは，変更するためにつくるべきである。開発時の変更，製造後の欠陥除去，機能改善，機能追加，できれば新環境への適応などの，ライフサイクルに沿って出現する変更要求に柔軟に対応して改造していけるべきである。

本論文では，「作ること」と「変更すること」を同じレベルで議論できる，ソフトウェア開発方法論，Decision-Map 法を提案した。本方法がシステム作りに適用できることは，2つの事例研究により確認した。[3][4] 図6レベル以下のソフトウェアツール DIFF[5]は実験的開発に成功している。本文中に述べた，計算機支援システムに対する要請は原理的なものであり，図1⑥部に関する考察である。紙面の都合上，計算機支援システムの形態的側面（図1⑤,⑦,⑧）については，ふれられないが，その基本的要請のみを明らかにしておく。「ソフトウェア開発は，RMAPの作成，DMAPの作成，ソースコードの実現というふうには進まない。むしろ，2.2，2.3の作業が並行的に進むことに特徴がある（先に進んでみなければわからない）。このとき，RMAPの分析，定義，DMAPの形成は作成者の頭の中に埋没する傾向がある。RMAP，DMAP間の情報の参照・引用機能，およびそれを保存する機能は充実されるべきである。2.3以降の作業は機械にやらせたい」。

文献 (1) E.W. Dijkstra, "Notes On Structured Programming", 「Structured Programming」, ACADEMIC PRESS(1972. 野下，川合，武市訳 "構造化プログラミング" サイエンス社. (2) W. Kent, 「Data and Reality」 North-Holland, 1978. (3) 今泉，落水，"ソフトウェア開発過程の記録法の一事例"，本研究会資料 32-1. (4) 落水，田中，平田，今泉，"電子メイルシステムにおけるメイルボックス設計の一手法"，本研究会資料 33-1。(5) 浦井，落水，"版管理機能を有する構造プログラミング支援エディタ DIFF"，情処論（投稿中）