大規模基幹系システム再構築時における開発支援ツールの導入効果について

みずほ情報総研（株）

■ 執筆者Ｐｒｏｆｉｌｅ ■

成田　謙介

2009 年　みずほ情報総研（株）入社
基幹系システム担当

■ 論文要旨 ■

近年は、Web・クラウドサービス上での開発案件が主流であるが、メインフレーム開発も依然として発生している。メインフレームは、レガシー言語を用いた開発が多く、今後を見据えた人材確保・品質確保が課題である。

今回、基幹系システムの再構築を実施するにあたり、これらの課題を解消するために開発支援ツールを導入した。具体的には、プログラム自動生成と単体テストケース自動生成機能があるツールを使用し、言語スキルを保有しない要員での開発及び、テストを目指す。

ツールを導入した結果、開発要員確保やプログラム品質の向上を図ることができた。6,800 本プログラムの新規開発及び単体テストを 3 ヶ月述べ 1,000 人月で実施し、開発期間・品質指標値についても計画時の基準に達した。

今後の保守フェーズにおいても、継続して開発支援ツールを使用することで、品質確保に加え、人材の流動化やメンテナンス性の向上が期待できる。

■ 論文目次 ■

１．はじめに ……《 3》
１．１　背景 ……《 3》
１．２　大規模基幹系システム再構築の概要 ……《 5》

２．本論 ……《 6》
２．１　開発支援ツールの導入目的 ……《 6》
２．２　開発支援ツールの機能 ……《 6》
２．３　ツール導入結果 ……《 9》

３．おわりに ……《 14》
３．１　評価 ……《 14》
３．２　メンテナンスフェーズに向けて ……《 14》

■ 図表一覧 ■

図１　開発アーキテクチャの経年推移……《 4》
図２　開発言語の経年推移……《 4》
図３　基幹系システムの概要図……《 5》
図４　製造工程作業フローとInterdevelop機能紐付け……《 7》
図５　日本語設計書エディタのイメージ……《 9》
図６　COBOLプログラム生成イメージ……《 9》
図７　分岐網羅ケースイメージ……《 10》
図８　セルフチェックリスト抜粋……《 11》
図９　開発支援ツールでのテストイメージ図……《 13》

表１　サブシステムＡの開発規模・テスト項目数……《 6》
表２　製造工程作業項目とツール機能の紐付け……《 7》
表３　プログミング工程の作業工数……《 11》
表４　単体テスト工程の作業工数……《 12》

１．はじめに

１．１　背景

近年、ソフトウェア開発はクラウドサービスを用いる傾向であるが、依然としてメインフレームで稼動する基幹系システムも多く、特に金融系システムではメインフレームでの開発が主流である。
そしてその開発スタイルには一般的に下記の特徴がある。

・大規模開発（大人数）
・ウォーターフォール型の開発
・レガシー言語（COBOL/アセンブリなど）を使用

業務システム内には複数のサブシステムがある場合が多く、各サブシステム・機能の開発工程スケジュールが重複するため、ピーク時には数百名の要員が必要である。そのため、スキルセットを保有したＳＥ、要員の確保が課題であった。
また、大人数で設計書、テスト仕様書を作成した場合の記載内容差異、品質のバラつきについても長年の課題である。

これらの課題を解決するために、従来は人材育成や、ドキュメントの雛形化、開発ルール作りを行い、ヒューマンスキル・業務経験で補う手法が取られてきた。

しかしながら、図１・図２に示すとおり、メインフレーム開発案件、及び COBOL 開発案件は低下する傾向にある。そのため、大規模プロジェクトの場合、新規人材が見込めず、経験者の減少に伴い、今後は人材確保がより一層厳しくなることが予想される。

今回、メインフレームの再構築プロジェクトにおいて従来の開発スタイルの課題、特に最も要員を必要とする製造工程の課題に着目し、開発支援ツールを導入することで、ヒューマンスキルに依存せず高品質・短期間での開発を目指す。

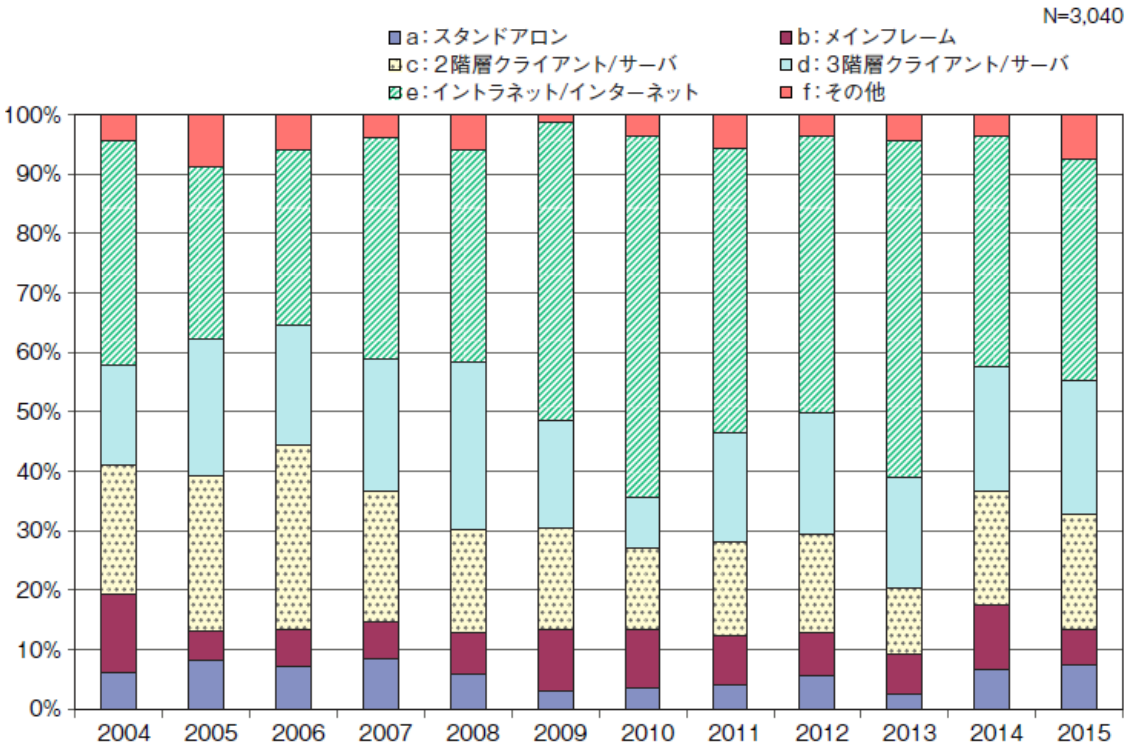


図１　開発アーキテクチャの経年推移

（引用『ソフトウェア開発データ白書 2016-2017』、2016 年、43 ページ図表 4-4-7）

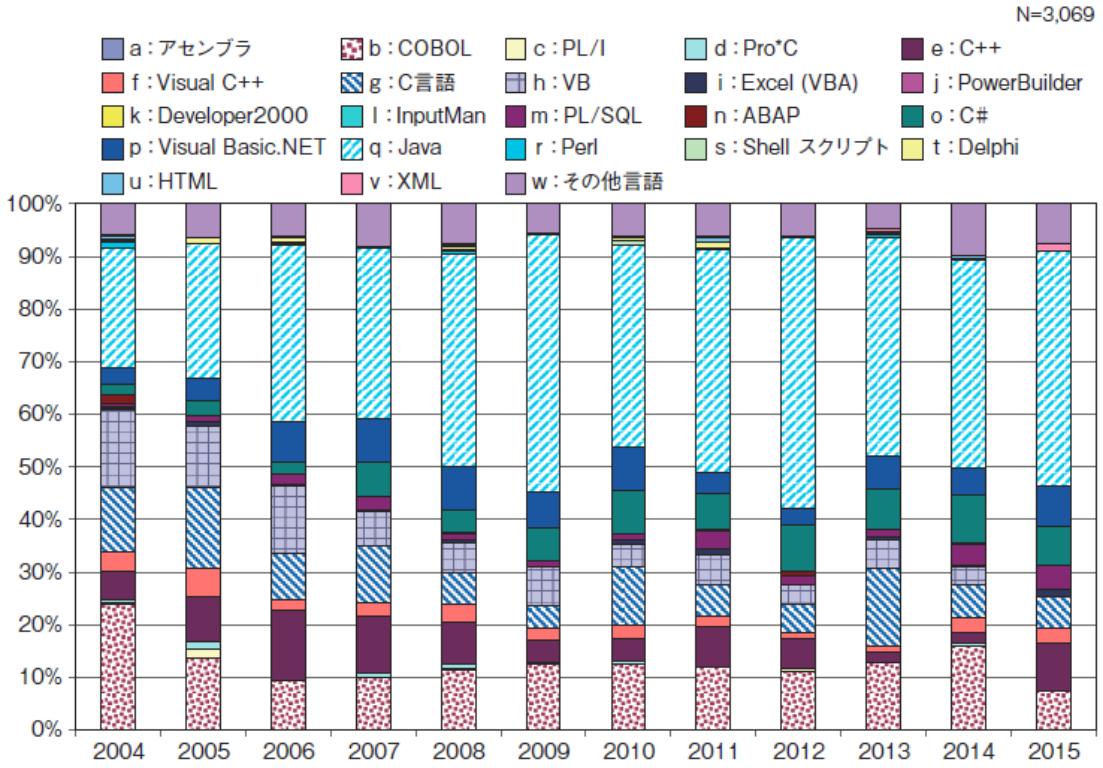


図２　開発言語の経年推移

（引用『ソフトウェア開発データ白書 2016-2017』、2016 年、47 ページ図表 4-4-15）

１．２　大規模基幹系システム再構築の概要

今回、開発支援ツールを導入したプロジェクトの概要について述べる。

１．２．１　プロジェクト概要

弊社は、基幹系システムのソフトウェア開発を担っている。今回、お客さまへのサービスや利便性の向上を一層進めていくことを目的に既存システムを再構築・一元化するプロジェクトを発足させた。

開発手法は、既存システムの部分改修やシステム移行ではなく、基幹系システム全面刷新である。既存サブシステムについて、最適な業務単位に集約及び再整理し、メインフレーム上で再構築する。サブシステム内のプログラムは新規作成とする。

なお、本論文では、基幹業務システムの１サブシステム（以降サブシステムＡ）の製造工程で実施した手法・成果について述べる。

図３に基幹系システムの概要図を示す。

＜新システムへの移行（イメージ図）＞

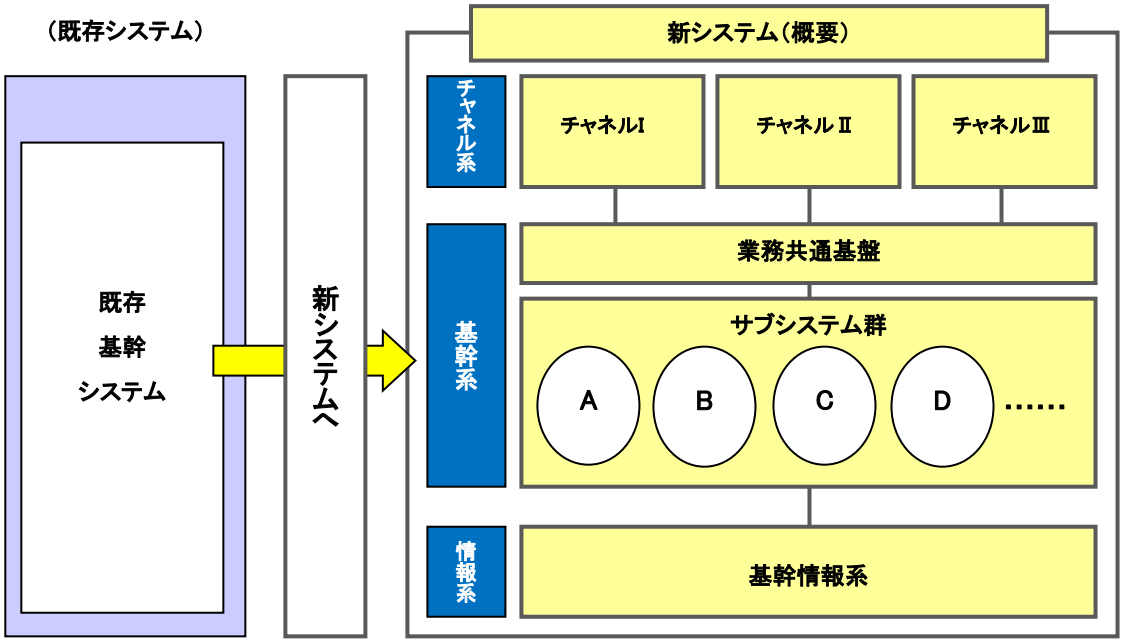


図３　基幹系システムの概要図

１．２．２　製造工程の定義

本論文では製造工程を以下の 4 作業と定義する。

① プログラム設計書作成
② プログラミング
③ 単体テスト仕様書作成
④ 単体テスト

なお上記の工程については、メインフレーム上ではなくすべてクライアント（Windows）環境で実施する。

１．２．３　工程スケジュール・開発言語

サブシステムＡの製造工程スケジュールは約 3 ヶ月。サブシステムＢ以降についても同様のスケジュールである。また、サブシステムＡの開発言語は COBOL とする。その他サブシステムの言語は COBOL または Java の何れかである。

１．２．４　構築規模

表１にサブシステムＡのプログラム構築規模、単体テスト項目数を示す。
また、再構築プログラム本数はオンライン処理プログラム約 5,500 本とバッチ処理プログラム約 1,300 本の計 6,800 本である。

開発規模		開発規模計：	新規：	修正	ﾃｽﾄ項目数：
単位：	KStep	5796.1	5796.1	0.0	318,012

表１　サブシステムＡの開発規模・テスト項目数

１．２．５　要員

サブシステムＡの開発要員（管理者を除く）は 3 ヶ月で延べ 1,000 人月、月平均 330 人月である。

２．本論

２．１　開発支援ツールの導入目的

基幹系システムの再構築を実施するにあたり、製造工程に開発支援ツールを導入する「理由」と「期待する導入効果」について述べる。
開発支援ツールは富士通社製品 Interdevelop Designer を採用する。

２．１．１　ツール採用理由

(1) 大量生産時の成果物均質化
大規模プロジェクトにおいて、型決めされたアプリケーション開発手法にて統制を図ることにより、成果物（プログラム／テスト品質等）を均質化する。

(2) コーディング誤りの軽減
日本語設計書から COBOL ソースを自動生成することで、製造工程の効率性を向上し、コーディング作業での作り込みリスクを排除する。

上記に加えて、パイロット開発を実施し、開発体制・技術サポート体制が現実的かの検証を行うと共に、当社作業者によるユーザビリティ検証を実施し、当ツールを選択することに問題がないことを確認した。

2．1．2　ツールへの期待効果

・プログラミング・テスト要員（数百人/月）の確保
・プログラム・テスト品質確保
・プログラミング・テスト期間短縮

2．2　開発支援ツールの機能

Interdevelop Designer は業務プログラムの開発・保守に特化した製品であり、製造工程の各種作業のサポートを行う。図４に製造工程作業フローと Interdevelop 機能紐付けを示す。

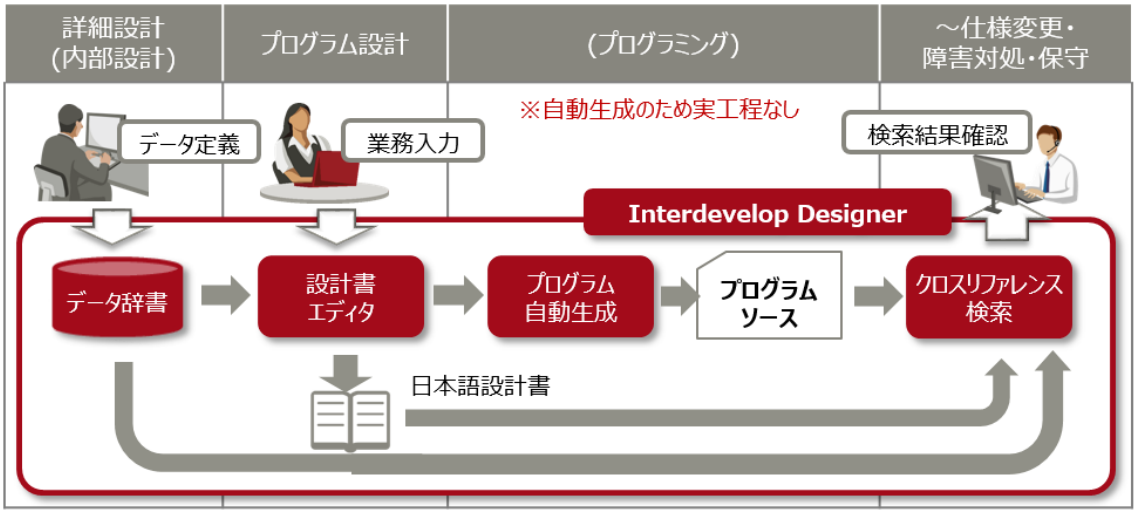


図４　製造工程作業フローと Interdevelop 機能紐付け

（引用『富士通ソリューションマップ』、１ページ、2018）

製造工程の各種作業項目とツール機能の紐付けを表２に示す。

製造工程作業項目	開発支援ツール	
	機能名	概要
（事前作業）	データ辞書	設計に関わるデータを一元管理 事前に日本語設計書エディタ内で使用する項目をデータ辞書に登録
（事前作業）	（プロセス仕様書）	プログラム概要を記載したExcel文書 設計書エディタへの入力を容易にするために事前にプログラム処理概要を記載したドキュメントを準備
プログラム設計書作成	設計書エディタ	プログラム処理を日本語で記述するツール 構文をメニューから選択し、項目名と組み合わせて記述
プログラミング	プログラム自動生成	設計書エディタからCOBOLプログラムを自動生成
単体テスト仕様書作成	テスト支援	テスト仕様書の自動生成、Excel還元 開発者にてバリエーションの追加、予想結果を入力
単体テスト		スタブ、ドライバの自動生成 テスト結果をテスト仕様書（Excel）へ自動反映

表２　製造工程作業項目とツール機能の紐付け

２．３　ツール導入結果

基幹系システム構築で開発支援ツールを導入した目的と導入結果について記述する。

２．３．１　COBOL 開発要員の確保

(1) ねらいと手法

現在のシステム開発では、Web 系開発、Java 言語での開発が主流である。
一方、メインフレーム開発、COBOL 言語の案件は減少傾向にある。今後も同様の傾向になると想定されるため、メインフレームでの COBOL 開発経験者の確保は難しくなっていく。
今回、開発支援ツールを使用し日本語設計書から COBOL プログラムを自動生成することにより、言語固有スキル・経験を保有しない担当者でも開発を可能とすることで要員確保を目指す。

(2) 成果

ａ．開発要員確保

製造工程では、3 ヶ月で延べ 1,000 人月の工数を要し、1 ヶ月あたり平均 330 人程度で開発を実施した。
約半数がメインフレーム及び COBOL 言語での開発が未経験であった。
設計書エディタは日本語で扱え直観的な操作が可能なことから、混乱なく利用が定着し、COBOL 知識がなくとも着手は可能であった。工程初期段階においては設計書エディタの習熟度が不足していたことにより、生産性が上がらない傾向にあった。そのため、製品マニュアルのみではなく実際の作業手順に沿ったマニュアルを作成し、併せて、各領域代表者に対してサンプルプログラム作成研修を実施することで、理解度の促進を図った。
後半以降習熟度が上がるにつれ、当初想定していた生産性数値に近い結果となった。
結果、予定期間内に製造工程を完了することができた。

ｂ．開発拠点分散

開発支援ツールはメインフレームではなく Windows 上で実施するため、ホスト接続がされていない開発拠点においても作業可能であり、100 人月程度をニアショア開発及びオフショア開発に委託できたことも要員確保につながった。

２．３．２　プログラム品質確保

(1) ねらいと手法

従来はメインフレーム上で直接プログラミングを実施していたため、担当者ごとのロジックの差異、入出力項目定義誤りによる手戻り、プログラム仕様書とプログラムソースの不一致が生じていた。特にプログラム仕様書とプログラムソースの不一致は、障害発生時の原因究明の見誤りや影響調査時の誤認につながるが、現行システムではヒューマンチェックに依存していた。
今回、日本語設計書エディタにて「入出力項目」及び「構文」を選択して記述することで担当者ごとのロジックの差異を解消する。また、プログラムを自動生成し、プログラム仕

様書とプログラムソースの不一致を解消することでプログラム品質の向上を図る。

(2) 成果

a．設計書記述レベルの統制

日本語設計書エディタを使用することで、設計書は担当者によるロジックの差異を少なくすることができ、記載レベルの統制が図れた。図５、６に設計書エディタでの記述イメージ、COBOL プログラム生成イメージを示す。

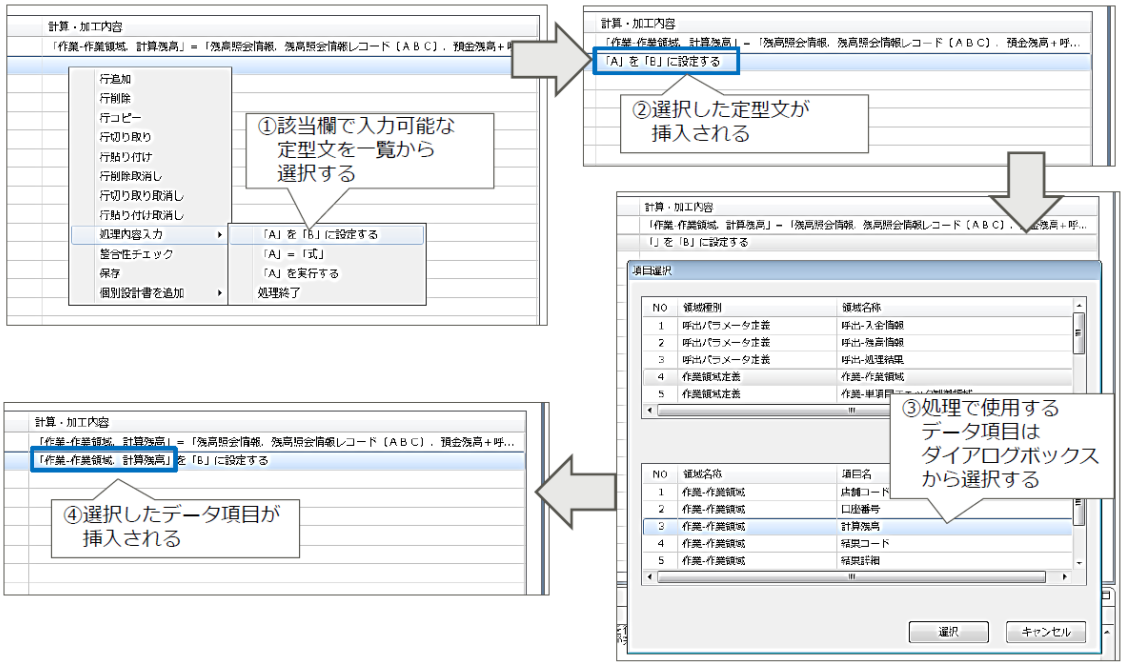


図５　日本語設計書エディタのイメージ

（引用『InterdevelopCOBOL 説明書（概要）』、2014 年、11 ページ）

```
033070*****************************************************************
033080*     計加－２－当日取引のチェック処理
033090*****************************************************************
033100 PRC-0008 SECTION.

～省略～

034220*  ／／＠＠０６２３－Ｓ－４＠＠
034230*( 計算・加工処理仕様－計加－２－当日取引のチェック処理：2400
034240*/ 「呼出－呼出パラメーター定義－１．Ｌ－プログラムインターフェー
034250*/ ス入力固有部．Ｌ－取引チェックパターンコード」を「作業
034260*/ －作業領域．Ｌ－集団項目．Ｌ－取引チェックパターンコー
034270*/ ド」に設定する
034280***                          - 24000
034290      MOVE LSHKNZDCHKPTNCD OF
034300        PRO1623-P01
034310                             TO
034320      LSHKNZDCHKPTNCD OF WRK-0001.
```

日本語設計書の内容がコメント文となる。

項目はデータ辞書に登録された英字名でソース生成する。

図６　COBOL プログラム生成イメージ

b．不良発生件数の低下

現行システムの不良摘出密度の品質指標値(0.98)に比べ、今回の不良摘出密度は（0.3）と不良発生件数は 1/3 程度となった。これは、プログラムが自動生成されることに伴い従来開発のコーディングミスが減少していることが奏功していると分析しており、品質にも問題はないと評価する。

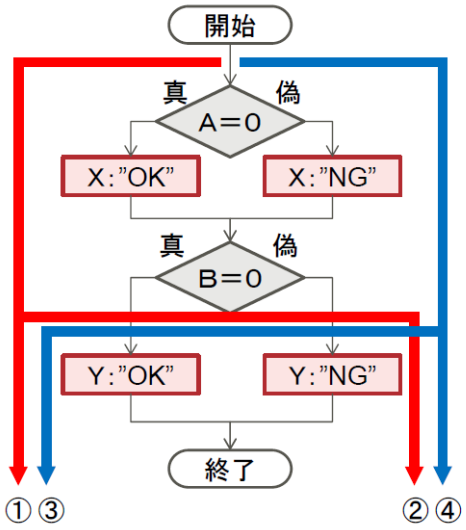
(3) 設計書エディタ使用時の考慮点

設計書記述において使用できる構文が限定されており、手組みより設計の自由度は低いが、その分、記述レベルの統制を図ることができる。一方、COBOL プログラミングで使用している文法、例えば、EVALUATE 文に対応する日本語構文はなく、IF 文に対応する日本語構文を複数回使用して記述する必要がある。

2．3．3　テスト品質確保

(1) ねらいと手法

ツールから分岐網羅ケースを自動生成することで、テスト項目の抜け漏れや担当によるテスト粒度の差異をなくし、プログラム品質の確保に充足するテストケースを実施する。図７に自動生成される分岐網羅ケースイメージを示す。



テストケース

ID	条件	真偽	抽出ケース
①	A = 0 B = 0	真 真	A = 0 B = 0
②	A = 0 B = 0	真 偽	A = 0 B ≠ 0
③	A = 0 B = 0	偽 真	A ≠ 0 B = 0
④	A = 0 B = 0	偽 偽	A ≠ 0 B ≠ 0

テストデータ

入力値	
A	0
B	0
A	0
B	1
A	1
B	0
A	1
B	1

期待値

期待値	
X	OK
Y	OK
X	OK
Y	NG
X	NG
Y	OK
X	NG
Y	NG

図７　分岐網羅ケースイメージ

（引用『InterdevelopCOBOL 説明書（概要）』、2014 年、18 ページ）

(2) 成果

ケース自動生成により全プログラムに対し分岐網羅を充足しており、網羅性は問題なしと思料。また、網羅ケースは自動生成されるため、担当者は閾値やロジック確認のテストケース追加に注力することができる。担当者が追加したケースの具体例は以下のとおり。

例１：1～10 の値を許容するコード項目について、0、１、2、9、10、11 の
　　入力バリエーションを実施
例２：同一作業領域を 2 回使用するケースで初期化の正当性を確認

上記のとおり、テストケースの十分性についても必要なテストケースが充足していることをレビューを通して検証しており、テスト品質に問題はないと評価する。

(3) 自動生成ケース内容の考慮点

自動生成される分岐網羅ケースが、以下の理由で完全には生成されない場合がある。

・デットロジックがあり網羅できない

・分岐条件が複雑であり網羅できない

上記の場合はアラームが出力されるため、担当者が自動生成ケースの内容を確認、追加する必要がある。

２．３．４　プログラミング期間短縮

(1) ねらいと手法

開発支援ツールのプログラム自動生成機能を使用することにより、従来実施していたプログラム仕様書作成とプログラミングが同一作業となるため、開発期間の短縮化が図れる。

(2) 成果

表 3 のとおり従来開発では 1 プログラム平均 3 人日を要していたが、今回開発では平均 1 人日で実施することができた。これは事前に設計書エディタに記述するデータ項目・業務ロジックを準備していたため、エディタ入力時間を削減できたことによる部分が大きい。

開発作業の差異

従来開発		今回開発	
作業内容	工数	作業内容	工数
①プログラム設計書作成	2人日	①設計書エディタ入力	1人日
②プログラミング	1人日	②プログラミング自動化	0人日

表３　プログミング工程の作業工数

コーディングミスや項目誤りで単体テストエラーなる事象も減少したため、手戻りによる再プログラミング（不良対応工数）についても削減。加えて、エディタ使用時のセルフチェックリスト（図８）を展開し、設計書記述規約を事前に確認することをルール化し、ソース生成時のエラーによる手戻りを軽減した。

観点	No	チェック項目	チェック
3. 処理仕様			
詳細情報	(5)	Noは、規約に準拠し記述されていること。 (Interdevelop設計書記述規約「5．5　ナンバリングの規約」)	□
	(6)	メインフロー ・条件／処理内容は、Interdevelop設計書ひな型「処理仕様(メイン)」に準拠し記述されていること。	□
	(7)	業務固有の処理仕様 ・条件は、プロセス仕様書「(2)業務プログラムフロー」の「業務処理実行条件」が記述されていること。 ・処理内容は、プロセス仕様書「(2)業務プログラムフロー」の「業務処理名」の実行が記述されていること。	□
	(8)	Interdevelop設計書ひな型の処理仕様 ・条件／処理内容は、Interdevelop設計書ひな型の「処理仕様」に準拠し記述されていること。	□
	(9)	多岐条件は、「評価」を使用していること。 (Interdevelop設計書記述規約「4.1.9.　多岐選択文」)	□
	(10)	条件文に、「 "NOT　="と"OR"の組み合わせ」を使用していないこと。 (Interdevelop設計書記述規約「4.1.8.　条件文　(2) 条件文2(複数条件)」)	□
	(11)	条件文に、計算式を使用していないこと。(例. A＝B＋C) (Interdevelop設計書記述規約「4.1.8.　条件文　(1) 条件文1(単純条件二方向条件分岐)」)	□

図８　セルフチェックリスト抜粋

(3) 事前準備の考慮点

前工程の詳細設計工程時に従来開発では実施していなかったデータ辞書登録、プロセス仕様書作成の作業が必要となる。使用する全項目に対して辞書登録を事前に実施する必要があり、初期は時間を要するものの対応プログラム本数が多いほど、登録済みのデータ項目を流用できるため、開発は効率的になる。

２．３．５　テスト期間短縮

(1) ねらいと手法

条件網羅のテストケースが自動生成されるため、テスト仕様書作成期間の短縮が可能。また、テストデータ作成がエクセル上で可能なため、テスト時間の短縮が可能。更にテスト結果検証は期待値と結果がアンマッチの部分は可視化されるため、検証時間についても短縮できる。

(2) 成果

表 4 のとおり従来のテストでは 1 プログラム平均 3 人日を要していたが、今回開発では平均 2 人日で実施することができた。

テスト作業の差異

従来開発		今回開発	
作業内容	工数	作業内容	工数
①単体テスト仕様書作成	1.5人日	①単体テスト仕様書作成 （条件網羅ケース自動生成）	1人日
②単体テスト	1.5人日	②単体テスト （テストデータ作成 実行・検証作業のの短縮）	1人日

表４　単体テスト工程の作業工数

これは、自動生成によりテストケース作成の時間が従来開発より削減されたこと、及びテストの準備期間についても削減された効果である。テストデータ作成がエクセル上で実施でき、JCL や COBOL の知識がなくともテスト可能であり、外部モジュール（スタブ・ドライバ）が自動生成される点が期間短縮の大きな要因である。

図 9 に開発支援ツールでのテストイメージを示す

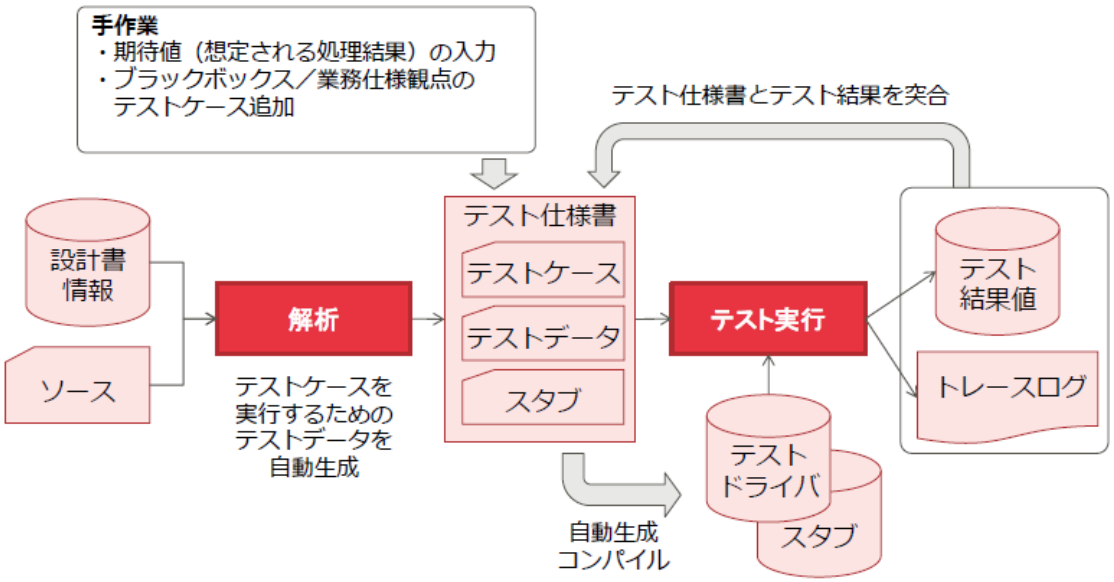


図９　開発支援ツールでのテストイメージ図

（引用『InterdevelopCOBOL 説明書（概要）』、2014 年、17 ページ）

また、過去の追加ケースも含め仕様書を流用してテストケースの自動生成が可能。そのため修正後の再テスト実施時のテスト仕様書作成時間についても削減できた。

(3) ケース自動生成時の考慮点

ケース自動生成の際に、STEPS 数が多いプログラムの場合 2～3 時間程度時間を要することがあった。そのため、修正後の再テストケース自動生成に時間を要する場合は、過去のテストケースに手修正でケース追加を行い、時間短縮を図った。

３．おわりに

３．１　評価

基幹系システム再構築は数十年に一度であり、開発手法を見直す良い機会であった。ウォーターフォール型の大規模開発では、サブシステムや機能が異なっても同一期間に平行して開発を行うことが多い。本プロジェクトにおいても、品質を損なわずに量産化できる仕組みが必要であった。

今回、開発支援ツールを使用し、3 ヶ月延べ 1,000 人月で 6,800 本の新規プログラム作成及びテストを完了することができ、量産化を実現できた。

現行システムの製造工程の課題であった要員確保やドキュメント統制についてもツールを使用した効果があったと評価する。

特に大規模開発において作業担当者を確保できた点は、今後の要員確保の課題を解消できるに価する結果である。

開発工数は、前工程でのデータ辞書登録や、プロセス仕様書の事前作成が必要なものの、開発要員は平準化でき、相対的に開発工数も削減できることから一定の効果はあると評価する。

３．２　メンテナンスフェーズに向けて

今回リリースした基幹系システムは、今後数十年間稼動予定である。リリース後のメンテナンスフェーズに「期待する効果」と「想定される課題」について述べる。

３．２．１　期待する効果

(1) 人材の流動化

今回サブシステムＡにて使用した開発支援ツールは、Java 言語で開発を実施しているサブシステムＢでも使用している。（Java プログラム自動生成）
そのため、開発言語にとらわれずプロジェクトの繁閑に応じて開発要員を他サブシステムへ流動的に配置することが可能であり、要員の効率的な稼動が期待できる。
加えて日本語でプログラムを記述する手法は、新規参入者の障壁が低く、初期構築時と同様にメンテナンスフェーズの要員調達に貢献できる。

(2) メンテナンス性の向上

日本語設計書記載方法の構文が開発支援ツールで統制されているため、プログラム修正を実施する際にも初期構築時と同様に担当者の言語スキルに依存せずに記述レベルが一定となる。

(3) 影響確認作業の効率化

既存機能の修正を実施する際にプログラム間インターフェース項目が、他のプログラムでどのように使用されているかを確認する必要がある。現行システムではメインフレーム上で JCL を実行し、紙で還元される内容をもとに確認を行っていた。開発支援ツールのクロスリファレンス機能を使用して Windows 上で実施、エクセル還元されるため、調査時間の短縮及び結果の分析が容易となる。

３．２．２　想定される課題

(1) 設計書バージョン管理

設計ドキュメントは電子化されたが、同一プログラムにおいて過去のバージョンに遡って修正箇所をトレースする機能は現状開発支援ツールにはない。現行システムでは同一プログラムが相当回数修正されているものもあり、過去の修正経緯や修正担当者が分からないと仕様確認に時間を要する場合がある。そのため、変更要件・修正プログラムバージョン・修正箇所を管理する資料の作成・周知が必要となる。

(2) データ辞書統制

設計書エディタ内で使用するデータ項目は事前にデータ辞書に登録する必要があるが、類似語が登録されているケースがある。

例：業務エラーコード、エラーコード

「業務エラーコード」で日本語設計書内の検索を実施した場合、「エラーコード」を使用している設計書は抽出対象外となるため、正確な影響調査が実施できない。また、プログ

ラム修正で未使用となったデータ項目も依然として辞書に存在するため、手作業での確認・項目廃止が必要となる。

(3) LOADモジュール管理

単体テスト後にホスト環境にプログラムソースを転送し、コンパイルを実施しLOADモジュールを生成するが、そのバージョン管理は手作業である。
複数案件で同一プログラムを修正する場合に、モジュール重複管理やテスト環境ごとのモジュールバージョン管理が必要となるため、運用ルールを作成する必要がある。

(4) 課題のまとめ

現状の開発支援ツールでは実装していない上記の統制・管理関連の機能は、既存システムと同様に継続してルール化・マニュアル管理を行う必要がある。

引用

松本　隆明：“ソフトウェア開発データ白書2016-2017”，独立行政法人情報処理推進機構（IPA），PP43、47

“InterdevelopCOBOL説明書（概要）”,富士通株式会社
,㈱富士通ミッションクリティカルシステムズ

“富士通ソリューションマップ”,富士通株式会社