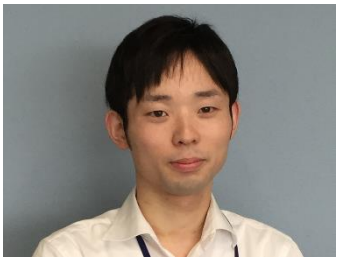
システム開発部門のあるべき姿
～ 知識・経験依存からシステム開発標準へ ～

山崎製パン株式会社

■ 執筆者Ｐｒｏｆｉｌｅ ■



長通　卓也

2012 年　山崎製パン（株）入社
2015 年　計算センター　システム開発課

宮前　貴洋

2005 年　山崎製パン（株）入社
2015 年　計算センター　システム開発課

大村　泰之

2007 年　山崎製パン（株）入社
2015 年　計算センター　システム開発課

■ 論文要旨 ■

自社の基幹系システムを約７年かけて、メインフレームからオープンシステムに切替えた。しかしこの再構築によって、本当にステークホルダーの満足度を高めることができたのか、システムを稼働させることがシステム開発部門の目的になってしまっているのではないかという疑問が浮かんだ。これをきっかけに若手社員にてシステム開発部門としてのあるべき姿をあらためて考え、「ステークホルダーに満足感を与えられる」、「生産性の高いシステム開発ができる」、「適切なコスト管理ができる」の三つをシステム開発部門のあるべき姿とした。

システム開発課部門メンバーが意識せずともあるべき姿が実現できる手法を追及し具体的な対策を講じた結果、新入社員でもベテラン社員と同じようにあるべき姿に到達することができた。

■ 論文目次 ■

■ 図表一覧 ■

１．はじめに

１．１　当社の概要

当社は食パン、菓子パン、和菓子、洋菓子、調理パン・米飯などの主力製品群を中心に、ジャム、デザート、レトルト食品やビスケット、クッキー、米菓などバラエティー豊かな製品群を製造している。販売面においては、量販店やコンビニエンスチェーン店などを通じて全国の消費者の皆様に製品を提供している。また自社業態店舗として、コンビニエンスストアの「デイリーヤマザキ」や冷凍生地を活用したベーカリーカフェの「ヴィ・ド・フランス」などの事業も展開している（図１）。

図　１　当社取扱いアイテム

当社の情報システム部門である計算センターでは、「企業経営を通じて社会の進展と文化の向上に寄与することを使命とし、自主独立の協力体制を作り、もって使命達成に邁進する」というヤマザキの経営基本方針のもと、関係会社を含めたヤマザキグループ全体の、基幹システムからそのほか業務システムの企画、開発、運用、保守などを行っている。

1．2　背景

2015 年 3 月、計算センターでは、これまでメインフレームで稼働していた基幹系システムの一部をオープンシステム、Linux サーバにて再構築を行った。これにより、各事業所で個別に稼働していたシステムが、府中市（東京都）のデータセンターに集約された（図２）。また同時に大阪にバックアップセンターを設置し、災害対策にも備えた。

このプロジェクトは約 7 年の月日をかけて導入した大規模なもので、システム開発者は無事にシステムが稼動し満足していた。しかし、システムを利用している人達は喜んでくれているのか、システム開発部門の自己満足ではないのかと疑問を持ち始めるようになった。

本論文では、システム開発部門のあるべき姿を中心に据えて、現状との乖離を問題と捉え、原因を分析・追及し、課題解決に向けた具体的な対策を講じた結果、あるべき姿に近づいた成果について論述する。

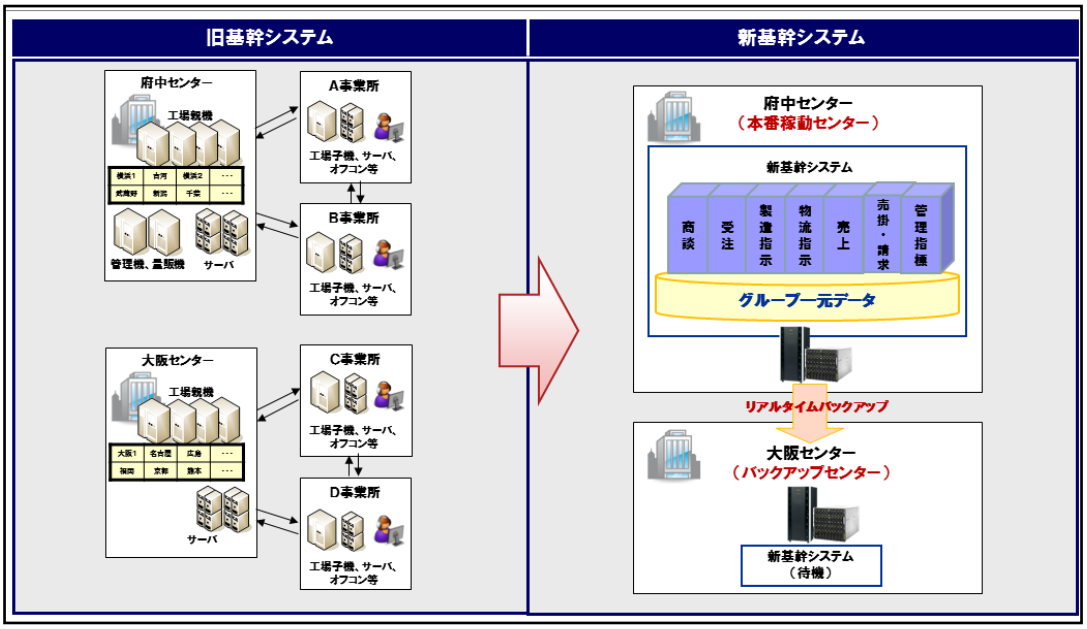


図　2　基幹システムの再構築イメージ

２．システム開発課のあるべき姿

２．１　若手が考えるシステム開発課のあるべき姿とは

前章のとおり、システム開発課のあるべき姿について検討する必要があると考え、入社10年目までのシステム開発課に所属している若手メンバーにて議論を始めた。

意見については、質よりも量を重視することとした。思いついた事柄をすべて付箋紙に一つずつ書いてホワイトボードに張り出して、ブレーンストーミングの手法をとった。図３のとおり、ホワイトボードに張り出してみると、非常に多くのあるべき姿が挙がった。

図　３　メンバーがあげたあるべき姿

２．２　システム開発課が目指すべきあるべき姿の決定

ブレーンストーミングの結果、あるべき姿について多くの意見が出た。それらについて、ユーザー企業の情報システム部門であることを念頭に置いて内容の精査を進めた。精査には、マインドマップを利用した（図４）。結果、あるべき姿を以下の通り大きく三つのカテゴリに分類することができた。

(1) ステークホルダーに満足感を与えられる
(2) 生産性の高いシステム開発ができる
(3) 適切なコスト管理ができる

これらを、今回参加していない管理職をはじめとしたベテラン課員にも合意を得て、システム開発部門が目指すべきあるべき姿とした。

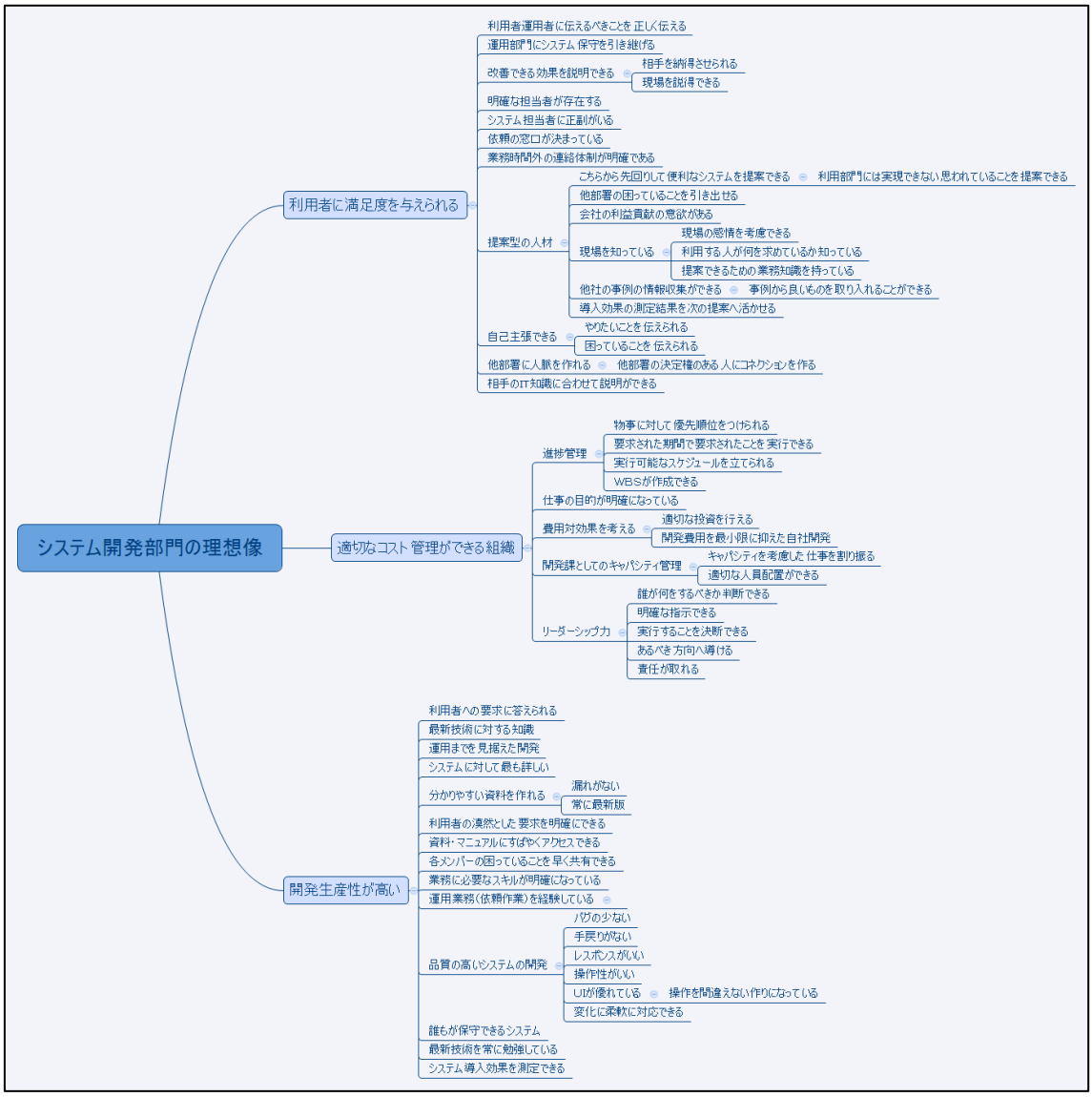


図　4　マインドマップを利用した分類

３．システム開発課の問題の設定

３．１　システム開発課のあるべき姿と現実との乖離

前章にて、システム開発課のあるべき姿を決めた。次に、あるべき姿と現在のシステム開発課の状況を明らかにした。あるべき姿を決めたときと同じように、ブレーンストーミングの手法をとり、現状の洗い出しを行った。その結果、あるべき姿とのギャップが見出され、問題点が明らかになった（表１）。

表 １　システム開発課のあるべき姿と問題

項番	あるべき姿	問題
1	ステークホルダーに満足感を与えられる	システム稼働後の効果測定を行っていない
		システムの品質を計る指標がない
		システムの品質が担当者によって安定しない
		利用者ニーズを把握できていない
		運用部門へのシステム引き継ぎ方法が人により異なる
		利用者向けマニュアルが分かりづらい
2	生産性の高いシステム開発ができる	基幹システムを集約したが、各事業所固有のプログラムが残っている
		資料やマニュアルの規約、フォームの管理、最新化が不十分である
		オープン化によりメインフレーム時代に比べてプログラム開発工数を要している
		グループ会社ごとにシステムが異なっており、限られた担当者しか保守できない
		事業所ごとに処理方式が異なる場合があり、限られた担当者しか保守できない
		最新技術の知識が乏しく、活用できる技術が限られている
3	適切なコスト管理ができる	要件定義で漏れが多く、事後の追加要件が発生している
		SI ベンダーの提案をそのまま採用している
		開発にかかる適切なコスト感覚が身についていない
		適切な人員配置ができていない

３．２　解決すべき問題の選定

あるべき姿に到達するために前項で明らかになった問題を解決する必要がある。しかし、最初からすべての問題解決に取り組むことは困難であると判断した。システム開発課内でアンケートをとり、解決すると効果が高いと思われる問題として以下の３つを選定した。「ステークホルダーに満足感を与えられる」というあるべき姿については、ステークホルダーをシステム利用者に絞ったうえで、特に問題になっていると考えられたものを選定した。

(1) システム稼働後の効果測定を行っていない

(2) オープン化によりメインフレーム時代に比べてプログラム開発工数を要している

(3) SI ベンダーの提案をそのまま採用している

４．解決策の設定と実施、およびその効果について

それぞれの問題に対して、当社の製造現場でも実施している「なぜなぜ分析」により原因を追究した。

次に、導き出された原因を除去するための「取り組むべき解決策」を設定し、実施、効果を測定した。この一連のプロセスについては、問題解決の基本的な手順を参考にした（図５）。解決策については、システム開発課メンバーがそれに従って開発をしていくことで、意識せずにあるべき姿（「ステークホルダーに満足感を与えられる」、「生産性の高いシステム開発ができる」、「適切なコスト管理ができる」）に近づくことができるものを目指した。

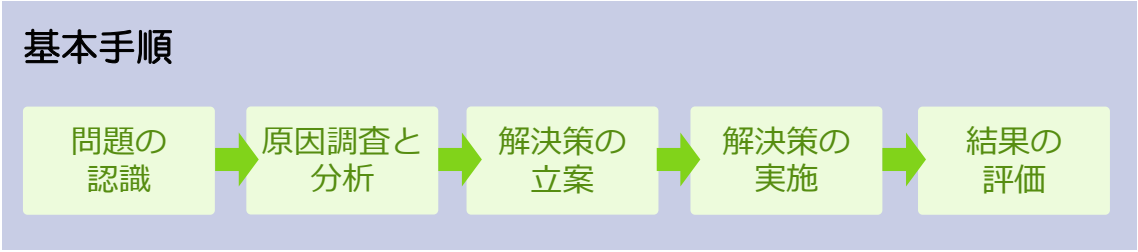


図　５　問題解決の基本手順

４．１　システム稼働後の効果測定を行っていない問題

システム稼働後の効果測定を行っていないという問題について、なぜなぜ分析により原因を追究した結果、「システム導入自体が目的になってしまっている」、「効果測定が手順化されていない」、「利用者からの要望があって初めて開発・改良に着手する構造であるため、それ以外で改善のアクションを起こさない」などの原因が導きだされた。以下、これら原因を除去するための対策と実施内容、得られた効果について述べる。

４．１．１　問題原因除去のための対策と実施内容、得られた効果

＜解決策の立案＞

「システム導入自体が目的になってしまっている」点については、これまでにも利用者からの使用感を電話やメール等で受け取る機会はあったが、稼働後に利用者の声を直接聴く機会はなかった。利用者の本音を聴き取るためには、実際に事業所に赴き対面で話を聴くことが近道だと考え、20 箇所の事業所に赴くこととした。

＜解決策の実施＞

最初の事業所へ赴いた際、意見交換会のような打ち合わせを想定して臨んだが、目的が曖昧なまま始めてしまい、あまり活発な意見交換がされなかった。また、事業所ではこの訪問を業務と捉えてもらえず、人員も時間も割いてもらえなかった。これを反省し、活発な意見交換がされなかった点については、次回以降の事業所訪問の目的を「新基幹システム導入による業務効果を定量的に把握し、結果がでるまで改善を繰り返す」として事業所へ案内し、効果を測る一環として事前に確認事項をまとめたヒアリングシートを配布した。人員や時間を割いてもらえなかった点については、訪問先の上長（課長）に訪問の意義、得られる効果等を説明し、業務として取り組んで貰えるよう人員手配をお願いした。この方式で各事業所を巡り、得られた意見を持ち帰り、取りまとめを行った。

＜結果の評価＞

新しい基幹システムを導入後、初めて現場ユーザーからのフィードバックを得ることができ、特にシステムが一か所に集約されたことによるメリットについて非常に多くの意見を得ることができた。集めた意見については持ち帰り早急に解決することで、実際に利用者からは使いやすくなったとの声があった。今回全事業所に赴き、利用者と対面で話をすることで、開発側と利用者側の関係も親密になり、電話やアンケートでは得られない生の声を引き出すことができた。

また、聴き取った効果の中に、「デニッシュ製品ラインにて製造過剰数が前年比で55.9%削減できた」というものがあった。この聴き取り結果に着目し、今回導入した新基幹システムをどのように利用したのか詳細に聴き取りを行った。新しい基幹システムでは「発注速報」という受注を受けたタイミングで、リアルタイムに製造数量を把握できる機能を実装している。その事業所のデニッシュ製品ラインでは、発注速報を利用することで正確な仕込み生地量が早期に把握できるようになり、その結果、今回の削減結果につながったということであった。

聴き取りにより確認できたデニッシュ製品ラインの大幅な業務改善を、計算センターが提供している社内ポータルサイトで紹介したところ、他事業所のデニッシュ以外のラインにおいても生産過剰数を半分に抑制することができ、大きな改善効果を得られた。製造過剰の削減はどの事業所においても力を入れている事であることから、今回システム導入後の評価、改善を実施したことで、利用者の満足度向上に繋がったといえる。

問題の原因のひとつであった「効果測定が手順化されていない」点については、今回実施した聞き取り方法や、訪問時のノウハウ等を手順の一部として採用し、今後さらに改善していく。

以上のことから、システム開発部門が目指すあるべき姿のうちのひとつである、「ステークホルダーに満足感を与えられる」ということについて、一定の結果が得られたと考えられる。

「利用者からの要望があって初めて開発・改良に着手する構造であるため、それ以外で改善のアクションを起こさない」点については、能動的に事業所を訪問したことで、改善要望や、新規システム開発依頼を聴くことができたが、アクションまでには至っていない。今後も定期的に要望を聴き取り、改善のアクションを起こしていきたい。

４．２ オープン化でメインフレーム時代に比べプログラム開発工数を要している問題

オープン化でメインフレーム時代に比べプログラム開発工数を要しているという問題について、なぜなぜ分析により原因を追究した結果、「メインフレーム時代の命名規約や、コーディングテンプレートが適用できなくなった」、「スキルをもった人材が不足している」などの原因が導きだされた。以下、これら原因を除去するための対策と実施内容、得られた効果について述べる。

４．２．１　問題原因除去のための対策と実施内容、得られた効果

＜解決策の立案＞

COBOL について、メインフレームで採用していた自社作成のコーディング規約、及びプログラム記述テンプレートが存在しているが、作成されたのは 20 年ほど前である。生産性の向上において非常に有効であったので、オープン系システムにおいても同様に作成することとした。

＜解決策の実施＞

コーディング規約については、SI ベンダーの開発フレームワークを参考にさせて貰い、当社オリジナルのものを作成した（図６）。さらに SQL の記述ルールなどもまとめ、SQL 規約を作成した（図７）。

プログラム記述テンプレートについてはメインフレーム版のものを参考に作成した（図８）。このテンプレートは、単純なデータベースからの SELECT のほか、INSERT＋UPDATE、テーブルマッチングなど、使用頻度が高いパターンを網羅しており、用途によって使い分けられるようにした。またテンプレートを一度理解するだけで様々なプログラムに応用でき、コーディング技術の習得にかかる工数を最大限削減できるように工夫した。

Java については、今回初めて社内で採用した言語であり、COBOL のようにノウハウがなく手探り状態であったが、SI ベンダーの助言を受けたり、外部の研修に参加したりするなどして完成させた。

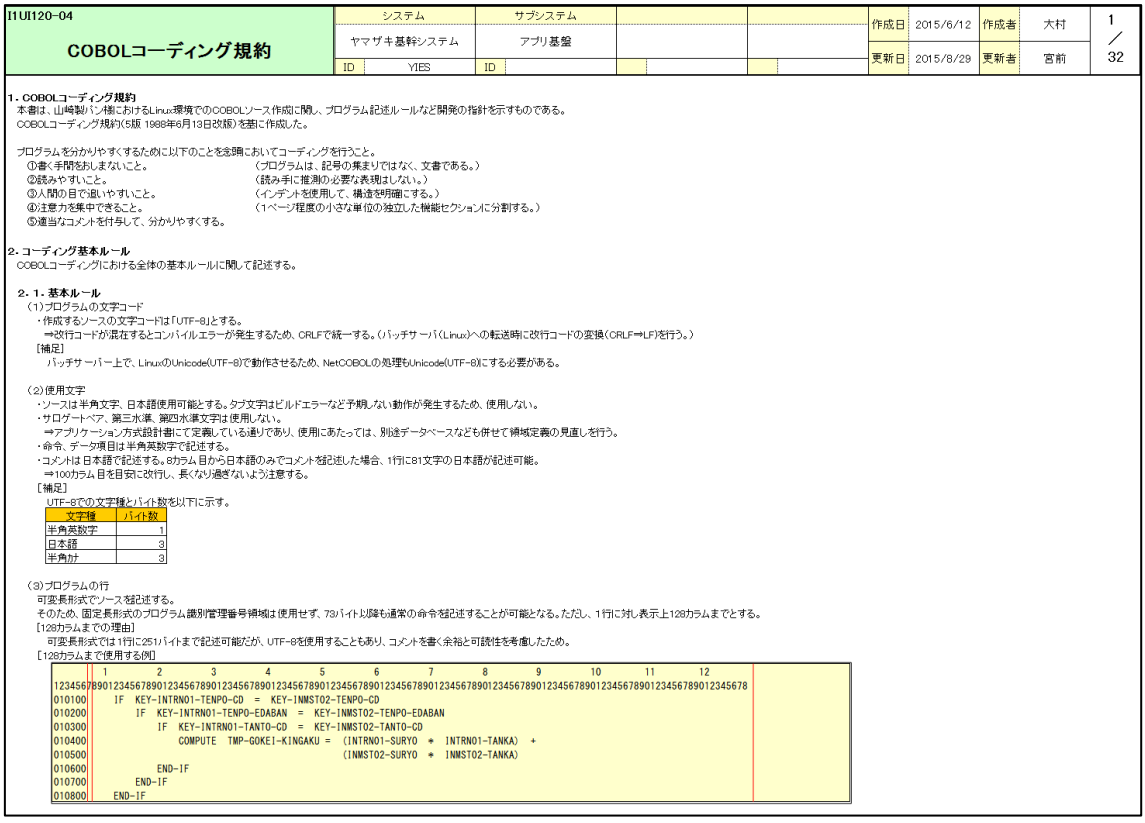
当社はユーザー企業であるため、情報システムの教育を受けたメンバーが少ない。経験の乏しいメンバーでも品質が落ちないように、分かりやすく記載することを心がけた。

＜結果の評価＞

コーディング規約を作成したことで、変数の命名法やインデントの入れ方、コメントの記述レベルが統一され、プログラムの可読性が増した。新機能追加時に改修箇所特定の迅速化や、思わぬデグレード、手戻り防止に役立っており、生産性の向上につながっていると考えられる。また、プログラム記述テンプレートにより、テンプレートに沿って構造化されたプログラムは、この部分のロジックだけ再利用したいといった場合に容易に切り出して再利用が可能であるためコーディングにかかる工数の削減に繋がっている。

問題の原因であった「メインフレーム時代の命名規約や、コーディングテンプレートが適用できなくなった」点については、今回オープン版のコーディング規約、プログラム記述テンプレートを作成したことで解消できた。

「スキルをもった人材が不足している」という点については、プログラム記述テンプレートに従えばスキルがない新入社員でも定型的なプログラムであれば開発することができた。しかし、独自性の高いプログラムの作成では、新入社員とベテランとの間に工数差がみられ、テンプレートが存在してもある程度のスキルが伴っていなければ大きな効果は得られないと分かった。今後は規約の整備と共に、それに対応できる最低限のスキルを持った人材の育成にも注力していきたい。



I1UI120-04

COBOLコーディング規約

システム	サブシステム			作成日	2015/6/12	作成者	大村	1 ／ 32
ヤマザキ基幹システム	アプリ基盤			更新日	2015/8/29	更新者	宮前	
ID YIES	ID							

1. COBOLコーディング規約

本書は、山崎製パン様におけるLinux環境でのCOBOLソース作成に関し、プログラム記述ルールなど開発の指針を示すものである。
COBOLコーディング規約(5版 1988年6月13日改版)を基に作成した。

プログラムを分かりやすくするために以下のことを念頭においてコーディングを行うこと。
①書く手間をおしまないこと。　（プログラムは、記号の集まりではなく、文書である。）
②読みやすいこと。　（読み手に推測の必要な表現はしない。）
③人間の目で追いやすいこと。　（インデントを使用して、構造を明確にする。）
④注意力を集中できること。　（1ページ程度の小さな単位の独立した機能セクションに分割する。）
⑤適当なコメントを付与して、分かりやすくする。

2. コーディング基本ルール

COBOLコーディングにおける全体の基本ルールに関して記述する。

2. 1. 基本ルール

(1)プログラムの文字コード
・作成するソースの文字コードは「UTF-8」とする。
⇒改行コードが混在するとコンパイルエラーが発生するため、CRLFで統一する。(バッチサーバ(Linux)への転送時に改行コードの変換(CRLF⇒LF)を行う。)
[補足]
バッチサーバー上で、LinuxのUnicode(UTF-8)で動作させるため、NetCOBOLの処理もUnicode(UTF-8)にする必要がある。

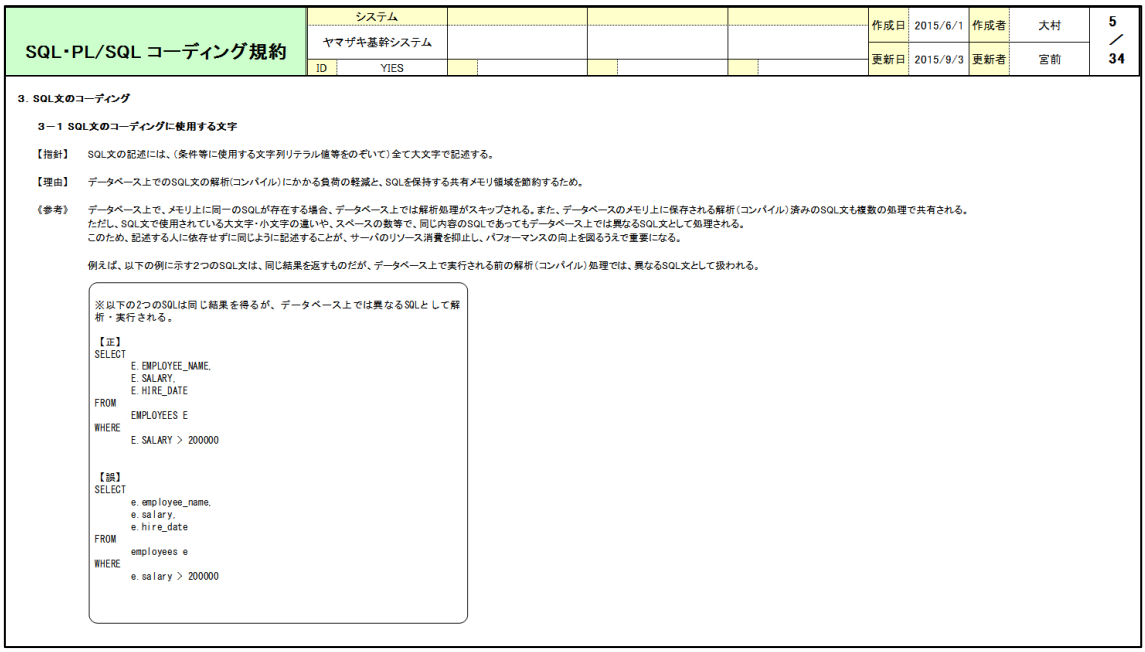
(2)使用文字
・ソースは半角文字、日本語使用可能とする。タブ文字はビルドエラーなど予期しない動作が発生するため、使用しない。
・サロゲートペア、第三水準、第四水準文字は使用しない。
⇒アプリケーション方式設計書にて定義している通りであり、使用にあたっては、別途データベースなども併せて領域定義の見直しを行う。
・命令、データ項目は半角英数字で記述する。
・コメントは日本語で記述する。8カラム目から日本語のみでコメントを記述した場合、1行に81文字の日本語が記述可能。
⇒100カラム目を目安に改行し、長くなり過ぎないよう注意する。
[補足]
UTF-8での文字種とバイト数を以下に示す。

文字種	バイト数
半角英数字	1
日本語	3
半角カナ	3

(3)プログラムの行
可変長形式でソースを記述する。
そのため、固定長形式のプログラム識別管理番号領域は使用せず、73バイト以降も通常の命令を記述することが可能となる。ただし、1行に対し表示上128カラムまでとする。
[128カラムまでの理由]
可変長形式では1行に251バイトまで記述可能だが、UTF-8を使用することもあり、コメントを書く余裕と可読性を考慮したため。
[128カラムまで使用する例]

```
      1         2         3         4         5         6         7         8         9         10        11        12
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
010100     IF  KEY-INTRNO1-TENPO-CD  =  KEY-INMST02-TENPO-CD
010200         IF  KEY-INTRNO1-TENPO-EDABAN  =  KEY-INMST02-TENPO-EDABAN
010300             IF  KEY-INTRNO1-TANTO-CD  =  KEY-INMST02-TANTO-CD
010400                 COMPUTE  TMP-GOKEI-KINGAKU =  (INTRNO1-SURYO  *  INTRNO1-TANKA)  +
010500                                               (INMST02-SURYO  *  INMST02-TANKA)
010600             END-IF
010700         END-IF
010800     END-IF
```

図　6　コーディング規約サンプル



SQL・PL/SQL コーディング規約	システム				作成日	2015/6/1	作成者	大村	5
	ヤマザキ基幹システム				更新日	2015/9/3	更新者	宮前	／ 34
	ID YIES								

3. SQL文のコーディング

3－1 SQL文のコーディングに使用する文字

【指針】 SQL文の記述には、(条件等に使用する文字列リテラル値等をのぞいて)全て大文字で記述する。

【理由】 データベース上でのSQL文の解析(コンパイル)にかかる負荷の軽減と、SQLを保持する共有メモリ領域を節約するため。

《参考》 データベース上で、メモリ上に同一のSQLが存在する場合、データベース上では解析処理がスキップされる。また、データベースのメモリ上に保存される解析(コンパイル)済みのSQL文も複数の処理で共有される。
ただし、SQL文で使用されている大文字・小文字の違いや、スペースの数等で、同じ内容のSQLであってもデータベース上では異なるSQL文として処理される。
このため、記述する人に依存せずに同じように記述することが、サーバのリソース消費を抑止し、パフォーマンスの向上を図るうえで重要になる。

例えば、以下の例に示す2つのSQL文は、同じ結果を返すものだが、データベース上で実行される前の解析(コンパイル)処理では、異なるSQL文として扱われる。

※以下の2つのSQLは同じ結果を得るが、データベース上では異なるSQLとして解析・実行される。

```
【正】
SELECT
        E.EMPLOYEE_NAME,
        E.SALARY,
        E.HIRE_DATE
FROM
        EMPLOYEES E
WHERE
        E.SALARY > 200000


【誤】
SELECT
        e.employee_name,
        e.salary,
        e.hire_date
FROM
        employees e
WHERE
        e.salary > 200000
```

図 7　SQL 規約サンプル

```
000100 @OPTIONS MAIN
000200* ALL RIGHTS RESERVED, COPYRIGHT(C) 2015
000300* YAMAZAKI BAKING CO., LTD ALL RIGHTS RESERVED.
000400*-----------------------------------------------------------------*
000500* システム名      ：ヤマザキ基幹システム
000600* サブシステム名：業務共通
000700* プログラム ID   ：PEDIT07
000800* プログラム名    ：テンプレート７
000900* 作成日          ：YYYY/MM/DD
001000* 作成者          ：ＮＮＮＮＮＮＮＮＮＮ
001100* 機能概要        ：ＤＢ（ＴＲＡＮ,ＭＡＳＴ）をキー項目によってマッチングする。
001200* 動作形態        ：メイン
001300*-----------------------------------------------------------------*
001400* 変更日：YYYY/MM/DD  変更者：ＮＮＮＮＮＮＮＮＮＮ
001500* 変更内容：要件番号：XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
001600*-----------------------------------------------------------------*
001700/
001800 IDENTIFICATION          DIVISION.
001900 PROGRAM-ID.             PMATS.
002000 ENVIRONMENT             DIVISION.
002100 CONFIGURATION           SECTION.
002200 SPECIAL-NAMES.
002300     CONSOLE             IS  CSL
002400     ENVIRONMENT-NAME    IS  ENV-NAME
002500     ENVIRONMENT-VALUE   IS  ENV-VALUE
002600     ARGUMENT-NUMBER     IS  ARG-NUMBE
002700     ARGUMENT-VALUE      IS  ARG-VALUE.
002800*
002900 INPUT-OUTPUT            SECTION.
003000 FILE-CONTROL.
003100 DATA                    DIVISION.
003200 FILE                    SECTION.
003300*---------------------------------*(ＮＮＮＮファイル          )*
003400*FD  OTMST.
003500*    COPY    YZXZ200C.
003600*
003700 WORKING-STORAGE         SECTION.
003800*
003900**--------------------------------*(共通作業領域              )*
004000     COPY    YZXZ010C.
```

図 8　COBOL テンプレートサンプル

４．３　SI ベンダーの提案をそのまま採用している問題

SI ベンダーの提案をそのまま採用しているという問題について、なぜなぜ分析により原因を追究した結果、「RFP（提案依頼書）を作っていない」、「相見積りをとらず、一社のみに発注している」、「最新技術、最新機器に対する知識が乏しく、提案を評価できない」などの原因が導きだされた。以下、これら原因を除去するための対策と実施内容、得られた効果について述べる。

４．３．１　問題原因除去のための対策と実施内容、得られた効果

＜解決策の立案＞

これまでのプロジェクトでは SI ベンダーを選定する際の発注先選定基準を明確に設けておらず、見積りを一社からしか採らなかったり、付き合いが長いベンダーにお願いしたりという理由で SI ベンダーが選定されてきた。この選定方法では本当に必要としている要件を実現できなかったり、提案を鵜呑みにして無駄なコストがかかったりしてしまう可能性があった。そういった事態を防ぐために、明確な評価基準として、RFP（提案依頼書）と発注先選定基準書を作成した。

＜解決策の実施＞

RFP（提案依頼書）については、自社内だけではノウハウが不足していたため、外部から専門の講師を招き、作成手法や盛り込むべき観点を取り込み、それを参考にプロトタイプを作成した（図９）。加えて、図１０のような評価基準を「発注先選定基準書」としてまとめ、チェック漏れや、人によって評価観点が変わることを防いだ。また重要度（ウェート）をそれぞれの項目にもたせることで、プロジェクトごとの特性に柔軟に対応できるものとした。

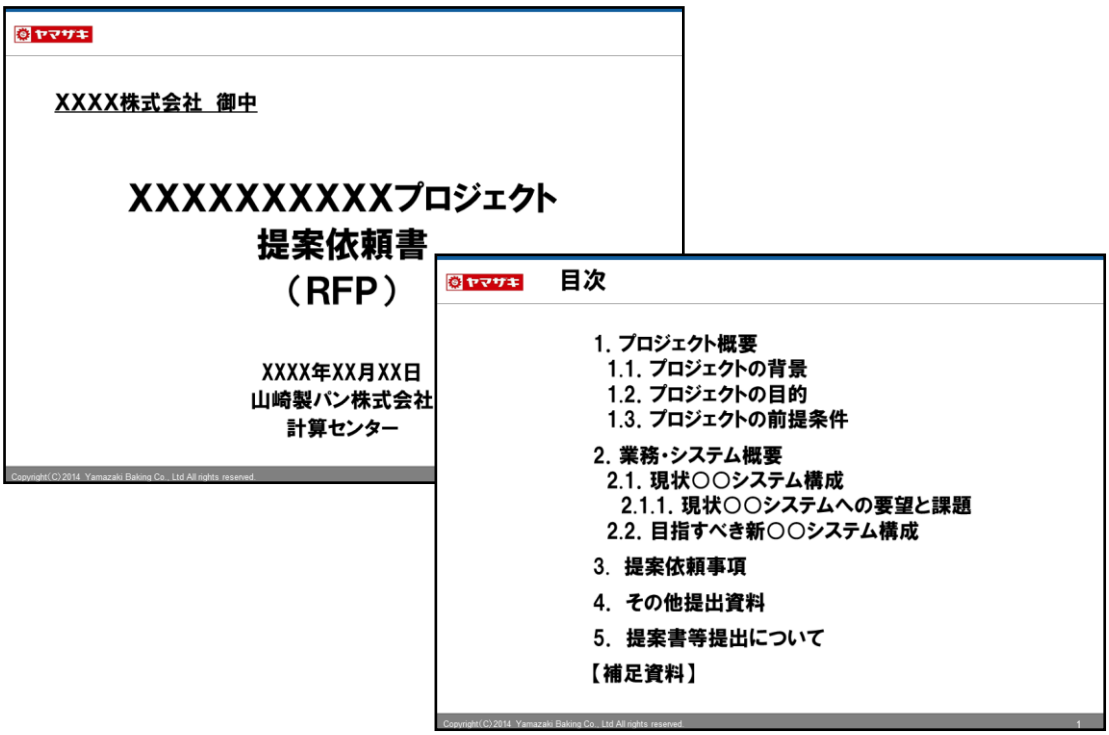
＜結果の評価＞

2015 年度中に当社のネットワーク設備を更新する計画があり、そこで今回作成した RFP（提案依頼書）テンプレートを適用した。実際に六社の SI ベンダーに送付し、返答結果から、当社の規模とコスト、要求を満たす一社を選定することができた。

また今回 RFP を作成し効果を得られたことにより、SI ベンダーに頼らず本当に我々が必要とする基準を設けることが大切だと気が付いた。適切なコストで開発を行うには常にコスト意識を持っている必要があるが、管理職やベテラン社員でない限りコストを考慮に入れて開発を行っている社員は少ない。認識違いや要件漏れによる余計なコストは減少することが想定される。

今回、RFP（提案依頼書）テンプレートを作成したことで、導き出された原因である「RFP（提案依頼書）を作っていない」、「相見積りをとらず、一社のみに発注している」という点については、解消されたと考えられる。

「最新技術、最新機器に対する知識が乏しく、提案を評価できない」という点については、自社開発を続け、SI ベンダーの言いなりにならない提案を判断できる人材の育成に努める。



ヤマザキ

XXXX株式会社　御中

XXXXXXXXXXプロジェクト
提案依頼書
（RFP）

XXXX年XX月XX日
山崎製パン株式会社
計算センター

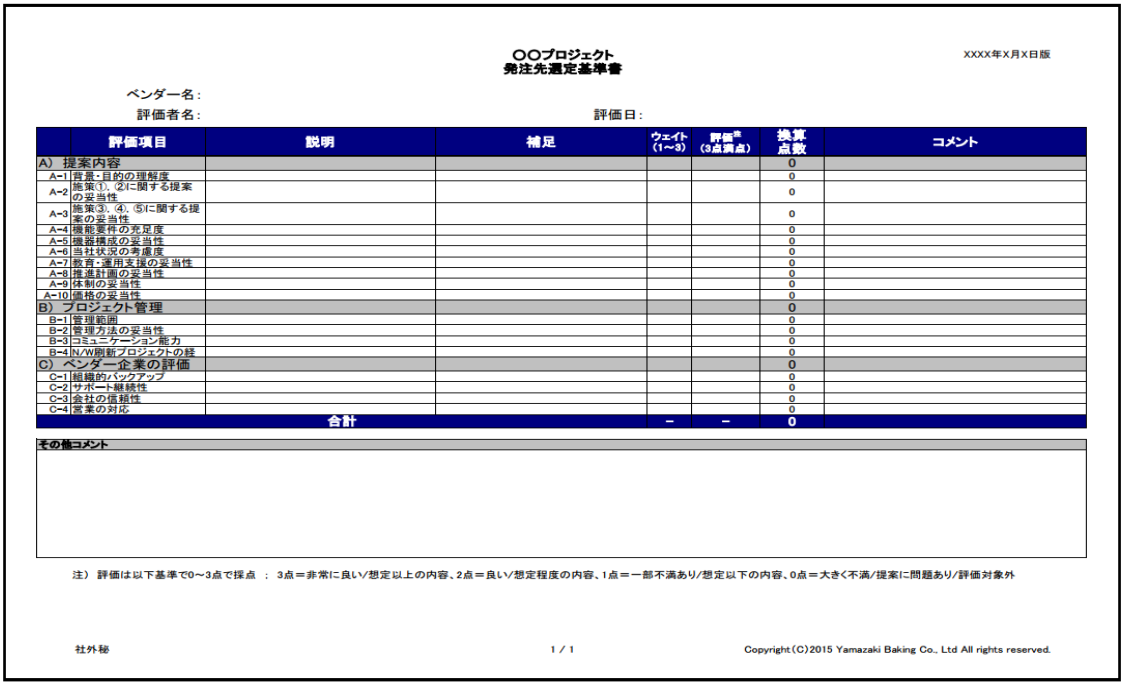
Copyright(C)2014 Yamazaki Baking Co., Ltd All rights reserved.

ヤマザキ　目次

1. プロジェクト概要
  1.1. プロジェクトの背景
  1.2. プロジェクトの目的
  1.3. プロジェクトの前提条件
2. 業務・システム概要
  2.1. 現状○○システム構成
    2.1.1. 現状○○システムへの要望と課題
  2.2. 目指すべき新○○システム構成
3. 提案依頼事項
4. その他提出資料
5. 提案書等提出について
【補足資料】

Copyright(C)2014 Yamazaki Baking Co., Ltd All rights reserved. 1

図　９　RFP（提案依頼書）テンプレート



○○プロジェクト
発注先選定基準書

XXXX年X月X日版

ベンダー名：
評価者名：　　評価日：

	評価項目	説明	補足	ウェイト（1～3）	評価点（3点満点）	換算点数	コメント
A）	提案内容					0	
A-1	背景・目的の理解度					0	
A-2	施策①、②に関する提案の妥当性					0	
A-3	施策③、④、⑤に関する提案の妥当性					0	
A-4	機能要件の充足度					0	
A-5	機器構成の妥当性					0	
A-6	当社状況の考慮度					0	
A-7	教育・運用支援の妥当性					0	
A-8	推進計画の妥当性					0	
A-9	体制の妥当性					0	
A-10	価格の妥当性					0	
B）	プロジェクト管理					0	
B-1	管理範囲					0	
B-2	管理方法の妥当性					0	
B-3	コミュニケーション能力					0	
B-4	N/W刷新プロジェクトの経					0	
C）	ベンダー企業の評価					0	
C-1	組織的バックアップ					0	
C-2	サポート維持性					0	
C-3	会社の信頼性					0	
C-4	営業の対応					0	
	合計			－	－	0	

その他コメント

注）評価は以下基準で0～3点で採点　：　3点＝非常に良い/想定以上の内容、2点＝良い/想定程度の内容、1点＝一部不満あり/想定以下の内容、0点＝大きく不満/提案に問題あり/評価対象外

社外秘　　1 / 1　　Copyright(C)2015 Yamazaki Baking Co., Ltd All rights reserved.

図　１０　発注先選定基準書

５．今後の展開

2015 年 3 月から 9 月までの半年間、当社のシステム開発部門が抱える課題について、解決策を実施してきた。今回選定した三項目では、問題点を解決していくことであるべき姿を達成することができた。

今回活動を実施していく中で、多くの問題を生み出している根本的な原因は、システム開発に対して「明確なルールが存在しない」ことによると考えた。解決に取り組んだ三種類の問題からも、これまで各担当者の「知識」と「経験」に多くを依存したシステム開発を行っており、体系化されたシステム開発が行われていないことが露呈された。

今回、課題解決の向けて取り組んできた内容の多くは、自分たちでルールや、具体的な方針を決めたものである。このことから、当社のシステム開発部門が抱える課題に対して、一定の成果が得られた。

以上のことから、体系化された「ガイド」のようなものが開発には必要であると改めて認識した。今後計算センターでは、システム開発のガイドとなるフレームワークとして、山崎製パン株式会社独自の「システム開発標準」を刷新していきたい。それはプログラムコーディングだけでなく、SI ベンダー選定や、開発環境の構築、また必要なドキュメント、記載レベル、リリース手順、効果測定など、「人によって異なる」可能性がある事柄を徹底的に排除して、誰が作っても、誰が運用しても自ずとあるべき姿が達成できるものとする。またハードや採用 OS、ブラウザ、ミドルウェア、採用言語、更には、開発モデルの選定（ウォーターフォール、アジャイル）にも踏み込んでいきたい。

将来的には、自社以外のグループ会社全体にも適用し、システム開発、及び保守のさらなる効率化を目指す。

６．おわりに

システム開発標準が完成することで、今、抱えている問題は自ずと解決されるであろう。課題は常に生まれるものであり、システム開発標準もそれに応じたものに更新していかなければならない。今回の開発課標準策定が一過性のものにならないよう、常に問題意識をもって取り組んでいきたい

今回は入社 10 年目までの若手メンバーにて活動を行った。実際自分たちが開発してきたシステムが本当に役に立っているのか、無駄なものを作っているのではないか、自信がもてない状況が少なからずあった。この活動の中で、実際に面と向かって「便利になった」、「使いやすい」という声を聴くことができたり、開発工数の削減が目に見えるようになったりしたことで、仕事に対するモチベーションを上げることができた。

この勢いをそのままに、今後も計算センターでは飽くなき追求を続けていきたい。

参考文献

- 小倉 仁志：問題解決力がみるみる身につく 実践 なぜなぜ分析，日経ビジネス人文庫（2013）