Visual Studioで構築するエンタープライズシステム

Application Architecture for .NETの利用例

最終回 ソース管理と開発環境

株式会社CSKシステムズ
IT生産技術部
中垣 健志
NAKAGAKI, Kenji

チーム開発の問題点

N君がこの.NET関連部署に異動してきて、もうすぐ一年が経とうとしています。いくつかの社内のプロジェクトに対して支援や指導を行なう中で、.NET Frameworkやアーキテクチャについてだいぶ理解が深まりました。しかし支援活動の中で気がついたのは、実際のプロジェクト内で発生している問題は実装技術以外のものも多いということです。

とくに深刻だったのは、チーム活動において無駄な作業が発生することです。昨今、エンドユーザーが高品質なシステムを短期間かつ低コストで求めるようになったため、その分開発チームのメンバーにかかる負荷は増加の一途をたどっています。開発プロジェクトの短期間／低コスト化により、みな自分の作業で手一杯で、他人の作業にまで目が届かなくなります。その結果、すでに他の人が完了した作業を重複してやってしまったり、完成したコードを間違って書き換えてしまったりなど、考えられないようなミスが発生します。

人は、神様と違って完全ではありません。忙しく追い込まれてしまうと誰しも間違いを犯してしまいます。そのような間違いを少しでも防ぐためにも、仕組みとしてチーム開発をサポートすることの必要性をひしひしと感じているN君でした。

レベル >>> Level

1 2 3 4 5

言語 >>> Language

- Visual Basic
- C#

ツール >>> Tool

- Visual Studio .NET 2003
- SQL Server 2000
- Visual SourceSafe 6.0d
- IIS 5.1

トラブルのない開発のために

エンタープライズシステムは複雑で大型な企業システムです。しかも、いつまでに作成しなければならないという期限が存在します。たとえ通常の3倍の効率で働ける人がいたとしても、ひとりで作り上げられるものではありません。必然的にチームが結成され、共同してシステム構築を行なっていくことになります。

チーム開発において何もルールがない状態でシステム構築を行なうと、最終的にいろいろな人がいろいろな場所にいろいろなコードを実装した、さまざまなプログラムが作成されます。このようなプログラムはよく「スパゲッティプログラム」と呼ばれます。プログラム中のルーチンが別のルーチンを複雑に呼び出し合いひとつひとつに分離できないさまを、絡み合ったスパゲッティに見立てているのです。このようなシステムが出来上がってしまうと、保守が非常に

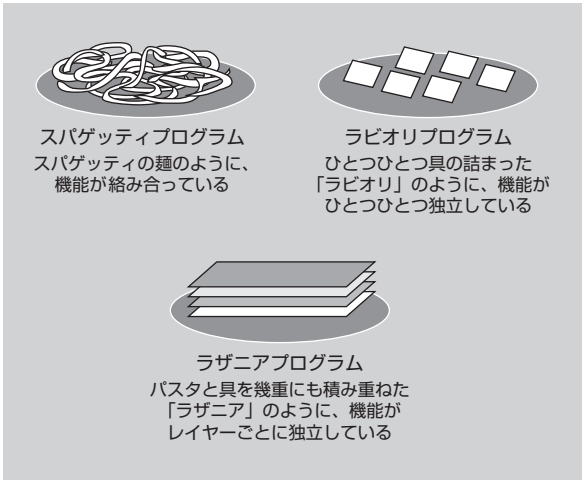
大変になります。運用中に発見された小さなバグの修正や、エンドユーザーからのリクエストによる小さな機能追加をする際でも、巨大なシステム内で絡み合っているプログラムをすべて解析して、影響のないように組み込まなければなりません。そのような作業を繰り返すうちにシステムのエントロピーは増大していき、やがて“カオス（混沌）”となってメンテナンスは不可能となります。

このような状態を防ぐために、プロジェクトリーダーは大きなシステムを「サブシステム」という単位に分割します。そしてサブシステムごとに担当者を割り振り、システム構築を行ないます。サブシステム同士は極力お互いの依存関係をなくし、プログラムが複雑に絡み合うことを防ぎます。こうして出来上がったシステムは、スパゲッティに対して、具の詰まったパスタ“ラビオリ”に見立てられることがあります。大きなシステムの中に、サブシステムというラビオリがいくつも入っているのです。「ラビオリプログラム」の問題点は、プログラムがまったく共有されていないことによるメンテナンスコストの増大です[注1]。システムを利用する企業の業務ロジックが変更された場合、すべてのラビオリについて改修を行なわなければならなくなってしまいます。

「お互いのプログラム同士の影響は抑えたいけれども、改修作業はなるべく少なくしたい」

このような相反する要求を実現するために、最近のエンタープライズシステムでは「レイヤーアーキテクチャ」が推奨されています。レイヤーアーキテクチャとは、システムを複数の「レイヤー」により分割して管理する方法です。レイヤーが積み重ねられた状態から、パスタと具を幾重にも積み重ねた“ラザニア”に見立てられることがあります。ひとつひとつが完全独立しているラビオリに対して、ラザニアは上下の層が緩やかに依存関係を持っています。しかしスパゲッティのように絡み合ってはいないため、一枚一枚の層を容易に分割して改修することが可能です。

図1：パスタに見立てたエンタープライズシステムのアーキテクチャ



以上の3つのアーキテクチャを図にしたものが図1になります。

Application Architecture for .NET（以下AAfN）は、ラザニア型のレイヤーアーキテクチャの一種です。レイヤーアーキテクチャに沿って構築されたシステムでは、それぞれのレイヤーは「インターフェイス」という約束事により緩やかに結合されています。そのため、インターフェイスの変更のない内部的な修正は、他のレイヤーには影響を与えません。またプログラムが果たすべき責務は一箇所にまとめられています。たとえば、会社の業務が変更された場合でもビジネスロジックレイヤーのプログラムさえ修正すれば、その業務が影響するプレゼンテーションレイヤーのすべての画面が修正内容を享受することができます。

◆ラザニア型のレイヤーアーキテクチャで開発するときの注意点

このように概念的には優れたラザニア型のレイヤーアーキテクチャですが、実際にチームでシステム開発を行なうときにはソースコードの共有に関する問題が発生します。ラビオリ型のサブシステム分割による実装時には、各サブシステムの担当者は他の担当者のソースコードを共有する必要はほとんどありません。したがって各担当者は完全に独立して作業を行なうことができます。しか

注1）その他に、ラビオリを作った人がいなくなるとラビオリの内部を知る人がいなくなってしまうという、組織としての問題も発生します。