



TEMPLATE SPECIALIZATION

Kathleen Dollard
kathleen@mvp.org

msmvps.com/blogs/kathleen
www.appvenue.com

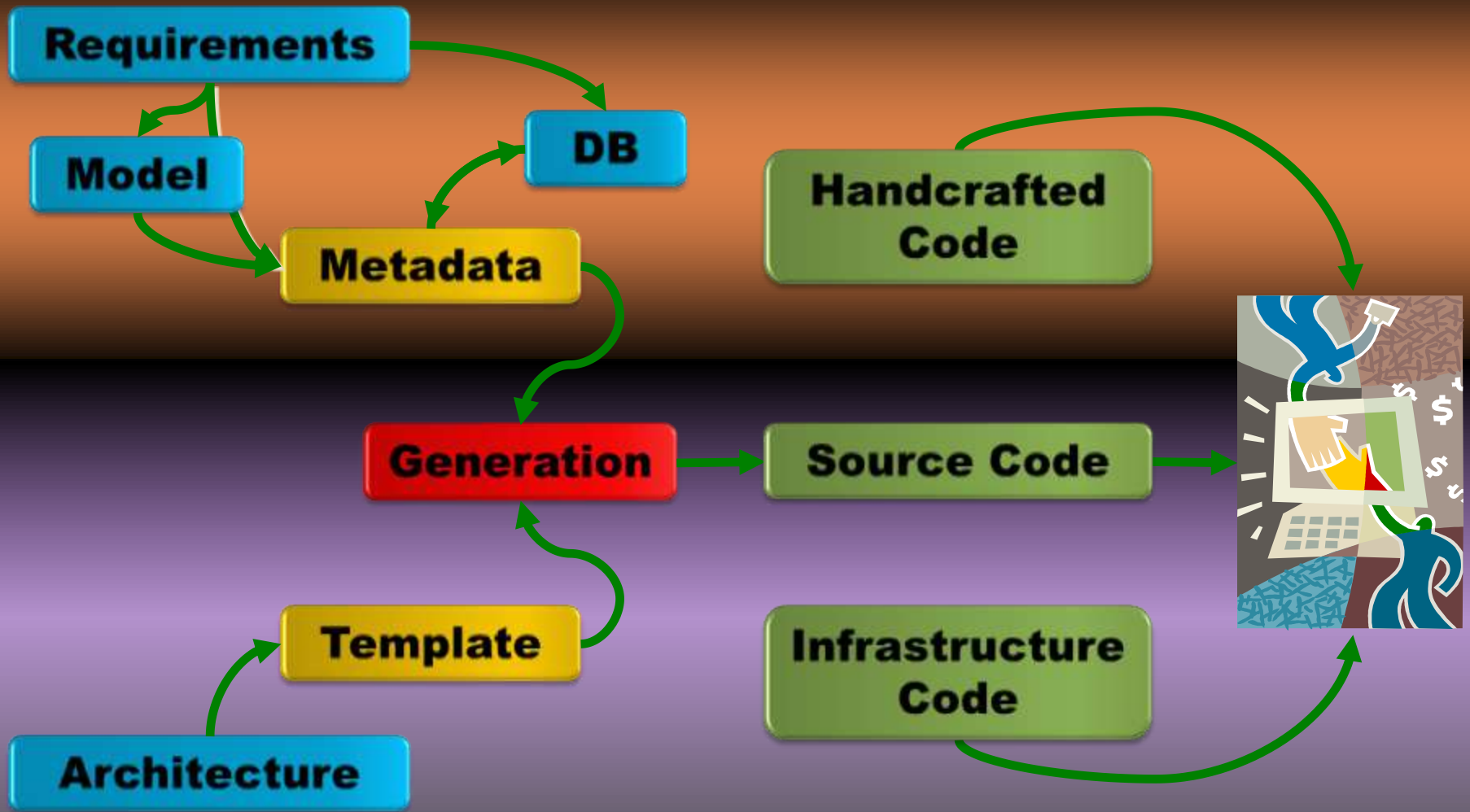
Bio

- First programmed in 1972 in junior high school
 - ▣ I hated programming except on personal platforms
- Middle of a three generation line of programmers
 - ▣ I encouraged my mother and son to program
- Microsoft MVP (since 1998)
- Wrote “Code Generation in Microsoft .NET” Apress, 2003
- Author “Ask Kathleen” column for Visual Studio Magazine
- Member of the INETA speaker’s bureau
- Speak about 50 times/year
- Member of a few insider’s groups advising Microsoft
- Guest on .NET Rocks and Hanselminutes
- Chief technologist of AppVenture
- I’ll follow this conference with a UG tour, Cambridge Thursday
- I’m going to Nepal to help build a library in October

Perspectives and Agenda

- I am thrilled to talk to other folks involved in code gen and modeling, so I'm here to learn
- I believe writing code is a sign of failure
 - ▣ But for now, we need code to comprehend and debug
- Agenda
 - ▣ My perspective (where I'm coming from)
 - ▣ What's the problem we're interested in
 - ▣ What are the current solutions – pros/cons
 - ▣ Where can we go from here
 - How do we get to exchangeable architectures

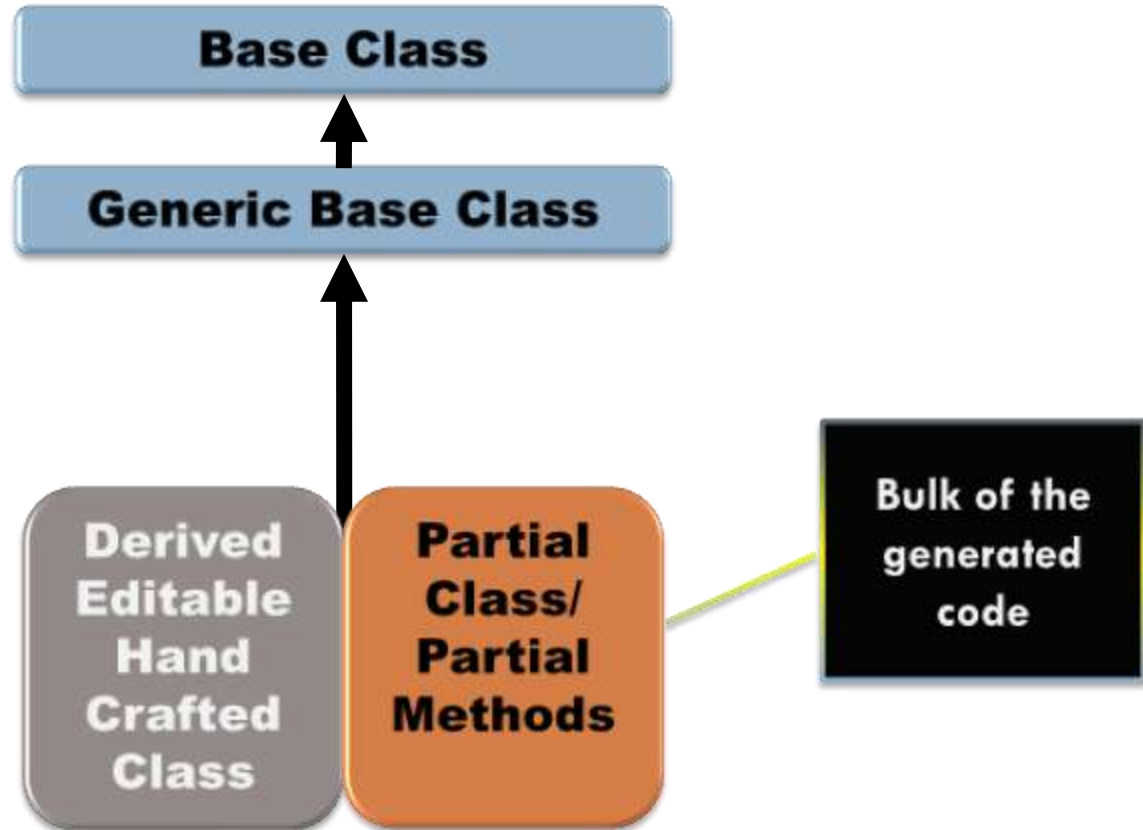
The Big Picture of Code Generation



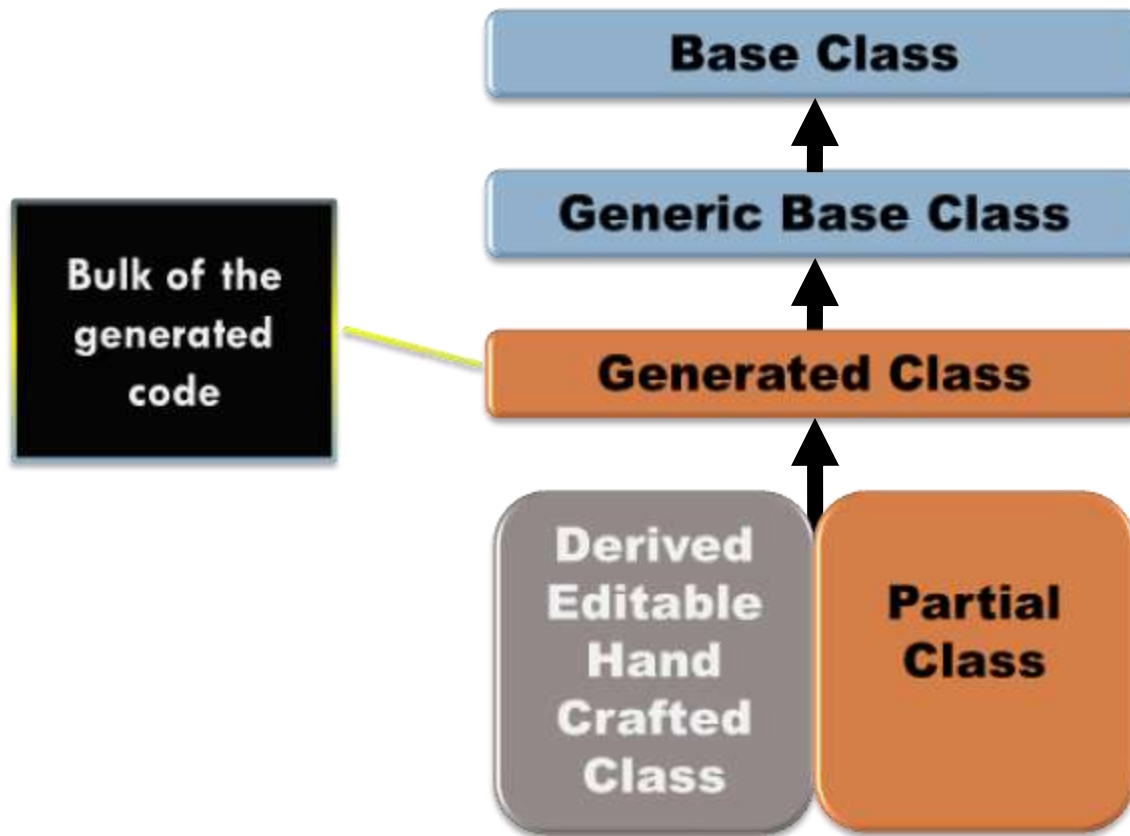
Code Generation Principles

1. Code Generation is under your control
 2. Metadata is distinct and morphable/mappable
 3. Generation is easy and fits into dev process
 4. Handcrafted code is sacred and protected
 5. Generated code is equal or higher quality
- A. Code gen should pay back on first project
 - B. Code gen should be easy

Designing Protection - Partials



Designing Protection - Derived



Some Problems with Templates

- Stack of templates, programmer wants small change
 - ▣ Internal in .NET 1.1 STD templates
- No reuse strategy
 - ▣ Every template starts over
- All output tested after all generation
 - ▣ Generation systems need testing too
- No feature choices
 - ▣ Can't do the Chinese menu thing
- Architectures are not interchangeable
 - ▣ Can't easily evolve
 - ▣ Can't try them on for size
- Templates have no governance

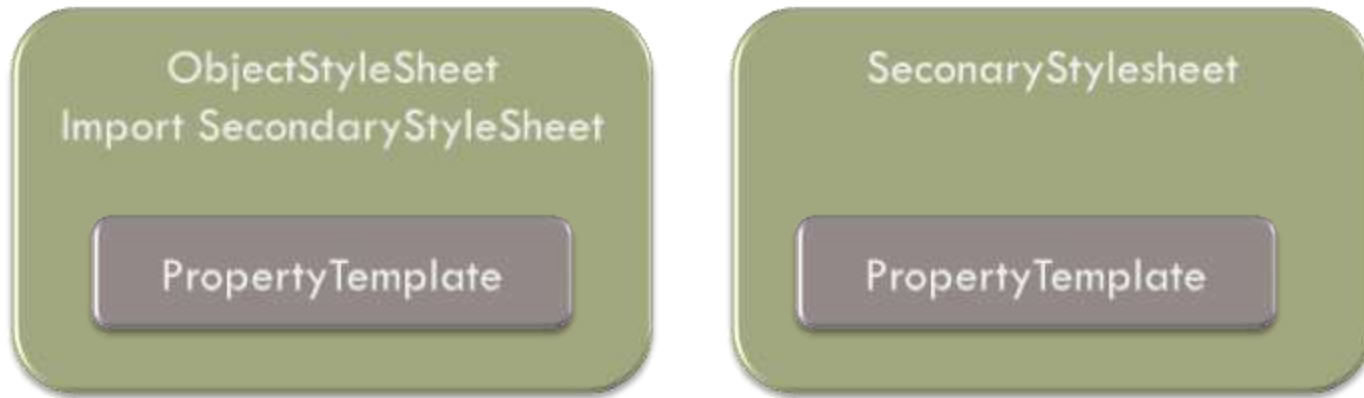
- Generally, people do not treat code generation as an application and process of value in and of itself
- From a generation perspective, modeling is the necessary precursor, the creation of metadata
 - ▣ The format doesn't matter
 - The primary benefit of DSL is bringing new people to the table
- Can we apply SRP (Single Responsibility Principle) to templates?

Why Does Code Generation Matter?

Why Not Just Talk about Modeling?

- Code generation is as an expression of architecture
- Tentative architecture principles
 1. Good architecture is critical
 2. Architecture must evolve to remain good
 3. Over time, entropy transforms hand crafted architectures into big balls of mud
 4. We must exchange architectures for business knowledge to outlive technology
 5. Exchangeable architectures will lead to architectural coalescence

Replacement Solution (XSLT primarily)



Pros

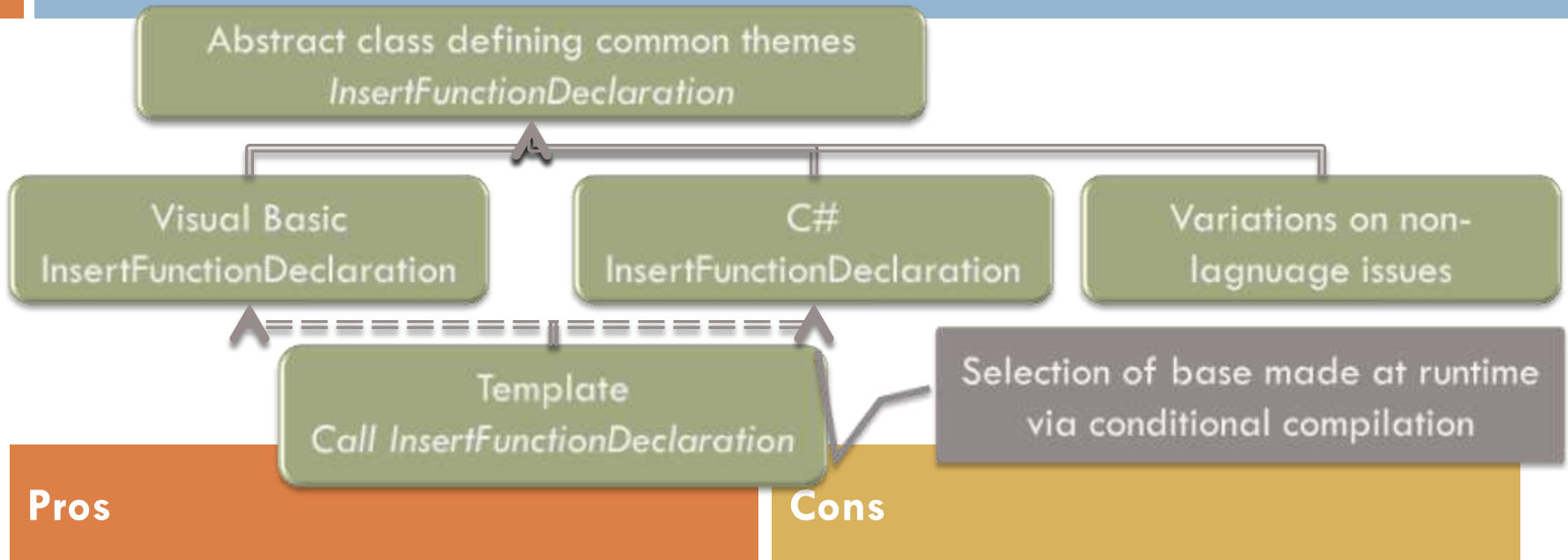
- ❑ Important technique within XSLT

Cons

- ❑ It's XSLT
- ❑ Tight coupling, the entry point knows of the 2nd

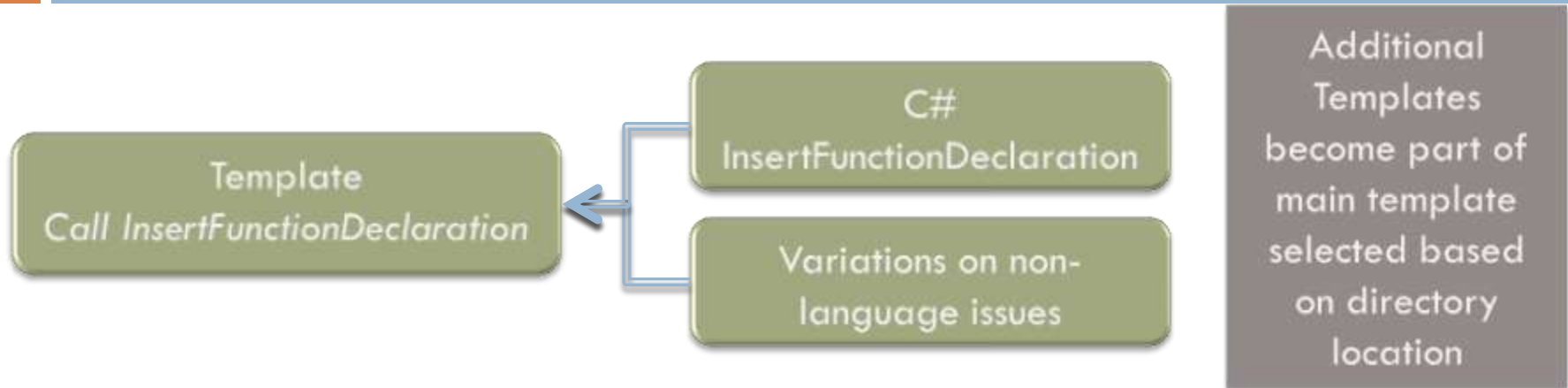
Derivation Solution

(XML Literal Templates in VB9, Code Spit)



Include Solution

(T4, CodeSmith, and Similar ASP.NET Patterns)



Pros

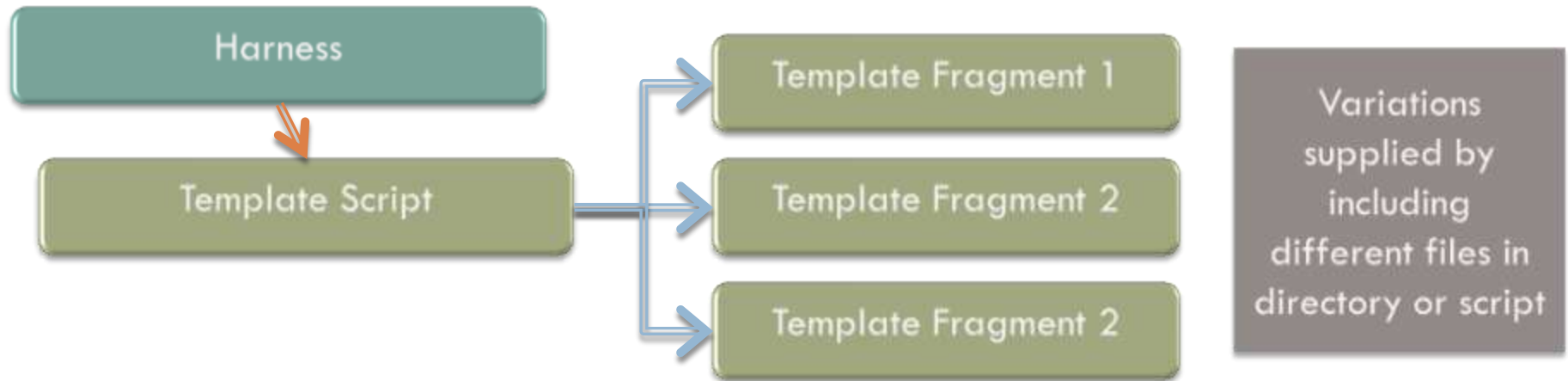
- ❑ Reduces potential for typos in common constructs (declarations, properties...)
- ❑ Reuse of common constructs

Cons

- ❑ Fragmented templates
- ❑ Directory dependent and in T4, relative to entry template

Aggregation Solution

(T4, CodeSmith, and any file based)



Pros

- ❑ Simple to understand
- ❑ Flexible
- ❑ Can document fragments

Cons

- ❑ Reuse challenging, script reuse unlikely
- ❑ Fragile
- ❑ Template tasks must be tightly defined or chaos
- ❑ Proprietary scripting

What Does it Make Sense to Reuse

- Naming
- Property definitions
- Child definitions
- CRUD

Naming

- Metadata could provide all the names
 - ▣ Creates tight coupling with metadata
 - Metadata structure often shared between groups
 - ▣ Makes metadata excessively complex
 - ▣ Rarely have Intellisense for metadata
- Metadata wrappers could provide the naming
 - ▣ Wrapper would require modifications for other naming schema
- Could create naming methods
 - ▣ Do they belong in code or templates?
 - ▣ Either couple to template or via **MEF**
- Given “GetFieldName” method,
 - ▣ Should it receive **property** or name?

What Does it Make Sense to Reuse

- Naming
- Property definitions
- Child definitions
- CRUD

Kathleen's Happy Place (NOT!)

- 500,000 to 3,000,000 architectures
- Architectures are derived from white papers
- Microsoft data technologies haven't succeeded
 - ▣ So they create more...
- No capacity to “try architectures on” for fit

Kathleen's Happy Place (NOT!)

- 500,000 to 3,000,000 architectures
- Architectures are derived from white papers
- Microsoft data technologies haven't succeeded
 - ▣ So they create more...
- No capacity to “try architectures on” for fit
- Architecture evolution avoided or manual
- Code generation has failed for many groups
- Massive amount of code is being written by hand
 - ▣ Sometimes obsolete before release
- Attempted solutions increase knowledge surface area

Kathleen's Happy Place (NOT!)

- 500,000 to 3,000,000 architectures
- Architectures are derived from white papers
- Microsoft data technologies have not succeeded
 - So they create another one...
- No capacity to “try architectures on” for fit
- Architecture evolution avoided or manual
- Code generation has failed for many groups
- Massive amount of code is being written by hand
 - Sometimes obsolete before release
- Attempted solutions increase knowledge surface area
- We've been writing code the same way for about 40 years
- Technology advancement seems to have found Moore's Law
- Risk is enormous – huge sums of money goes to failed projects – trust dissipates
- Stress is high – we don't have enough coders
- No one is competent in the traditional senses of the word in software engineering
- We cannot continue to succeed unless we have massive change
- <add more>

Kathleen's Happy Place

- An ecosystem for exchangeable architectures
 - ▣ The natural reaction to people being able to choose architectures is coalescence on a few
 - ▣ We see this even in white paper based architectures
 - ▣ True expression of the vision
 - ▣ Architectures can evolve smoothly
- How many of the problems on the previous page go away if we generate 75-90% of our code
 - ▣ How many more go away when 75-90% of remaining replaced with rules engines/workflows/other techniques
- We express business process in architecture
 - ▣ Get technology out of our apps
 - ▣ Every line of code we write should be acknowledged as failure

What Do We Need?

- Standard metadata formats that can be morphed into each other
 - ▣ How much variation is there in metadata?
 - ▣ We common ways to address semantics of this problem
- A small handful of architectural models
 - ▣ Start with pipeline, we know we're going to pass data on
 - ▣ SOA, domain/service, client server
 - ▣ Each with predictable boundaries and interfaces
- Reporting and UI can both be generalized (UI is easier)
- Identification of customization drivers
 - ▣ Authentication, validation, notification, calculated fields, localization, processes/workflow, what else?
 - ▣ Create a set of interfaces for each
- Issues are orthogonal, so it's a cross product at the start
 - ▣ We can be sparse, if we define the problem correctly

Thank you!

- BoF?
- kathleen@mvp.com
- www.msmvps.com/blogs/kathleen
- “Ask Kathleen” column in Visual Studio Magazine
www.VisualStudioMagazine.com
 - ▣ T4 and MEF articles recently
- *Code Generation in Microsoft.NET*, Kathleen Dollard, Apress, 2003