# A Foundation for the Pillars of Software Factories

**Tom Fuller**
**Blue Arch Solutions Inc., Tampa, FL., United States**

**Abstract**

The complexity involved in designing and developing solutions has increased dramatically over the past 30 years. As your application portfolio evolves, there are process strategies that can help your organization overcome the problems that plague software development today. Promoting reusability by adopting production line methodologies will ensure broader success of the systems delivered using these processes. Reusability rarely happens by accident, and using strategic processes like architecture-driven development will make discovery of reusable components an intentional step as opposed to an opportunistic one.

## 1. Introduction

The most valuable artifacts that any architect can produce are those that can be applied across numerous problem domains. This versatility is why patterns, frameworks, guidelines, reference models, and automation tools are core deliverables from any process iteration. Delivery strategies that focus on architecture must extract and apply proven techniques for solving challenging application problems. Architects will typically be embedded at the project level but remain knowledgeable of enterprise scope. Over time, the collection of these extracted best practices forms a library for other teams to use as they compose their solutions.

It is immediately evident that the software-factory pillars and the delivery goals of an architecture-driven process are in sync. Using standards-based deliverables, like software-factory schemas and pattern languages, to group and describe your enterprise architecture components can take your enterprise architecture to the next level. Capturing reusable artifacts with these templates gives your organization a consistent way to deliver reusability. The methodology will then extend beyond the construction and delivery of the product and focus on post-delivery return on investment measurement, education, and the long-term road map for these deliverables.

Introducing architectural guidance within any software development life cycle will improve the overall effectiveness. It is this process and delivery shift that is at the heart of the software manufacturing revolution. This article explains how to use an architecture-driven process and the software-factory pillars to change how you structure teams and deliver solutions. The goals here are to explain what it means to use that process and what type of expected deliverables result from each transition phase of the software development life cycle. Additionally, the article will explain the benefits of intentional discovery, implementation, and measurement of reusable architecture.

## 2. Managing Increased Complexity

Anyone who has worked in the software industry will tell you that developing business productivity software is very challenging. Often, companies become overwhelmed by the lack of consistency and bloated costs that typical software cycles create. There are four syndromes that contribute to this increased difficulty:

- *The moving-target syndrome:* This syndrome is an unavoidable aspect in the software industry. There will always be a constant evolution of frameworks, patterns, strategies, and technologies. This continuum will often make your effective solution today an ineffective one at some point in the future.
- *The perfect-storm syndrome:* It is always difficult for any company to find the right number of

developers, at the right time, with the appropriate vision and budget. If any one of these items is not correct the solution will suffer and can potentially become ineffective.

- *The Goldilocks syndrome:* One of the joys in solving logical problems is discovering the most innovative and efficient solution for a technical problem. However, this delivery rarely considers the right solution based on cost. As a result, many solutions are overengineered or underengineered. Very often this engineering is a matter of perspective, and striking a balance between the best solution and the *right* solution is challenging.
- *Grandpa's favorite-chair syndrome:* It is in our nature to gravitate toward things that we understand. That is why software is often architected to avoid as much change as possible. The limited risk that comes from using code that has already been proven to work is immeasurable. This approach will result in solutions that are added to in unnatural ways.

Adding to this complexity is the migration from consolidated, monolithic applications to highly-scalable, distributed systems. As the strategy has changed (see Figure 1), so has the focus. Today we need to find an even higher level of abstraction than objects or components. It is based on this need that we introduced repeatable architecture patterns and frameworks. The all-important transition to an architecture-driven process is a catalyst for the discovery and publication of these reusable artifacts.

Software is not the first industry to see its output increase in complexity over time. It only takes a moment to consider how much more complicated current automotive or construction deliverables are now versus 50 years ago. How have these industries managed to meet the high demand for their increasingly complex product? The answer lies in the shift away from pure craftsmanship and toward manufacturing in both instances. By capturing and repeating best practices for building well-known aspects, the limited resources available for development are able to focus on those things that are truly unique.

Based on the preceding information, it is clear that the software industry is faced with a similar dilemma. The complexity has increased, and at the same time the demand has skyrocketed. The concept of software manufacturing is not a new idea. In fact, there are many well-respected software engineers that have been giving it thought since the late 1970s. However, until recently most of the frameworks, patterns, and strategies were still very immature.

We stand at the cusp of a major revolution in how business productivity software is delivered. As businesses begin to use architecture to drive their delivery of software they will find ways to isolate consistent portions of their enterprise applications and capture them in a way that can be reapplied through automation. These concepts provide the best chance to date to help deliver quality software while managing all of the inherent complexity that comes with it.

## 3. Defining the Vision/Scope for All Process Iterations

One of the first steps for any organization interested in moving to an architecture-driven process is to define the vision and scope for the enterprise architecture. Without this definition, it becomes almost impossible for solution architects to make good decisions about where and when to introduce architectural patterns. You might think that service-oriented architecture (SOA) is an example of the vision for your enterprise architecture. In reality, SOA is an example of a delivery strategy that can help you to adhere to the vision for your enterprise architecture.

A vision statement should be concise and devoid of any implementation biases. The vision statement should "paint a picture" of where the architecture team wants the applications to evolve to. Here are some examples that could be used to help focus the architecture team as solutions are being delivered using architecture:

- All application deliverables will focus on quality through embracing and extending proven enterprise architecture artifacts. Over time, new solutions should be built completely through composition and customization of enterprise architecture frameworks.
- All application deliverables will efficiently use resources within the enterprise infrastructure to solve business productivity demands. Using tailored tools and processes, 75 percent of a

custom application will be constructed, tested, and deployed automatically.

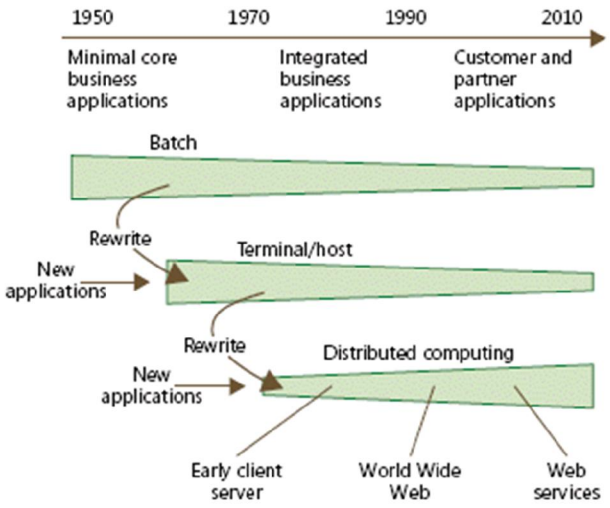


Figure 1. Time-phased trending in application architecture patterns

The scope of your enterprise architecture is separate from the vision. What needs to be considered when defining the scope of the enterprise architecture is whether or not the practices and patterns that are being managed by the solution architects are within specific technology disciplines, business areas, or application styles. The broader the scope the more challenging it is to manage complexity. However, if the scope is too narrow, you will risk decreasing the impact your architecture patterns can have. The best way to manage scope is to determine a grouping strategy based on variation. For example, the strategies and patterns of the engineering and infrastructure group may be very different than those of the application-delivery group. As long as there is a shared strategy for how to consistently apply those patterns, managing them separately is acceptable, which starts to show the reason vision and scope for enterprise architecture are so important.

In most, if not all, organizations it takes a combination of people with different spheres of concern to deliver applications successfully. If the burden of managing the architecture vision and scope is not shared, then the leadership and direction will become fragmented and inconsistent. Other strategists in the organization will spend less time debating issues if synergies are found between the various scoping groups. This synergy is without a doubt one of the most important steps in starting to build architecturally sound applications.

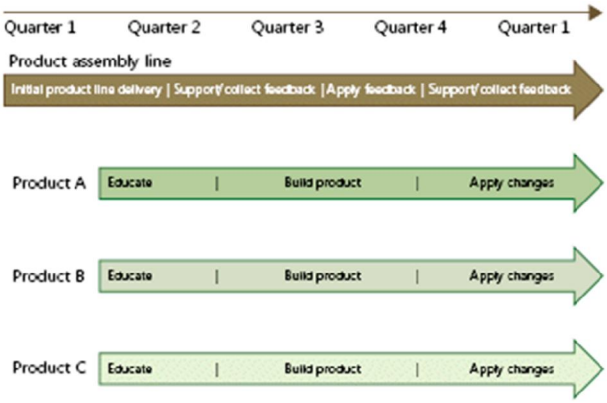


Figure 2. The product line development approach

## 4. Building Your Assembly Line

An architecture-driven process is focused primarily on shifting the control for delivering solutions to the architecture team. Specifically, the solution architect that is embedded in the delivery team will be responsible for determining how quickly an application can move through the phases of development. This determination is primarily in an effort to work on strategically delivering applications more efficiently in the future. So how then do solution architects "prove their worth" within each of these iterations? This value is where software factories and production line delivery of applications is key. The architecture team is working constantly to construct and improve on the software assembly line within one of the predefined enterprise architecture scope groups.

The key components of a software factory or product line are all focused on one key goal: abstract those portions of the application that do not vary, and guide the creation of variants by using pragmatic constraints. Here are the four core pillars of the software-factory initiative:

1. *Software product lines:* Architects must focus on how to find those portions of an application that can be abstracted because they are consistent. Once they are discovered they should be delivered ahead of the products that will use them. This approach promotes an intentional step of finding and delivering reusability. In all

likelihood these assets will fall in line with the scoping groups defined for your enterprise architecture.

2. *Guidance in context:* There are a couple of levels of variability when it comes to components within the software product line. One level is that which can be automatically built "hands free." These components are usually very low level and require essentially no decision making by the product-delivery team. The next level is that which can vary in a controlled way, which is where guidance comes in. When a product developer can choose from a set of constraints to build an application component from a finite set of variations, the software factory should support that. This variability is not the whole picture though. Also consider the "in context" portion of this process. When you can provide context-sensitive guidance, there are benefits to be gained from providing something as simple as tailored help!
3. *Architecture frameworks:* Frameworks (often described using a factory schema) within your software factory provide a way to group all of the building blocks that will be used by the product developers. There are a number of components in any software deliverable that are potentially reusable. The framework will capture and deliver best practices in an effective way.
4. *Model-driven development (MDD):* Models provide a mechanism for representing complicated software components using visual abstractions. This mechanism typically helps simplify the design, development, and support of those components. Making models a critical component of your software factory requires you to think differently about design documentation. Models must always reflect the current running code if they are to remain useful in the support and maintenance of software. Historically that has not been the case, and MDD is an effort to fix that shortcoming.

As an enterprise architecture proponent, each of these pillars is critically important. Every product that is delivered will look to extend and/or consume numerous portions of the software factory. As a solution architect these are the tools you bring to the table to help drive every product toward well-established best practices. Without these tools every product has to be built from scratch. This style is often referred to as "one off" and is considered very inefficient.

As an enterprise matures in its architecture-driven practices so do the components of the software factory. Collecting new enhancements and delivering new versions of the factory components facilitates perpetual innovation through the architecture team. The hardest part about transitioning toward an architecture-driven process and a software-factory approach is starting. We discussed previously the scope definition for the enterprise architecture. Scope continues to be a basic element of concern for all software-factory deliverables. In the case of business productivity applications economies of scope and scale can both be achieved:

- *Economies of scope:* To justify the cost of an architecture component it has to be useful to a number of products. One strategy for identifying components that exhibit economies of scope is to group products based on implementation style. For example, if your applications will be built using a distributed model focusing on Web services across tiers, then it will be beneficial to build a factory component that can help guide the delivery of a Web service. Very often these stereotypes are common across every application. Another possible economy of scope benefit is the delivery of enterprise services. If a large number of applications depend on the same data or business subprocesses, then it may be an ideal candidate for a services-based approach.
- *Economies of scale:* It is very rare to find circumstances where business productivity applications can benefit from economies of scale. If a factory component is going to have an economy of scale your organization will need to benefit from that component being created the same way multiple times. An example could be enterprise data dictionaries that are made available as part of every new data dictionary. If the component can be reapplied "as is," then it can be said to exhibit economies of scale.

Even when a solution architect is in control of the progress of product delivery, it is still challenging to find reusable

factory components. This difficulty is especially true when you consider cost justification. A product line methodology is required to discover factory components with broad enough scope. When a factory product line phase is introduced, the building blocks can be built, and the architecture team can get ahead of the delivery curve. Without this methodology, there are always difficult cost/benefit decisions that determine the architectural direction.

What eventually forms are two separate delivery teams focused on completely different aspects of software. One team is responsible for envisioning, designing, delivering, and training of reusable assets in the scoping group. The other team(s) is responsible for learning, consuming, and giving feedback on those components. The most important item to note in Figure 2 is that the product assembly line team must be given the opportunity to get ahead of the product delivery teams. Once established, it becomes easier to drive work through the feedback loop that will ensue. If the teams begin too much work without a clear understanding of the foundation then it will be impossible to avoid one-off project development.

This concept of grouping applications based on commonality into product groups or families is a critical step in moving toward an architecture-driven process and software-factory development. These product families should have dedicated architecture resources and common infrastructure. These families can also help provide consistency in the patterns that are discovered and applied in an enterprise. Typically, the context and forces will remain consistent within a product group. Architects are able to benefit from the lessons learned when applying these solutions to similar application contexts. This consistency translates into effectiveness within the architecture group.

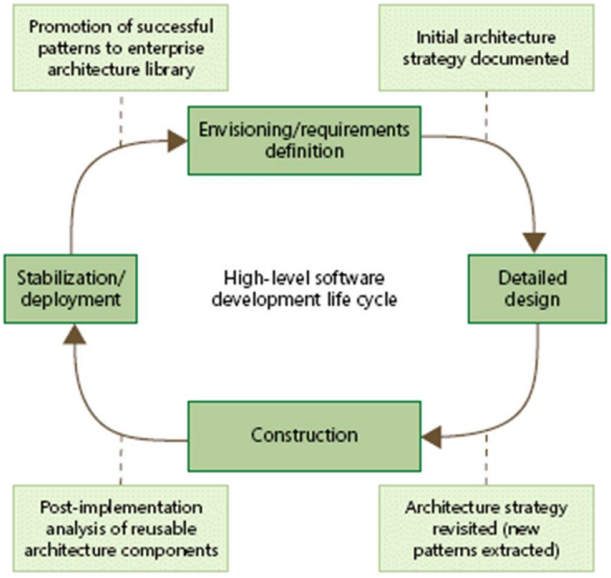


Figure 3. Architecture-driven processes require architecture-focused deliverables during every transition.

It is impossible to talk about processes or reusability without mentioning agility. Does driving delivery through architecture create an agile process? In the traditional sense, an architecture-driven process would probably not be considered an agile process. However, if you are looking at the fundamental goal of an agile process, architecture-driven processes and software factories do translate into higher productivity and adaptability levels. Based on this fact, it is safe to say that driving a process with architecture can foster agility in application delivery.

Always remember that developing a technical solution is simply a series of refinements and abstractions. Depending on your frame of reference you may be attempting to decompose or refine a business problem, or you may be trying to design higher-level abstractions to demonstrate what low-level machine instructions should do with user input. Developers and analysts will always struggle to find the sweet spot when it comes to abstraction and refinement. Too much refinement and time is wasted; too much abstraction and complexity is increased—another application of the Goldilocks syndrome.

In the end, a business productivity application needs to be able to respond to change. Rarely does a product team know the final picture of a solution during the initial release of an application. Embracing change through

planning for iterations is paramount to the success of any delivery process. Architecture-driven processes naturally move the product-family delivery team toward iterative development by introducing a feedback loop for the factory components.

Less code typically results in higher productivity. Architecture-driven processes and software factories are built on the concept of abstracting what is already known and guiding what varies, which is a powerful strategy in managing complexity. Consistent and reusable software combined with an up-front understanding that an application will change is what being agile is all about!

## 5. Prerequisites for an Architecture-Driven Process

There are a number of prerequisites before you can begin an architecture-driven process, some have been mentioned previously, and others are listed here as a sort of readiness checklist:

- A role on the project must be focused on the application architecture. This role requires knowledge of existing enterprise architecture patterns and being dedicated to helping deliver applications that meet an enterprise architecture vision.
- A long-term vision for architecture of the enterprise should be established. This vision will help to simplify decision making and support making good decisions that will integrate well with the overall enterprise vision. This vision is articulated through enterprise architecture road maps, business capability matrices, and enterprise framework maturity models.
- Well-defined deliverables must be established for each of the transitions. These can include architecture strategy documents, pattern templates, reusable component analysis reports, and an enterprise architecture library.
- There must be agreement from the project sponsors that transitions among phases cannot take place if the agreed-upon architecture deliverables are not complete. Otherwise, the drive toward expedient delivery can often short-circuit any architecture-driven effort.
- An agreed-upon strategy for resolving architectural anomalies should exist before starting. This agreement will help to mitigate the risk of becoming paralyzed by any lack of "buy in" for the enterprise architecture initiatives. Additionally, this risk-aware approach will help avoid application teams succumbing to antipatterns to meet expedient delivery demands.

Once these prerequisites have been satisfied you can safely begin delivering architecture-focused applications. The tactical introduction of architecture deliverables will ensure that you take the time to proactively build or consume reusable architectural components.

As applications are delivered in any organization, some type of delivery process must be followed. The high-level steps shown in Figure 3 represent those that are commonly found in all development life cycles. It is in the transitions between these phases where architecture should become a focus, and it is this focus that will help your enterprise transition away from one-off and *siloed* application delivery and toward the cohesive development of applications that adhere to an enterprise architecture vision.

*Transition 1 (envisioning to detailed design):* In this early phase of the project it is critical to start looking for already existing architectural assets (patterns, services, framework components, and guidelines) that can be consumed by the new application. This search will bring to the surface questions about availability, performance, and maturity of these existing components. As an architect, the focus on reuse should help to drive the initial architecture strategy documentation. Delivering a plan that helps the application deliver a high-quality application that leverages as much of the existing enterprise architecture as possible is how you measure your success. Deliverables might include:

- Architecture strategy overview: an enterprise architecture component consumption report, expected enterprise architecture variants, and planned architecture pattern usage report
- Service-level change requests for existing components
- Recommendations for new enterprise architecture components

*Transition 2 (detailed design to construction):* Once the initial strategy is in place the detailed design can be built to realize the high-level architecture vision. Through a series of refinements, the architecture strategy is either adopted or modified based on the context in which it is applied, which is where the architect role on the project becomes amplified. Helping the project team to make practical decisions about the modifications to the enterprise architecture standards should be his or her focus. In most cases the existing standards should be used as is to avoid inconsistency and added complexity.

It is not always possible to use the existing architecture assets in their current state, and revision requests are sure to be needed. There are also new architectural patterns and styles that can be discovered during this phase. Key deliverables during this transition include architectural component change requests, a new architectural pattern definition, recommendations for new enterprise architecture components, and a revised architecture strategy overview.

*Transition 3 (construction to stabilization):* Once the application has been built, the focus shifts to quality and post-implementation analysis. To continue to gain widespread acceptance of the architecture patterns being applied, the successes and failures should be documented and communicated to the sponsors of the application. As the patterns being used mature, the likelihood of failures diminishes. The goal here is to show how much time was saved and how much quality was introduced by the focus on architecture. Deliverables include a reusable, assets-consumed overview; a change/extension cost analysis; and an existing asset improvement report.

*Transition 4 (stabilization to next iteration):* An effort to communicate best-of-breed solutions back to your company is a critical part of being focused on improvement. Once the application has stabilized, the architect should go through an exercise that helps to educate the enterprise to the new patterns and antipatterns discovered during this application's life cycle. Promoting best practices and cross-training other project teams are the only ways to ensure perpetual knowledge growth in your organization. The key deliverables in this phase are mainly for enterprise education, and they include a revised, enterprise architecture library catalog; a cross-team, architectural best practices session; and training materials for new architectural patterns.

## 6. Taking the Next Step Toward Industrialization

Once an organization has practiced architecture-driven delivery and considers the process mature, there are steps that can be taken to automate the delivery of applications. All of the architectural components that are delivered as part of the product line are ideal candidates for automation. These components will quickly go through a number of iterations and their consistency will become clearer as the product line team has to adapt to product-level variations.

Not until you understand these variations and establish the constraints should you remove the code completely from the product developers. Understanding what to abstract and automate is complicated and requires some level of trial and error. Many code-generation techniques appear on the surface to be beneficial, but always remember that a developer's confidence in generated code can be lost in an instant. Managing this perception requires a partnership with product developers through some process iterations and guided adoption of best practices.

This next step requires sophisticated tool support. These tools must be capable of providing an open API for developing model-driven tools, integrated wizards for guidance, and context-based capabilities to seamlessly incorporate architecture best practices into the product developers' workspace. These tools have the ability to change the productivity levels of application developers immensely.

Over time the percentage of well-known application components will increase and in parallel so will the automation benefit. Architects will always be attracted to automation for architecture components, but be careful to first understand what it is that you are automating. Most modern tools do a great job at creating separation between automated tool output and custom code, but that separation does not fix the issue of perception or accuracy.

To maintain momentum during product delivery you cannot solve the same problems over and over. The cure comes in the form of creating and guiding consumption of

codified architectural best practices. Adopting a process that is driven by architecture will shift application delivery into a more proactive "prepare for the future" mindset. With the increased complexity of and demand for business productivity applications comes a need to transition away from one-off development. Most, if not all, engineering disciplines have learned this lesson and will mimic the success of previous iterations. The application development world is no different. An architecture-driven process will facilitate the collection of the key building blocks of productivity.

The seemingly unattainable goals of software industrialization are quickly becoming a reality. The software factories movement makes great strides in giving companies a way to measure the improvement or value proposition for architecture. Once a repeatable artifact is discovered, its *replayability* and *variability* have to be evaluated before absorbing the cost of building a guidance package or a designer. Discovering consistency, increasing productivity, and embracing change are the backbone of agility in application development. Combined, architecture-driven delivery and software factories will help lead us toward the next generation of software development.

## References

[1]Cusumano, Michael. Japan's Software Factories: A Challenge to U.S. Management. New York, NY: Oxford University Press, 1991.
[2]Wegner, Peter. Research Directions in Software Technology, "Conference Proceedings, 3rd International Conference on Software Engineering." Cambridge, MA: MIT Press, 1978.
[3]Greenfield, Jack, Keith Short, Steve Cook, and Stuart Kent. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Indianapolis, IN: Wiley, 2004.
[4]Clements, Paul, and Linda Northrop. Software Product Lines. Boston, MA: Addison-Wesley Professional, 2002.

**Tom Fuller** is CTO and senior SOA consultant at Blue Arch Solutions Inc., and an architectural consulting, training, and solution provider based in Tampa, FL. He is also the current president of the Tampa Bay chapter of the International Association of Software Architects (IASA), holds a MCSD.NET certification, and manages a community site dedicated to SOA, Web services, and Windows Communication Foundation (formerly "Indigo").