設計方法論に基づいたCASEツールの落とし穴

4U－8

金子　博　松本　憲幸　岡山　敬
株式会社　東芝

1.はじめに

CASEツールは、開発の標準化や効率向上などを目的としており、その1つに方法論に従ったソフトウェアの開発を機械的に支援することがある。CASEツールによって支援される開発方法論に従う限り、標準の開発規約の範囲でスムーズに開発を行うことが可能である。一方、肝心の方法論自体がカバーする範囲が狭いことや現実に適用する際に不都合が生じる可能性を秘めており、実開発においては容易にこの方法論範囲からはみ出したり、矛盾に衝突したりする場合がある。

本稿では、不完全ともいえる開発方法論に従順なCASE機能を実開発に適用する際に生じる不都合とその改善策について述べる。

2.問題点と対策

(1) CASEツールへの取り掛かりの問題

CASEツールの有効性は、従来の膨大であった下流工程での作業量を上流に移動することにより全体の作業量及び開発費用の減少が可能であるとされている。通常のユーザがCASEツールを導入する際に最初に抱くであろう疑問は、現在の開発が前者のパターンに陥っているのか後者にあてはまるのか、さもなくば特異な負荷分散となっているのかという現状の把握であり、CASEツールを適用するべきかどうか、またどの工程に適用すると最も効果的であるかの判定である。

まず、開発方法論は混沌の中にある開発を正規化するという視点で見た場合、CASEツールの導入ユーザに対して最初に行うべき支援はユーザのソフトウェア開発の現状の分析・評価である。

つまり、CASEツールを適正に適用するためには、CASEツールの開発支援機能の導入に先立ってあるいは同期して利用可能な工程管理・工程評価支援機能のサポートが必要となる。

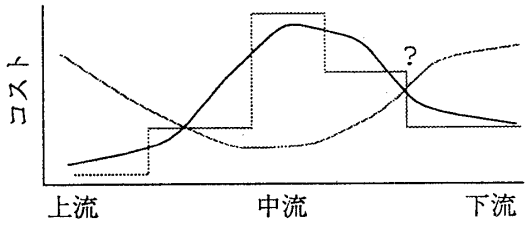

図1　CASEツール導入前の分析結果

(2) 構造化分析/設計の位置づけの不都合

構造化分析は、複雑化されたシステムの要求分析を目的とし、通常は上流に位置づけられる。

構造化分析手法の全体は、システムの抽象設計しか想定していないことは提唱者自身[1]が述べている事実であるが、それ故この様な方法論に忠実なCASEツールでは、一般的な処理の定義能力をもった状態遷移図(STD)やDFD/CFDを記述することさえも中下流ではサポートされない。STDは、イベント駆動型で動作するタスクやモジュールの内部仕様を定義するのに適しているし、アプリケーションの画面遷移の記述にも適している。また、データフロー図は処理の並列性を単純に定義する手段であり、マルチスレッド処理を行うプログラムの仕様の記述に適している。

このケースでは、方法論に従順であるが故により高度な設計への制約を生じてしまう。このような方法論に縛られないために、各種設計図の自由な作図・配置を可能としながら、これらの設計図を方法論に従って管理することも可能な形態でCASEツールを作成する必要がある。

(3) 設計図とプログラムとの双方向変換の困難

フローチャートなどアルゴリズムを表現した設計図からプログラムへの自動変換およびプログラムからフローチャートへの逆変換により設計図とプログラムとの間の整合性を保つことが可能である。一方、フローチャートが1つの確定したアルゴリズムを記述するのに対して、プログラムには選択コンパイルマクロにより潜在的に複数の異なっ

"The traps of CASE tools based on design methodologys"
Hiroshi Kaneko, Noriyoshi Matsumoto,
Takashi Okayama,　TOSHIBA Corporation

たアルゴリズムを表現する能力がある。

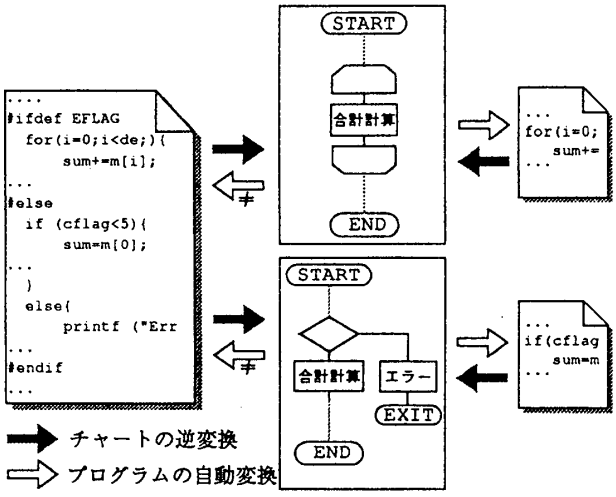

図２　双方向変換の問題点

これによって、チャートからプログラムへの変換に際しては情報量の保存が可能であるが、プログラムからチャートへの逆変換に際してはチャートはプログラムに潜在するアルゴリズムの内の1つを具現化することになり情報量の欠損が生じる。このため、逆変換されたチャートから再度変換して作成されるプログラムは、一般には逆変換前のプログラムとは異なるものとなる。

故に、プログラムとチャートの間で高いレベルの整合を取るためには、チャートに対するコンパイルマクロに相当する拡張またはプログラムとチャートを1:n対応とする等の対策が必要となる。

(4) ユーザ要求に見る開発の流れの不都合

CASEツールの支援対象となるソフトウェアは、4−8bitマイコン制御等の小規模システム開発からプラント制御等の大規模システム開発まで規模も仕様言語も異なる。CASEツールはある程度大きなシステムの開発を想定した方法論に基づくため、特に小規模システム開発者からは通常のCASEツールが用意する構造化分析→構造化設計→構造化プログラミングの流れに対して、以下のような要求がある。

i) 構造化構文をもつ言語による開発者

プログラム自体が構造化されている場合には、プログラミングの段階で構造化でき、フローチャート等によるアルゴリズムの構造化設計は生産速度の点でメリットが少く、構造化分析の結果を直接プログラムに変換したいという要求がある。

ii) アセンブリ言語による開発者

プログラミング言語自体が構造化されていないため、構造化設計を利用してプログラムに変換したいという要求がある。この場合、フローチャート等はアルゴリズムを表現する目的ではなく、アルゴリズムの1実現形態としてのコードイメージを表現するために使用され、構造化エディタ的に使用される。

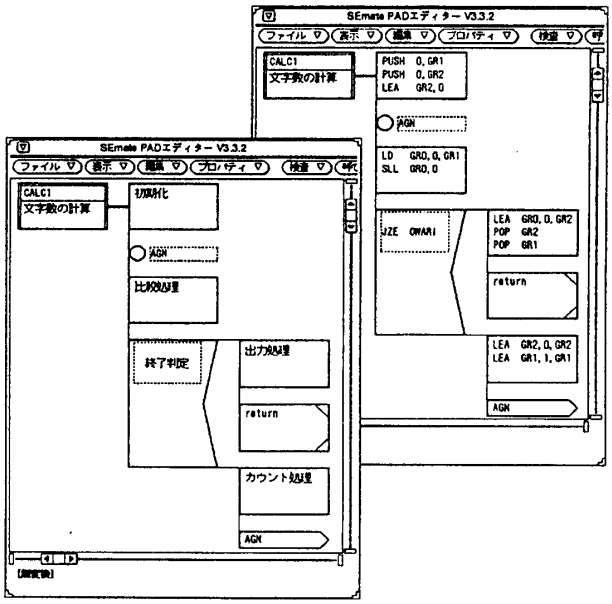

図３　PAD記述例

これらのケースは、現実的な開発効率の向上を得るためには開発内容に応じたCASEツールの個別工程支援の結合関係を変更し、開発の流れの組み替えが必要となる可能性を示している。

3. おわりに

以上、現在のCASEツールに潜む問題点とその対策案に付いて述べてきた。これらのことを勘案し、更に様々な要求にあったより柔軟で利用価値の高い次期CASEツールを現在開発中である。また、開発言語や開発対象などにも合わせた設計方法をサポートしたツールの開発が必要であろう。

参考文献

(1) Hatley他:リアルタイム・システムの構造化分析,日経BP社,1989
(2) Aho他:Data Structures and Algorithms ,ADDISON-WESLEYPUBLISHING社,1987
(3) マーチン他:ソフトウェア構造化技法,近代科学社,1986