Moving to Software Factories

Jack Greenfield and Keith Short, Architects, Visual Studio Team System, Microsoft Corporation.

Industrializing Software Development

Total global demand for software will grow by an order of magnitude over the next decade, driven by new forces in the global economy like the growing role of software in social infrastructure, by new application types like business integration and medical informatics, and by new platform technologies like web services, mobile devices and smart appliances. Without comparable increases in productivity, total software development capacity seems destined to fall far short of total demand by the end of the decade. What will change to provide the massive increase in capacity required to meet demand? It is not likely to come from adding developers. Instead, software development methods and practices will have to change dramatically to make developers much more productive.

Other industries multiplied their capacity by moving from craftsmanship, where whole products are created from scratch by individuals or small teams, to manufacturing, where a wide range of product variants is rapidly assembled from reusable components created by multiple suppliers, and where machines automate rote or menial tasks. They standardized processes, designs and packaging, using product lines to facilitate systematic reuse, and supply chains to distribute cost and risk.

What will industrialization look like in the software industry? We cannot know with certainty until it happens, of course, but we can make educated guesses based on the way the software industry has evolved, and on what industrialization looks like in other industries. The key is to leverage experienced developers by encapsulating their knowledge as reusable assets that others can apply. Patterns demonstrate limited but effective knowledge reuse. The next step is to move from documentation to automation, using languages, frameworks and tools to automate more of the software life cycle.

Automating Software Development

Can we automate software development? Of course, we can, and we have already. Widget frameworks and WYSIWYG editors, for example, make it easier to build and maintain graphical user interfaces, providing benefits like device independence and visual assembly. Database design offers similar forms of automation. Looking closely at how this was done, we can see a recurring pattern.

- After developing a number of systems in a given problem domain, we identify a set of reusable abstractions for that domain, and then we document a set of patterns for using those abstractions.
- We then develop a run time, such as a framework or server, to codify the abstractions and patterns. This lets us build systems in the domain by instantiating, adapting, configuring and assembling runtime components.
- We then define a language for the domain and build tools that support the language, such as editors, compilers and debuggers, to automate the assembly process. This helps us respond faster to changing requirements, since part of the implementation is generated, and can be easily changed.

The Role of Models

Raising the level of abstraction for developers using higher level languages, such as modeling or visual assembly languages, is one of the key elements of this pattern. We are using UML for documentation, but we are using models based on small, focused, domain specific languages (DSLs) for automation. DSL based tools can help developers define and assemble components, such as web services, generate their implementations using framework completion, and capture metadata used to automate validation, packaging, deployment, configuration management, test generation, defect tracking and many other aspects of the software life cycle. We are using high fidelity DSL based models as first class software development artifacts.

Is that Model Driven Architecture (MDA)? No, not quite. Like MDA, we are interested in models. However, we are less concerned with portability and platform independence than MDA, and more concerned with productivity. While stereotypes and tags can be used to decorate UML models, experience shows that more precise language features are required to support compilation, debugging, testing and other development tasks. Unlike MDA, we do not propose to use UML where programmatic manipulation of models is a key requirement. We use UML for discussion, sketching diagrams on whiteboards and napkins, using the UML static structure and sequence chart notations that are now almost universally recognized by developers.

Not Just Models

While models play an important role, they are not the whole solution. Scaling up to higher levels of productivity will require the ability to rapidly configure, adapt and assemble independently developed, self-describing, location independent components to produce families of similar but distinct systems. It will require a transition from craftsmanship to manufacturing like the ones we have seen in other industries, and it will eventually produce more advanced earmarks of industrialization, such as supply chains, value chain integration, and mass customization. This vision cannot be achieved with models alone. Domain specific patterns, frameworks, and tools must also be developed and applied systematically, and must be aligned with both the product architecture and the life cycle process.

Also, we must address the processes by which we analyze requirements, develop software and deploy it to our datacenters. While prescriptive methods optimize for complexity not change, modern agile methods optimize for change not complexity. To scale agile methods requires the ready availability of best practices, reusable content and patterns. To ensure prescriptive methods are not overly rigid requires flexibility and variability in their description.

We are using a methodology, based on these ideas, called software factories.

Software Factories

A software factory is a product line that configures extensible development tools like Microsoft Visual Studio Team System (VSTS) with packaged content and guidance, carefully designed for building specific kinds of applications. A software factory

contains three key ideas: a software factory schema, a software factory template and an extensible development environment:

- Think of the software factory schema as a recipe. It lists ingredients, like projects, source code directories, SQL files and configuration files, and explains how they should be combined to create the product. It specifies which DSLs should be used and describes how models based on these DSLs can be transformed into code and other artifacts, or into other models. It describes the product line architecture, and key relationships between components and frameworks of which it is comprised.
- The software factory template is like a bag of groceries containing the ingredients listed in the recipe. It provides the patterns, guidance, templates, frameworks, samples, custom tools such as DSL visual editing tools, scripts, XSDs, style sheets, and other ingredients used to build the product.
- An extensible development environment such as VSTS is like the kitchen where the meal is cooked. When configured with the software factory template, VSTS becomes a software factory for the product family.

To press this analogy further, the products are like meals served by a restaurant. Software factory stakeholders are like customers who order meals from the menu. A product specification is like a specific meal order. The product developers are like cooks who prepare the meals described by the orders, and who may modify meal definitions, or prepare meals outside the menu. The product line developers are like chefs who decide what will appear on the menu, and what ingredients, processes, and kitchen equipment will be used to prepare them.

An Example

For example, we might design a software factory schema for building thin client eCommerce applications using the Microsoft .NET Framework, C#, the Microsoft Business Framework (MBF), Microsoft SQL Server Yukon, Microsoft BizTalk Server and the Microsoft Host Integration Server – a broad but useful family of applications. We might use this software factory schema to configure VSTS to become a software factory to build members of this family.

The software schema might contain a DSL to represent configurable requirements (perhaps using feature models), a DSL for describing business processes, a DSL for describing logical and physical data models, a DSL to describe Web page navigation, a DSL to describe interactions between Web services, a DSL to describe logical deployment policies, and a DSL to describe business entities. These DSLs have transformations and constraints defined between them, such as rules which describe how Web services map to source code structures in ASP.NET, or rules by which the mapping from Web services to business process activities should be reconciled. Architectural features of the family, such as 3-layer detailing for service implementations would also be specified. The software factory schema also describes the common features of products in the family, and specifies the variations permitted, and the effects of configuring variable features on the architectural elements of the family.

We would use the extensibility features of VSTS to host a software factory template based on this software factory schema. Included in the software factory template would be content from Microsoft Patterns and Practices group such as the User Interface Block for building navigable user interface flows and other software frameworks; Team Foundation Server policies for check-ins, project structures and work item descriptions and workflows; Visual Studio Enterprise templates that define policies for project structures; Microsoft Solutions Framework (MSF) micro-processes and guidance; custom extensions to Visual Studio Team Architect tools for special web service contracts and custom datacenter deployment configurations suitable for hosting large-scale eCommerce applications.

Equipped with this software factory, a development team could rapidly punch out a variety of eCommerce applications, each containing unique features based on the unique requirements of specific customers. The team would configure the architecture, software components and frameworks using configuration mechanisms for the variable features described in the software factory schema. Additionally, this factory could be used to create an ecosystem by making it available to third parties, who could extend it to rapidly build eCommerce applications incorporating their value-added extensions.

Conclusion

We realize that the term software factory is controversial – conjuring up for some visions of mindless automatons stamping out applications with all creativity in the process removed, or perhaps of previously unrealized visions of application development without programmers. On the contrary, the key to meeting demand on an industrial scale is to stop wasting the talents of skilled developers on rote and menial tasks. We must make better use of these few but valuable resources than expending them on the construction of end products that will require replacement when the next major platform release appears, or when changing market conditions make business requirements change, which ever comes first. We need a way to leverage their skills and talents by asking them to encapsulate their knowledge as reusable assets that others can apply.

Software factories are possible today. They represent an attempt to learn from other industries facing similar problems, and apply select patterns of automation to existing manual development tasks. Software factories exploit recent advances in software product line practices, component specification and orchestration techniques, domain specific higher level languages and extensible development environments like VSTS. Software factories make it faster, cheaper and easier to build applications – even for applications that are not initially perceived as being part of a family. Remember, ongoing maintenance of an application is akin to producing successive applications in a family. Instead of having to build a whole application from scratch, developers using a software factory build only the parts where the application differs from other members of the family. The rest is provided by the software factory.

For further information, please see the book we have written called *Software Factories: Assembling Applications using Patterns, Models, Frameworks and Tools,* published by John Wiley, and check out other references available from **http://msdn.microsoft.com/architecture/overview/softwarefactories.**

For further information on Visual Studio Team System, see http://msdn.microsoft.com/vstudio/teamsystem.