新しいイテレーション型開発

XAMLが可能にするWPF (Windows Presentation Foundation)での
デザイナ/開発者のコラボレーションの変革

Microsoft®

目次

新しいイテレーション型開発

XAML が可能にする WPF（Windows Presentation Foundation）での
デザイナ / 開発者のコラボレーションの変革

執筆：Karsten Januszewski、Jaime Rodriguez

はじめに

優れたソフトウェアの開発には、洗練されたビジュアル要素、対話機能、適切なコンテンツ、応答性など、さまざまな要件が求められます。このような広範な要件に的確に対処するには、複数のスキルが必要です。例としては、魅力的なビジュアル要素を作成するグラフィック デザイナのスキル、コンテンツの整理 / 優先順位付け / 関係付けを行う情報アーキテクトのスキル、コンテンツ システムとバックエンド システムにビジュアル要素を統合する開発者などのスキルなどが挙げられます。しかしながら、こうしたすべてのスキルを 1 人の個人が習得することは、きわめて難しいと言えます。したがって、ほとんどの場合、優れたソフトウェアを開発するには各スキルに該当する役割を担う複数の個人の連携が必要です。これらの役割は、通常「開発者」と「デザイナ」の 2 つに大きく分けられます。前者にはアーキテクト、プログラマ、テスターなどが含まれ、後者にはグラフィック デザイナ、インタラクティブ デザイナ、ビジュアル デザイナなどが含まれます。

デザイナと開発者でチームを組み、より優れたエクスペリエンスを作成しようという試みは既に実践されていますが、そこで効率的なコラボレーションが生まれることはありませんでした。マイクロソフトの新しいユーザー インターフェイス プラットフォームである WPF（Windows Presentation Foundation）[1] では、この 2 つの役割のコラボレーションに重点が置かれており、革新的な方法でソフトウェア開発を行うことができます。その成果は、複数のアナリスト レポート[2] やさまざまなソリューションの導入事例[3] において明らかになっています。

マイクロソフトでは、次のような面から新たな戦略に取り組んできました。

- 役割のコラボレーションにおいて、統合の簡素化、曖昧さの排除、冗長性の低減を実現するために、ユーザー インターフェイスを表現する新しいマークアップ言語 XAML（Extensible Application Markup Language）を構築しました。

- このユーザー インターフェイス用の共通言語に加え、アプリケーションを構築するプロセスにおいて特定のタスクに最適化された新しいツール セットを用意しています。これらの新しいツールの多くは、Expression Studio と Visual Studio 2008 [4] に実装されています。

1 WPF の概要については、http://msdn2.microsoft.com/ja-jp/library/aa970268.aspx を参照してください。

2 Burton Group の XAML を参照してください。また、「Declarative Programming Advances in .NET 3.0」（http://www.burtongroup.com/Research/PublicDocument.aspx?cid=885）、および Forrester の「Why Windows Presentation Foundation Will Dominate Thick Client Development」（http://www.forrester.com/Research/Document/Excerpt/0,7211,38241,00.html）も参照してください（英語）。

3 WPF や Expression を使用したソリューションに関する導入事例の一覧については、http://www.microsoft.com/casestudies（英語）および http://www.windowsclient.net（英語）を参照してください。

4 Expression Studio の詳細については http://www.microsoft.com/japan/products/expression/default.mspx を、Visual Studio については http://www.microsoft.com/japan/msdn/vstudio/ を参照してください。

これらの新しいプラットフォームとツール セットを使用することによって、ソフトウェア開発プロセスのコラボレーション ワークフローの構築が可能になり、生産性の向上、市場投入時間の短縮、デザインの再現性の向上が実現されます。このドキュメントでは、上記のビジョンをさまざまな角度から考察し、“新しいイテレーション型開発”（デザイナと開発者の間で何度も繰り返しが可能な開発）のメリットを具体的に実現するために必要な方法について、理論と戦略の両面から概説します。

このドキュメントの内容は、調査および執筆者の実際の使用体験に基づいています。調査は、現在 WPF を使用しているさまざまな組織を対象に実施されたもので、各組織の見解が随所に盛り込まれています。また、Expression 製品チーム自体へのインタビューも掲載されています。同チームは、プラットフォームとツールに対して豊富な知識を備えており、かつ、Expression 製品そのものが WPF を使用して構築されています。

このドキュメントの構成

このドキュメントの内容は多岐にわたり、そのすべてが読者の目的に一致するというわけではありません。ドキュメント内のさまざまな解説事項は、それぞれ異なる目的に対応しています。読者は、自分の目的に関連した箇所に目を通すようにしてください。

「XAML による革命」は、WPF プラットフォームの使用経験がなく、この新しいイテレーション型開発の基本事項についての理解を深めたいと考えるユーザーを対象としています。テクノロジを中心とした内容で、XAML が開発者 / デザイナの新しいコラボレーションを可能にする理由を詳しく説明しています。

続く 2 つのセクションは、役割とツールを中心とした内容で、ワークフローの概要を説明しており、プロジェクトの管理、調整を担当するユーザーを対象にしています。これらのセクションは、デザイナと開発者の日常の業務が新しいプラットフォームとツールによってどのように変化するのかという点についても説明しているため、デザイナと開発者にも役立ちます。

最後の 2 つのセクションは、特にデザイナと開発者に向けて書かれており、このプラットフォームの導入を検討しているユーザーに実践的なヒントとテクニックを紹介します。これらのセクションは、主に、生産ラインに具体的に携わるユーザーを対象とした構成になっています。

Silverlight について

マイクロソフトの Silverlight は、リッチでインタラクティブ性を持つ Web エクスペリエンスを実現するためのクロスプラットフォーム プラグインであり、UI デザインには XAML が使用されています。このドキュメントに記載されているツールの多くは、Silverlight アプリケーションで使用できる XAML を生成します。このドキュメントには Silverlight に関連する内容が多数含まれますが、次のような理由から Silverlight 自体は取り上げていません。まず、Silverlight 1.0 は正式にリリースされていますが、Silverlight プロジェクトを構築するための関連ツールの多くは、まだアルファ版やベータ版の段階にあります。そして、このドキュメントの中心的なテーマは、デスクトップ用の RIA（Rich Interactive Application）の構築であり、Web アプリケーションではありません。Web アプリケーションは性格を大きく異にするものであり、今後、別のドキュメントで取り扱う必要があります。

XAML による革命

WPF アプリケーションにおいて、開発者 / デザイナの円滑なコラボレーションの鍵となるのは XAML です。ただし、XAML のみによってこの新しいコラボレーションが実現されるわけではありません。基盤に存在するのは WPF プラットフォームであり、これが XAML を介して表現され、効率的なコラボレーションが実現されるのです。したがって、XAML の動作のしくみ、XAML による WPF の表現方法、および全体的なシステムのアーキテクチャを理解することが非常に重要になります。「マークアップ言語を使用する理由」と「XAML と他のマークアップ言語の違い」のセクションをお読みいただければ、このプラットフォームに独自の可能性についての洞察と理解がさらに深まります。

マークアップ言語を使用する理由

現在、ユーザー インターフェイスを表現できるマークアップ言語は複数存在します。たとえば、HTML、XUL、SVG、WordML などが挙げられます[5]。その中でも、特に HTML によって、マークアップを使用したユーザー インターフェイスの表示が普及しました。XML ベースのマークアップ言語は、ユーザー インターフェイスで表示される階層や親 / 子 / 兄弟の関係を表現するのにとても適しています。

マークアップ構文が普及した大きな要因は、人間とコンピュータの双方が、マークアップ構文をすぐに判読できることです。これはマークアップの使いやすさを示す重要なポイントであり、特に、手続き型のプログラミング言語を習得するのにかかる時間を考えるとそのメリットは明らかです。たとえば、多くのユーザーは、ソフトウェアの開発経験はありませんが、HTML 構文に対して特に抵抗を覚えません。なお、ここでは、実際のマークアップそのものを理解することは、それほど重要なことではありません。マークアップ構文の生成にはツールが不可欠であり、HTML WYSIWYG エディタの普及が HTML 浸透の鍵となっています。

XAML もまた、人間とコンピュータの双方がすぐに認識でき、ユーザーはツールとテキスト エディタの間を簡単に切り替えることができます。これは大きなメリットです。

XAML と他のマークアップ言語の違い

XAML は既存のマークアップ言語の多くの機能を搭載するだけでなく、他のテクノロジにはない新機能と拡張機能を搭載しています。XAML は、表現性、包括性、および拡張性という 3 つの点において非常に優れています。

表現性

XAML は広範かつ高度な表現性を備えています。コントロール（ボタン、リスト ボックスなど）、レイアウト機能（パネル、グリッドなど）、そしてテキスト機能（フォント、書式など）まで、考えられるすべての機能が含まれています。以下に、HTML と XAML の違いを示すごく簡単な例を挙げます。ここでは 1 個のボタンの含まれたページが表示されています。

5　UI マークアップ構文の一覧については、http://en.wikipedia.org/wiki/User_interface_markup_language（英語）を参照してください。

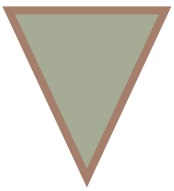
```
<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
   <input type="button" value="Click Me" />
 </body>
 </html>
```


```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
 presentation">
   <Button Width="100" Height="50" Content="Click Me" />
</Page>
```

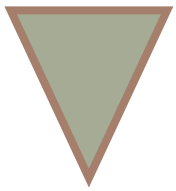
ボタンを格納したページ。上が HTML 構文で、下が XAML 構文

XAML では、コントロール、テキスト、レイアウト以上のものを表現できます。また、ベクタ グラフィックをより具体的に表現する場合に、形状、パス、傾きなどを記述できます。こうした詳細な表現には、既存のベクタ ベースの言語と同様の構文を使用するため、他のベクタ ベース言語から簡単に移行できます。

```
<rect x="1" y="1"
 width="398" height="398"
 fill="none" />
<path d="M 100 100 L 300
 100 L 200 300 z"
 fill="#A3A993"
 stroke="#A8806C"
 stroke-width="3" />
```



```
<Canvas xmlns="http://schemas.microsoft.com/winfx/2006/
 xaml/presentation" Width="398" Height="398">
<Path Data="M100,100L300,100 200,300z" Fill="#A3A993"
   Stroke="#A8806C" StrokeThickness="3"/>
</Canvas>
```



三角形のベクタ ベースの表現。左側が SVG 構文で、右側が XAML 構文

これは、2D ベクタ グラフィック表現の域を超えた、3D の表現に相当します。XAML はカメラ、ライティング、行列変換、メッシュなどの 3D シーンをネイティブで表現できます。

包括性

XAML では、単なるビジュアル要素をはるかに上回る視覚的表現が可能になります。XAML は、まさにこの点において、WPF の基盤となる機能を活用しています。XAML がデザイナ / 開発者のコラボレーションを支援する最も重要な領域は、スタイル、トリガ、コントロール テンプレート、データ テンプレート、データ バインディング、およびアニメーションです。

- **スタイル：**スタイルは、コンテンツとルック アンド フィールとを切り離すための実証済みのテクノロジです。WPF は XAML によるスタイルの定義をサポートするだけでなく、スタイルをトリガとテンプレートに結合するときに、コンテンツとルック アンド フィールとを切り離した状態で、リッチなユーザー インターフェイスを構築できる優れた柔軟性を備えています。

- **テンプレート：**WPF には、以下に示す 2 種類のテンプレートが用意されています。

    - コントロール テンプレート：デザイナは基本的な動作を変更することなく、コントロールのビジュアル要素を完全に再定義できます。

    - データ テンプレート：デザイナはビジュアル要素のセットを構築して、特定のデータ タイプを表現できます。カスタム コントロールを定義して、データとユーザー インターフェイス間のすべての対応付けを実行する必要はありません。

- **データ バインディング：**WPF では、データ バインディングがよく利用されています。XAML の優れた拡張性によって、データ バインディングはマークアップから（または Expression Blend などのツールから）アクセスできるため、デザイナはユーザー インターフェイスとビジネス オブジェクト /XML データとの間で、基本的なバインディングを簡単に設定できます。

- **アニメーション：**WPF は、XAML を介して表現およびトリガできる高度なリッチ アニメーション システムを実装します。デザイナは、マウス入力時のボタンのグロー エフェクトを始めとするユーザー イベント用の対話機能を、コードを記述することなく構築できます。

これらの XAML のすべての機能はツールによって制御され、XAML の動作の基本実装の詳細が表に現れることはありません。

拡張性

厳密に言うと、XAML は言語そのものではなく、.NET のシリアライズおよび初期化用の言語です。このため、XAML は .NET オブジェクト グラフ、カスタム コントロール、新しいアニメーションなど、WPF プラットフォームの機能以上のものを表現できます。ある意味では、XAML は XML6 の表現コードと見なすことができます[6]。

デザイナの課題と開発者の課題の分離

XAML の表現性と包括性の副産物として、ユーザー インターフェイスとビジネス ロジックが分離されるというメリットがあります。開発者はデザイナが作成した XAML ファイルを忠実に使用できます。一方、デザイナはユーザー インターフェイス、動作、アニメーションを構築できます。UI とデータとの基本的なバインディングを設定することも可能です。

6 オブジェクトのシリアライズと初期化が十分ではないシナリオでは、XAML によってシリアライズされるオブジェクトにアタッチされる動作をユーザーが作成できる “マークアップ拡張機能” がサポートされます。このようにして WPF で高度なデータ バインディングとリソース ルックアップを実現します。

役割とワークフロー

XAML を導入すると、デザイナと開発者との間に存在する壁を取り除くことができ、その結果、新たなコラボレーションが生まれます。デザイナと開発者双方のコメントを以下に紹介しましょう。

> 「XAML には、デザイナが開発者と協力して作業を行うことができるというメリットがあります。私は複数の Blend プロジェクトのそれぞれに多くの時間を費やしていますが、開発者とのやり取りは絶えず行われています。それはテニスに似ており、プロジェクトをボールのように相互に投げ合うのです。このようなやり取りによって、ツールとお互いの見解について少しずつ理解を深めていくことができます。結果として、プロジェクトとエンド ユーザーへのメリットが生まれます」
>
> （Yahoo! のデザイナ、Kalani Kordus 氏）

> 「XAML によって、デザイナと開発者との間の境界が取り除かれるため、デザインが完全に “凍結” してしまうという厄介な問題に直面することがなくなりました。つまり、基本的なビジュアル言語さえあれば、私の作業にほとんど影響することなく、最後までプロセスがスムーズに流れていきます。これまでのテクノロジでは、このような状況は起こりえませんでした。その理由は、プレゼンテーションとコードの双方で発生するイテレーションを迅速かつ並行して実施できなかったためです」
>
> （frog design の開発者、Lee Brenner 氏）

プロジェクトのイテレーションを非常にスムーズに実施することが可能になったという点が、この新しいコラボレーションの成果です。仕様の変更によってアプリケーション全体で根本的な作業のやり直しが発生してしまうような “単一方向の流れ” がなくなりました。これにより、デザイナと開発者のコラボレーションに新しい可能性が開かれ、対話によってさらにクリエイティブな開発を行うことが可能になります。

このような新しい潮流に適切に対応するために、この後のセクションではデザイナと開発者にとっての XAML の意義について学びます[7]。続いて、2 つの役割の間のワークフローについて検討します。

デザイナにとっての XAML の意義

XAML によってデザイナにもたらされる最大の変化は、成果物が実際のソリューションの一部となり、実際のソフトウェア エクスペリエンスに大きな影響を持つということです。変換のプロセスがなくなるため、開発者の傍らでビジョンを説明し、開発者がそれをコードに置き換える作業を見届ける必要がなくなりました。これにより、デザイナの影響力が飛躍的に高まります。デザイナの作業が、ワークフローそのものの一部になるのです。前述のように、仕様の変更が必要になった場合も、デザイナはプロセスに入り込み、プロジェクト全体への影響を与えずに変更を行うことができます。

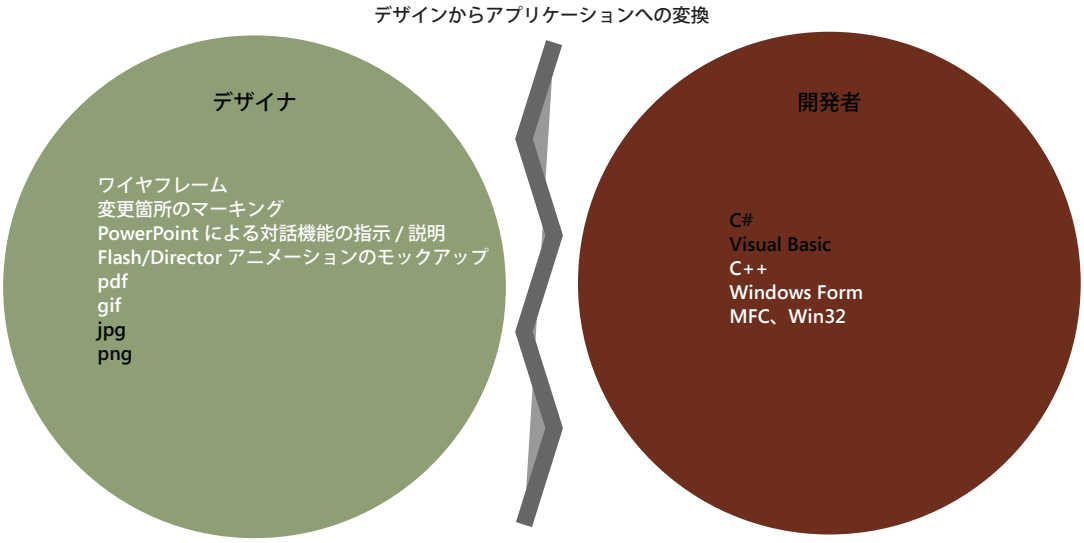
7 デザイナと開発者には、ツールとプラットフォームを習得するための先行投資費用が必要です。この投資がないと、ワークフローから得られる効率性の大部分が失われます。

新しいコラボレーション モデルでは、デザイナのプロトタイプが実際に使用されるプロトタイプになります。Expression Blend などの新しいツールによって、デザイナが開発者にコンセプトのモックアップ構築を支援してもらう必要がなくなります。デザイナは多彩な対話機能によって、サイズ調整イベントでボタンのクリックから画像のロードまでをワイヤアップすることができます。結果として、開発者の支援なしに高度なプロトタイプを構築できるため、プロセスでの生産性が向上します。

このようなプロトタイプを構築して承認が得られた場合、それが最終製品に使用される可能性は高くなります。デザイン プロセスの単なる副産物にはなりません。frog design の Robert Tuttle 氏は、「プロジェクトの初期段階のプロトタイプは 100 ～ 50% の割合で無駄になっていました。WPF を使用した開発によって、プロトタイプが無駄になる割合が 20% にまで下がりました」と述べています。

ここで重要な点は、デザイナがソフトウェア開発のプロセスに密接に関与する場合、ソフトウェアの観点からも作業の位置付けを十分に理解する必要があるということです。一般的なソフトウェア開発の柱となる要素について考えてみましょう。パフォーマンス、保守性、バージョン管理、品質管理、安定性などがこれに相当しますが、デザイナの作業はこれらのいずれの項目にも影響を与える可能性があります。そこで、教育やトレーニングを受けて、開発に参加する意味を理解することが必要になります。多くの場合、デザイナは何らかのツールを使用して XAML を構築するため、ソフトウェア エンジニアリングの観点からは必ずしも適切とは言えないものが出来上がる可能性があります。したがって、デザイナはツールの本質を理解する必要があります。詳細については、このドキュメントの 3 番目のセクション「ツール」で説明しています。

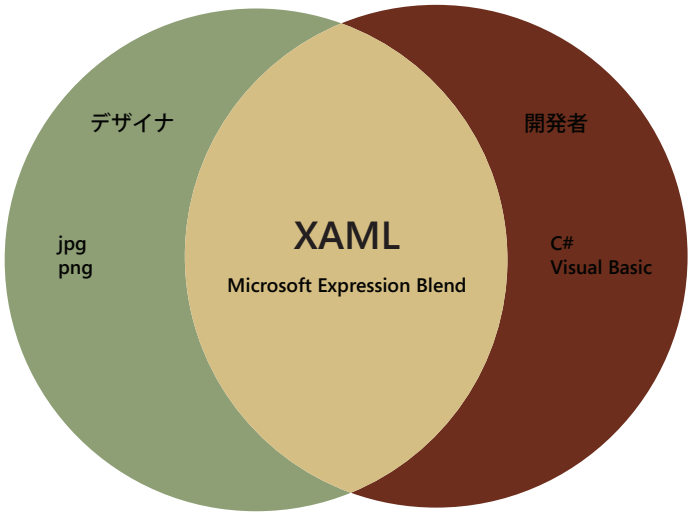
従来型の開発



■ “新しいイテレーション型開発” でも使用するもの

□ “新しいイテレーション型開発” では不要になるもの

新しいイテレーション型開発



従来の方法では、デザイナはソフトウェアのモックアップを構築する場合に、開発者にビジョンを再実装してもらう必要がありました。これは無駄の多いプロセスでした。新しい方法では、XAML がデザイナと開発者の共通語となります。

開発者にとっての XAML の意義

開発者の立場は、XAML の登場によって変化しました。XAML は表現性が高く、デザイナ向けのツールによって簡単に作成できるため、これまでは開発者が担当していたユーザー インターフェイスの機能の多くを、デザイナが担当できるようになります。これにより、開発者の負荷は大幅に軽減され、他のエンジニアリング領域に専念することができるようになります。もっとも、このことによって開発者はいくつかの所有権を手放すことになります。

XAML によってもたらされるその他の変化には、開発者がプロジェクトを構造化してワークフローの最適化を推進することが必要になるという点があります。そこで、初期プロジェクトの構築や、ディレクトリ構造、リソース、モジュールの設定などのタスクが、非常に重要になります。XAML によってプロジェクトを成功に導き効率化を達成するには、初期計画が重要であり、その多くは開発者が担当します。

以上のようなことから、開発者は XAML を学ぶ必要があると言えます。このドキュメントを執筆するにあたっての調査では、“XAML に精通していること” が WPF での作業における開発者の必須スキルであるということに対して、だれからも異論はありませんでした。2nd Factory のシニア エクスペリエンス アーキテクトであり、Expression Blend に関する書籍も著している 東 賢氏 は、「プロジェクトが複雑になるにつれて、XAML そのものに対してさらに注意を払う必要があります」と述べています。XAML の多くは、デザイナによってツールで作成されます。開発者はそうしたツールによって作成された成果物の意味を理解する必要があります。また、開発者はツールで作成されるさまざまな資産の統合と構成を担当するため、その点からも XAML への理解は不可欠です。実際の WPF 開発では、XAML は具体的な実装以上のものです。ツールに依存しきって作成する対象についての理解を欠いたままでいると、最終的に目指すべき結果を得ることができなくなってしまうでしょう。

調理場の料理人たちの役割と責務

この新しいコラボレーションでは、デザイナが構築プロセスに参加する機会が増え、デザイナの資産がアプリケーションに直接利用されます。したがって、このような資産を正確に導入するプロセスを確立することが非常に重要になります。チームの規模、スキル、および新しいテクノロジに関する知識はプロジェクトに応じて異なるため、推奨されるチーム モデルを 1 つに絞ることはできません。しかしながら、早期採用のさまざまなプロジェクトで最新のテクノロジを採用してきた経験に照らすと、プロセスの管理には 2 つの標準的なアプローチが存在すると考えられます。1 つ目のモデルは、**デザイナ / インテグレータ / 開発者のモデル**です。ここでは、ソフトウェア構築プロセスに新しい役割が追加されています。インテグレータの役割は、プロジェクトのビジュアル デザイン要素の統合と最適化です。2 つ目のモデルは、**デザイナ / 開発者のモデル**です。新しい役割は導入せず、境界を明確化して質の高いコミュニケーションを行うことによって、デザイナと開発者による並行作業、およびビジュアル デザインとアプリケーション機能の統合を実現します。両モデルには、それぞれメリットとデメリットがあります。また、チームのスキルに基づき、各モデルにはさまざまな構成が考えられます。

チームによってスキルは異なるため、推奨されるチーム モデルを 1 つに絞ることはできませんが、すべてのモデルに共通するベスト プラクティスが 1 つあります。それは、「それぞれのタスクや成果物に対する所有者を特定する」というものです。このベスト プラクティスについて説明するために、2 種類のチーム モデルを詳細に分析します。

デザイナ / インテグレータ / 開発者のモデル

「ソフトウェア開発プロセスにおいて、XAML はデザイナと開発者の境界を越える新しい役割を作り出した」という考え方があります。この「新しい役割」に対して付けられた最も実用的な名称が「インテグレータ」です。これは IdentityMine によって提出された名称であり、IdentityMine はこのトピックに関して膨大な考察を行っています。IdentityMine のテクニカル プログラム マネージャである Paul Alexander 氏は、「インテグレータは開発者のニーズを理解するとともに、デザイナのニーズもサポートして、デザインそのままの魅力的なアプリケーション UI を構築し、同時に開発者の記述したコードによってコンセプトを確実に実現します」と述べています[8]。

理想的なインテグレータは、デザイン センスに優れ、WPF と XAML に精通している人物です。インテグレータは、XAML および開発者とデザイナ間のインターフェイスの所有者として、XAML ファイルを常に最適化します。必要に応じた XAML の構造化とモジュール化は、インテグレータの役割です。したがって、インテグレータは XAML の専門家として、継承、スタイル、リソースのルックアップなどの WPF のコンセプトを理解する必要があります。

インテグレータを配置する利点の 1 つは、デザイナと開発者の負担を軽減できることです。インテグレータを配置することによって、デザイン チームは資産をプロジェクトに効果的に組み込むために XAML を構成する必要がなくなります。デザイナが新しいツールを習得したがらない場合や、要求の厳しいプロジェクトの中でその時間が取れない場合、デザイナは使い慣れたツール（Adobe Illustrator など）で資産を構築し、XAML としてエクスポートします。こうした状況では、XAML をプロジェクトに統合するために必要となる修正をインテグレータが行います。同様に、開発者が XAML によるコードへの影響への対処に手間をかけたくないという場合は、インテグレータがそのサポートを行うことができます。特に、短期のプロジェクトや要求の厳しいプロジェクトでは、経験豊富なインテグレータを配置すれば、すべてのビジュアル要素をプロジェクトにすばやく組み込み、かつ、その工程で開発者 / デザイナを啓発することもできます。

一方、インテグレータの役割には潜在的なマイナス面もあります。XAML の知識はチーム全体で共有する必要があり、インテグレータという単一障害点（SPOF）を作るのは望ましいことではありません。また、大規模なプロジェクトでは、デザイン固有の〝反復性〟から、インテグレータが資産をほとんど構築できず、資産の洗練や見直しを行うこともできないため、インテグレータがプロセスのボトルネックになってしまう可能性があります。開発者からインテグレータへの委任は、頻繁に繰り返されるタスクであり、委任のたびに不要なオーバーヘッドが発生します。そのほか、インテグレータが XAML をプロジェクトに統合するときの微調整時に、デザイナのオリジナル ビジョンを気付かずに変更してしまうという危険性もあります。さらに、プロジェクトの規模が拡大して複数のインテグレータを配置することになると、作業の重複が発生して管理が困難になります。結果として、デザイナと開発者の負担が増大します。

8 http://blog.identitymine.com/blogs/paul_alexander/archive/2007/07/10/doing-a-wpf-project-got-an-integrator-cool-do-you-know-what-to-do-with-him.aspx（英語）

プロジェクトとチームはそれぞれに異なるものであり、XAML による新しいワークフローによってプロジェクトを成功へと導く重要な要素として、インテグレータの配置を一律に推奨することは困難ですが、ここでは、新しいプロジェクトを開始する場合の 1 つの可能性としてこの役割に言及しました。

デザイナ / 開発者のモデル

デザイナ / 開発者のモデルでは、新しい役割は不要ですが、グラフィック要素と対話型要素に関する資産をプロジェクトに組み込む方法を管理するプロセスを設定する必要があります。ここでは、ワークフローの最適化のためにハーベスト モデルとコラボレーション モデルという 2 つのアプローチが使用できます。

ハーベスト モデル

ハーベスト モデルでは、開発者がデザイナから資産を ” 収穫（harvest）” します。したがって、資産を統合する責任は開発者側にあります。このシナリオでは、デザイナはプロジェクトで直接使用する資産を構築しますが、ソース コントロール レポジトリに格納するファイルは確認しません。さらに、デザイナは完成したアプリケーション自体をコンパイルして実行することはできません。デザイナは独立した小規模なプロジェクトで作業を行い、実際のプロジェクトへの資産の統合や移行は開発者に委任します。ハーベスト モデルは、デザイナと開発者の明確な役割委任を実現します。

このモデルでは、デザイナがプロジェクトにどの程度貢献できるかは、デザイナの WPF ツールの習熟度によって異なります。たとえば、デザイナがツールからエクスポートしたグラフィックの資産を受け渡すだけの場合もあれば、デザイナが実際の WPF アニメーション、スタイル、およびテンプレートを作成して、開発者に組み込んでもらう場合もあります。このワークフローでは、デザイナは WPF の機能に非常に精通していても、プロジェクトからは一歩距離を置いているということになります。

ハーベスト モデルのメリットは、デザイナがプロジェクトへの影響を把握していなくても、プロジェクトに直接貢献できることです。デメリットは、デザイナがプロジェクトから分離されており、イテレーションで依然として受け渡しの手続きを必要とすることです。

コラボレーション モデル

もう 1 つのアプローチは、デザイナがプロジェクト自体を直接管理するモデルです。このモデルでは、デザイナによるすべての変更がプロジェクトに即座に反映されます。したがって、デザイナは開発者と連携してソリューションを実際に構築することになります。デザイナは XAML ファイルをソース コード レポジトリに格納でき、多くの場合は、アプリケーションをコンパイルして実行できます。このシンプルなモデルは、シームレスなイテレーションを可能にします。多くのプロジェクト、特に革新的なプロジェクトでは、このことがコラボレーションを強力に支援します。これはあらゆるモデルにとっての長期的な目標であり、モデルが進むべき方向でもあります。デザイナがプラットフォームについての経験を重ね自信を深めるにつれて、プロジェクトにより直接的に関与できるようになり、最終的にはプロセスの簡素化が実現するのです。

これらの 2 つのアプローチの根本的な違いは、デザイナと開発者との境界です。この境界を定める方法に関しては、明確な規定はありません。現実においても、コードを記述できるデザイナとグラフィック / 対話型要素のデザインに対処できる開発者とが増加するにつれて、この境界は曖昧になってきています。スキルに関係なく、プロジェクトに関与する各役割の境界を明らかにすることがポイントになります。XAML の ” 所有者” を明確に理解することが重要です。そのほかに重要な要素としては、こうした資産のパッケージ化 / 受け渡しの戦略があります。これについては、ベスト プラクティスについてのセクションで説明します。

ツール

XAML はテキスト エディタのみでも編集できますが、RAD（Rapid Application Development：迅速なアプリケーション開発）ツールを使用してユーザー インターフェイスを構築すると、さらに生産性が向上します。XAML は XML をベースとしており、XAML を編集またはエクスポートできるツールは数多く提供されています。このドキュメントでは、すべての市販ツールを取り上げることはできませんが、よく使用されているツールを紹介します。以下の説明では、ツールを 3 つのカテゴリに分類しています。すなわち、XAML をターゲットとするネイティブ ツール、XAML をエクスポートするツール、および WYSIWYG XAML エディタです。

ネイティブ XAML ツール

このカテゴリのツールは、XAML を生成するデザイン サーフェスを表示します。これらは、本質的に “双方向” ツールです。つまり、デザインを変更すると XAML が変更され、XAML を変更するとデザインが変更されます。

Expression Blend

Expression Blend は、XAML ベースのアプリケーション用の対話型デザイン ツールです。Blend を使用すると、XAML の表現力を最大限に活用できます。レイアウト、グラフィック、コントロール、テンプレート、スタイル、アニメーション、データ バインディング、標準的な 3D などを自在に操作できます。Blend という名称のとおり、デザイナ、対話型デザイン ツール、開発者の連携の下にデザインを実現できます。デザイナは、Blend を使用して、ビジュアル、スタイル、テンプレート、アニメーションの作成や、アプリケーションのルック アンド フィールの構築に関連するさまざまなタスクを手掛けます。開発者は、Blend を使用してさまざまな UI 関連作業を手掛けますが、多くの場合、Visual Studio のデバッグ機能と豊富なコード編集機能を合わせて活用します。

Visual Studio 2008

Visual Studio 2008 には、WPF アプリケーションを構築するためのビジュアル エディタ[9] が用意されています。このエディタは XAML 対応の優れたテキスト エディタで、IntelliSense 機能、リッチ レイアウト、ドキュメント ナビゲーション機能、サード パーティ コントロール用のホスティング / 拡張モデルを実装します。ただし、スタイル、テンプレート、およびアニメーション対応のビジュアル エディタは装備していません。Blend と Visual Studio は同じプロジェクト システムを使用しており、両方とも XAML 形式へのシリアライズが可能です。したがって、2 つのツール間の作業をシームレスに切り替えることができます。

各ツールの特長

Microsoft® Expression Blend™

- 高度なレイアウト
- スタイル
- コントロール テンプレート
- トリガ
- データ テンプレート
- アニメーション
- ベクタ グラフィック

Microsoft® Visual Studio® 2008

- 高度なレイアウト
- WPF プロジェクト テンプレート
- イベント ワイヤアップ
- C#、Visual Basic
- コード編集機能
- XAML 編集機能
- デバッグ機能
- 展開

XAML をエクスポートする既存のツール

このカテゴリのツールおよびプラグインは XAML をエクスポートしますが、XAML をネイティブに取り扱いません。つまり、XAML を直接処理することはできません。XAML のインポートでは、ツールからプロジェクトへの一方向のフローが生成されます[10]。

9 このデザイン ツールのコードネームは “Cider” です。Cider は Visual Studio 2005 で使用できますが、CTP（Community Technical Preview）としてのリリースであり、サポート対象外です。

10 各種コンバータ（HTML、Flash など）は含まれません。

Microsoft Expression Design

Expression Design は、グラフィック デザイン要素を構築できるベクタ ベースのツールです。Expression Design では、XAML とラスタ ベース イメージの両方をエクスポートできます。また、XAML が対応していない表現性のきわめて高いグラフィックも自在に作成できます。デザイナはビジュアル要素を作成して、XAML でサポートされていない要素を簡単にラスタ化できます。なお、ブレンド モードやパスのテキストなど、XAML としてエクスポートできず、現時点ではラスタ化用にサポートされていない機能も一部存在します。

Adobe Illustrator

Illustrator は、業界で定評のあるベクタ ベースの描画ツールです。多数のデザイナがこのツールを使い慣れているため、Illustrator で UI を構築してから XAML としてエクスポートする手順がよく使用されます。Illustrator から XAML を生成するには、次の 2 つの方法があります。

1. プラグインを使用して Illustrator から XAML をエクスポートする[11]。

2. Expression Design を使用して AI ファイルをインポートしてから、XAML にエクスポートする。

Adobe Illustrator で作成したビジュアル要素の中には、XAML に変換できないものもあります。特に、Adobe Illustrator のエフェクトやフィルタを使用する場合には注意してください。

XAML をエクスポートする 3D ツール

WPF は 3D をサポートし、Expression Blend は 3D シーンの基本操作をサポートしますが、マイクロソフトの各種ツールは現時点では 3D モデルの構築には対応していません。既存の 3D 資産を XAML に変換するための変換プログラムを、http://blogs.msdn.com/mswanson/articles/WPFToolsAndControls.aspx から入手することができます。また、Expression Blend では .obj ファイルを直接インポートすることができます。

Expression Design や Adobe Illustrator と同様、これらの 3D 変換プログラムは一方向のツールであり、XAML としてレンダリングした場合に忠実度が損なわれる可能性があります。

WYSIWYG XAML エディタ

WYSIWYG（What You See Is What You Get）方式の XAML エディタは、WPF アプリケーション開発の中心的な役目を果たします。XAMLPad、XAMLCrunch、Kaxaml[12] などのエディタが有名です。これらのエディタは複数の分割ビューを実装した軽量のユーティリティ アプリケーションであり、無償で配布されています。画面の半分の領域で XAML を入力し、残りの領域でその XAML によるビジュアル レンダリングを確認できます。Visual Studio 2008 と Blend 2 はこの分割ビュー機能を実装しているため、こうしたエディタを使用する機会は今後少なくなっていくと思われますが、余分なオーバーヘッドのない軽量エディタはやはり便利です。

11 現在、2 つの Illustrator プラグインがあり、それらは以下のサイトで確認できます。
http://blogs.msdn.com/mswanson/articles/WPFToolsAndControls.aspx（英語）

12 XAMLPad は Windows SDK、XAMLCrunch は http://windowsclient.net/downloads/folders/controlgallery/entry2314.aspx（英語）、Kaxaml は http://notstatic.com/archives/49（英語）で、それぞれ確認できます。

ツールの比較

以下に示す表は、WYSIWYG エディタを除いた各種ツールを比較し、使用可能な機能の概要を示したものです。

	Expression Design	Expression Blend	Visual Studio 2008	Adobe Illustrator
レイアウト	静的、絶対位置に配置	動的	動的	静的、絶対位置に配置
スタイル	非対応	ビジュアル エディタ	非対応	非対応
テンプレート	非対応	ビジュアル エディタ	非対応	非対応
リソース	エクスポート オプションとして	ビジュアル エディタ	非対応	非対応
コーディング サポート	非対応	基本エディタ	リッチ IntelliSense	非対応
デバッグ	非対応	非対応	対応	非対応
XAML の ラウンド トリップ	一方向	対応	対応	一方向
XAML の エクスポート	損失防止のため機能に制約あり	対応	対応	損失大。ツールは WPF よりも多機能
アニメーション	非対応	ビジュアル エディタ	XAML エディタのみ	非対応

デザイナのベスト プラクティス

デザイナにとって、WPF を導入するのは非常に簡単です。各ツールは既存のデザイン ツールに類似しており、ビジュアル要素のドラッグ アンド ドロップにより、魅力的なユーザー インターフェイスを簡単に作成できます。ただし、そのようなアプローチでは、成果物となるアプリケーションが適切に実行されないか、あるいは保守性の面で最適化されない場合があります。以下に、実際のプロジェクトに基づいて得られたベスト プラクティスをご紹介します。

デザインでのコスト編成

印刷のデザインでは、資産の構築時に資産の印刷コストを削減するための意思決定を頻繁に行います。これには、カラーの低減、インパクト フォントの選択、イメージ サイズ、ディテールなどがあります。アプリケーションのデザインでも、WPF アプリケーションのデザイン時にコストを最小限に抑える方法を検討する必要があります。デザイン内のすべてのビジュアル要素のコストを把握することは、メモリや CPU を多量に消費しすぎない応答性に優れたアプリケーションの構築には不可欠です。レンダリング コストは、ターゲットのユーザーとアプリケーションを実行するハードウェアに見合ったものであることが必要です。次に、コストの編成によって差が生じる可能性のある領域についていくつか説明します。

- 実行時、XAML の描画はオブジェクトのコレクションを生成し、ベクタ ベースの描画プリミティブを表現します。このモデルには、ベクタ描画をディスプレイの解像度に適合させることができ、描画プリミティブのレンダリングを GPU（別名：ビデオカード）で処理できるというメリットがあります。なお、非常に詳細なデザインの場合は、多数のオブジェクトとリソースが生成されて実行時にインスタンスが作成されるため、ラスタ イメージと比較して、メモリや CPU の消費量が大きくなる点に注意が必要です。プログラムで操作する必要のない多数のオブジェクトを含む詳細なデザインでは、ラスタ イメージや **DrawingImage**[13]（WPF でサポートされる特殊なベクタ ベースのイメージ）を作成することをお勧めします。ラスタ イメージを使用すると、解像度を変更したときにベクタ ベースのスケーラビリティが損なわれますが、これは各解像度で複数のイメージを作成することによって回避できます。**DrawingImage** はベクタ ベースであり、あらゆる解像度に適合します。

- WPF の最も魅力的な機能の 1 つに、アプリケーションで使用できる強力な 3D エンジンがあります。ただし、最適化された保持モード アーキテクチャのために、他の 3D プログラムから WPF に資産を組み込む場合に、いくつかの問題が発生します。まず、WPF 3D エンジンでは、DirectX や OpenGL などの他の 3D エンジンよりもオーバーヘッドが大きくなります。したがって、3D デザインのコストを編成して、頂点とライト（特にポイント ライト）の数を最小限に抑える必要があります。そのような問題はありますが、一方で、WPF プロジェクトに効果的に組み込まれた 3D の優れた事例が多数存在します。ターゲットとなるハードウェアのベンチマークを綿密に実行し、デザイナがデザインのコスト編成を実施しさえすれば、3D を使ったシャープなビジュアル要素をアプリケーションに付加できます[14]。

- **BitmapEffect** は WPF の高度な機能の 1 つです。デザイナが、そのルック アンド フィールの使用を検討する場合、作業は慎重に行う必要があります。これにより、すべての関連コンテンツがハードウェア アクセラレーションなしにレンダリングされてしまうためです。パフォーマンスを低下させることなく、**BitmapEffect** の視覚効果を利用するために、いくつかの回避策が用意されています。適用する視覚効果が静的なものである場合（サイズ変更のない要素のドロップ シャドウなど）、描画テクニックを駆使して同じエフェクトをシミュレートできます。視覚効果が動的なものである場合は、開発者と連携して、**RenderTargetBitmap** などのビットマップ API を使用してエフェクトを適用してから、そのルック アンド フィールをキャッシュできます。

上記の 3 つは、アプリケーション開発成功の妨げとなる資産をデザイナが構築する可能性のある既知の領域ですが、すべてが網羅されているわけではありません。重要なことは、デザインがアプリケーションのパフォーマンスに与える影響をデザイナが認識しておく必要があるという点です。パフォーマンス関連の問題を早期に回避することは、後からそれを修復するよりもはるかに簡単です。

13 http://msdn2.microsoft.com/ja-jp/library/system.windows.media.drawingimage.aspx を参照してください。

14 WPF で 3D を使用する場合の有用なヒントについては、http://blogs.msdn.com/wpfsdk/archive/2007/01/15/maximizing-wpf-3d-performance-on-tier-2-hardware.aspx（英語）を参照してください。

再利用の促進

以下に示す複数の理由から、デザイナが XAML 資産を再利用するテクノロジを習得することは重要です。

- 似かよった多数の項目を新規作成するのではなく、既存の資産を再利用すれば、パフォーマンスが向上します。たとえば、多数の項目を描画する場合にブラシを再利用すると、アプリケーションのパフォーマンスは大幅に向上します。再利用によって、Blend や Visual Studio 2008 でのデザイン時の負荷が軽減されるため、デザイナのパフォーマンスも向上します。

- 再利用によって XAML がコンパクトになるため、保守性が向上します。たとえば、ブラシ、スタイル、テンプレートを再利用すると、アプリケーションのルック アンド フィールに影響を与える変更箇所は 1 つだけになります。

再利用を促進するための最善の方法はリソースを経由させることです。XAML は、要素全体で共有するリソース用のプロパティ バッグとして機能するリソース ディクショナリをサポートします。Expression Blend と Visual Studio では、リソース ディクショナリをネイティブに使用できるため、さまざまな場合に要素をリソースとしてパッケージ化できます。Expression Design では、リソースとして項目をエクスポートすることもできます（エクスポート機能の［ドキュメント形式］オプション）[15]。

一方向ツールについての理解

Expression Design、Adobe Illustrator などの一方向のツールやエクスポータを使用する場合、イテレーションを実行してワークフローのメリットが得られるように、注意を払う必要があります。ツールで XAML 資産をエクスポートした後は、その資産を直接変更しないことを強くお勧めします。XAML を変更せずに保持することによって、これらの XAML 資産を再びエクスポートしてプロジェクトに統合するときに生じる問題が最小限に抑えられます。

たとえば、次のシナリオについて考えてみましょう。デザイナがツールでボタンを作成して、XAML としてエクスポートします。次に、開発者がエクスポートされた XAML の名前などを修正します。その後、ボタンの色を変更する必要があることが決まりました。デザイナが一方向のツールに戻り、色を変更すると、エクスポートされる XAML の名前は、開発者が変更した名前とは対応しなくなります。結果としてそのボタンがプロジェクトで機能しなくなり、開発者が XAML を再度修正する必要が生じます。ここでは、イテレーションの最適化が損なわれています。

この問題を解決するには、ツールからエクスポートされた XAML を慎重に統合する必要があります。XAML 資産を再びエクスポートするときに XAML が破損してしまう可能性を最小限に抑えるには、以下に示すいくつかのテクニックを使用できます。

15 Adobe Illustrator から XAML に直接エクスポートするか、あるいはそれらの資産を Expression Design にインポートしてから XAML にエクスポートするかを選択する場合、Expression Design を経由させればリソース ディクショナリのネイティブ エクスポートがサポートされるというメリットがあります。

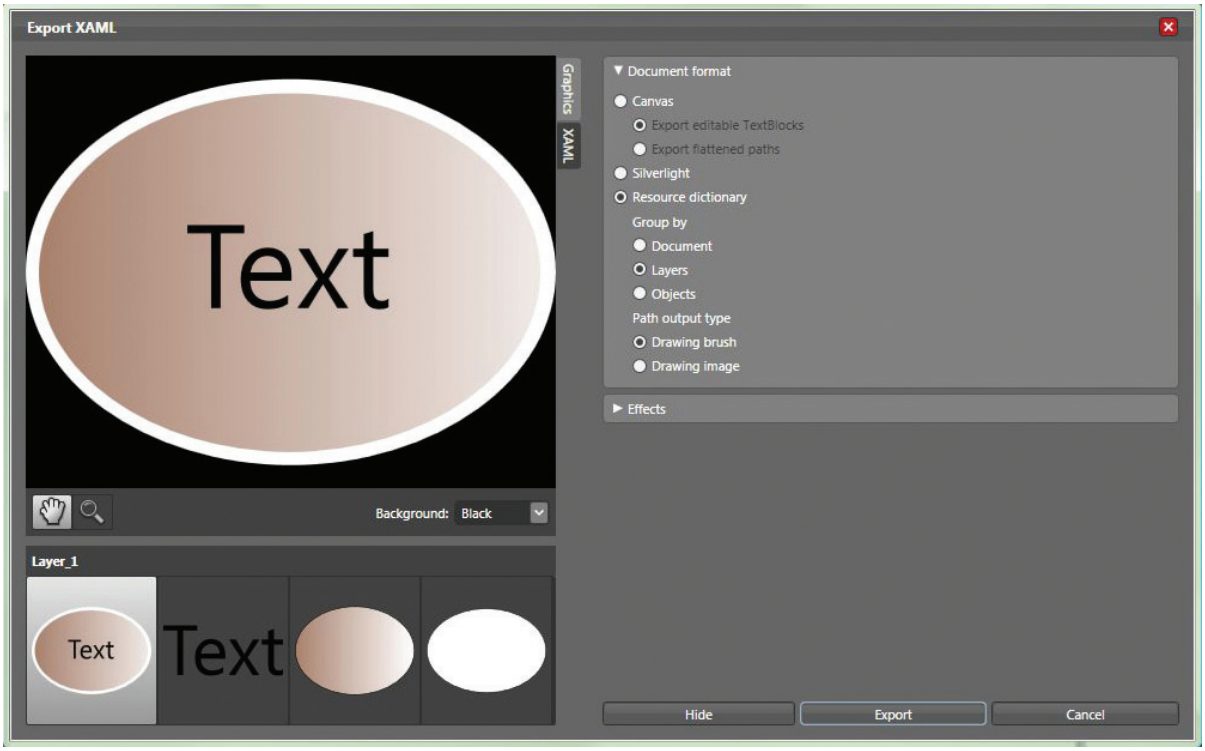
- Blend で “ロック” 機能を使用します。Expression Blend で XAML の親ノードをロックすると、その資産が誤って変更される可能性が減少します。

- インポートされた XAML を配置した場所が明確にわかるように、直接 XAML に詳細なコメントを付加します。

- エクスポートされた XAML をリソース ディクショナリとしてプロジェクトに組み込みます。すべてのリソース ディクショナリをツールからエクスポートした資産とするような、わかりやすい運用方法を確立すれば、誤ってリソースが編集されることがなくなり問題の回避につながります。

グループ オブジェクト

デザインの多くは、論理エンティティまたは論理オブジェクトとしてグループ化する必要があります。このグループ化を最初から実行しておくと、開発プロセスでの統合が迅速化されます。次に示す Expression Design の丸いボタンのテンプレートをご覧ください。これは 2 個の楕円とその内部のコンテンツから構成されています。これをオブジェクトとしてエクスポートして、適切にグループ化すると、ボタン全体が再利用可能なエンティティになります。ただし、オブジェクトをグループ化しなかった場合は、3 つの個別パーツとしてエクスポートされます。オブジェクトをグループ化した方が、単一のエンティティとして扱いやすいことは明らかです。



Expression Design と Expression Blend は、共にオブジェクトとして再利用できるグループを作成する機能を搭載しています。

デザイナがプロセスを理解するには、WPF のしくみを詳細に把握することが必要です。

解像度に依存しない論理ユニット

WPF は解像度に依存せず、要素にサイズを与える論理ユニットを実装します。WPF の各論理ユニットは 1/96 インチです。たとえば、<Rectangle Width="288" /> と定義された四角形は、システム解像度が 96 dpi、144 dpi など、どのような解像度であってもそれには関係なく、正確な 3 インチ幅になります。Blend と Visual Studio 2008 の内部処理時に、すべての要素が同じ縮尺として処理されるため、論理ユニットと物理ユニット間での変換処理は目には見えません。ただし、コンテンツ（特にラスタ イメージ）のインポート時には、WPF の解像度に依存しない機能に効果的に適応させるために、多少のスケーリングが発生します。このようなスケーリングについて、例を挙げて説明しましょう。

Expression Blend と Expression Design で、72 ピクセル / インチで作成され、幅が 216 ピクセルであるラスタ イメージをインポートすると、Blend ではイメージが 4/3（96/72）倍にスケーリングされます。イメージが 3 インチ幅（216/72）であったため、Blend では WPF でのサイズが 3 インチを維持するようにスケーリングされるのです。このイメージを 144 ピクセル / インチで作成していれば、Blend では 2/3（96/144）倍にスケーリングされます。

Expression Design でも、単位がピクセルである Adobe Illustrator ファイルを開く場合に同様のスケーリングが発生します。ポイント単位の Illustrator ファイルであれば、スケーリングは行われません（これは、ポイントが解像度に依存しないためです）。

動的レイアウトとコンテンツ モデル

WPF アプリケーションのレイアウトには、次の 3 つの基本オプションがあります。

1. 静的レイアウト。すべてが絶対的な位置に配置され、サイズは変更されません。
2. 動的レイアウト。UI 要素のサイズは使用可能なスペースに合わせて変更されます。
3. 上記 1 と 2 を組み合わせたレイアウト。

XAML をエクスポートするほとんどの一方向ツールは、絶対位置座標（静的レイアウト）に対応します。動的レイアウトを使用するには、Blend、Visual Studio 2008 などのネイティブ XAML ツールが必要です。

レイアウトを操作するときに、WPF のコンテンツ モデルを理解していると役に立ちます。WPF レイアウト コントロールには、1 つの子にしか対応できないものや、複数の子に対応できるものがあります。複数の子に対応するコントロールには、子のオーバーラップが可能なものと可能でないものがあります。こうした要件は取り扱いが面倒ですが、デザイナ向けのガイダンスとしては、以下のようにまとめることができます。

- 何の設定も加えずに、内部の子のオーバーラップが可能であるコントール（オーバーレイ エフェクトの作成が可能なコントロール）は、Canvas と Grid の 2 つです。

    - Canvas では、子は絶対的な位置に配置されます。子は相互にオーバーレイできます。ZIndex プロパティの値が最も高い子が前面になります。2 つの子の ZIndex 値が同じ場合は、最後に描画される子が前面になります。

    - Grid では、マージン指定による絶対座標を使用できます。または、アライメント、ストレッチ、およびマージンの指定による相対配置を使用できます。

    - コンテンツとして子を許可する他のあらゆる要素は、Grid、Canvas などの要素を格納できるため、任意の数のオーバーレイを柔軟に作成できます。

絶対座標で作成したレイアウトから動的レイアウトの一部の機能を利用するには、UI のスケーリングを実行します。この場合、2 つのテクニックがあり、WPF Viewbox コントロールを使用するか、または UI 要素への Scale 変換を適用します。後者の場合、すべてのコンテンツ（子）が UI 内でスケーリングされます。

テンプレートとスタイル

WPF では、コントロールのビジュアル構成を定義するコントロール テンプレートを使用して、コントロールのビジネス ロジック（動作）とルック アンド フィールを分離できます。

たとえば、次のサンプルでは、ボタンの既定のコントロール テンプレートは、ButtonChrome 内部の ContentPresenter です。

```
<ControlTemplate TargetType="{x:Type Button}">
 <Microsoft_Windows_Themes:ButtonChrome ... >
   <ContentPresenter ... />
 </Microsoft_Windows_Themes:ButtonChrome>
</ControlTemplate>
```

標準の Windows テーマに基づいたボタン

ButtonChrome の部分を Grid と Ellipse（背景用の円）で置き換えると、ボタンは次のように丸くなります。

```
<ControlTemplate TargetType="{x:Type Button}">
 <Grid ... >
   <Ellipse ... />
   <ContentPresenter ... />
 </Grid>
</ControlTemplate>
```

カスタム ボタン

コントロール テンプレートを使用すると、デザイナはボタン内部のビジュアル要素を置き換えることによって、どのような外観のボタンでも作成できるようになります。テンプレートを置き換えても動作は変わりません。このボタンは、特に追加の作業をすることなく、マウスのクリックとポイントに反応します。

コントロールのスケルトン（ビジュアル要素）をコントロール テンプレートで定義すると、WPF ではルック アンド フィールをカスタマイズしたリッチ スタイルがサポートできます。スタイルを使用すると、再利用可能な設定（または属性）のグループを作成して既存の UI 要素にアタッチすることによってルック アンド フィールをカスタマイズできます。

コントロール テンプレートとスタイルの違いについてよく質問を受けます。テンプレートは要素のビジュアル構成を置き換えるのに対して、スタイルは要素の既存の属性やプロパティを設定するのみとなります。

スタイルとテンプレートの連携については、さらに込み入った議論があります。ここでは説明しませんが、デザイナと開発者はこの 2 つのコンセプトを正確に理解しておく必要があります。これらのコンセプトが、新しいコラボレーション モデルを実行可能にする分離（コードからの UI の分離）機能の基盤となります。コントロール テンプレート、データ テンプレート、スタイル、リソース ディクショナリを活用することによって、開発者はロジック（動作）に、デザイナはビジュアル要素（UI）に、それぞれ専念することができます。

WPF アニメーション

WPF では、アニメーションはタイム ベースであり、フレーム ベースではありません。タイム ベースのアニメーションのメリットは、エンド ユーザー エクスペリエンスの一貫性が向上することです。タイム ベースのアニメーションでは 1 秒が固定した単位となります。フレーム ベースのアニメーションでは 30 フレームが 1.5 秒または 1 秒に相当しますが、これはアニメーションを実行するシステムに応じて異なります。一方、タイム ベースのアニメーションのデメリットは、アニメーションの作成や視覚化、およびアニメーションの前後の状態が少し複雑になることです。Blend を使用すると、アニメーションを作成して、実行をプレビューできます。開始点のない相対アニメーションの場合、始点 / 終点を視覚化することが困難になりますが、この制約はアニメーションの開始点（0:0:0）を設定することによって回避できます。表示を確認して、アニメーションのデザインが完了してから、開始点を取り除いて相対アニメーションにすることができます。

開発者のベスト プラクティス

開発者 / デザイナの新しいワークフローは、役割間のコラボレーションの強化をベースに構築されます。開発者がコードとプロセスを最適化すると、デザイナがツールの使用時に直面する問題が少なくなります。

Blend の導入

デザイナにとって最適化の鍵となるのは、Blend の導入です。Blend を使用して、Blend でプロジェクトを開くことができるようにすると、デザイナがプロセスに参加し続けることができます。Blend でプロジェクトを開くことができない場合、デザイナはプロジェクトのデザイン時ビューを取得できなくなるため、実質的にプロセスから排除されてしまいます。変更内容を確認するには、プロジェクトをもう一度コンパイルしてから再実行する必要があります。IdentityMine の Robby Ingebretsen 氏は、このような状況を「Blend でプロジェクトを表示できなくなると、作業サイクルが “微調整 - 微調整 - 微調整” ではなく、“微調整 - コンパイル - 実行” になってしまいます」と表現しています。

このような背景から、開発者がアニメーションなどを作成し、Blend でデザイナによるブラッシュアップや調整を行う場合は、コードではなく XAML を使用することをお勧めします。XAML での作業を習得することは、開発者にとっては新たなタスクになりますが、これにより最終的には生産性と創造性の向上が実現されます。デザインツールでプロジェクトを開くことができれば、デザイナが関与できる機会が得られます。一方、プロジェクトがコードで分離されると、デザイナがプロジェクトに直接アクセスすることはできなくなります。

Blend でプロジェクトを開けるということによってもたらされるメリットについて、Blend チームのケースを事例として取り上げてみましょう。Expression Blend のプログラム マネージャである Samuel Wan は、次のように述べています。「私たちは、Blend と、Blend を使用する Design のスキンの両方を構築しました。ツールでビジュアルなフィードバックを即座に得られることは、非常に貴重でした。Design の場合、わずか数か月という期間で 8 年目の製品のルック アンド フィールを徹底的に検討し直すことができました」

Blend の内部動作について

ここでは、開発者が理解しておくべき Blend の内部動作について解説します。Blend でシーンをデザインするとき、シーンで参照するものすべて（コントロール、ベクタ グラフィック、ビットマップ、ビデオなど）がメモリにロードされて、初期化コードが実行されます。つまり、初期化コードでエラーが発生すると、ツールに問題が発生して、コントロールをロードできなくなります。最悪の場合には、シーンがまったくデザイン サーフェスにロードされません。よくあるケースは次のようなものです。開発者がユーザー コントロールを作成し、患者情報などを表示します。コントロールの初期化コードでは、開発者はデータ コンテキストを解析して患者の名前と姓を取得します。データ コンテキストのないコントロールが Blend のデザイン サーフェスにロードされると、コードの解析処理が失敗し、Blend ではユーザー コントロールがロードされません。

このような問題を回避するために、WPF ツールでは **Microsoft.Windows.Design** 名前空間のデザイン時 API を提供しています。これにより、デザイナ内部で処理が実行されているかどうかを確認できます。これらの API を使用して、開発者は、デザイン環境を確認し、コントロールでのエラーの発生を回避する別のコードを実行できます。

データ優先型アプローチ

すべてのアプリケーションはそれぞれ異なるため、このアプローチが絶対的なベスト プラクティスであるとは言えません。しかし、大多数のアプリケーションでは、データとして格納されている情報を表示または取得し、それに対して XAML はデータ バインディング、データ テンプレートなどの強力なデータ処理機能を実装しています。ここで重要なのは、ツールの使用を開始するときにデータを定義する必要があるという点です。これにより、データ スキーマに基づいたコードをツールで自動生成できます。

たとえば Blend では、データ スキーマ（CLR オブジェクトまたは XML スキーマ）の簡単なドラッグ アンド ドロップ操作で、データ テンプレートを作成できます。デザイナがユーザー インターフェイスのレイアウト作業を開始する前にデータとスキーマが用意できていれば、デザイナと開発者の両者にとって時間の節約になります。デザイナは、ユーザー インターフェイスのレイアウトにとどまらず、さまざまな処理を実行できます。ビジネス オブジェクト スキーマに UI をバインドすれば、開発者が後からその作業を行う必要がなくなります。また、最初からスキーマを使用すれば、データ モデルの変更に伴って後からデザインを変更する必要もなくなります。

一貫性のある XAML ファイルの命名、構造化、文書化

XAML ファイルの管理におけるもう 1 つのポイントとして、XAML ファイル内の各要素の命名規則の指定があります。プロジェクトが複雑になるにつれて、要素を簡単に検索したり把握したりするためには、一貫性のある命名規則が重要になります。大きな XAML ファイルでは、その XAML コードを調べるときに迅速な読み取りが困難になります。開発者は、デザイナが使用した名前に基づいて、イベントをワイヤアップする必要があります。命名規則を指定することによって、このような作業の負担を軽減できます。これは開発者にとっては既になじみの作業ですが、一部のデザイナにとってはまったく新しい作業です。XAML を取り扱うデザイナは、将来的な保守性の面から命名規則に対応する必要があります。要素はコード内で名前によって参照されることがあるため、一度命名してしまうと変更できないようになっている点も理解しておく必要があります。

名前に関する注意事項：オブジェクトを命名する場合（x:Name =""、または Name=""）、XAML コンパイラはコード ビハインド クラスのメンバとして変数を生成します。ほとんどのシナリオでは、これは開発者にとって非常に便利であり、通常、実行時のコストは低くなります（メモリのバイト数と初期化コードの命令数が削減されます）。ただし、オブジェクトへの不要な参照については注意を払う必要があります。命名したオブジェクトを実行時にロードおよびアンロードする場合、クラスがまだメモリに残っているときにガーベッジ コレクタによってこの変数のメモリがクリーンアップされるように、作成される変数にヌルを設定する必要があります。

XAML ファイルの実際のサイズは非常に大きくなることがあるため、可読性と保守性の点で問題が発生する可能性があります。特に、これは XAML を手作業で修正する必要がある場合に問題となります。Blend には “XAML の表示” という名称の機能が実装されており、ユーザーはビジュアル ツリーの表示からすぐに XAML に移動することができます。

これは便利な機能ですが、XAML ファイルの一部であるテンプレートやリソースの確認がさらに困難になる場合があり、XAML の肥大化という問題を根本的に解決するものではありません。ここでポイントになるのが、XAML ファイルのモジュール化です。これは特に大規模なプロジェクトでは重要です。XAML ファイルのモジュール化のためには、XAML ファイルの各要素によって実行される内容を把握することが不可欠です。Paul Stovell 氏[16] はこれらの内容を見事に判別し、XAML ファイルを機能 XAML、リソース XAML のいずれかに分類しました。機能 XAML ファイルには、Window、Page、UserControl などの最上位の要素が含まれます。リソース XAML ファイルには、汎用リソース XAML ファイル、固有リソース XAML ファイルという 2 つのサブカテゴリがあります。汎用リソース XAML ファイルには、Brush や Style リソースなど、アプリケーション全体で使用されるリソースが格納されます。固有リソース XAML には、3D ジオメトリなど、特定の Window、Control、または Page 固有のリソースが格納されます。Stovell 氏が提案するディレクトリ構造を持つサンプル アプリケーションが、同氏のサイトで紹介されています。参考にしてみてください。

コード ビハインドの選択

WPF では、XAML によってコードをインラインで挿入できますが、これは推奨されません。コードを XAML ファイル内に配置すると、コードと XAML の分離が無効になってしまい、結果として保守が非常に困難になります。この操作はできるかぎり避けるようにしてください。イベント ハンドラの初期化を XAML またはコードのどちらで行うかという判断についても、注意する必要があります。すべてのイベント ハンドラの初期化をコードで行うと、イベント ハンドラが XAML ファイル内に分散されることはなく、プロジェクトがすっきりします。しかし、イベント ハンドラの初期化をすべてコードで行った場合、Blend の対話型デザイン ツールはプロジェクトやアプリケーションのさまざまな状況に効果的に対処できなくなります。イベント ハンドラの初期化を XAML で行うと、プロジェクトは Blend からアクセスしやすくなりますが、イベント ハンドラが複数の箇所で定義されるため、保守の面で困難が生じます。

プラットフォームのアーキテクト

ここで、テンプレート、スタイル、リソース ディクショナリ、コマンド、データ バインディング、依存プロパティ、トリガに関してさらに詳細に説明を行うと、一冊の書籍ができてしまうでしょう。これらの機能はすべて WPF で使用可能であり、XAML を介してサポートされます。したがって、デザイナ フレンドリであり、Blend から使用できます。「UI ロジックのコーディングでは、多くの場合、データ バインディング、トリガ、テンプレートを十分に使用できない」という確固たる経験則があります。しかし、前述のとおり、XAML は表現性、包括性、および拡張性に非常に優れた UI 言語です。XAML を使用すると、デザイナにはアプリケーションのユーザー インターフェイスと対話機能を構築する権限が与えられ、ビジネス ロジック、コンポーネント化、カスタム コントロールの作成、データ モデルの公開などのタスクは開発者に委任されます。

16 http://www.paulstovell.net/blog/index.php/xaml-and-wpf-coding-guidelines/（英語）

SNOOP による実行時の XAML の微調整

ユーザー インターフェイスの構築にツールを使用した場合に生じる固有の問題の 1 つは、デザイン時にアプリケーションのすべての状態をレンダリングできるわけではないということです。多くの場合、アプリケーションの状態が確認できるのは実行時に限られます。そのため、デザイナや開発者がアプリケーションで必要な個所を修正する機会は制限されています。そのような状況に対して、実行時に任意の UI プロパティの値を変更できる SNOOP[17] という魅力的なツールが提供されています。

Expression Blend と Expression Design のユーザー エクスペリエンスやスタイルを担当する Expression チームのデザイナである Aaron Jasinski は、これはさまざまなケースでユーザー インターフェイスを微調整するために必須のツールであると述べています。デザイナは、アプリケーションのライブ ビューを確認して、プロパティを変更することができ、続いて、開発者はその変更をコードに組み込むことができます。SNOOP は、完璧なデザイン ツールと言えるほどには洗練されていませんが、実行時のユーザー インターフェイスを修正する場合に大きな威力を発揮します。

あとがき

このドキュメントは、多くの人々の協力なしには完成できませんでした。グラフィック担当の Tim Aidlin 氏、貴重なフィードバックを提供していただいた多くのレビューアの皆様、ならびにこのプラットフォームを早くから導入してくださった企業の皆様のご協力に心から感謝を申し上げます。

17 http://www.blois.us/Snoop/（英語）