ソフトウェア工学　72－2
（１９９０．５．２９）

通信処理プログラムの移植に関する検討

中村　英児[*1]　佐々木　主税[*2]　松本　匡通[*2]

*1 ＮＴＴ、*2 ＮＴＴ情報通信処理研究所

移植を予め考慮したコーディングの移植性に対する評価について述べる。実際にコーディングがなされた通信処理プログラムを例に移植を妨げる要因の出現頻度、移植時の対処方法別出現頻度を調査し、移植を予め考慮したコーディングを行うことにより、移植時に変換すべき規模が１／４以下になることを得た。また移植を予め考慮したときの移植工数比の定量的評価方法を検討した。これにより、移植を考慮しない場合に比較して、システム全体の移植工数が約１／４以下に削減可能であることを得た。

A STUDY ON THE PORTABILITY OF COMMUNICATION PROCESSING PROGRAM

Eiji Nakamura[*1]　Chikara Sasaki[*2]　Masamichi Matsumoto[*2]

*1 NTT
*2 NTT Communications and Informations Processing Laboratories

1-2356 Take Yokosuka-Shi Kanagawa 238-03 Japan

It is described that programming in consideration of migration in advance increases program portability. Ratio of unportable elements is estimated , as an example , on a coded communication processing program . The program statements to be converted will become 1/4 by preconsidering migration. Quantitative evaluation method of migration labor is proposed and it is obtained that migration labor as whole system will decrease to 1/4 in comparison with nonpreconsideration.

１．はじめに

従来から既存ソフトウェアの有効利用や生産性向上のためにソフトウェアの移植の必要性が叫ばれてきた。移植性を高めるためにコンバータによる自動化や予め移植を考慮したコーディングによる方法がある。また予め移植を考慮したコーディングについてもその方策が多く述べられている[1]が、実例についてはほとんど報告されていない。さらに実際に移植を実施することによりどれほどの効果があるかを知る上で、移植工数の見積が重要である。現在までに移植工数についてモデルによる評価が報告されている[2]-[4]。参考文献[2]では移植工数(Ｅ)をプログラム規模(Ｌ)の関数$E = aL^b$(ａ、ｂは定数)として統計処理を行っているが、定数の値がシステムによって大きく異なっている。コンバータの効果を考慮した参考文献[3]では定量的な評価がなく、また参考文献[4]では定量的な値を出しているが、システム固有のデータを用いたモデルの導出を行っている。

そこで本稿では実際にコーディングされた通信処理プログラムを例にとり、移植を予め考慮したコーディングの移植性に対する有効性について述べる。

２．通信処理プログラムの移植[5]

２．１　通信処理プログラムの構成

プログラムの移植性、流通性を向上させるために通信処理プログラムは図１のような階層構成を採用している。すなわちＯＳはCTRON*仕様に準拠し、基本ＯＳと拡張ＯＳの２階層構成を取っており、基本ＯＳでハードウェアアーキテクチャを隠ぺいし、拡張ＯＳは基本ＯＳの上位に位置し、アプリケーションプログラム（以後ＡＰと記述する）の流通性を保証する。今回対象とする通信処理プログラムにおいては拡張ＯＳ、ＡＰを移植の対象としている。

*:CTRONは"Communication and Central TRON"の略。

TRONは"The Real Time Operating System Nucleus"の略。

２．２　移植を妨げる要因とプログラム構成

本稿における移植元と移植先の動作環境の相違を図２に、またこれらの環境の差異を元に抽出した非互換項目の一覧を表１に示す。ハードウェアアーキテクチャの違いによりアセンブラやハード依存システムコールが異なり（以後この非互換項目を「ハード直接依存」と呼ぶ。）、またアドレスサイズが異なるために言語上はエラーにならないが、処理上問題となる項目（以後「ハード間接依存」と呼ぶ。）が存在する。また使用言語は交換プログラム記述用言語である高級言語であるＣＨＩＬＬを用いるが、バージョンが異なるため非互換項目がある（以後

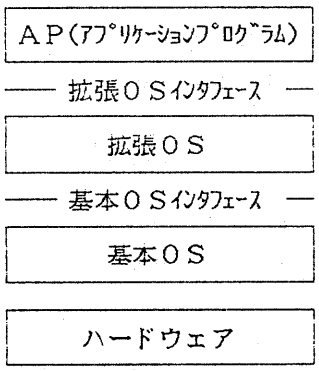

図１　移植対象プログラムの構成

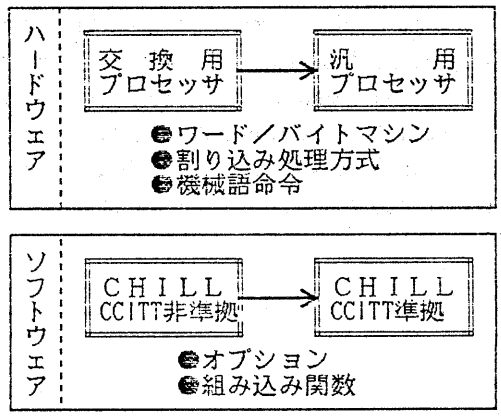

図２　移植前後の動作環境の相違

表１　非互換項目の一覧

原因	非互換項目	数	対処*
言語仕様依存	マクロ（インクルードファイルの読込など）	2	A,B,C
	オプション（プロシージャにシステム固有の処理を提供する）	7	C
	待行列（待行列データの定義、参照）	4	A,C
	局データ作成（局データ作成用の命令）	2	C
	その他(BASED変数、STATICO/S、ABNL変数等)	12	A,B,C
	組込関数（特殊な機能を提供する関数群）	13	A,B,C
ハード間接	CASE構造体（CASE付の構造体中にPTRモードがある場合、サイズが異なる）	1	B
	強制モード変換（アドレスサイズ不一致のためPTRモードとの変換不可）	2	B
ハード直接	アセンブラ（インラインの展開不可、アセンブラが異なる）	1	C
	ハード依存命令（ハード依存のシステムコール使用不可）	1	C
	計	45	

*:対処は、A→構文解析可能、B→コーディングを規定することにより対処可能、C→移植時に変換が必要（２．３節参照）

「言語仕様依存」とする。）。非互換項目は全部で４５項目あるが、その大半が言語仕様差に起因していることがわかる。

さらに本稿で検討の対象とする通信処理プログラムと類似のシステムについてサンプル的にいくつかのプログラム（モジュール）を選択して非互換項目の出現頻度について測定した。結果を図３に示す。ここで原因とは表１の分類方法を用い、出現頻度は全ステートメント数における非互換項目の出現ステートメント数およびその内訳を示している。

非互換項目の出現頻度は全体の規模に対し、７～８％であるが、その内訳内容がプログラム階層により大幅に異なっていることがわかる。すなわち基本ＯＳではハードウェアアーキテクチャを上位のプログラムである拡張ＯＳやＡＰに隠ぺいするためにハード直接依存関連の非互換項目出現が多く見られる。一方、拡張ＯＳ、ＡＰでは極端にその値が少なくなっており、移植性の向上への効果が予想される。

なお、ＡＰでハード間接依存の非互換項目の出現数が多く見られるのは、調査対象プログラム中にメモリサイズを算出する組込関数を多く使用しているプログラムがあり、これは移植前後でアドレスサイズやビットエンディアンが異なるために非互換項目となっているためである。

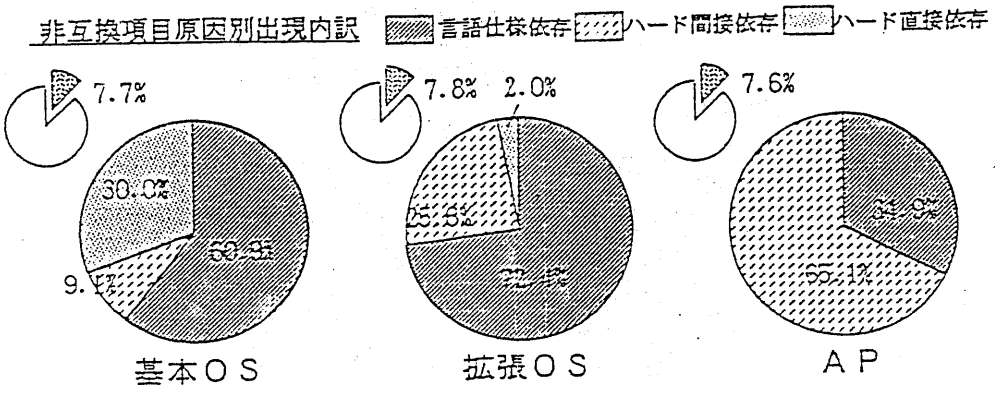

図３　非互換項目の出現頻度内訳

２．３　移植時の対処方法

前節においてプログラムの構成を階層化させることにより移植性が向上することを得たが、本稿ではさらに移植性を高めるために予め移植を考慮したコーディングを採用した際の有効性について検討する。

移植を予め考慮したコーディングの有効性を検証するために表１で示した非互換項目に対する対処方法を検討した。対処方法には大別して３種類考えられる。表１に各非互換項目に対する対処方法を示す。具体的には、各非互換項目について構文解析が可能であるもの（表中のA）、予め移植を考慮したコーディングを行うことにより互換あるいは移植がスムーズになるもの（同Ｂ）と移植時に変更が必要（同Ｃ）、という分類でまとめている。例えば、ハード依存部分は移植前に対処することができず、移植時の変更が必要となる。

表１に示す全ての非互換項目について対処方法を検討し、２．２節で用いたサンプルプログラムについて出現頻度内訳を調査した結果を図４に示す。

移植の対象となる拡張ＯＳ、ＡＰに注目すると、移植を予め考慮したコーディングを行うことにより、拡張ＯＳでは63.6%(コーディング規定で可能(48.1%)+移植時変換要(15.5%))→移植時変換要(15.5%)と約１／４に、ＡＰでは84.2%(84.2+0.0)→０%となる。すなわち、移植を予め考慮してコーディングすることにより、移植時に変更を要する規模が小さくなることがわかる。つまり、予め移植を考慮したコーディングの有効性が推定される。

３．移植を予め考慮したコーディングの有効性

３．１　移植工数の比較手法[6],[7]

前節において移植を予め考慮したコーディングを行うことにより、移植時に変更を要する規模がかなり削減されることを述べた。しかし、移植工数としてみた場合、移植時に変更を要する規模だけで判断することは困難である。そこで本節では移植を予め考慮することによる移植性に及ぼす影響を移植工数の面から検討する。

移植を予め考慮したコーディングを行う場合／行わない場合の移植工数の比較方法を検討する。表２に本稿で用いた移植工数の分類を示す。

これらより移植工数(Ｅ)は

$$E = Ep + Em + Et + Eb + Ec \quad \cdots\cdots \quad (1)$$

となる。

また本稿における前提条件として、

❶移植準備の工数は加味しない（Ｅp=0）

❷コンバータは既に存在している物を使用するとし、工数はかからない。（Ｅc=0）

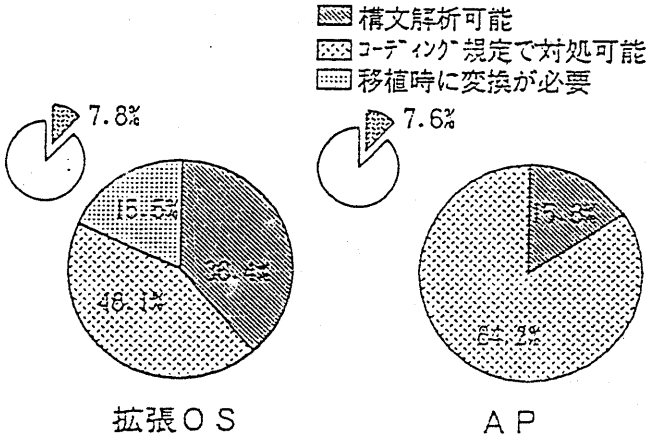

図４　非互換項目対処方法別内訳

表２　移植のプロセス

移植作業	工数記号	意　味
移植準備	Ep	移植環境の調査や方針決定、移植プログラムの準備など
変換	Em	言語仕様間、ﾊｰﾄﾞｳｪｱｱｰｷﾃｸﾁｬ間の相違の吸収
デバグ	Eb	バグの修正、テストプログラムの再走行
試験	Et	テストプログラムの走行
コンバータ作成	Ec	コンバータの作成

とすると、

$$E = Em + Et + Eb \quad \cdots\cdots\cdots\cdots \quad (1')$$

となる。

また

●変換作業に要する工数は変換規模に比例するとし、その比例定数をＣｍとする。

●デバグ工数は変換した部分そのもの(Ｎｄｊ)と変換した部分に関連するプログラム部分(Ｎｉｊ)において発生率Ｐで発生するバグ部分をデバグするのに必要な工数である。そこでデバグ工数は両者規模数の和に比例するとし、その比例定数をＣｂとする。

●オリジナルバグは存在しない。すなわちデバグ工数は変換した部分と変換部分に関連する部分のみを考慮すれば良い。

と仮定すると、Ｅｍ、Ｅｂはそれぞれ

$$Em = Cm \cdot \sum_{j=1}^{N} nj \cdot Hj \qquad (2a)$$

$$Eb = Cb \cdot P \cdot \sum_{j=1}^{N} nj \cdot (Ndj + Nij) \quad \cdots (2b)$$

Ｎ：非互換項目数

Ｃｍ、Ｃｂ：定数(工数／statement数)

ｎｊ：非互換項目ｊの出現数

Ｈｊ：　〃　　の変換規模

Ｐ：バグの発生率

Ｎｄｊ：非互換項目ｊの変換規模(＝Ｈｊ)

Ｎｉｊ：　〃　　の関連規模

となる。ここでｎｊ、Ｈｊ、Ｎｄｊ、Ｎｉｊは非互換項目毎に調査することにより把握できる。

従って、(２)式より、移植を予め考慮したコーディングを行う場合と行わない場合の変換工数やデバグ工数の比は、

$$\frac{Em1}{Em2} = \frac{\sum_{j=1}^{N1} nj \cdot Hj}{\sum_{j=1}^{N2} nj \cdot Hj} \quad \cdots\cdots\cdots (3a)$$

$$\frac{Eb1}{Eb2} = \frac{\sum_{j=1}^{N1} nj \cdot (Ndj + Nij)}{\sum_{j=1}^{N2} nj \cdot (Ndj + Nij)} \qquad (3b)$$

となり、算出可能となる。ここでＮ、Ｅｍ、Ｅｂ、はそれぞれ非互換項目数、変換工数、デバグ工数を示す。また添え字の１、２は移植を予め考慮したコーディングを行う場合／行わない場合を意味する。一方、試験工数については移植前後で同一の試験を行うと考え、

$$Et1 = Et2 \quad \cdots\cdots\cdots\cdots \quad (3c)$$

となる。

３．２　数値計算結果

(３ａ)、(３ｂ)式に基づき２．２節で用いたプログラムについて数値計算を施す。計算に用いたＨｊ(＝Ｎｄｊ)、Ｎｉｊの値および見積方法の一例を表３に示す。またｎｊのデータについては２．２節で測定した値を使用する。

拡張ＯＳ、ＡＰについてそれぞれＥｍ1／Ｅｍ2、Ｅｂ1／Ｅｂ2の値を計算した結果を図５に示す。移植を全く考慮しない場合の工数を１とすると、拡張ＯＳの変換工数が０．１５、デバグ工数が０．１８、またＡＰでは変換工数、デバグ工数ともに０．０３という値を得、移植を予め考慮することによりかなりの工数削減が得られることがわかる。

表３　計算に用いたＨｊ、Ｎｉｊの例

j(非互換項目)		Hj(=Ndj)(修正規模数)	Nij(関連規模)見積
マクロ	%TT	%TT直言規模分	なし
	%INCLUDE	×１（ファイルオープンのみ）	なし
オプション	OVERLAY	×１（削除）	プロシージャ規模
	SAVE	×４（アセンブラコード追加）	プロシージャ規模
	NONLOGUE	×１０＋call×５	プロシージャ規模＋call発行プロシージャ規模
CASE付きの構造化(PTR)		×１＋使用モジュール数×１	なし
待行列定義（FIFO/LIFO）INSERT,REMOVE		互換 ×１（修正）	――― なし
強制モード変換		×１＋使用モジュール数×１	なし
アセンブラインライン展開		×１	プロシージャ規模
組込関数	SIZE組み込み関数	×１＋使用モジュール数×１	なし
	FIX組み込み関数	×３（展開形に修正）	なし
	SHIFT組み込み関数	×３	なし
ハード依存命令の使用		×（１＋パラメータ数）	プロシージャ規模

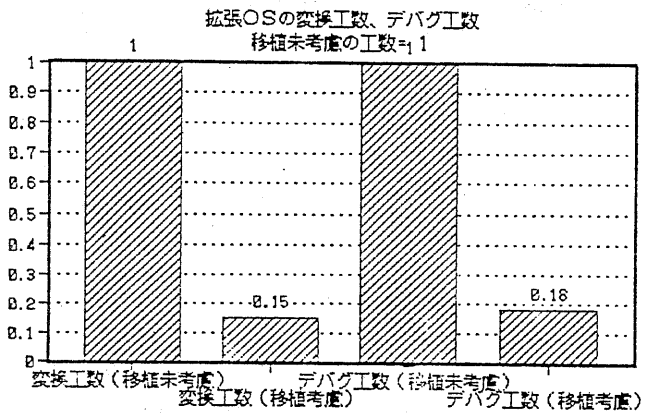

図５（ａ）　拡張ＯＳの変換工数、デバグ工数

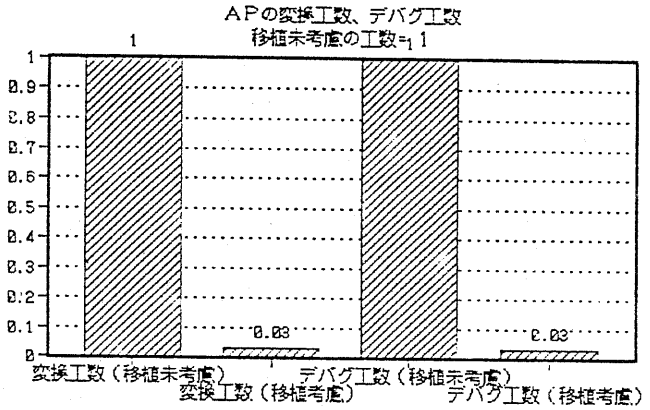

図５（ｂ）　ＡＰの変換工数、デバグ工数

３．３　通信処理システムの移植について

実際の通信処理システムのソフトウェアは複数のモジュールで１つのシステムを構成しているのが現状である。さらにモジュール群の中には移植時にハード環境が変化することにより、システム構成制御など強くハードウェアに依存している部分は処理方式の変更があり、新規作成を免れない可能性がある。そこで図６に示す通信処理プログラムシステムの移植の事例について移植工数の評価を行う。

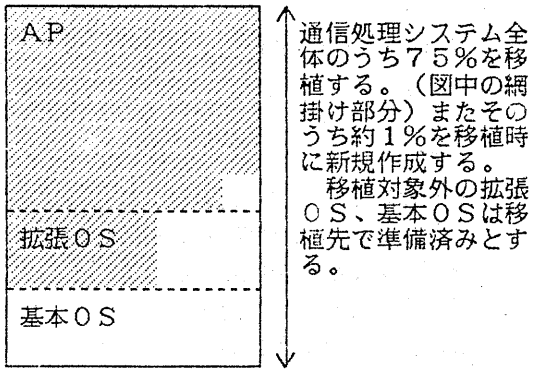

図６　移植対象の通信処理プログラム

図６のシステム全体の移植において移植を予め考慮した場合と考慮しない場合の移植工数の比較を行うにあたって以下の事項を仮定する。

●拡張ＯＳの移植対象外モジュールと基本ＯＳの全部は移植先で既にサポートされており、処理方式的に移植前の内容と変更はないものとする。

●拡張ＯＳ、ＡＰの非互換項目の出現頻度は事前にサンプル的に調査した拡張ＯＳ、ＡＰの非互換項目出現性に従うものとする。

●新規作成する工数は新規作成する規模に比例するとし、その比例定数は（２ａ）式の定数Ｃｍと同一であるとする。

●新規作成したモジュールのデバグ工数は新規作成した規模と新規作成した部分に関する規模の和に比例し、その比例定数は（２ｂ）式の定数Ｃｂと同一であるとする。

結果を図７に示す。移植を全く考慮しない場合を１とすると変換工数では０．１８、デバグ工数は０．２９に減少可能であることを示す。

また移植工数全体としてどの程度削減が図れるかを予想する。移植工数は（１’）式のように表される。ここで過去のシステム開発例から変換工数とデバグ工数の比を１：３、またテスト工数をここでは考慮しないと、図７の結果より、０．２６（＝(0.18×1+0.29×3)/4）という値を得る。すなわち移植を予め考慮することにより約１／４の工数削減が図れる。

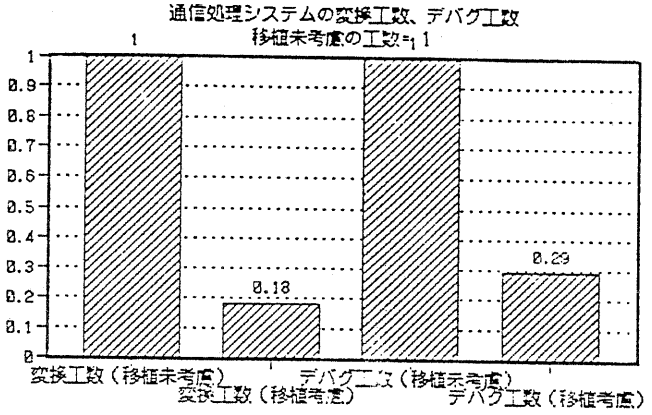

図７　通信処理システムの変換、デバグ工数比

４．おわりに

本稿では移植効率向上のために移植を予め考慮したコーディングの有効性について検討し、以下の結果を得た。

(1)移植を妨げる要因となる非互換項目の抽出を行い、その使用頻度をサンプルプログラムで調査した。基本ＯＳのハードウェアアーキテクチャ隠ぺいの効果により拡張ＯＳ、ＡＰでは極端にハード直接依存の非互換項目の出現が低くなっており、プログラムの階層構成の有効性を得た。

(2)今回調査したサンプルプログラムでは、移植を予め考慮したコーディングを行うことにより、移植時に変換すべき規模数は１／５以下に、またデバグ工数は１／４以下になることを得た。

(3)通信処理システムの移植について移植工数の評価を行い、１／４以下への工数削減が図れることを得た。

今後は今回行った移植工数の算出方法について試験工数や移植準備作業等を考慮して、さらに検討を深める。また今後実施される移植事例を元に本評価方法の妥当性を検証する予定である。

［参考文献］

[1]例えば A.S.Tanenbaum , "Guidelines for Software Portability", Software-prac. & Exper., vol.8 , pp.681-698 , (1978)
多田, "ＵＮＩＸのワークステーションへの移植性について", 情処論誌 , vol.26 , No.6 , pp.1033 -1040 , (1985)など

[2] J.R.Wolberg ,"Comparing the Cost of Software Conversion to the Cost of Reprogramming", SIGPLAN Notices , 16(4) ,pp.104-110 (1981)

[3] J.R.Wolberg ,"A Costiong Model for Software Conversion ", Software Prac. & Exper. , Vol.12 . pp.1044-1049 (1982)

[4] 金井ほか , "コンバータの効果とプログラム特性を考慮したプログラム移植工数モデルとその評価" , 信学研資 SE40-6 , pp.31-36 (1985)

[5] 中村ほか ,"通信処理プログラムの移植に関する検討" , 第４０回情処全大 , pp.1432 , (1990)

[6] 藤田ほか , "プログラムの移植について" , 情処学会誌 , Vol.21 , No.11 , pp.1128 -1135 (1980)

[7] 栄ほか ,"移植プロセスのモデル化と評価" , 信学研資 68-6 , pp.1-8 (1989)