情報システム開発におけるソフトウェア資産の上流シフトへの対応

松山浩士
(株)サイバーリンクス
ko-matsuyama@cyber-l.co.jp

鯵坂恒夫
和歌山大学
ajisaka@sys.wakayama-u.ac.jp

飯ヶ谷拓真
和歌山大学
s175002@sys.wakayama-u.ac.jp

要旨

プログラムコードを自動生成するアプリケーション開発ツールが普及するきざしがある. このツールはデータベーステーブルの設計も自動化するので, ソフトウェア開発のプロセス・視点・方法論がこれまでと大きく変わる. プログラムもテーブルも資産(asset)ではなくなり, 問題記述・要求仕様のパターンや記述法がそれに代わることになる. 本稿では, このような上流資産として望まれる性質を求めるための試みについて報告する.

1. はじめに

かつて高級言語が当たり前のものになってしばらくした頃, 超高級言語(VHLL)ないし第4世代言語(4GL)によるプログラム自動生成ツールが提案されたが, 以来約50 年間, 目立って進歩することはなかった. その理由は, 生成ツールの入力となるべき記述を個々のアプリケーションの要求仕様にきめ細かく対応させることや, 生成されるプログラムの実用的効率を確保する設計の自動化が難しかったことである.

現在に至って, これらの困難が解消される状況が整ってきた. コンピュータとネットワークのハードウェア性能が劇的に向上したために, 機械的に生成された結果をそのまま受け入れ, 性能を引き出すために込み入った設計を加えなくても, 利用品質として問題なくなったことによる. 特殊な独自仕様を与えられても, 高次正規形のテーブルに対するデータアクセスロジックをそのまま実装してもよくなった.

さらに情報システムについては, もうひとつ過去と異なるここ十数年の事情がある. すなわち, ウェブ, データベースやユーザ認証など, サーバ連携によるシステム構築が通例化したことである. それにともなって開発言語が単一のプログラミング言語だけではなくなり, 種々のスクリプト言語やフレームワークを調整するコンフィギュレーションファイルの記述など, 広範な技術スキルが求められるようになったため, 個々のシステムごとにいわゆるスクラッチでこれを開発していては, 求められる生産性と品質に対してコストが見合わなくなってきた.

このような状況を見計らって, データベースを扱う情報システムを対象とする新世代のアプリケーション自動生成ツールが登場してきた. これを活用したシステム開発法は当然ながら従来の方法論とは異なる. モジュールあるいはクラス設計を核とする考え方にとらわれていたのでは, むしろツールの強みを阻害することにもなりかねない. プログラムやデータテーブル設計を資産として活用し続けようという考えも改めたほうがよい. 問題の記述, 要求の仕様化に資産がシフトするので, より高い資産価値のあるものがどのような視点と手法によって得られるか明らかにすべきである.

この目標に向け, 本研究では具体的なツール事例として ”GeneXus”[1]を取り上げ, 目的とするシステム記述の内容とプロセスを確認する. 開発の重要なポイントがどこからどこへ移るのかを考察する.

2. GeneXus の概要

ひとつの開発プロジェクトに必要な情報群を集めたものを GeneXus ではナレッジベースと呼んでいる. ナレッジベースごとに Web アプリケーションや Windows アプリケーションなどの種別, ターゲット・プログラミング言語の指定などができる. ナレッジベースには 9 種類のオブジェクト(記述フォーム)を作成するが, とくに重要なトランザクション, Web パネル, プロシージャの3つについて以下に概説する.

2.1. トランザクション

トランザクションは, 業務上ひとまとまりの作業となる一連のデータ操作を定義する. まず, タプルデータの属性を決め, 個々の属性および属性間の規則を記述する. これらの情報をもとに GeneXus は正規化テーブルを生成する. 加えて, データの登録・変更・削除の画面が生成される. なお, 生成された画面に直接手を加えることもでき

るが，その時点でデータや操作の定義との関係が絶たれ，以後の変更に対する自動生成ができなくなる．

2.2. Web パネル

画面設計（トランザクションの記述により自動生成される画面以外のもの）のための定義記述群である．データの配置，受け渡し規則，システムおよびユーザからのイベント（コントロール）などが定義できる．

2.3. プロシージャ

データベースを読み出し，それにしたがってなんらかの出力（画面表示，帳票等印刷，ファイル書き込み）を作成するためのオブジェクトである．したがって，最も手続き的であり，出力レイアウトやパラメータに関する規則・条件を宣言的に書く部分もあるが，ソースコード記述が中心となる．

3. 開発重点のシフト

プログラム自動生成ツールを使うのであるから，プログラム論理設計・モジュール設計やプログラミングは，一部を残して大方なくなることは明らかである[2][3]．それ以外の大きな変化を二つ挙げて考察する．

3.1. データモデリングの上層シフト

データベースの三層スキーマは，上位から順に外部-概念-内部あるいは概念-論理-物理と呼ばれるが，データモデリングの作業はこれまでその中間層のスキーマ設計が中心であった．関係モデルではテーブルとタプルの設定であり，キー制約や正規化を十分に検討することであった．自動化ツールはその作業を代行する．したがって，重点は最上位層にシフトする．データモデリングがなくなり概念モデリング（外部スキーマ設計）に移行するといってもよい．

そこで記述するのは非正規表（樹形データ）でよい．属性値例を入力することにより反復群 (repeating group) が発見され，正規化された表が生成される．ここで再度，属性値入力をすることにより，正規化がさらに進むか完了かが判断される．

現実をより自然に（直感的に）反映するのは表より木であろう．正規表現の集まり帰着させるこれまでのデータモデリングは，全体が先にあり（定められていることが必要であり）それを部分に分ける分解指向の設計であった．外部スキーマ／概念スキーマを非正規表（樹形データ）で表現する概念モデリングはこの逆である．部分を集めて（成長し続ける，そのときどきの）全体を構成する統合指向システム構築法である．

概念モデリングにあたっては，システムに必要な概念群の全体を最初に捕捉しておく必要もない．例えば図書館システム例題について，貸出業務，棚返却業務，取寄せ業務など（関連するが）異なる業務に必要な画面・帳票のデータ定義は，どの順番で行っても最終結果は同じになり，また一部の定義でとどめても部分システムとして正常にはたらく．次々と現れる画面・帳票のデータ同定を繰り返すことでいわゆるインクリメンタル開発が自然に実現する．

SQL や DAO を含むプログラムはこれまで資産価値の高いものであったが，テーブル設計の必要性がなくなるとともに，それらは破棄されざるをえない．

3.2. 画面・帳票の自動化

ソフトウェア工学の教科書的な記述にはあまりみられないが，開発現場では画面・帳票の設計からスタートしフィックスするという慣例が根強くあった．それが業務上のまとまり作業を認識させるからである．従来はそれをもとに中間層のスキーマ設計をしていた．これは情報システム開発のハイライトのひとつであり，ソフトウェア/データベース技術者はこのスキルを手放したがらない，あるいはデータ中心の発想から抜けきれないかもしれない．

しかし GeneXus を用いた開発では，画面・帳票は設計対象ではなく，むしろ要求獲得の源泉となる業務的ドキュメントであり，それをもとにトランザクションを定義するという位置づけになる．一枚一枚の画面・帳票から起こされたトランザクション定義は，中間スキーマ設計を意識することなく，いかようにも積み重ねることができる．

先述のとおり，生成された画面には触れさせないというポリシーは，概念構造とトランザクションの構築に集中させるためとも考えられる．ユーザビリティや視認性という品質指標はたしかにあるが，属人性の強いものであるので，最大公約数的なところで折り合いをつけよう，エンタテイメントではなく業務なのだからそれでよかろうという姿勢だと思われる．

4. GeneXus を使った開発事例

データセンターにおける障害管理システムを事例として GeneXus（以下 GX）の活用について述べる．

4.1. 開発背景と目的

システムの障害が発生するとワープロソフトを使って障害報告書を作成している．報告書は障害の内容や経緯，原因や対策を書き管理者に報告される．報告後も根本原因の調査が続行される．その過程で発見される複数の原因や，個々の障害を誘発した誘因の連鎖関係を障害の根本対策が完了するまで持続的に管理しなければならない．根本対策が完了せず並行して管理しなければならない障害が増えてくると，個々の進捗状況を一括で管理できる障害管理システムが必要となる．

4.2. 開発方針

障害管理業務プロセスは，「障害管理作業」－「問題管理作業」－「変更管理作業」－「リリース管理作業」がある．現状はワープロソフトや表計算ソフトを使って場面ごとに報告書や一覧表など業務ドキュメントを作成し作業している．今回の開発は，前述した開発範囲をあえてはじめから意識せず，障害管理作業で使われている「障害報告書」の作成システムを開発することからはじめる．その後一つ一つの作業をシステム化し，最終的に障害管理業務全体のシステムの開発を目指す．

4.3. システム開発

4.3.1. 障害報告作成システムの開発

障害が発生すると責任者は障害報告書(図1)を作成し管理者に報告する．この報告書を作成するシステムを開発する．

図1　障害報告書

まず，障害報告書の属性を表の列に展開する．その後，属性値例を入力し反復郡を発見しては表の左端に詰めながら列の順番を置き換えていく．また，明細を発見していく（図 2）．

図2　障害報告書非正規表

次に GX のトランザクションを定義する．非正規表で発見した反復属性はトランザクションの外部化をおこなう．また，明細属性はレベルを下げ定義する（図3）．

コンパイルすると正規化されたテーブルとデータアクセスロジック，画面が生成される（図4）．

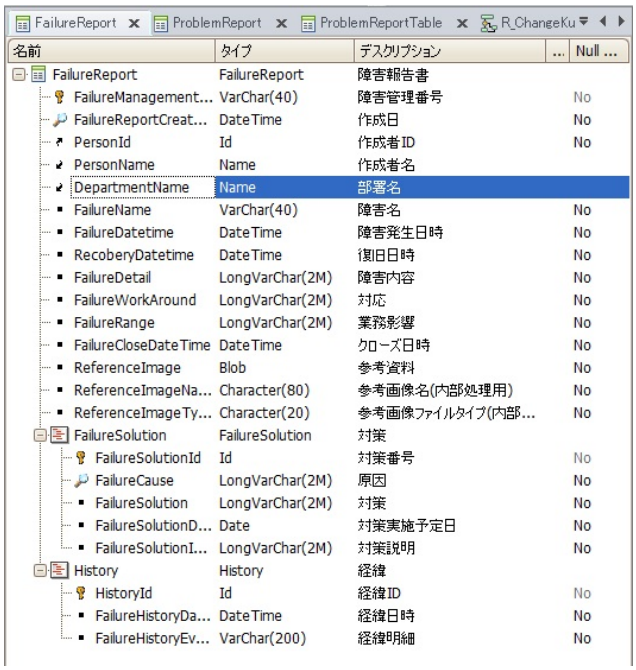

図3　障害報告書トランザクション定義

図4　自動生成された障害報告書作成画面

生成されたテーブル構造は図5の通りである．外部化した属性と明細指定した属性が正規化されテーブル生成されているのがわかる．

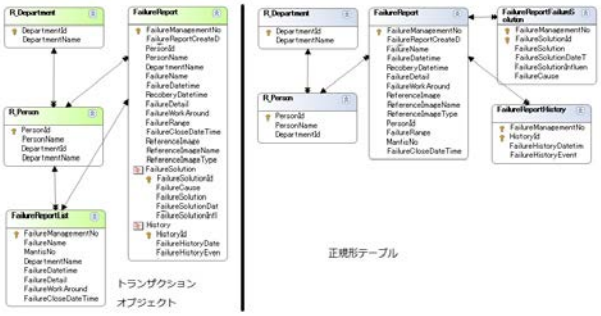

図5　障害報告書システムの正規形テーブル

4.3.2. 障害対策進捗管理機能の追加

障害対策の進捗を確認する為の一覧表を追加する．属性はほぼ障害報告書と同じで,「Mantis」という障害対応の記録が書かれている外部システムの管理番号が増えている(図6)．

部署名	障害管理ステータス	障害管理番号	Mantis管理番号	障害名	障害発生日時	障害内容	対応	クローズ日時
繰返	繰返							
クラウド基盤セル	問題管理	20151016-01	1563	紀三井寺データセンタ	2015/10/16 3時31分	紀三井寺データセンタ(本	① 新規接続ス	
		20151102-01	1447	海南DC仮想環境障害	2015年11月2日 10時23分	海南データセンタ内一般向	① 稼働してい	
	クローズ	20150910-01	1576	紀三井寺DC仮想環境	2015年9月10日 2時14分	紀三井寺データセンタ内仮	不調スイッチの	2015年12月31日 10時00分

図6　障害対策進捗管理非正規表

GX のトランザクションを追加する(図 7)．概念が同じ属性は既に稼動している障害報告書のトランザクション定義と同じ名前で登録する．そうすることで生成されるテーブルも同一概念として扱われテーブルと画面が自動生成される(図 8)．

図7　障害対策進捗管理表トランザクション定義

図8　障害対策進捗管理画面

4.3.3. 問題管理機能の追加

障害の原因と対策,そして緊急度とインパクトから導出した優先度を管理する機能を追加する．設計と開発の手順は先の二つと同様である．生成された画面とテーブルは次の通りである(図 9, 図 10)．

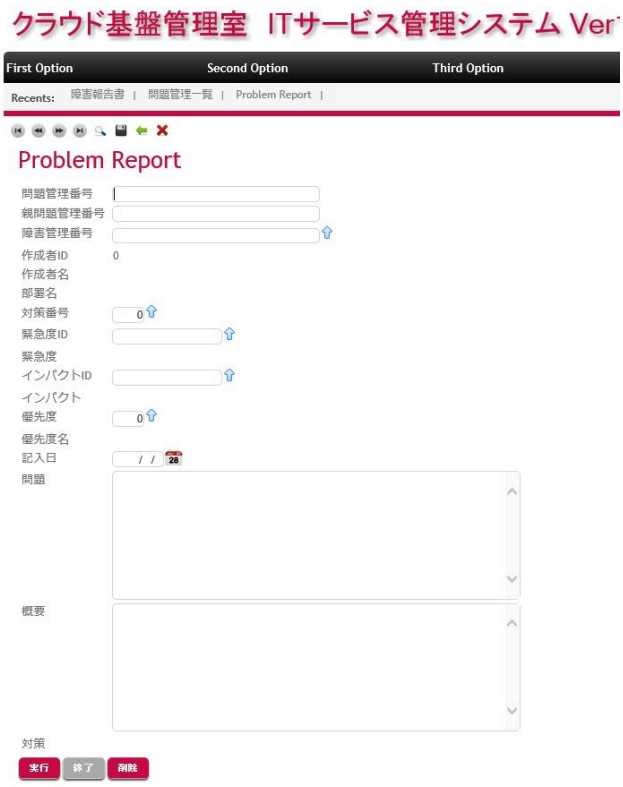

図9　問題管理画面

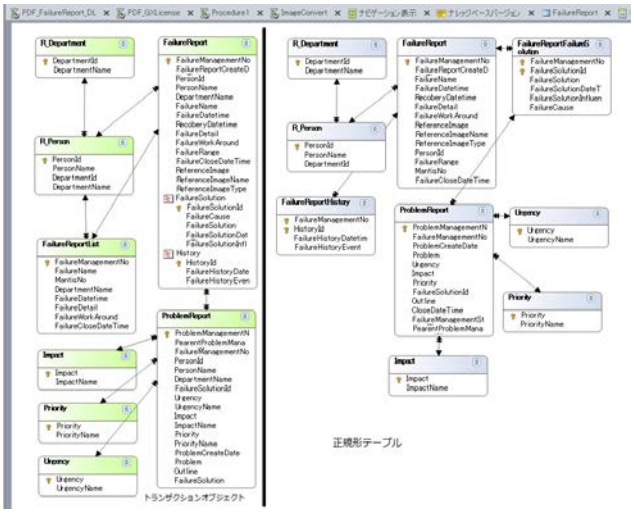

図 10　問題管理が統合された正規形テーブル

4.4. 考察

ハードウェア性能の向上により,機械生成された高次正規形のテーブルに対するデータアクセスロジックを利用してもパフォーマンスに問題がなくなり. 4 GL はそういった環境下で有用性を発揮できるようになった.

その結果プログラムやテーブル設計（データモデリング）に開発工数かからなくなり, ソフトウェアの資産性が上流シフトしている. それにもかかわらず開発現場ではあまりその意味が明確に認識されていない.

ソースコードが自動生成されることで,プログラミング工数が削減され, 単体テストや結合テストの工数も削減される. そのメリットは感じているものの, 開発手法は従来のままである. まずシステム開発の範囲を決め, 要求を洗い出し,データモデリング（テーブル設計）をおこない,そこから画面や帳票とのデータアクセスロジックを実装するというやり方だ. これは全体が先にあり（定められていることが必要であり）それを部分に分ける分解指向の設計である.

テーブルやデータアクセスロジックに変更があってもその都度自動で再編成できることで, プログラムの構造化, カプセル化, モジュール化, など, 流用性や保守性に動機づけられた技術へのニーズはなくなる. ソフトウェア開発の自動化がもたらす, 資産の上流シフトへの対応として, 部分を集めて（成長し続ける, そのときどきの）全体を構成する統合指向システム構築への思想的転換の可能性を明らかにした.

5. おわりに

プログラムが自動生成されるということは, 単体テストや結合テストもなくなるということである. しかし, システムテスト・総合テストはなくならない. もともと verification ではなく validation の性質をもつこの最終テストは, どのように計画すればよいのか. 自動生成ツールを使う開発と従前の開発では違うのかどうか, 今後の重要な検討課題である.

もうひとつの興味あるテーマは, 完全にはなくならないプログラム記述はどのような性質のものなのか, それをできるだけ宣言的な記述の解釈で実現することにどのような得失があるのかを精査することである.

さらに大きな問題は, 人間の思考的工数が上流へ集中できるようなった, 集中するしかなくなったときに, 様々に考えうる概念モデルのなかから, なぜあるひとつを選択

するのかという意図，理由 (rationale) を説明する方策であろう.

参考文献

[1] http://www.genexus.com/global/home?en
[2] 大澤文孝：GeneXus 入門，日経 BP 社（2012）
[3] ウィング：はじめての GeneXus，カナリアコミュニケーションズ（2011）