ソフトウェア工学 80－7
（１９９１．７．１８）

EAGLE/P ﾃﾞｨｸｼｮﾅﾘを用いた ﾌﾟﾛｸﾞﾗﾑｼﾞｪﾈﾚｰﾀの開発と適用効果

大野　治　　森岡　洋介　　降旗　由香理

ファコム・ハイタック　（株）　ハイタック本部

構造化ソフトウェア開発支援システムＥＡＧＬＥ/Ｐにおけるプログラム生成方式では、生成対象となる処理がファイルおよびレコードレベルの処理に限られていた。このため、プログラムの内容となるチェック、編集処理等の生成が不可能であり、生成率向上の点においては限界に達していた。

そこで、プログラム生成機能の強化を図り、データ中心アプローチの考え方を取り入れた、新しいプログラム生成機能を開発した。これは、項目単位の処理を定義した独自のディクショナリを開発し、項目レベルの処理を生成可能とするものである。

そして、本ツールをプロジェクトに適用した結果、プログラムの生成率の著しい向上とこれに伴う、信頼性および生産性の向上という効果を得た。

Development of a program generator using EAGLE/P dictionary and effect

Osamu Ohno　　YOSUKE MORIOKA
Yukari Furuhata

FACOM-HITAC LIMITED　HITAC DIVISION

EAGLE/P is a structuralized software development support system. Processing that it becomes a generation object was limited to disposing of a file handling and a record handling level by this program generator. Therefore it was impossible to generate processing of a check and an editing of item handlling level and so on .

Then we have developed the new program generator in which the way of thinking of data oriented approach is incorporated.

This develops the original dictionary that defines processing of an item unit, and generation of processing of an item handling level is enabled.

And then, this toolwas applied to a project. In consequence the generation rate of a prrogram improved remarkably. An effect that reliability and productivity improved was gained by this tool.

1．はじめに

EAGLE/P は増大するソフトウェアの需要に応えるために開発された、構造化ソフトウェア開発支援システムである。これまで本システムは、生成方式が不完全であったため、生成率の低さに問題があった。このため、今後増大の一途を辿ると予想されるシステム開発の規模を考え、プログラム生成方式を改良し、生産性の向上を図る必要が生じてきた。

そこで、ソフトウェアの部品化にデータ中心アプローチの考え方を取り入れることにより、ディクショナリの再開発を行い、プログラムの生成機能の強化を図った。そして、実際のプロジェクトに適用した。

本稿では、この新しいEAGLE/P におけるプログラム開発の考え方と適用効果について述べる。

2．EAGLE/P プログラムジェネレータの概要

2.1 プログラムジェネレータの位置付け

プログラムジェネレータは、プログラムの骨格(スケルトン)として、標準パターンを用意し、これを基に、プログラム仕様情報と外部仕様(画面、帳票、ファイル、レコード等)情報を付加することによって、プログラムを自動的に生成するツールである。

今回、新たにディクショナリを開発し、この情報をプログラムにとり込むことができるようにした。(図2-1 参照)

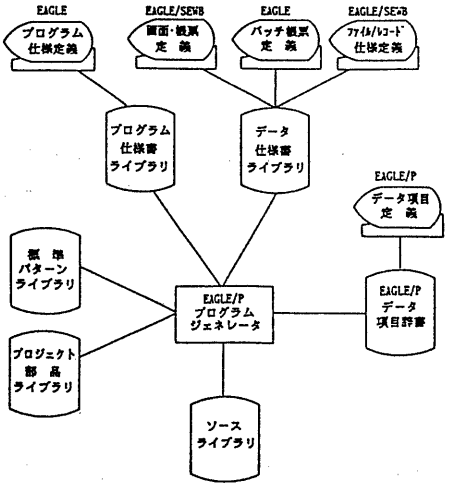

図2-1 ﾌﾟﾛｸﾞﾗﾑｼﾞｪﾈﾚｰﾀの位置付け

2.2 項目単位の処理の生成

EAGLEでは、プログラムの構造を、図2-2に示すようなファイル操作、レコード操作および項目操作の各レベルから構成している。

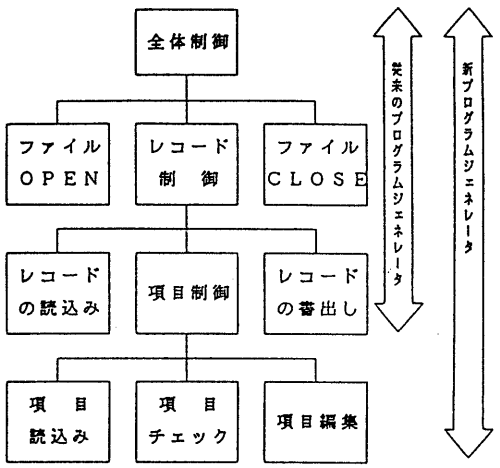

図2-2 ﾌﾟﾛｸﾞﾗﾑの制御構造と生成範囲

従来のプログラムジェネレータでは、プログラムの骨格(スケルトン)としての標準パターン、プログラム仕様情報および外部仕様情報により、レコード操作のレベルまでをカバーしたに過ぎない。

したがって、プログラムの内容となるチェック、編集等の項目レベルにおける各処理が生成できないことになる。結果として生成後の追加コーディングに頼らざるを得なくなり、この追加コーディングがプログラムの生産性および信頼性を低下させる要因となっていた。

そこで、今回の新しいプログラム生成では、プログラムのパターンおよび外部仕様からもう一歩踏み込み、項目レベルの処理を生成対象とした。そして、これを実現するために、項目操作の段階での情報を定義した、独自のディクショナリを開発した。

3．ディクショナリの開発

3.1 データ中心アプローチの考え方の適用
－データ項目単位の部品化－

従来の部品化の考え方は、個々のプログラムに対して部品を機能分割により定義するものであった。このため、部品の定義は設計者によって異なり、部品の選択は人間が行わなければならないため、大規模プロジェクトや複数プロジェクトにわたる再利用が困難であった。

そこで、部品を、プログラム単位ではなく、データ項目単位に定義する方法を採用した。定義は、個々のプログラム生成時ではなく、システム設計の段階ですべてのデータ項目を定義する。これにより、システムに含まれるすべてのデータ項目を資源として一元管理することが可能となる。

ここで、従来のディクショナリとの相違を明確にするために、図3-1 を用いて、説明する。図3-1 は「生年月日」というデータ項目を例にとり、システム内で1つのデータ項目に対して行われる処理をモデル化したものである。

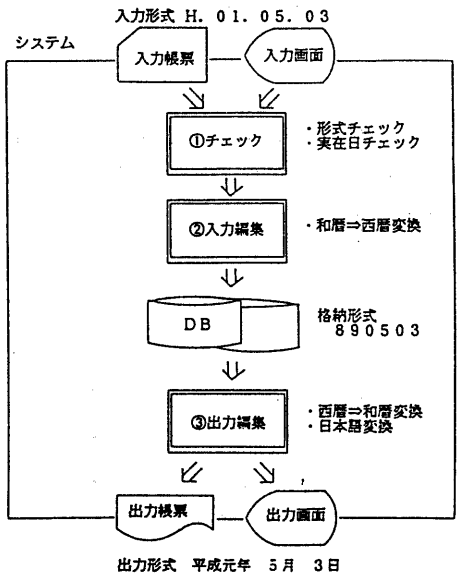

図3-1 ﾃﾞｰﾀ中心ｱﾌﾟﾛｰﾁの考え方の適用

従来のディクショナリでは、帳票、画面を通してシステム内に入力される項目（H. 01.05.03）とシステムから出力される項目（平成元年５月３日）を各々登録する。しかし、本ディクショナリにおいては、１データ項目（生年月日）につき、１登録であり、これはディスクの格納形式で登録される。（890503）そして、これを基準として、人間がどのように入力を行うか、出力を行うかについて、各々をそのデータ項目の入力編集、出力編集処理として定義するのである。

この点を踏まえた上で、図3-1 に詳細な説明を加える。

システムに対し、'H.01.05.03' という形式の「生年月日」のデータが与えられると、①その形式および妥当性をチェックする処理（チェック処理）を行い、②データ項目を格納するときには、'890503' という西暦日付の形式に変換されてDBに格納される（入力編集）。そして、このデータを出力する場合には、③格納状態のデータから出力形式 '平成元年5月3日' に変換するため

に和暦および日本語変換処理を行う。(出力編集)

このようなディクショナリの実現によりシステム全体を支える基盤を確立すると同時に、この基盤を通じてデータ資源の有効活用を図ることが可能になった。

3.2 ディクショナリの構造

ディクショナリは、宣言部と手続部より構成される。宣言部では各データ項目に関する名称、属性等を定義する。手続部ではデータ項目に対する処理を、チェック、入力編集、出力編集に分けて定義する。(図3-2参照)

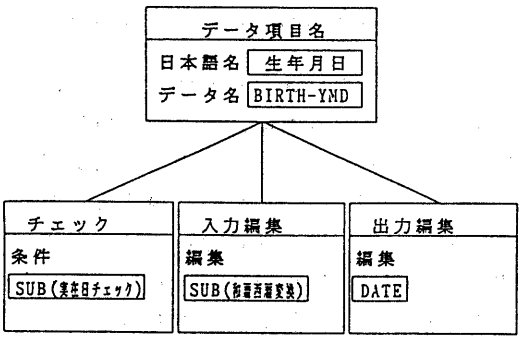

図3-2 ディクショナリの構造

本ディクショナリの構造を決定する際に考慮した点を以下に述べる。

①チェック、編集処理の定義はキーワードによる選択方式とした。

キーワードとは処理機能をまとめて1語で表現したものである。これにより定義の記述量を最小限に抑えディクショナリの品質を高めることを可能にしたと考える。

しかし、適用業務に柔軟に対応するために、COBOL言語により直接処理を記述することも認めている。

尚、キーワードは表3-1 に示すような約15種類を揃え、この組合せも自由であり、広い範囲の機能を定義することを可能にしている。

表3-1 キーワード一覧

No		キーワード	機能
1	チェック条件	ALPHABETIC	英字チェック
2		NUMERIC	数字チェック
3		SPACE	スペース不許可
4		LVALUE	LOW-VALUE不許可
5		ZERO	ZERO不許可
6		CONST(定数,・・・)	定数チェック
7		AREA(範囲,・・・)	範囲チェック
8		SUB(ｻﾌﾞﾙｰﾁﾝ,ﾊﾟﾗﾒﾀ,・・・)	サブルーチンによるチェック
9		FREE	COBOL文法による記述
1	編集処理	DATE	日付編集
2		TIME	時刻編集
3		COMP(計算式)	計算
4		STRING	文字列結合
5		SUM(ブレークキー)	集計
6		SUB(ｻﾌﾞﾙｰﾁﾝ,ﾊﾟﾗﾒﾀ,・・・)	サブルーチンによる編集
7		FREE	COBOL文法による記述

②1つの項目に対し、複数通りのチェック、及び編集処理を定義可能とした。

データ項目に対する処理、特に出力編集処理は、出力媒体によって異なる場合が多い。このため、データ項目名に別名を付加することにより異なった処理を選択できるようにした。

③従属項目の定義を可能とした。

従属項目を持つ項目(例えば、「年月日」は、年、月、日の従属項目を持つ。)に対し、従属関係を内部構造として定義できる。これは、恒久的な親子関係を持ったデータ項目群をまとめて1つのデータ項目として定義することであり、上位項目の定義中で従属項目に対する定義が行えることを意味する。この機能によりディクショナリに登録するデータ項目数を削減できるとともに、従属項目に関する定義がファイル情報の定義から不要となるため、システム全体の保守性を高めることが可能になる。

4．プログラム生成方式

4.1 入出力定義情報からのプログラム生成

前述の通り、本プログラムジェネレータでは、プログラムの骨格である標準パターンにプログラム仕様情報、外部仕様情報、さらに、ディクショナリに登録した各処理の情報を付加してプログラム生成を行う。ディクショナリの情報を取り込むにあたっては、ディクショナリに登録されたチェック条件および編集処理のうち、必要な項目に対してのみパターンに取り込む必要がある。そこで、自動的に必要なデータ項目の選択を行い、さらに、これにより、作業効率の向上と定義ミスの防止を実現した。

方法として、入出力媒体に含まれるデータ項目に着目し、その関係から取り込むべきデータ項目を自動的に選択するようにした。

図4-1 を用いて具体的に説明する。

まず、データ仕様書ライブラリの参照により、情報を取り込む必要があるデータ名が決定される。

（帳票定義‥‥ﾃﾞｰﾀ名「氏名」「現住所」
画面定義‥‥ﾃﾞｰﾀ名「氏名」「生年月日」）

次に、標準パターンライブラリから、該当する標準パターンを選択することにより取り込まれる処理と位置が決定される。

（帳票定義‥‥ﾃﾞｰﾀ名「氏名」「現住所」の出力編集処理）
（画面定義‥‥ﾃﾞｰﾀ名「氏名」「生年月日」のチェック、入力編集、出力編集の各処理）

＊ 帳票出力のパターンには、出力編集の処理のみを取り込むようになっているため、ﾃﾞｰﾀ名「氏名」「現住所」のチェック、入力編集の各処理は取り込まない。

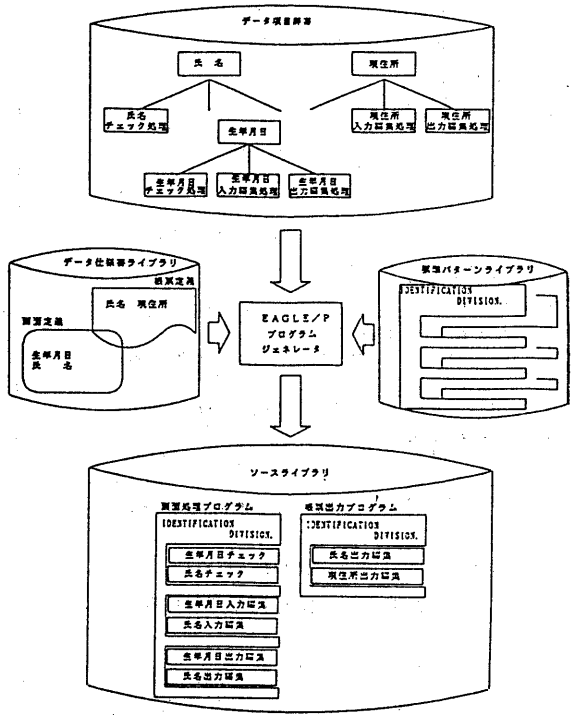

図4-1 プログラム生成方式

4.2 各種処理の展開

ディクショナリにおける処理の定義方法とそのプログラムへの展開内容について、例を用いて具体的に説明する。

(1) 出力編集処理の展開

図4-2（次頁）をもとに説明する。

データ名「合計給与」は「課コード」をコントロールブレークキーとしてブレークが生じた場合の給与の合計を示す。

この内容をディクショナリでは、出力編集欄に集計用のキーワード'SUM'とコントロールブレークキー'課コード'を記入するだけで定義できる。そして、これを反映し、プログラムでは、①集計用作業領域の確保、②これをゼロクリア、③加算する命令文、④出力項目へ転送するMOVE文が各々展開される。尚、これらの展

開位置は、各々作業領域、集計、前、後処理と定められている。

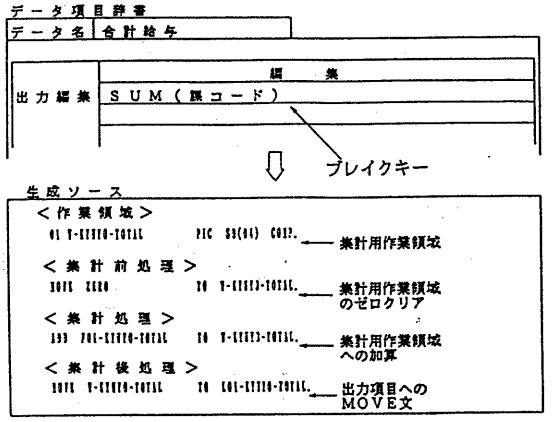

図4-2 出力編集処理の生成例

(2) チェック処理の展開

図4-3 をもとに説明する。

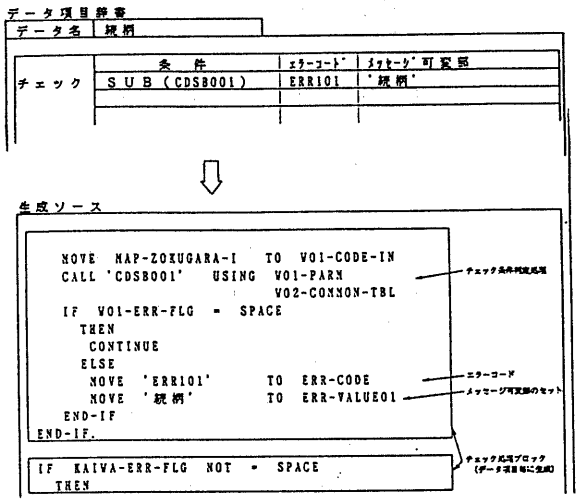

図4-3 チェック処理の生成例

データ名「続柄」はサブルーチンCDSB001' によりチェックされ、エラー時にはエラーコード 'ERR101'とエラーメッセージに '続柄'を出力する。この内容をディクショナリではチェック処理欄にキーワード 'SUB' サブルーチン名、エラーコードおよびメッセージ可変部を記入するだけで定義できる。そして、これを反映し、プログラムでは、①チェック条件判定処理、②エラー処理が各々展開される。

以上、単なる集計の命令文、あるいはサブルーチンのCALL文にとどまらず、関連した一連の処理を展開できる点が本ツールの特徴であり、これ故に、利用効果の高さが期待できるのである。

4.3 ＭＯＶＥ文の生成

データ項目の中には、編集処理を定義する必要がないものがある。すなわち、ファイル、あるいはＤＢに格納されているデータを、形式を変えることなく、画面、あるいは帳票に出力するだけのデータ項目である。事務系アプリケーションプログラムでは、特にこのような項目が多数存在する。

そこで、ディクショナリ中に、あえて編集処理の定義をしない場合には、ＭＯＶＥ文を生成するようにした。図4-4 に例を示す。

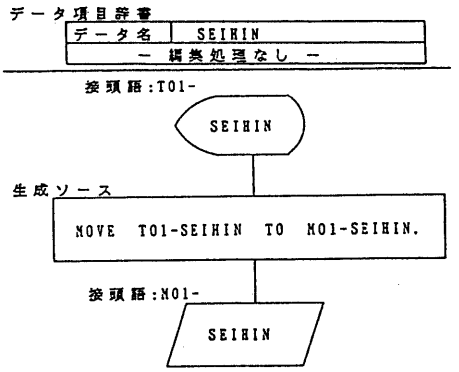

図4-4 単純なＭＯＶＥ文の生成例

このことにより、以下に示す2点の実現を可能にした。

①ディクショナリの定義量、ひいてはディクショナリの作成工数を削減する。

②ディクショナリが100％完成していない場合でも、プログラム生成ツールが充分機能する。

5. プロジェクトへの適用

本プログラムジェネレータを実プロジェクトに適用した。このプロジェクトにおける単体プログラムの開発規模は以下に示す通りである。

(1) 開発規模(業務プログラム)

1	ﾊﾞｯﾁ系	規模 (ks)	492
		ﾌﾟﾛｸﾞﾗﾑ数(本)	637
2	ｵﾝﾗｲﾝ系	規模 (ks)	344
		ﾌﾟﾛｸﾞﾗﾑ数(本)	768
3	合計	規模 (ks)	836
		ﾌﾟﾛｸﾞﾗﾑ数(本)	1405

(2) 開発期間　7か月　(90.9～91.3)

(3) 開発人員　約40名

6. EAGLE/P プログラムジェネレータの適用効果

6.1 評価方法

本プログラムジェネレータは、従来のプログラム生成機能では生成が不可能であった、プログラムの内容となるチェックおよび編集処理を自動生成することによって、生成率を向上させる目的をもって開発された。このため、適用効果の評価のおいては生成率を重視し、これが信頼性および生産性の向上にどのような影響を及ぼしているかという観点で行った。

そして、評価方法は、従来と今回のプログラム生成ツールを用いて開発されたプログラムの各データを比較することにした。データは、「開発グループ間の作業環境」の相違から生じるデータの相違がないように同一グループのプログラムに限定した。さらに、工程の途中で仕様に変更、追加があったものを除外し、データの整備されているもののみを抽出した。

6.2 信頼性における効果

6.2.1 バグ密度による考察

信頼性の指標として、生成率とバグ密度を取り上げる。表6-1 に従来と今回のプログラム生成ツールを用いて開発されたプログラムの生成率とバグ密度を示す。

尚、バグ密度のデータの数値は、従来のツールを用いた場合のデータを100とした相対数値とする。

表6-1 生成率とバグ密度の比較

(ﾌﾟﾛｸﾞﾗﾑ数：従来 43本、今回 44本)

項番	項目		従来	今回
1	生成率 (%)		68	81
2	1ﾌﾟﾛｸﾞﾗﾑの規模(ks)		0.65	0.58
3	バグ密度	単体ﾃｽﾄ(UD)	100	87
		結合ﾃｽﾄ(CD)	100	73
		総合ﾃｽﾄ(SD)	100	10
		合計	100	66

表6-1より、本ツールを用いたプログラム開発について、

①生成率が向上していること。

②全工程においてバグ密度が減少していること。

③従来のツールを用いた場合のバグ密度との差は工程が進む程大きくなっていること。

がわかる。

以上のことから、バグ密度の減少は、生成率の向上により、信頼性の高いロジックを多く生成できるようになり、プログラマによる修正ロジックが減少したことによると推測できる。さらに、③から、信頼性の向上は後工程に引継がれ、むしろ工程が進むほど適用効果が顕著になることが予想される。

6.2.2 バグ密度に影響を与える因子

バグ密度の減少が生成率の向上によるこ

とを確認するために、生成率とバグ密度の関係を調べた。その結果、図6-1を得た。尚、データは、個々のプログラムによるばらつきを是正するために、生成率を10%毎に区切り、各区間内における生成率およびバグ密度の平均値を採用した。縦軸は生成率60%台の単体テスト時のバグ密度を100とした相対数値である。

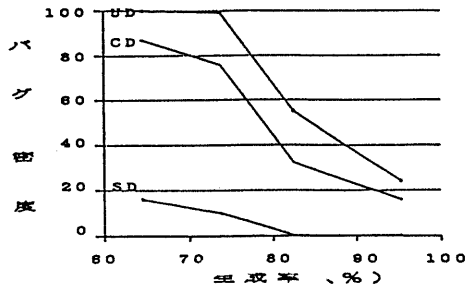

図6-1 生成率とバグ密度

これより、バグ密度には生成率との関係がみられ、生成率の向上によるバグ密度の減少が確認できた。

6.3 生産性における効果

生産性の指標として、生成率とコンパイル回数を取り上げる。図6-2に今回のプログラムジェネレータを用いて開発されたプログラムの生成率とコンパイル回数の関係を示す。尚、6.2.2節同様に、データは生成率を10%毎に区切り、各区間内における生成率およびコンパイル回数の平均値を採用した。縦軸は生成率60%台のコンパイル回数を100とした相対数値である。

これより、生成率の向上により、コンパイル回数が減少することがわかり、「生成率の向上は生産性の向上をもたらす」ことを確認できた。

さらに、作成工数を調べたところ、今回は従来に較べて1.4倍になっているという結果を得た。以上より、生成率を向上させた本ツールの適用が、生産性の向上をもたらしたと考えている。

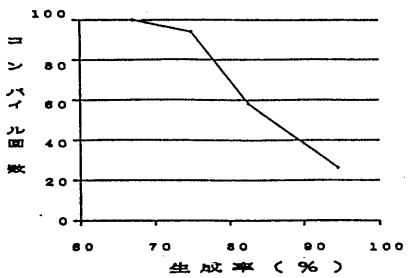

図6-2 生成率とコンパイル回数

7. おわりに

本稿では、データ中心アプローチの考え方を取り入れた、ディクショナリとプログラムジェネレータを開発し、プロジェクトに適用した。その結果、本ツールの適用により、プログラムの生成率が向上し、これに伴う、信頼性と生産性の向上が認められ評価を得た。

今回報告したシステム開発方法では、ディクショナリから大きな恩恵を受けられる反面、その内容はディクショナリの品質に左右されるという問題がある。したがってディクショナリの作成し易さ、検証の容易さを図る必要がある。

上述の問題を克服して、今後、ソフトウェアの生産性および信頼性のより一層の向上を推進したい。

参考文献

(1) 森岡他：「EAGLE/P データ中心アプローチ支援ディクショナリの開発」情報処理学会第41回全国大会，4G-8 PP.5-205～5-206，1990

(2) 堀内　一著：「データ中心システム設計」オーム社

(3) 高橋他：「データ中心アプローチによるシステム開発」情報処理学会第33回全国大会，3F-3