プログラム開発における自動設計の一考察

4J－6

北川直治[1]　岩本　朗[1]　中沢　通太[2]　山本　浩[3]　片山　栄司[2]　青山　幹雄[2]
[1]富士通名古屋通信システム㈱　[2]富士通㈱　[3]富士通九州通信システム㈱

1．まえがき

開発プロセスとプログラムがそれぞれ類似のパターンに帰着できる点に着目し、部品合成により詳細設計の一部を自動化する方法と支援環境を開発した。この方法とプロトタイプによる試行結果を報告する。

2．背景と狙い

2.1 背　景

ソフトウェアは様々な要求を柔軟に実現する手段である。従って、要求の変更や追加などにも容易に対応できる。しかし、ソフトウェアの開発者は人であり、この柔軟性の故に人の趣向やスキルの違いにり、以下の問題が発生する。

①開発技法やプロセスなどの規約を全開発者へ周知徹底させることが難しい。

②開発者の個人差による作成ドキュメントのバラツキが発生する。

一方、効率や品質の向上が追求され、一方法として、プログラム再利用の研究と適用が進められている。しかし、実開発における再利用には以下の問題がある。

①技術は日々発展しており、それを実現する新規プログラムが開発される。しかし、新技術を実現する機能追加は既存ソフトウェアの構造を歪曲してしまったり、継承を損なうことがある。

②プログラムの理解には多大な時間と費用が掛かるが、その利用および効果が保証されているわけではない。

2.2 狙　い

新規機能の追加でシステム機能は向上するが、それは既存の問題を解決することに他ならない。この解決には既存の技術や知識をベースにしており、既存プログラムの改良か、代替プログラムの開発である。この観点から、新版数プログラムと旧版数プログラムを比較してみると、その類似性が確認できる。しかし、実際には再利用が進んでいない。この原因は以下と考える。

①作業標準の徹底を人に頼っているため、作成したプログラムが再利用できるほど標準的となっていない。

②既存プログラムを十分に理解していないため、再利用すべきプログラムが見落とされている。

一方、開発における作業時間配分を調査すると、前工程の開発者や他モジュールの開発者からの情報伝達に費やされる時間が多いことが確認できた。その原因を追求すると、以下の問題が明らかとなった。

①設計書等による情報伝達が容易でない。

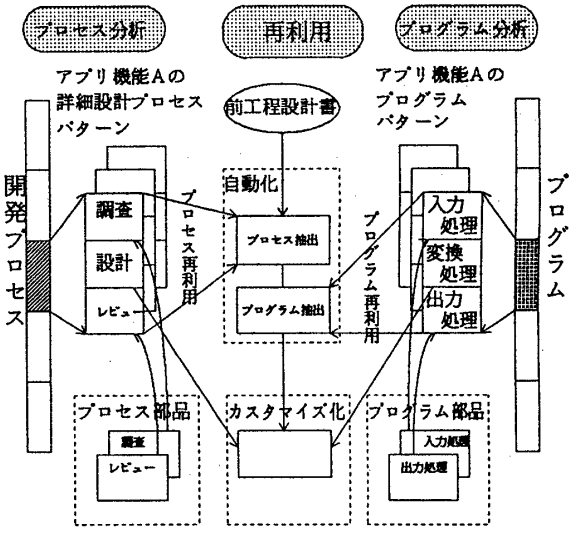
②開発者の知識や技術が不足している。

以上の分析から、開発者と支援環境について、分担とその関係を工夫すれば、再利用と規約の遵守が向上し、効率と品質が向上すると考えた。

3．自動化の方法

3.1 自動化の概念

本稿で提案する自動化の概念を図－1に示す。基本設計、詳細設計などの各工程を詳細に分割すると、分割された作業は単純となり、これはプログラムについても同様に言える。また、工程内作業の違いは開発機能の違いであり、機能毎にプロセスとプログラムのパターン化によって再利用が図れると考える。更に、プログラム構成の類似性をパターン構成の中に埋め込むことにより、部品合成による自動化が可能であると考えた。しかし、全ての部分を単純化、パターン化できるわけではなく、自動化すべき部分とすべきでない部分とを分離する。



図－1　自動設計の概念

An Automated Program Design System based on Software Reuse
Naoji KITAGAWA[1], Akira IWAMOTO[1], Michitaka NAKAZAWA[2], Hiroshi YAMAMOTO[3], Eiji KATAYAMA[2], and Mikio AOYAMA[2]
[1] FUJITSU NAGOYA COMMUNICATION SYSTEMS LIMITED, [2] FUJITSU LIMITED,
[3] FUJITSU KYUSHU COMMUNICATION SYSTEMS LIMITED

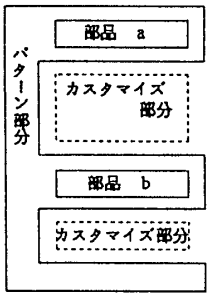
①再利用：プログラムの再利用はプロセスのパターン化とプログラムのパターン化より成り立っている。
②プロセスのパターン化：前工程の設計書から自工程の開発機能が決定されるが、既存プログラム構造や機能による分類に基づき開発プロセスがパターン化できる。
③プログラムのパターン化：分類されたプログラムは処理構造は類似しているが、処理するデータ構造が異なる場合が多い。このプログラム構造をパターン化することにより、プログラムの理解が容易となる。

3.2 自動化に到る過程

部品合成による自動化を実現する作業手順を以下に示す。

(1)プロセス分析
①作業の分離／分割：各工程の作業を開発者による差異が発生しないまでに分割、分類する。
②作業の関係づけ：分割、分離した各作業の関係を明確にする。
③作業のパターン化：分離、分割した各作業を入力、処理、出力などとして、明確なフォーマットを決める。
④作業の分離：分離、分割した作業を自動化できる作業とできない作業に分ける。

(2)プログラム分析
①プログラムの分離／分割：既存、または旧ソフトウェアを内容の類似性が確認できるまで分割、分類する。
②プログラムの関係づけ：分割、分離した各プログラムの関係を明確にする。
③プログラムのパターン化：分離、分割した各プログラムのスケルトンを決定する。
④プログラムの分離：分離、分割した各プログラムを設計者を必要とする部分としない部分に分ける。



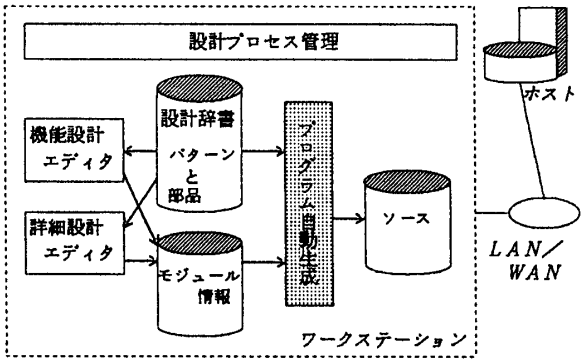
図－2　プログラム構造

このようにして抽出したプログラム部品の構造を図－2に示す。

3.3 自動化の支援環境

本自動化を支援するために、図－3に示す支援環境を開発した。パターン化されたプロセスとプログラムを辞書として登録しておき、前工程の設計情報から一様な変換により設計できるものを対象として自動化する。非一様な変換部分は開発者による設計とする。各システムは、次のような構成となっている。

①設計エディタ：前工程である機能設計の設計情報を入力する機能設計エディタと開発者によるカスタマイズ設計をする詳細設計エディタからなる。
②設計辞書：パターンを構成する部品と、プロセスおよびプログラムのパターン化されたデータから成る。
③モジュール構成情報：開発したモジュール個々の構成管理情報である。
④設計プロセス管理：プロセスのパターンの管理情報。
⑤自動生成：設計辞書と構成管理情報により、ソースプログラムを自動生成する。



図－3　自動化支援環境

4．試行結果

4.1 適用対象

(1)対象工程：詳細設計
(2)対象ソフトウェア：大規模交換ソフトウェアのデータ管理アプリケーション

4.2 適用効果

(1)定量的評価
①開発効率：生産性が４０%向上した。
②品質：混入バグが１２%減少した。

(2)定性的評価
①開発ドキュメントの理解が容易になり、保守が容易となった。
②プロセスのパターン化により、規約が徹底できた。
③プログラムの定型化により、再利用が推進できた。
④各工程間のドキュメントの関係が明確となった。

5．今後の課題

本試行でのパターン化はモジュール単位、工程単位であり、モジュールの型が変更となる場合に柔軟に対応できない。今後、パターン化の単位を検討し最適化を図りたい。

6．まとめ

プロセスとプログラムのパターン化に着目した、ソフトウェアの自動設計の方法と支援環境について報告した。今後、本手法に基づき実開発に自動化を適用していきたい。