特集論文

設計書駆動型アプリケーションフレームワーク“ARMS”

上村和久*
Kazuhisa Uemura

Design Document Driven Application Framework "ARMS"

要　旨

三菱電機ITソリューションズ㈱(MDSOL)では，システム構築ガイド“MBRAINS(The Methodology of Business Production and Integration Management System)”を整備・活用している。MBRAINSは，ウォーターフォール型開発モデルがベースであり，“設計フェーズ”“製作フェーズ”“試験フェーズ”で構成するMDSOLのシステム構築標準である。

今回，設計フェーズ完了後，製作フェーズを実施することなく，ダイレクトに試験フェーズが実施可能になる“設計書駆動型アプリケーションフレームワークARMS(All Round Making System)”を開発した。

ARMSを活用した開発では，次の効果が得られる。

(1)　ノンコーディングによる生産性向上

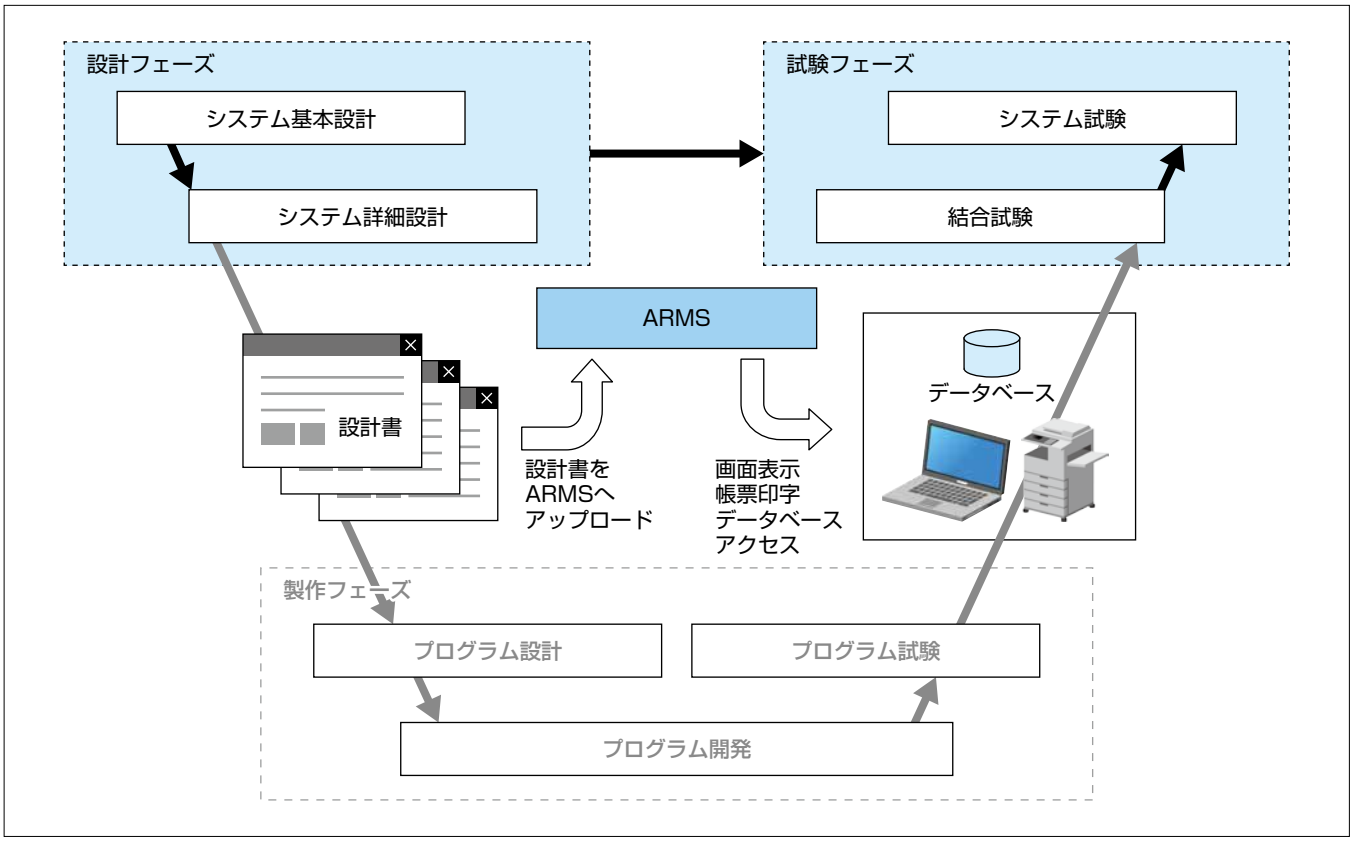
ARMSでは，ソースの自動生成を行わず，設計書をARMSへ取り込むだけで，画面表示，帳票印字，画面・帳票からのデータベース・ファイルアクセスが可能である。その結果，生産量の大幅削減が実現する。

(2)　フロントローディングによる品質確保

設計書を作成後，製作フェーズを実施せずに試験フェーズを実施できるため，設計で作り込んだ障害の早期検出が可能になる。

(3)　システム改修作業期間の短縮

システム運用開始後の保守作業では，設計書を変更すれば対応完了になるため対応期間の短縮が可能になる。



ARMSを利用した開発モデル

ARMSは，ウォーターフォール型開発モデルを踏襲しつつ，製作フェーズを実施せずに，設計書をARMSへアップロードするだけでシステムを動作させることが可能である。プログラムの作成を行わないため，生産量の大幅削減が実現して品質確保も容易になる。また，設計した画面・帳票をすぐに確認できるため，顧客との認識合わせも容易である。

*三菱電機ITソリューションズ㈱

1. ま え が き

システムの稼働形態は，C/S(Client/Server)からWebへとシフトしている。これを実現する開発モデルは，従来のウォーターフォール型に加えて，プロトタイプ型，スパイラル型，アジャイル型といった開発モデルが確立されてきた。MDSOLでは20年，30年と長期間利用するパッケージ製品を開発・販売することから，ドキュメントの重要度が高く，全フェーズでドキュメントが確実に作られるウォーターフォール型での開発を主としている。本稿ではウォーターフォール型での課題と，その解決策として開発したARMSの導入効果について述べる。

2. システム開発での課題

2.1 生産性向上に関する課題

ウォーターフォール型開発モデルでの生産性向上アプローチとしては，設計書からプログラムのソースコードを自動生成する仕組みが採用される。この仕組みでは，自動生成されたソースコードだけではシステムとしては動作せずプログラム開発を伴う。自動生成されたソースコードだけではシステムとして十分な機能を実現できないため，自動生成後のソースコードを一部追加・変更してシステムを完成させることになる。自動生成されたステップ数を含めると生産性が高く見えるが，全体にわたって試験を実施する必要があり，期待したほどの生産性向上は得られないことも多い。さらにシステムの改修作業では，自動生成の仕組みが有効に作用せず，逆に生産性が低くなる要因になる。

2.2 品質確保での課題

ウォーターフォール型開発モデルで開発する場合，設計フェーズで作り込まれた障害は，設計フェーズでのレビューで取り除くが，実際には完全に取り除けず，残存障害としてその後の開発ステップが進行する。最終的には，試験フェーズで残存障害を検出して対応することになる。つまり，開発モデル前半で検出すべき障害が，開発モデル後半に検出されて対応する形になる。この対応に当たっては，設計フェーズへ立ち返るという手戻り作業を伴うため，システム運用開始間際に短期間で多大な作業時間が必要であり，品質が低下するという課題がある。

2.3 ドキュメントに関する課題

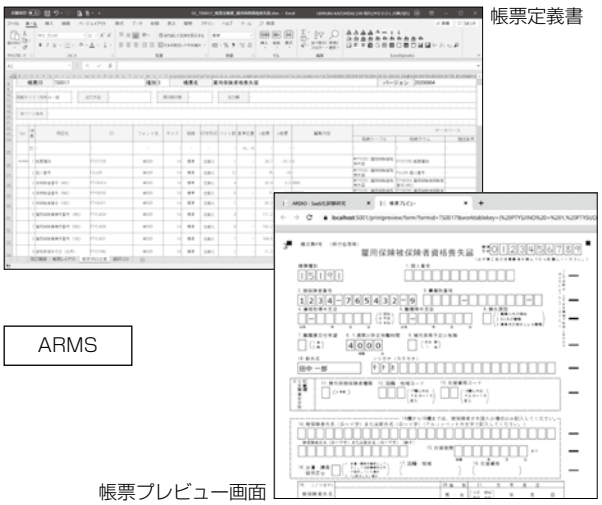
ウォーターフォール型開発モデルで開発したシステムの場合，法改正など運用後の仕様変更作業で，ドキュメントの改訂が不確実なものになり，結果的にプログラムのソースコードとの不一致が発生するケースがある。そのため，ウォーターフォール型開発モデルでは，ドキュメントとプログラムを一致させる作業にパワーをかける必要があるという課題がある。

3. 設計書駆動型アプリケーションフレームワークARMS

今回開発したARMSでは，ウォーターフォール型開発モデルと同じくシステム基本設計を実施する。ここで設計する情報は，ユーザーインタフェース仕様である“画面定義書”“帳票定義書”になる。これらの設計内容は，従来のシステム基本設計，システム詳細設計で実施する内容と何ら変わりない。次にARMSを利用した設計の流れを述べる。

ARMSを利用し，設計のアウトプットである設計書(画面定義書，帳票定義書)を取り込むと，Web画面が表示され帳票プレビューが可能になる(**図1**)。

(a) 画面定義書によるWeb画面表示



(b) 帳票定義書による帳票のWeb画面表示

図1．ARMSを利用した設計・実行イメージ

⑴　画面設計

画面設計では，システムで利用する画面を構成するために必要な情報を設計し，画面定義書としてまとめる(**表１**)。

画面レイアウトは，ARMSが提供する専用のビジュアルエディタを利用して作成する(**図２**)。作成する画面については，検索・入力といったテンプレートを意識することなく自由に項目を配置することが可能である。作成した画面レイアウトは，そのままWebシステムの画面として表示可能である。

画面項目定義は，各画面を構成する要素の属性を定義する。表示フォーマット，入力フォーマット，表示桁数といった定義に加えて，必須項目などの指定，画面項目のコンボボックス，リストボックスの値リストに設定すべき情報を定義する(**表２**)。ここで定義された属性に基づいて，ARMSが画面表示及び各項目の制御を行う。

アクション仕様では，各項目のイベント(ボタンのクリックや，コンボボックスの選択など)に対応した動作を設計する。動作としては，後述のオブジェクト制御，項目転送の指定を行う。ビジネスロジックが必要な場合については，その機能について設計する。

表１．画面定義書で設計する内容

設計項目	設計内容
画面レイアウト	ビジュアルエディタで作成
画面項目定義	画面項目の属性を定義
リストデータ	選択候補として設定する情報を定義 リストデータとして値を列挙するか，データベースから情報を抽出して設定することも可能
アクション仕様	イベントに応じた振る舞いを定義
オブジェクト制御	活性／非活性，表示／非表示といった状態を定義
項目転送	画面・データベース・ファイル間の項目転送・編集を定義

図２．画面ビジュアルエディタのイメージ

表２．画面項目定義で設計する内容

画面項目定義	ARMSでの動作概要
表示 フォーマット	値を表示する際の書式を定義 (カンマ編集や，日付の和暦表示などが可能になる。)
入力 フォーマット	入力した値のチェック形式を正規表現で定義 (入力値が不正の場合は，エラーを表示する。)
表示桁数	表示桁数を超える情報は，ツールチップで全桁表示
必須	必須項目であることを設定 必須項目が未入力の場合は，エラーが表示されるとともに，登録ボタンが非活性となるように制御を実施

オブジェクト制御では，画面項目の活性状態(活性／非活性)，表示状態(表示／非表示)を一覧で定義する。

項目転送では，画面・データベース・ファイル間の双方向転送の設計を行う。

⑵　帳票設計

帳票設計では，システムで出力する帳票を構成するために必要な情報を設計し，帳票定義書としてまとめる(**表３**)。

帳票レイアウトは，ARMSが提供する専用のビジュアルエディタを利用して作成する(**図３**)。帳票レイアウトも画面レイアウトと同様，ヘッダ部，フッタ部といったテンプレートを意識することなく自由に項目を配置することが可能である。さらに，イメージをオーバーレイ表示させた状態で，イメージに重ねて項目をデザイン可能なため，位置決めを容易に行うことができる。

帳票項目定義は，各帳票の要素の属性を定義する。印字する際のフォント情報に加えて，数値フォーマットの指定も可能である。

項目転送では，データベース・ファイルから帳票項目への転送を設計する。項目転送時に必要な編集として，項目の結合や分割といった指定にも対応している。

これら設計が完了した画面定義書と帳票定義書をARMSに取り込むと，Web画面が表示され，画面項目定義に応じた表示・入力ができ，各項目を操作した際には，アクション仕様に応じた動作として，オブジェクト制御や項目転送が行われる仕組みである。また，帳票を印字することが可能である。そのため，設計後すぐに画面・帳票の試験を実施することが可能になる。

このようにARMSを使用したシステム開発では，テンプレートを意識した画面・帳票ではなく，ニーズに合致した画面・帳票を作成可能であり，かつ，これまでの開発プロセスを実施しつつ，製作フェーズを省略することが可能になる。

表３．帳票定義書で設計する内容

設計箇所	設計内容
帳票レイアウト	ビジュアルエディタで作成
帳票項目定義	帳票項目の属性を定義
項目転送	帳票・データベース・ファイル間の項目転送・編集を定義

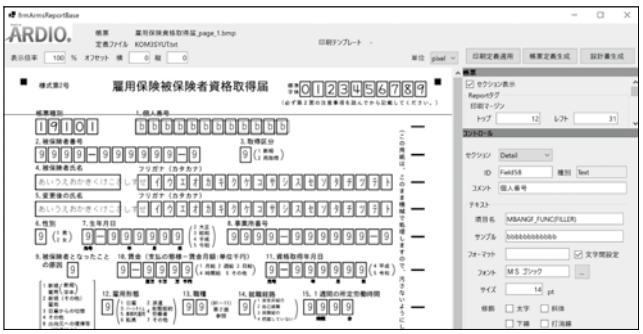


図３．帳票ビジュアルエディタのイメージ

4. ARMSによるシステム開発での課題解決

4.1 生産性向上に対する解決

画面表示を行うアプリケーションをVB.NETでプログラム作成を行う場合とARMSを利用する場合の生産量の比較を行った。

画面を構成するプログラムの内訳は，主に，画面初期化部分，画面制御部分(オブジェクト制御や入力データのチェック及びメッセージ表示)，データベースとのデータ転送部分(参照・更新)及び，ビジネスロジック部分である。ARMSでは，これらのうちビジネスロジック以外については，ノンコーディングで動作させることが可能である。ビジネスロジックについても，ARMSが提供する機能を利用して開発することが可能であり，生産量を削減できる。実際に比較してみると約1/10のステップ数で同等機能を実現できた(**図4**)。

ARMSでは，プログラムの作成を行わないため，試験対象量が減り，試験工数を削減させることが可能である。さらにシステム改修作業による変更でも，設計書改訂作業だけで対応が完了するため，プログラム変更によるデグレーションを発生させることなく対応可能になる。

4.2 品質確保に対する解決

ARMSでは，設計フェーズ完了後，すぐに試験が実施できるため，ウォーターフォール型開発での最初の開発ステップで残存障害を検出することが可能になる。その結果，残存障害の対応期間に余裕ができるとともに，手戻り作業も最小限になった。このように，ARMSを利用することによってフロントローディングによる品質確保が実現できる。実際のシステムでは，画面定義や帳票定義だけでは対応できず，独自のロジック記載が必要なケースも発生する。この場合は，ARMSを利用して設計書からコーディングに有益なスケルトンを出力することが可能である。このスケルトンには，画面定義書のアクション仕様として設計したビジネスロジックの機能がコメントとして出力されるため，トレーサビリティが確保され，ビジネスロジックの実装で抜け・漏れの防止につながる。

このように，ARMSを利用した開発では“フロントローディング”“トレーサビリティ確保”によって，システム全体の品質向上が実現できた。

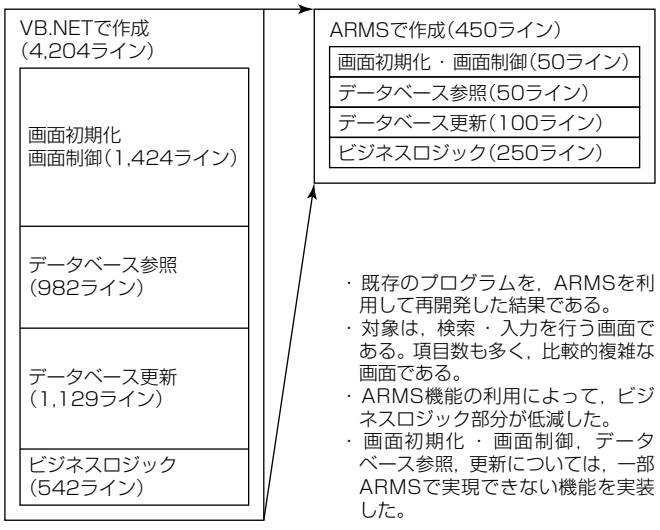


図4．生産量削減の実例

4.3 設計書の課題に対する解決

ARMSでは，設計書(画面定義書，帳票定義書)をインプットとしてシステムが稼働する。そのため，システム運用開始後に法改正が行われるなどシステムの改修が必要になり，リスト値の追加や，入力項目，帳票項目の追加があった場合でも対応が容易である。設計書の必要部分を改訂し，改訂後の設計書をARMSへ取り込んで動作させるだけで完了する。これによって，常に設計書と運用しているシステムの状態が一致するため，設計書とプログラムが不一致になる問題が解決できる。

5. むすび

ARMSを開発した2020年度は，コロナ禍で開発者が集まって作業することは困難な状況であったため，クラウドオフィスを利用することにした。クラウドオフィスでは，全員が常時接続しているため，“呼び出す必要がなくいつでも容易にコミュニケーションが取れる”“チーム単位でエリアを決めて作業するため，必要なメンバーとだけ会話ができる”など，集まって作業するのと遜色なく開発を進めることができた。

現在は，2,000ラインを超えるMDSOLパッケージ製品を，ARMSを活用してリニューアル中である。この開発でもARMSの利点を活用し，設計後，すぐに動作させメンバー全員で細かな動作まで確認できておりARMS導入の効果が発揮できている。また，この開発もクラウドオフィスで行っており，アフターコロナでもARMS＋クラウドオフィスを活用し，ARMSを利用したシステム開発を拡大させていく。