数理解析研究所講究録
第525巻 1984年 292-314

部品化ソフトウェア ～ その概念と課題 ～

富士通(株) 国際情報社会科学研究所
小林 要 (Kaname Kobayashi)
中川 徹 (Toru Nakagawa)

1. はじめに

どのようなソフトウェアを開発すべきなのか、という問題は、ソフトウェア工学にとって、最も基本的な課題の一つである。一方では、ソフトウェアの利用者側の要求を分析することが重要となり、他方、ソフトウェアの生産者側には、生産性や品質、および開発の方法論が問われることになる。

『部品化ソフトウェア』の概念は、ソフトウェアのあるべき姿の一つとして考えられたものであり、ソフトウェア工学の技術開拓や目標設定の役割の一端を担うものと考える。また、ソフトウェアを開発するに際して、モデル構築を行ううえでも重要な役割を果すものと考える。

部品化ソフトウェアの発想は、ソフトウェアを考える視点の分析を通じて得られた。以下では、視点をどのように分析して、部品化ソフトウェアの概念を得たかを示す。

2. 開発過程をとらえる視野

ソフトウェア開発の実態をどのように把握するべきか。歴史的に整理してみると，ソフトウェア開発の作業実態そのものの変遷もさることながら、ソフトウェア開発の全体像をとらえる視野が広がってきたことに気が付く。

ソフトウェア開発をとらえる才1の視野は、図1に示すように、「プログラミング」として、人間の思考過程に重点を置いた視野であった。この視野で考える限り、書き易い、読み易い、理解し易いこと、など、人間一人一人の作業との相互作用の解決に力点が置かれがちである。

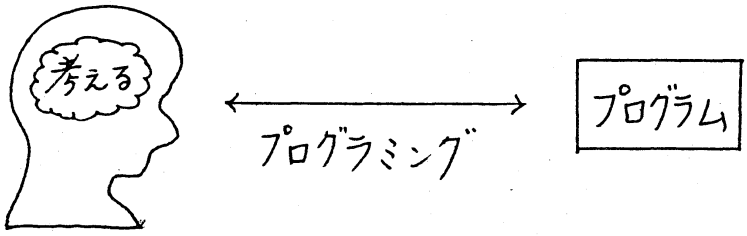


図1：ソフトウェア開発．才1視野

ソフトウェア開発をとらえる才2の視野は、図2に示すように、「ソフトウェア・ライフサイクル」として、「工程」という概念で考え直した視野であった。工程に分けることによって、作業内容の区別を行えば、作業の目標も詳細化され、技術も向上する。さらには，集団でソフトウェアを開発する図式が得られるようになったと言えよう。

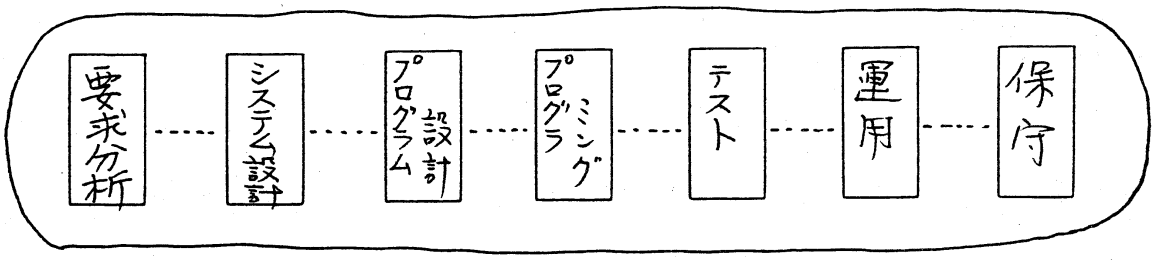
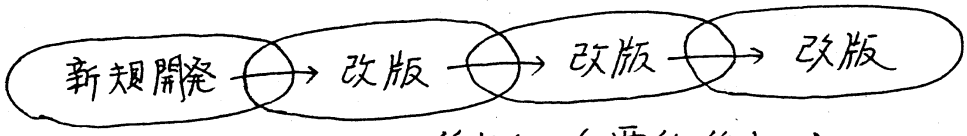


図2：ソフトウェア開発．オ2視野

ソフトウェア・ライフサイクルの図式に対し、実際の作業現場から、必ずしも現実に即していない面があるとの批判があった。工程図式は、新規開発には良いが、旧版の改良などについて対応しにくい、という批判である。ソフトウェアの蓄積が多くなるにつれ、改版などの作業も多くなった。このため、次の視野でとらえることが重要となった。図3に、オ3視野を示す。

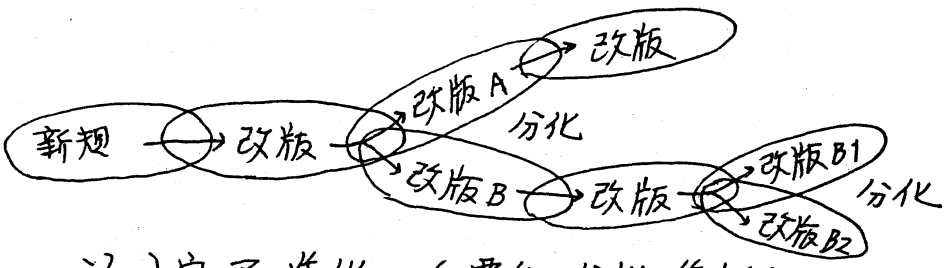
ソフトウェア開発をとらえるオ3の視野は、「進化」の視野である。進化を考える視野にも様々あるが、最初のとらえ方は、遷移としてとらえる視野であった。図3における、一つ一つの○印は、開発や改版などの単位に相当する。この図式によって、作業者が、かなり長期に渡って、同様なソフトウェアを相手にする実態が反映されてきた。



ソフトウェアの進化（遷移進化）

図3：ソフトウェア開発．オ3視野

ソフトウェアの進化も、時が経過するにつれ、遷移的な進化だけでなく、枝分かれし、分化する。このため、ソフトウェアの進化を念頭に置いたツール群も新たな対応がせまられることになる。図4に、オ4の視野を示す。



ソフトウェアの進化（遷移・分化進化）

図4：ソフトウェア開発．オ4視野

オ4の視野で開発をとらえ直してゆくと、ソフトウェアの構成要素の管理の重要性に気がついてゆく。また、類似性の高いソフトウェアが複数存在してくる。ソフトウェア・ファミリーという概念が必要になってきた。しかし、これでも不足な面が生じる。それは、ソフトウェアの類似性を考え直す視点の不足である。

ソフトウェア開発のオ5の視野は、類似性の高いソフトウェア群の進化をとらえようとする視野である。この場合、進化の中に、遷移、分化以外に、統合という視点が加えられる。いわゆる離合集散が生じるのである。図5にその オ5視野を示す。

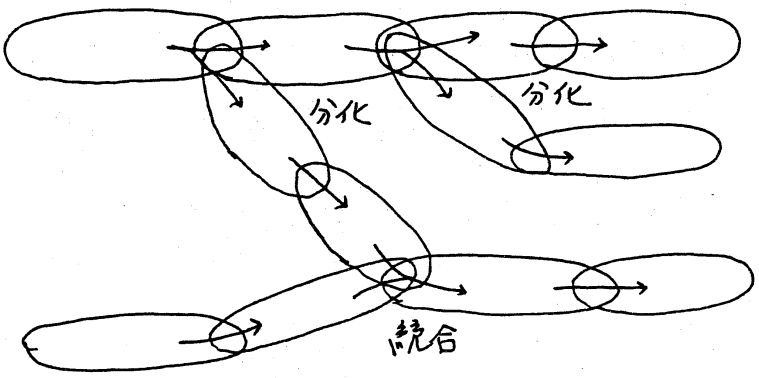


図5：ソフトウェア開発、オ5視野

図5に示すオ5視野は、様々に解釈できる。例えば、いわゆる既存ソフトウェアの部分の流用、再利用、あるいは、機能の統合、整理などが説明される。一方では、類似性の高いソフトウェア群の全体的な管理の必要などである。

ここで重要なことは、オ5視野を前提にしたときの新しい技術である。つまり、複数の類似性の高いソフトウェア群が統合、分化、遷移をすることが予測される場合、どのようなソフトウェアを開発してゆくべきかという技術課題がある。「部品化ソフトウェア」は、この技術課題に対して答えようとするものである。

3、ソフトウェア製品をとらえる視野

ソフトウェア開発をとらえる視野は、開発行為に注目した場合の視野であった。これに対し、開発される製品に注目し

た視野の広がりがもう一方にある。図6は、製品をとらえる基本的な視野を示す。これをソフトウェア製品の才1視野と呼ぼう。

製品の才1視野では、生産者と消費者の図式が採用されている。ソフトウェアの場合の消費者は、一般に、利用者と呼ばれる。生産者が提供したソフトウェア製品を利用者が使用する、という素朴な発想である。

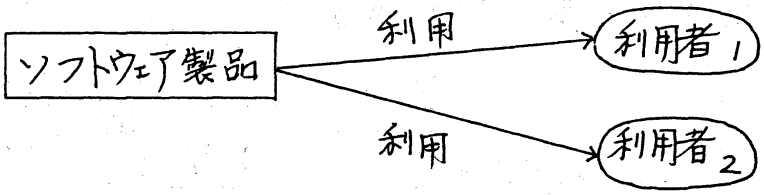


図6：ソフトウェア製品、才1視野

しかし、ソフトウェア製品の利用の実態は、必ずしも才1視野におさまりきらない。利用者には、利用者個別の実行環境があるために、各々の実行環境に適した形態に修正、または適応させてから実際に利用されることが多い。特に、ソフトウェアパッケージ製品の場合には、カストマイジングと称して、ソースプログラムの手直しが行われることが多い。

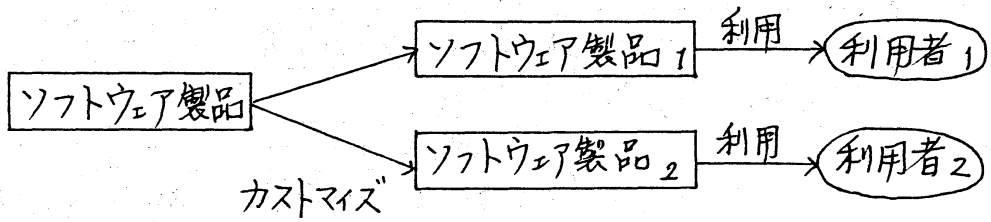


図7：ソフトウェア製品、才2視野

図7に、ソフトウェア製品の才2視野を示す。製品の才2視野の特徴は、製品にも、元となる製品と、カストマイズされた製品とが存在することになる。元となる製品には、カストマイズ容易であることが要請される。

製品視野を考える場合、前節で述べたソフトウェアの進化を考慮に入れる必要が生じる。ソフトウェアの進化は、製品のうち、カストマイズ後の製品にまず生じ易い。図8は、この点に注目した才3の製品視野である。

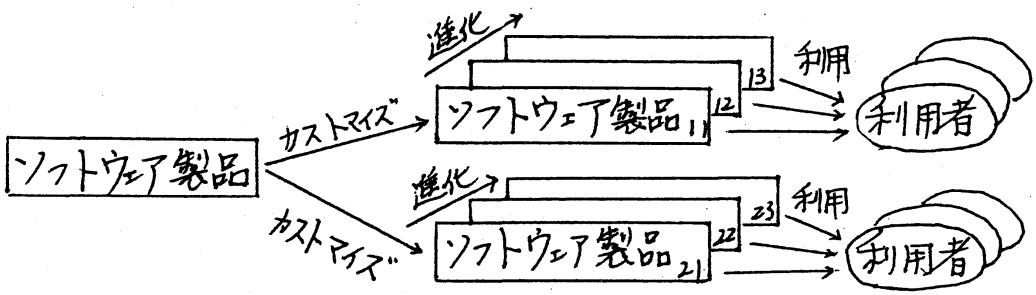


図8：ソフトウェア製品、才3視野

元のソフトウェア製品も当然、進化するであろうから、実際には、図9の才4視野で考える必要が生じる。この場合、どの版のどこをカストマイズしたか、という点で複雑な状況を呈する。

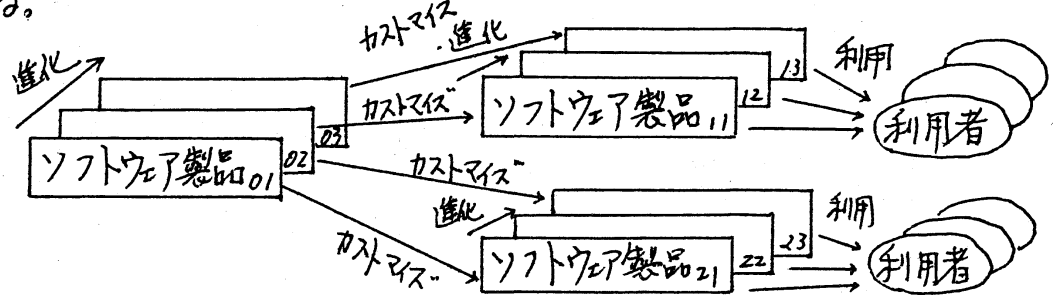


図9：ソフトウェア製品、才4視野

図5のようなソフトウェア進化の視野と組み合わせると、さらに複雑な状況になることがわかる。これを、ソフトウェア製品オ5視野と呼ぶことにしよう。図で表現するには複雑になりすぎて混乱するので、図は略す。

このように、ソフトウェア製品をとりまく状況は、進化とカストマイジングにより、類似性の高いソフトウェア群をなす状況にあると言える。部品化ソフトウェアのねらいは、かような、類似性の高いソフトウェア群に対し、効果的に対処しようとする点にある。

4. 変化への対処

カストマイジングは、元の製品を母体に、複数の利用者へ変化させる空間的な変化としてとらえることができる。進化は、製品の時間的な変遷としてとらえることができる。問題は、図10のような、時間的、空間的な変化の中でソフトウェアをとらえ、かつ、類似性の高いことを考慮に入れて、変化にどう対処するべきかということになろう。

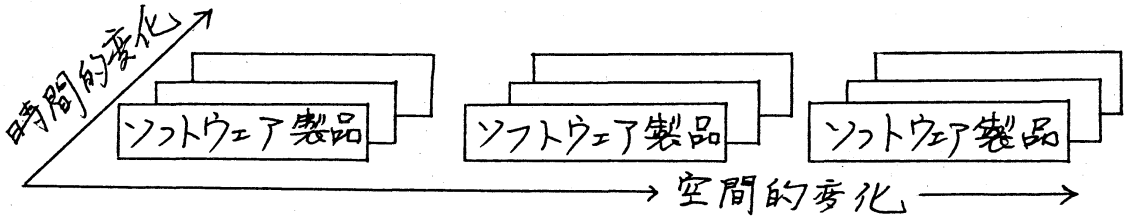


図10: ソフトウェアの変化

時間的変化は、将来の変化の予測と、過去の蓄積の活用という2つの課題を提起する。一方、空間的な変化は、どのような多様性があるかを分析することと、多様性に対処するカストマイズの技術開拓という2つの課題を提起する。

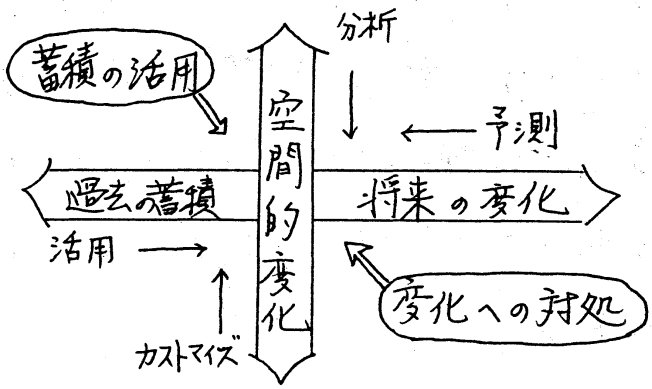


図11：変化への対処と蓄積の活用

図11は、ソフトウェアの変化への対処、蓄積の活用の重要性を示すものである。部品化ソフトウェアの考えは、この図11の課題に答えるためのものである。すなわち、ソフトウェアの変化に対処するには、全体的変化を部分的変化で吸収し、全体的個別性を部分的汎用性に分離し蓄積、再利用しようとする方向で、それも計画的に行おうという考えである。

ソフトウェアを部分に分ける技術は、ソフトウェア工学の歴史の中でも、重要な役割を果してきた。次節で、その歴史的発展をふり返ってみることにする。

5. モジュール分割技術への反省

表1にモジュール分割の考え方の進展を表にしている。M. V. Wilkesにはじまると言われるサブルーチンの概念は、抽象化という発想の原点になったと考えられる。Dijkstra と Hoare [1] は、抽象化にも構造があることを示し、構造化という発想の原点となった。構造化の対象は、制御構造，データ構造の両方であり、ともに、連接、選択、繰り返しという基本構造が重要であることがわかった。これら3構造要素をファイルにも導入したのがJackson 法であろう[2]。

モジュラー・プログラミング[3]、複合設計[4]、デシジョンテーブル[5]などは、いわゆるモジュール化の指針を与えようとするアプローチであり、構造化の考え方とあいまってプログラミング方法論の議論をかもし出した。

Dikstra [6]、Parnas [7]、Liskov [8] の流れは、カプセル化と呼ばれる考え方で結ばれる。抽象機械を想定し、O関数、V関数、などの抽象機械操作命令群をインタフェースとするモジュール化によって、情報隠蔽、情報局所化を図ろうという考えで代表される。データ抽象化は、この考えをさらに一歩進めて、抽象データ型という型の概念に対応させようとしたもの[9]と考えられる。

最近の動向は、Smalltalk 80 [10] や Objtalk [11] などに見ら

表1. モジュール化技術の流れ

発想	方法	関連	年代
抽象化	サブルーチン呼出し	Wilkes	1940 1950
構造化	連接・選択・繰返し 構造化プログラミング Jackson 法	Dijkstra Hoare Jackson *	1960
モジュール化 (情報局所化)	モジュラー・プログラミング 複合設計 ディシジョンテーブル	Armstrong Myers London *	1970
カプセル化 (情報隠蔽)	抽象機械 O関数, V関数	Dijkstra Parnas Liskov	
データ抽象化 (型)	抽象データ型 データカプセル化	Liskov Zilles	
対象化 (クラス化)	オブジェクト指向 プログラミング	Smalltalk Objtalk	1980
部品化 (ソフトウェア部品)	ソフトウェア部品 部品化ソフトウェア	電子協[12]化 富士通	

※印は年代不整合

れるオブジェクト指向プログラミングであろう。そこには、対象化、あるいは、クラス化の発想があり、抽象化の構造がさらに充実してきたといえる。

部品化の発想は、モジュール化の流れとしてとらえきれな

い側面を持つが、本研究で目標とする部品化ソフトウェアの構築には、新たなモジュール分割技術が要請される。ただし、これまでの構造化、カプセル化、抽象データ型、クラス化、などの技術をふまえたものである。そこで、まず、これらの考えを整理しておこう。図12にこれら一連の流れを、ADT（Abstract Data Type）の"考え方"としてまとめている。

図12の横軸は、クラス化の軸、縦軸は、いわば構造化の軸である。クラス化は、より一般形にすることを意味し、構造化は、構造的な取扱いをより充実させることを意味している。図12は、あくまでも、データ構造に重点を置いた発想であって、ソフトウェア一般を述べているわけではないが、歴史的な考え方の流れを認識するうえで重要であろう。

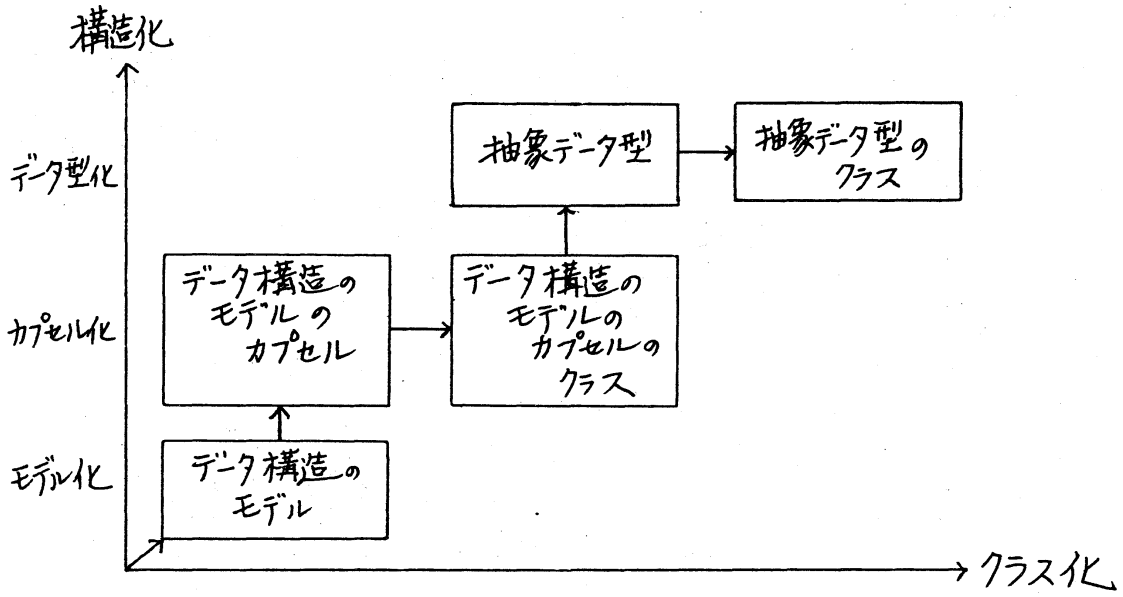


図12：ADT（抽象データ型）の『考え方』

6、ソフトウェアの3側面

ソフトウェアを部分に分けようとする際に、モジュール化の技術が教えてくれた着眼点は、情報の扱い方であったと言える。例えば、情報の局所化、情報の隠蔽、あるいは、型宣言による情報の明示などである。しかし、ソフトウェア単品の分割にとって重要なこれらの着眼点だけでは、部品化技術は構築できない。ソフトウェア群を相手に、何をどう分割でき、何が蓄積できたかを反省する必要がある。

図13は、ソフトウェアの実体を把握する際に不可欠な3つの側面を示している。機能的側面とは、ソフトウェアの意味を把握する側面である。一方、ソフトウェア実体は、何らかの媒体の上で記述、表現されて存在する。これを記述表現的側面という。さらに、何らかの実行環境の上でソフトウェアが実行されると、ハードウェアがコントロールされ、実際のダイナミックな挙動が生じる。これを性能挙動的側面という。

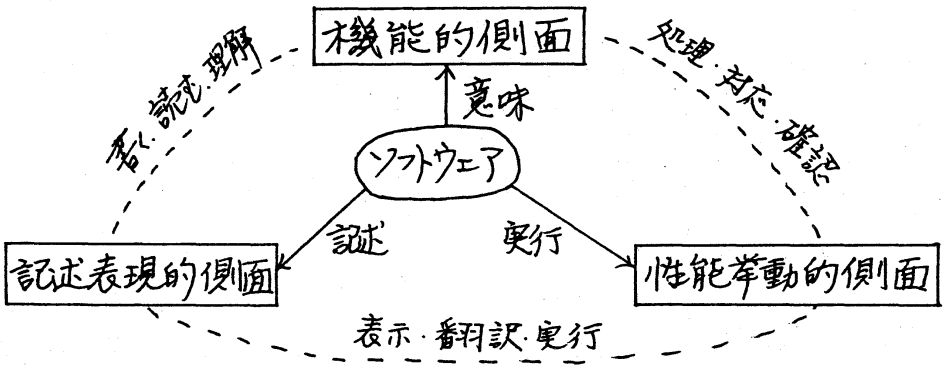


図13：ソフトウェアの3側面

ソフトウェアを部分に分けようとする場合、意味的に分けようとしても、記述表現の上でうまく分離できるとは限らない。また、挙動上のまとまりは、必ずしも意味的まとまりと一致するとは限らない。一方、ソフトウェアの蓄積は、記述表現の側面において物理的になされることになる。

逆に言えば、記述上のまとまりをいくら蓄積しても、それが、機能的まとまりの上で活用しにくい限り、役に立ちにくい。さらに、機能的まとまりと記述的まとまりが対応していたとしても、性能挙動的側面が良くなければ役に立ちにくい。3側面がうまくまとめられたソフトウェア部品を発見することは容易でない。種類や数も限られてしまうことになる。

まとめるレベルにも様々なレベルが考えられる。図14に示すように、システムレベル、(サブシステムを含む)、プログラムレベル、モジュールレベル、さらにはモジュールを構成するフラグメントのレベルがある。

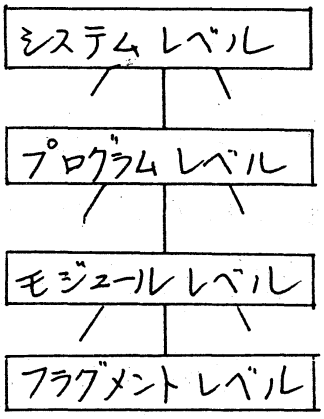


図14: ソフトウェアの構成要素レベル

どのレベルでどのように分けることができるのかを追及しなければならない。こうした背景を考慮しながら、ソフトウェアの部品化の技術を次に考えることにしよう。

7. ソフトウェア部品化技術

ソフトウェア部品化技術は大別すると2つのアプローチがある。1つは、『ソフトウェア部品ライブラリ・アプローチ』であり、もう1つは『部品化ソフトウェア・アプローチ』である。用語が類似しているので混同され易いが、発想の原点が違う。ライブラリ・アプローチの主眼は、蓄積の活用にある。一方、部品化ソフトウェア・アプローチの主眼は、進化とカストマイズへの効果的な対処方略にある。

図15に、ライブラリアプローチ、図16に部品化ソフトウェア・アプローチを示す。両者はともに、ソフトウェア群を考慮に入れ、群全体への貢献を目標とし、技術蓄積の活用、将来の多様な変化に対処しようとしている点で一致する。

7.1 ソフトウェア部品ライブラリ・アプローチ

図15において、まず、複数のソフトウェアの中から、類似性の高い部分に注目して抽出し、収集し、洗練して、ライブラリに登録する。部品ライブラリには、大きく分けて、3種類ある。一つはB型（ブラックボックス型）部品ライブラリ、である。これは、各部品の内部詳細に立ち入らずに再利用できるものをいう。次2は、P型（パターン型）部品ライブラリである。これは、記述パターンの再利用に主眼のあるもの

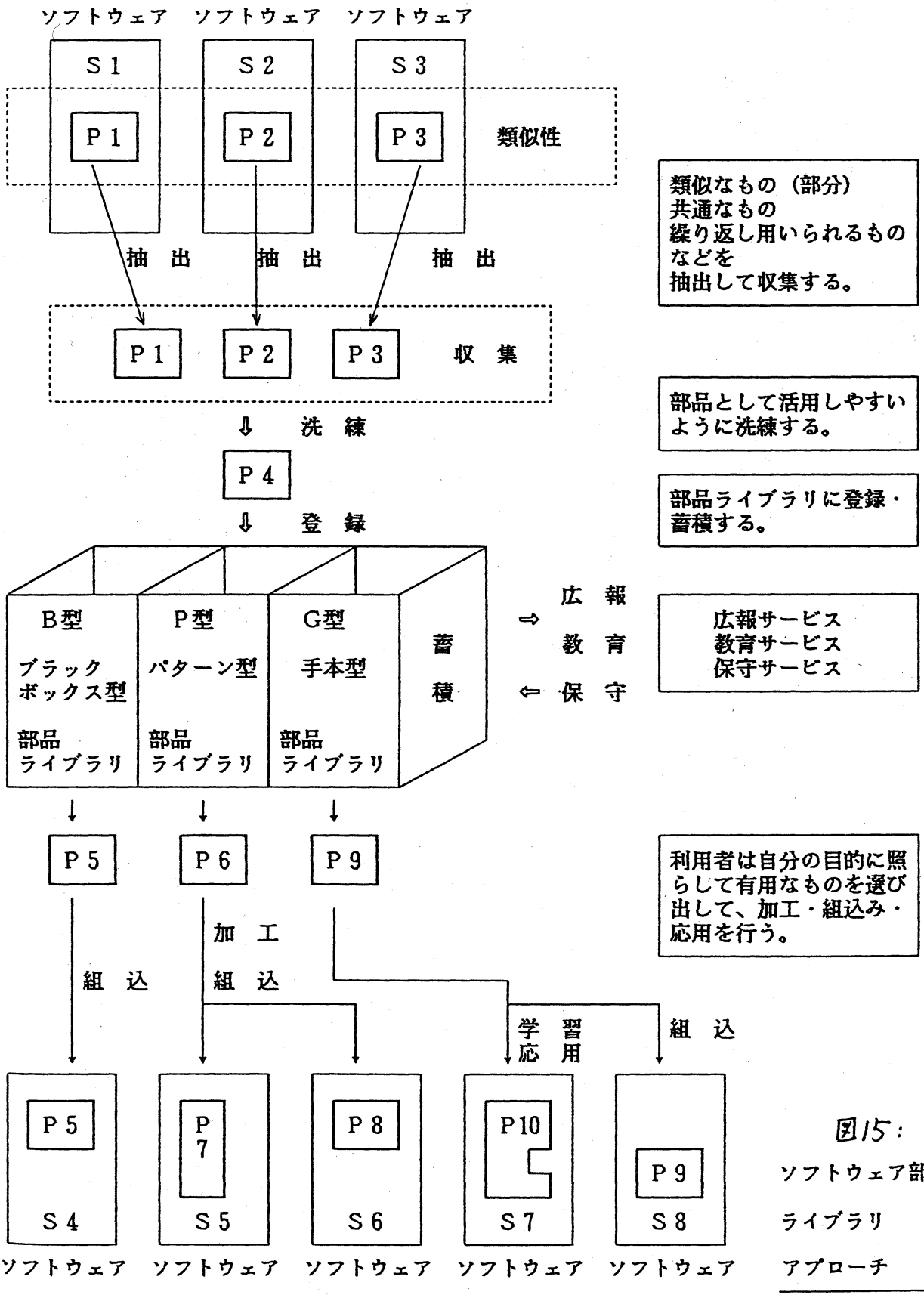
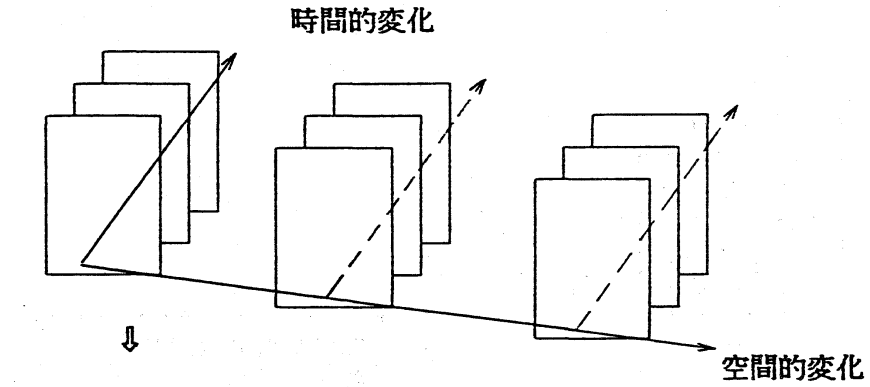
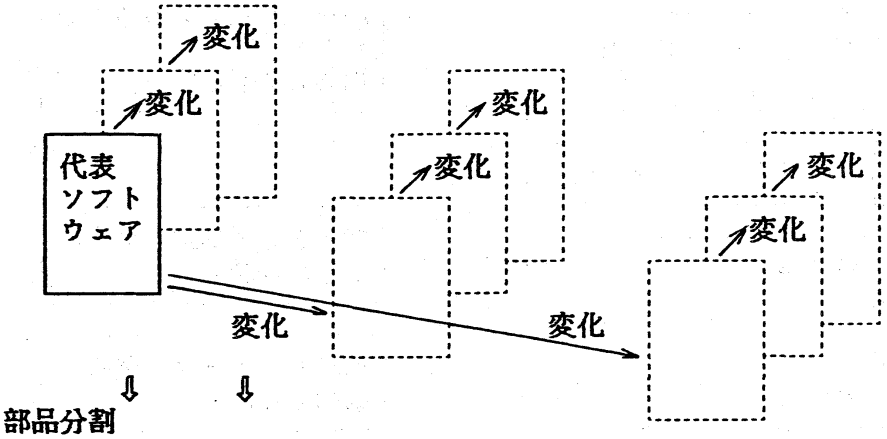


図15：
ソフトウェア部品
ライブラリ
アプローチ



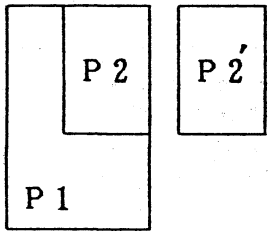
類似なソフトウェア群を想定する。



代表となるソフトウェアを考える。
これ以外は代表ソフトウェアからの変化と考える。

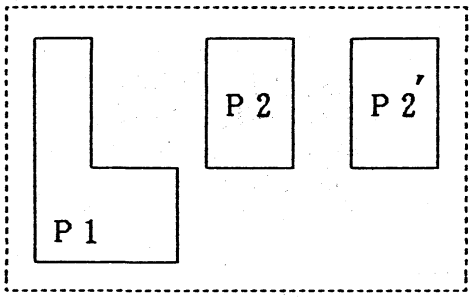
変化を考慮に入れて、変化に対処できるよう代表ソフトウェアを部品に分割。
さらに、変化に対処できるように、他の部品も準備する。

部品分割



変化に向けての部品

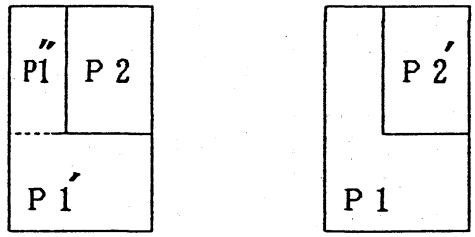
⇩ 管 理



部品化ソフトウェアの
データベース

これらの部品を統一的に管理
（データベース化）

⇩ 進化・その他　　⇩ カスタマイズ・その他



変化への対処

部品を組合せて変化に対処
（構成管理、等）

図16：部品化ソフトウェア・アプローチ

をいう。第3はG型（手本型：Good example のG）部品ライブラリである。これは、記述そのものや、機能そのものよりもその背景にあるテクニック、アルゴリズム、考え方、などの活用を意図するものである。

B型は機能提供、P型は記述労力提供、G型は技術知識提供に効果が大きい。それぞれ、使いがってが違う。B型は、組込み、P型は加工して組込み、G型は学習・応用して組み込み、などの作業を要す。

7.2 部品化ソフトウェア・アプローチ

図16において示すように、部品化ソフトウェアのねらいは、時間的変化、空間的変化を念頭に置いた類似性の高いソフトウェア群へいかに効果的に対処するか、という課題に接近する積極的な側面を持つものである。

そこではまず、代表的なソフトウェアを考えて、それを核にした形で、他のソフトウェアと比較し、変化部分を見定める。さらに、ソフトウェア群を生成するために必要な様々の部品を準備しながら、ソフトウェア分割を図る。その際、従来の単品生産と決定的に違う点は、分割され準備された部品のすべてを組み立てなくても、何らかの製品となりうるという点である。図16でいうとP2'は交換用部品として、変化の

ためにあらかじめ準備されたものであることを意味し、P1、P2だけで十分に1つの製品となることを示している。

このように、計画的に準備されたソフトウェア部品群は、類似性の高いソフトウェア群の生成に役立てられる。そのためには、さらに、部品群をデータベース化して、全体的な関係の中で一貫性を管理する。この管理のもとで、様々な進化やカストマイズに対処してゆこうとするのである。

8、部品化ソフトウェア

図16に示したように、部品化ソフトウェアのアプローチでは、計画的にソフトウェア群を生産しようとする多品生産の概念がある。したがって、必要な技術も、

① できるだけ、独立性の高い部分に分ける、ことの他に、
（従来のモジュール化技術）

② 当初から、複数のソフトウェア製品の実現を意図する、
（進化とカストマイズのための技術）

などの技術が重要となる。また、生産性の概念も、従来の、単品生産における生産性ではなく、複数品生産における生産性のワク組みで、とらえ直すことが必要となる。

部品化ソフトウェアは、そこから、複数の製品を産み出す母体としての役割を果すわけで、ソフトウェア群を産み出す

に必要なしかけを用意しなければならない。その場合、単に産み出し易いだけでなく、産み出されるソフトウェア群が、進化に適応し、カストマイズ要件にも適合することが要請されるため、その生成範囲も広くなければならない。

既存のソフトウェア技術の中で、部品化ソフトウェアに役立つと考えられるものは、図12のADTの考え方で示したような、クラス化の技術である。すなわち、類似性の高いソフトウェア群を何らかのソフトウェアのクラスとして実現し、それを母体に、ソフトウェア製品をインスタンシエート（実体化）する。

例えば、部品化ソフトウェア C_0 を、C_1, C_2, C_3 から成るものとする。　　$C_0 = \{C_1, C_2, C_3\}$

C_1, C_2, C_3 は、それ自身、何らかのクラスを代表しているものとし、クラス部品と呼ぶ。クラス部品は、様々に実体化できるとする。例えば、C_1 からは、P_{11}, P_{12}, P_{13}, のように生成できるとする。さらにまた、ソフトウェア製品とするには、C_1 のインスタンスと C_2 のインスタンスとでも実現され、C_1 と C_3 のインスタンスの組み合せでも別の実現が得られるとする。こうすると、図17に示すように、様々の組み合せが得られ、その中で進化やカストマイズの要件に適するものが多いほど、良い部品化ソフトウェアであるということになろう。

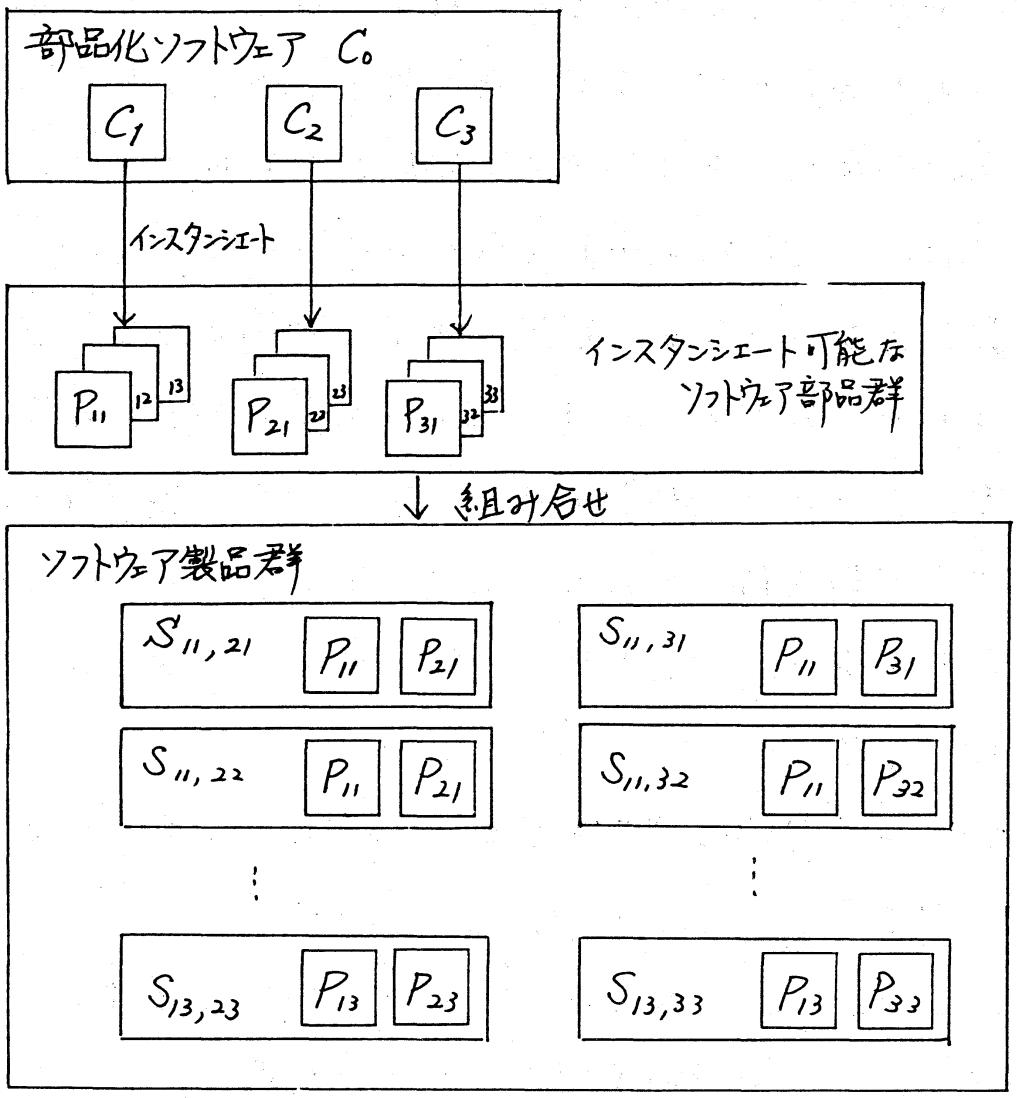


図17：部品化ソフトウェアのクラス部品による組合せ

部品化ソフトウェアC_0は、どのように作れば良いであろうか。進化やカストマイズのために将来の変化を予測することが必要であるが、どこまで予測できるであろうか。こうした問題のためには、開発対象とするソフトウェアの分野における、技術の蓄積をよく分析することが重要であろう。したがって、部品化ソフトウェアのためのモジュール分割の技術は、

ⓐ 独立化 （独立性の高い部分への分割）

ⓑ クラス化 （一般性を高める）

ⓒ 実体化 （クラス化されたものから逆に個別化する）

ⓓ 組み合せ （個別化されたものを組み立てる）

などを目標に置きつつ、対象とする世界の分析（普遍性の高い概念と、多様性の高い概念との分離、進化の歴史から得られる将来の予測など）を行うことのための技術に相当してくると考えられる。

もちろん、すべてを予測することは困難であるから、母体となる部品化ソフトウェア自身も進化することを念頭に置いて、できるだけ全体的変化を部分的変化によって吸収できるように工夫することが重要である。

9. おわりに

ソフトウェアの構造化の努力は必須ではあるが、その目的が定められない限り、選択の余地は大変大きい。部品化の努力は、部品化ソフトウェア・アプローチにおいては、変化への対応のための構造化が重要であると言える。その意味において、従来の『構造化ソフトウェア』の時代から一歩前進して、『部品化ソフトウェア』の時代に突入したのではなかろうか。

謝辞

本研究所北川敏男会長、辻ヶ堂所長付からは有意義な助言を受けました。社内のソフトウェア関連諸氏には、様々のアイデアを提供していただいております。謝意を表したいと思います。

参考文献

[1] Dijkstra,E.W., Hoare,C.A.R., and Darl,O.-J.: Structured Programming, Academic Press, (1972).

[2] Jackson,M.A.: Principles of Program Design, Academic Press, (1975).

[3] Armstrong,R.M.: Modular Programming in COBOL, Wiley (1973).

[4] Meyers,G.J.: Composit/Structured Design, Van Nostrand Reinhold, (1978).

[5] London,K.R.: Decision Tables, Auerbach, (1972).

[6] Dijkstra,E.W.: The structure of the "THE" - multiprogramming system, Comm.ACM,Vol.11,No.5, (1968).

[7] Parnas,D.L.: On the criteria to be used in decomposing systems into modules, Comm.ACM, Vol.15, No.12, (1972).

[8] Liskov,B.H.: The design of the Venus operating system, Comm.ACM, Vol.15, No.3, (1972).

[9] Liskov,B.H., and Zilles,S.: Programming with Abstract Data Types, Proc. Symp. VHLL., (1974).

[10] Goldberg,A. and Robson,D.: SMALLTALK 80, The Language and its Implementation, Addison-Wesly, (1983).

[11] Rathke,C.: ObjTalk-Primer, Stuttgart Univ., West Germany, (1982).

[12] 日本電子工業振興協会：ソフトウェアエンジニアリングに関する調査，ソフトウェアの部品化，56-C-416，昭和56年3月，(1981).