プログラム移植性向上手法の検討

2K−8

宮崎義之*　田中利清**　花沢満**
*NTTネットワークシステム開発センタ　**NTT情報通信網研究所

1．はじめに

ソフトウェアの生産性向上のために、プログラムの再利用が望まれている。再利用を促進するためには、プログラムの移植にかかる工数の削減が必要である。そのためには、以下の2つのシステムでの条件の違いを吸収できるよう対策を行なうことが必要である。

・新規開発時に動作を確認したシステム（以下、移植元と略）
・移植により動作を確認しなければいけないシステム（以下、移植先と略）

それには、移植を行なうプログラム（以下、移植対象プログラムと略）の移植性向上対策だけでなく、周りの環境（コンパイル環境、試験環境等）も合わせて考慮することが必要となる。著者らは、移植元と移植先で同じソフトウェアインタフェースが提供される場合を対象に実際に移植作業を行った。本稿では、移植を通じて明確になった移植上の問題点（以下、移植阻害要因と略）を基に、移植工数を削減するための手法を提案する。

2．実際の移植における手順、工数の整理

今回の移植は、以下の条件のもとで行った。

(a) 移植対象プログラムの品質が高い（移植元において高い品質が保証されており、潜在バグは存在しない）
(b) 開発者と移植作業者が異なる（移植作業者は移植対象プログラムの内部構造を理解していない）
(c) 移植先と移植元でコンパイルツール（コンパイラ、リンカ等）、コンパイルマシン、ハードウェアアーキテクチャ、デバッグツールが異なる

移植の作業を11の手順に分類して、各手順でかかった工数を測定した。（図1、図2）

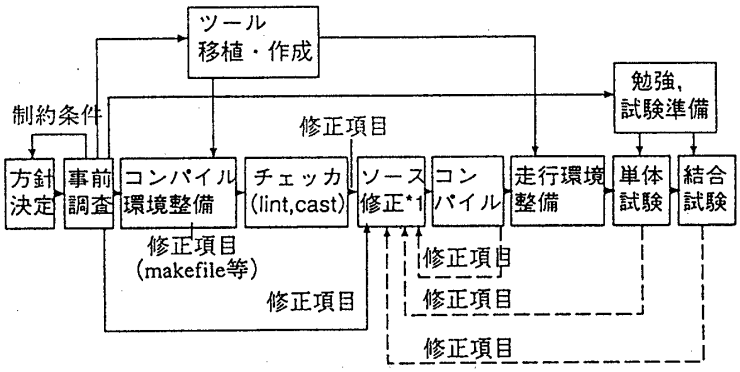
なお、移植先と移植元で同じソフトウェアインタフェースが提供されているが、実際にはアーキテクチャに依存する部分とインプリメントに依存する部分の差が存在した。

3．問題点の抽出と対策の検討

移植作業工数の分析結果から以下の問題点がわかり、それらを解消するための対策を検討した。

(1) 勉強工数が必要（移植先仕様、移植対象プログラム仕様・構造理解、ツール操作習熟）

この中で、移植対象プログラム仕様・構造理解が大部分を占め、試験手順書作成と、試験実行のために行なったものであることが判明した。これに関しては、次の見解を持った。



*1：アセンブラルーチン，システムコール登録ルーチン等の新規作成を含む

図1　移植作業手順

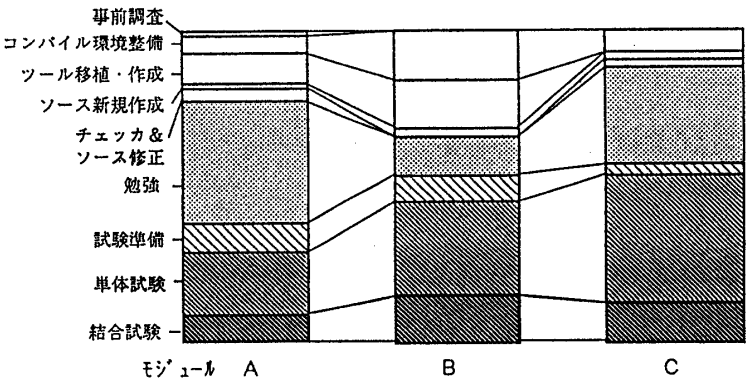


図2　移植作業工数

A study of improving the program portability
Yoshiyuki MIYAZAKI, Toshikiyo TANAKA, Mitsuru HANAZAWA
NTT

・試験手順書が十分詳細化（作業手順、入力内容、確認内容等まで記述）されていれば、試験実行のためにあらかじめ内部構造まで理解する必要はない

［対策］

試験手順書の詳細化、再利用化を図る（以下の(2)と密接な関係がある）

(2) 試験準備工数が必要（試験手順書、テストプログラム作成）

これは、移植先と移植元でのデバッグツールが異ること、試験手順書とテストプログラムが移植先で使用できる形で残っていなかったためであることが判明した。これに関しては、次の見解を持った。

・試験手順書、試験プログラムを再作成するのは無駄である（再利用することで工数を減らすことができる）

［対策］

試験方法を移植先に依存しないようにする

(3) モジュール単体試験は余り効果的ではない（モジュール単体試験では問題点が検出されなかった）

これは、コンパイルツールが異るときの問題点が、(i)コーディング規約により回避可能、(ii)文法チェッカで容易に検出可能、(iii)検出された問題点は容易に修正可能なためであることが判明した。これに関しては次の見解を持った。

・他のモジュールとのインタフェースを重点的にチェックすることが必要

・移植を阻害する要因は新規開発時にほぼ解消することができる

［対策］

新規開発時に移植上の問題点を作り込まないようにする

(4) 移植先と移植元でのインタフェースの差が存在する（アーキテクチャに依存する部分、インプリメントに依存する部分）

これは、ソフトウェアインタフェースでの隠蔽が難しい部分であり、必ずしも統一すれば良いというものではないことが判明した。これに関しては次の見解を持った。

・インタフェースの差がある部分は限られており、差があることを前提に作れば良い

［対策］

インタフェースの差を容易に吸収できる作りとする

4．対策の実現性と有効性の検討

それぞれの対策について、実現性と有効性を検討した。

(1) 試験手順書の詳細化、再利用化を図る

・試験手順書の記述をキー操作レベル、確認する具体的数値まで詳細化する

(2) 試験方法を移植先に依存しないようにする

・デバッグツールを使用しないで試験実行できるようにする

(3) 新規開発時に移植上の問題点を作り込まないようにする

・コーディング規約を決め、文法チェッカでコーディング上の問題点を検出する

(4) インタフェースの差を容易に吸収できる作りとする

・あらかじめ、パラメータ化しておく

・移植先の変更部分を容易に把握できるようにする

これらを実現する上で問題となる項目を以下に示す。

(A) 試験をどこまで環境に依存せずに行なえるか

(B) 文法チェッカでどこまで規約違反を検出できるか

(C) インタフェースチェックの手順書化、プログラム化

このうち、(B)、(C)に対しては現在検討を進めており、評価を行っているところである。(A)に対して検討を行った結果、テストプログラムの作成条件等を考えることにより、テストプログラムの起動及び試験結果の収集は移植先環境に依存しないように実現でき、試験手順書の再利用と詳細化の条件を満たすことが可能であることがわかった。

具体的には，以下の課題を解消することで実現できる。なお、今回試行した例を（）内に示す。

・テストプログラムの起動方法を統一する
（例：システム初期起動でロード、起動を行なう）

・作業者との入出力を統一する
（例：入出力関数を規定し、使用を義務づける）

・結果の判定に必要な情報を取得できるようにする
（例：移植対象プログラム内部に出力ルーチンを埋め込む）

・テストプログラムの移植性を高める
（例：特殊なライブラリは使用しない）

5．まとめ

移植阻害要因の解消には、新規開発時、移植時の両方で移植性向上対策を行う必要がある。

また、移植するソフトウェアだけでなく、試験環境も併せて再利用を考えることで、効果的に移植工数を削減することができる。

6．おわりに

今後はこの方法を実際の移植に適用して評価を行なう予定である。

試験環境に対しては、今後の課題として以下がある。

・移植確認として十分な試験項目

・出力ルーチン埋め込みが問題となるソフトウェアに対する試験方法

［参考文献］

① 林他：ソフトウェア流通阻害要因の抽出と考察、第４３回情処大予稿(1991)