# 状態遷移表設計によるソフトウエア開発プロセス改善

コニカ㈱ オフィスドキュメントカンパニー 機器開発統括部 第3開発センター　渡辺 智 ／ 滝 研司 ／ 黒畑 貴夫

## 1　はじめに

複写機をはじめとする情報機器は多機能、高性能化し、それを制御するソフトウエアも肥大、複雑化しています。また、商品サイクルも年々短くなり、開発期間の短期化が求められています。これらの機器を制御するソフトウエアの開発においても、このような要求に対応するための対策が望まれています。

従来のソフトウエア開発は、要求仕様書からソフトウエア設計仕様書を作成し、この設計仕様書を元にソフトウエアの最終段階であるソースコードを作成していました。

しかし、要求仕様書の大部分が図表でなく文章で書かれているために、ソフトウエア設計者が要求仕様書の文章を読解してソフトウエアを設計する段階で、誤解が生じたり、思考の漏れや抜けが生じる可能性があります。対象となるソフトウエアが複雑になればなるほどこの傾向は顕著となっています。また、ソフトウエア設計仕様書もほとんどが文章で書かれているため、デザインレビュー等においては、レビューの参加者が、設計内容をビジュアルに把握する事ができず、期待通りのレビューを行う事ができないといった問題も起こります。

さらに、時事刻々変化する市場ニーズへの対応や開発段階での課題に呼応して要求仕様が変更される事がありますが、従来は変更内容が記載された仕様書や変更部分が修正された仕様書をソフトウエア担当者が読解し、直接ソースコードを修正しています。このため変更部分の影響について漏れや抜けが生じ、仕様変更に伴う不具合が発生する事があります。

今回、上記の問題点を改善すべく、複写機のフィニッシングシステムおよびカラープリンタシステムの制御ソフトウエア設計に状態遷移表設計手法を導入しました。また、合わせて状態遷移表設計を支援するCASE（Computer Aided Software Engineering）ツール「ZIPC」の導入効果についいても評価を行ったので、その結果と有効性について紹介します。

## 2　情報機器制御システム制御の特徴

情報機器は、機器に組み込まれたコンピュータシステム（組込みシステム）によって制御されており、次のような特徴があります。[1]

第一に、組込みシステムはユーザーによる操作やセンサ信号等の外部からの事象（イベント）に対して、ランプを点けたりモータを回す等の処理（アクション）を行う反応型（リアクティブ）システムである事があげられます。このようなリアクティブなシステムでは、内部の状態と外部からの事象により制御（処理）を行う事が多いです。[2]

情報機器の一つである複写機について考えてみると、原稿をセットしてからコピーボタンを押してコピー動作を開始する手順を処理するためには、原稿がセットされた時点でその状態を内部に保持し、コピーボタンが押された時点で内部に保持した原稿がセットされているという状態と照らし合わせて、コピー動作（処理）を開始します。ただ、単純に思えるこの動作も、実際にはコピーボタンが押された時に機器がウオームアップ中なのか、すでにウオームアップが完了した状態なのかといったように、多くの状態が存在する事になります。

第二には、制御対象に予測・制御不可能な要素が含まれているために入力と出力の関係が一意に定まらない事があげられます。

例えば複写機では、給紙開始指令を出してから紙が通過検知センサに到達するタイミングは、紙の特性や環境要因等により複写動作の各紙毎で異なっており、同一のタイミングで制御する

事は不可能です。第一の特徴で示した多くの状態において、様々なタイミングで事象が発生する事となります。

即ち、複写機をはじめとする情報機器の制御設計においては、複雑にからみあった状態と、これらの状態において様々なタイミングで発生する事象を如何に整理するかが課題となります。

## ３　状態遷移表設計手法の導入

### ３．１　　目的

本手法導入の目的は、状態遷移表を使ってソフトウエア設計を行い、状態と事象を明示的に整理する事により、ソフトウエア設計段階で発生する不具合を低減する事です。

Fig.1は、複写機のフィニッシングシステムの開発段階で発生したソフトウエア不具合について、不具合の発生原因を、要求仕様そのものの不備、設計段階でのミス、コーディング段階でのミスの３つに大別して示したものです。

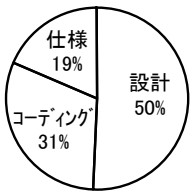


Fig.1 不具合発生の原因となった工程

Fig. １から、不具合の多くは設計段階で発生しており、ソフトウエア不具合を低減させるには、設計段階での不具合低減が有効である事がわかります。

設計段階で不具合が発生する原因として、次のような事があげられます。

第一には、設計が設計者の頭の中で行われており、その結果を明示的に示すものがないため十分なデザインレビューが行われないこと。第二には、納期圧力の関係から設計の一部を端折ってコーディング作業を開始してしまい、コーディング中に設計漏れや設計抜けに気付いても設計フェーズにもどらずその場のコーディングで対処してしまうこと。第三には、仕様変更があった場合、設計者が頭の中で考えていた初期設計時の情報が失われており、変更仕様の織込みに際して、影響範囲や関連個所の把握が十分に行えずに設計漏れや設計抜けが生じやすいこと。

状態遷移表設計手法の導入は、これらの問題を解決し、設計段階での不具合を低減する事にあります。

### ３．２　導入

状態遷移表設計手法は、制御対象となるシステムがとり得る全ての状態と、そのシステムで発生する全ての事象を洗い出し、この２つの要素より構成される表を作成し、ある状態である事象が発生した時の処理をその表の升目（セル）に記載すると共に、処理実行後にシステムがとる状態を記載する事により、システム全体の動作を設計するものです。Fig.2に状態遷移表の概要を示します。

| 事象＼状態 | | ｱｲﾄﾞﾘﾝｸﾞ | 給紙中 |
|---|---|---|---|
| | E＼S | 1 | 2 |
| ｺﾋﾟｰｷｰON | 1 | 2<br>給紙ﾓｰﾀｽﾀｰﾄ | 不可 |
| 給紙ｾﾝｻｰON | 2 | 無視 | 3<br>給紙ﾓｰﾀ停止 |

Fig.2 状態遷移表（例）

今回、状態遷移表設計手法を２つの機種開発の一部に導入しました。

一つは、複写機のフィニッシングシステムの制御ソフトウエア設計への適用です。今回設計手法の導入対象としたのは、フィニッシングシステムのカバーシートフィーダー機能です。カバーシートフィーダー機能とは、複写機本体から排出されてくるコピー紙に対して、予めフィニッシングシステムに用意された紙を表紙や裏表紙として挿入する機能です。ユーザーが、予めフィニッシングシステムに用意された紙をどこに挿入するかを指定するオペレーション機能と、紙を搬送して複写機本体から排出されてくるコピー紙と合わせるために機構部品を制御す

る搬送制御機能とから構成されています。今回は、この両方の機能のソフトウエア設計に手法を導入しました。

もう一つは、カラープリンタの状態管理機能です。プリンタの制御ソフトウエアは、外部との通信を行う通信機能、機械の状態を監視しプリント作業全般について指令を出す状態管理機能、画像を形成する画像形成機能、紙の搬送を制御する紙搬送機能、トナーを紙に定着させる定着機能、特殊な動作指令に対応して動作する特殊機能より構成されていますが、今回は状態管理機能のソフトウエア設計に手法を導入しました。

# 4　結果

## 4．1　不具合発生率の低減

Fig.3は、フィニッシングシステムにおいて状態遷移表設計手法を採用してソフトウエア設計を行ったカバーシートフィーダー機能と、従来の設計手法でソフトウエア設計を行ったその他の機能でのソフトウエアの不具合発生率を示した表です。手法の導入により設計段階での不具合発生率が従来の約1/2に低減されています。

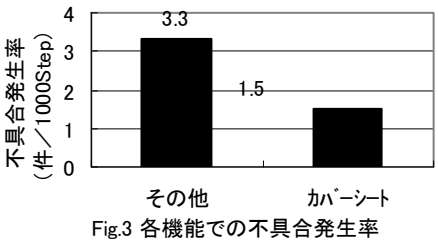

Fig.3 各機能での不具合発生率

Fig.4は、同様にカラープリンタにおいて状態遷移表設計手法を採用してソフトウエア設計を行った状態管理機能と、その他の機能でのソフトウエアの不具合発生率を比較した表ですが、このシステムにおいても、手法の導入により設計段階での不具合発生率が従来の約1/4に低減されています。

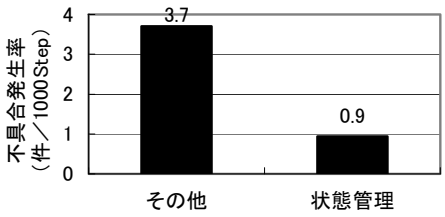

Fig.4 機能別の不具合発生率

## 4．2　仕様変更への対応

今回のカバーシートフィーダー機能の開発では、初期設計で機能の基本的な部分に対応した小規模な状態遷移表を作成し、開発の進捗に伴って発生する要求仕様を、状態・イベントの追加として折り込み、状態遷移表を拡張する方法を採用しました。

Fig.5は、初期設計での状態遷移表と最終的な状態遷移表のセル数を比較したものですが、この方法により開発の進捗に合わせて初期の約2.5倍の規模まで状態遷移表を拡張し、且つ、日程の遅延なく前記のように高い品質のソフトウエアを提供できました。

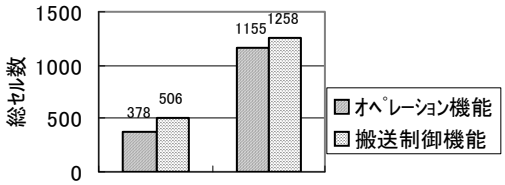

Fig.5 各設計段階でのセル数

# 5　考察

## 5．1　設計における手法の有効性

今回ソフトウエア設計に状態遷移表設計手法を導入した機能は、ユーザーインターフェイスの役割を担うオペレーション機能、紙搬送のための機構制御を担う搬送制御機能、プリンタシステムの状態監視と全体の制御を担う状態管理機能といったようにそれぞれ異なった機能ですが、これらは全て有限の状態を持ち、外部からの事象の変化を受けて、何らかの処理を行うリアクティブなシステムであると考える事ができ

ます。状態遷移表設計は、この様なシステムの設計段階において発生する不具合の防止に有効な方法であると言えます。これは、状態遷移表を活用して設計を行う事により、システムがとりうる全ての状態とイベントを洗い出すと共に、状態遷移表の全てのセルについてそこでの処理を考えるため、設計段階での漏れや抜けを防止できるからです。

## 5．2　デザインレビューでの有効性

Fig.6 は、フィニッシングシステムのカバーシートフィーダー機能の初期設計段階で作成した状態遷移表をもとにデザインレビューを行った際のデザインレビューでの不具合指摘件数とその後に指摘を受けた不具合の件数割合を示したものですが、不具合の多くがデザインレビューにより指摘されている事がわかります。これは、仕様を状態遷移表で表す事により、従来設計者の頭の中で行われていた設計がビジュアル化され、デザインレビューの参加者が設計内容を理解しやすくなり、適切な指摘が可能になったためと考えられます。

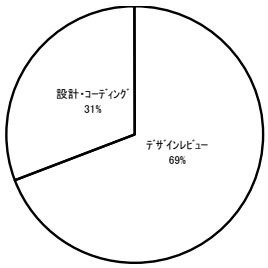


Fig.6 不具合指摘件数

## 5．3　仕様変更への適応性

Fig.7 は、状態遷移表設計手法を導入してフィニッシングシステムのカバーシートフィーダー機能を開発した際の工程別作業工数の割合を示したものです。

状態遷移表設計手法を導入したソフトウエア開発では、設計作業が全体の作業量の約６割を占めています。初期設計において設計の作業量

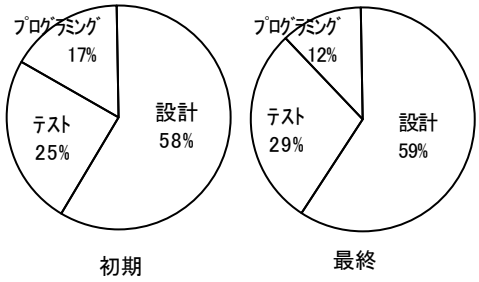


Fig.7 工程別作業工数

が大きなウェイトを占める事は容易に想像されます。しかし、仕様変更を行った最終段階においても、なお、設計作業が初期設計と同等のウェイトを占めるのです。

これは、状態遷移表設計では仕様変更時に、追加または削除される状態とイベントを整理した後、状態遷移表を修正するためです。即ち、仕様変更時も初期設計時と同様に状態遷移表による設計作業が行われるため、仕様変更作業後でも作業全体に占める設計作業のウェイトは変わらないのです。この再設計作業により、仕様変更に伴う設計漏れ、設計抜けを防止する事ができます。

そして、状態遷移表設計では、最新の仕様に対して、常に最適な設計がされている事になります。これにより、ソフトウエアの最新の設計状態を容易に把握できると共に、たとえ不具合が発生しても現状分析が容易であり、短時間で原因解析が可能であるという効果が期待できます。

## 6　状態遷移表設計支援ツールの評価

ソフトウエア設計に状態遷移設計手法を導入し、その有効性を確認する事ができましたが、今回の手法導入において幾つかの課題があった事も事実でした。

そこで、この課題を解決するため、状態遷移表設計手法を支援する CASE ツール「ZIPC」についても調査とその評価を行いました。

ZIPC については、ツールの各種機能や、ZIPC

を実際の開発業務に活用した事例が紹介されており、今回はこのZIPCを採用しました。[3] [4]

今回の手法導入の中で出てきた課題は、状態遷移表の肥大化と、状態遷移表とソースコードの不一致です。

フィニッシングシステムのカバーシートフィーダー機能の開発においては、市販の表計算ソフトを活用して状態遷移表の作成を行いました。しかし、Fig.5にも示したように、オペレーション機能、搬送制御機能とも状態遷移表のセル数が1000を超えており、全体を一度に把握する事が難しくなると同時に、状態遷移表のコンピュータの画面への表示、紙へのプリントアウトといった物理的な面での制約も発生しました。

また、フィニッシングシステムの開発では、ソースコードの作成は、状態遷移表を見ながらソフトウエア作成者がコードを作成する作業を行いましたが、この際に入力ミスが発生しました。今後、より多くのシステム開発に状態遷移表を採用した場合、この作業は大きな課題になると考えました。

今回の状態遷移表設計において、状態遷移表が肥大化した大きな要因は、セルの中に条件判断を上手く記述できない事にありました。今回のように市販の表計算ソフトを活用して状態遷移表を作成した場合は、操作上、セルの中を分割する事が難しく、新たな状態を作成して対処する事となります。この結果状態の数が多くなり、状態遷移表が肥大化してしまいました。これに対して、ZIPCはセル内での条件分割機能があります。フィニッシングシステムのオペレーション機能について、ZIPCの条件分割機能を使用して状態遷移表を再設計しました。その結果、状態遷移表のセル数を約1/6にでき、ZIPCが状態遷移表の肥大化防止に有効である事が確認できました。

ソースコードの不一致については、ZIPCが持つソースコード自動生成機能を活用する事によって改善が可能であると考え、今回状態遷移表設計手法を導入したカラープリンタの画像形成機能の一部について、ZIPCを活用して状態遷移表による再設計を行い、ZIPCの自動生成機能によるソースコードの自動生成を行いました。

その結果、状態遷移表で記述した分岐条件等が自動的にコード化され状態遷移表とソースコードの不一致が回避されたと同時に、ソフトウエア設計者によって入力するソースコードが1/3に軽減されました。また、最終的に生成されたオブジェクトコードは、ZIPCを使用せずに作成した時とほぼ同等でした。

ZIPCの活用は、以上のような効果がありますが、導入の初期段階ではツールに習熟するための時間が必要となります。

## 7 むすび

情報機器の制御ソフトウエア設計に状態遷移表設計手法を導入する事によりソフトウエア設計段階での不具合発生を低減できる事を確認しました。同手法は多数の状態と事象を管理するプログラムの設計に有効な手法であると考えます。

今後、同手法をより多くの開発に適用していくと共に、支援ツールも含めより効果的な導入方法についても検討していく必要があると考えます。

**参考文献**


[1] 高田広章,情報処理,Vol.38 No.10,870,(1997)

[2] 川口晃,岸知二,門田浩,情報処理,Vol.38 No.10,881,(1997)

[3] 渡辺政彦,リアルタイム制御CASE,初版,電子開発学園編,電子開発学園出版局,(1993)

[4] 渡辺政彦,石田哲史,戴志堅,リアルタイム環境へのCASE導入, 初版,電子開発学園編,電子開発学園出版局,(1994)



(わたなべ さとし / たき けんじ / くろはた たかお)