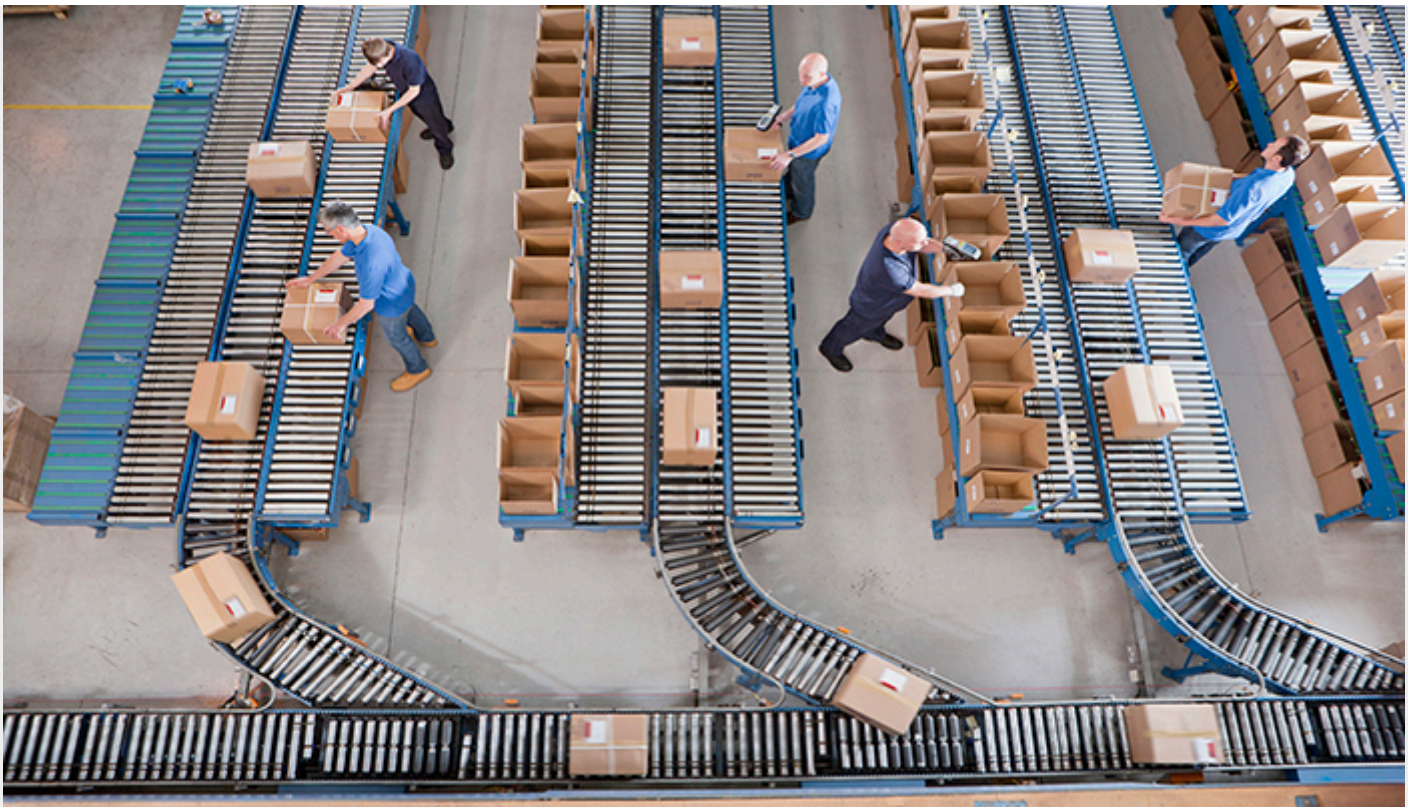
HOW & WHY TO BECOME A SOFTWARE FACTORY



When it comes to software development, you've likely already evolved from Waterfall approaches to more modern, DevOps-based approaches.

But an even more mature approach to software development is the software factory. Initially a lofty goal, becoming a software factory is something that companies across all industries are considering as a means to getting quality software to market sooner.

Is your company pushing streamlined, clean software on a regular basis? If not, becoming a software factory—and embracing the factory's core concepts of AI and machine learning—is something to consider.

So, let's take a look at the software factory concept.

What is a software factory?

Companies like Google and Netflix have set the gold standard in software development, with many updates and releases pushed daily in order to fix bugs, strengthen code, introduce new features, and handle unpredictable scaling.

This factory approach to software—churning out new software quickly, easily, and frequently—is called a software factory. Software factories roll out high-quality products and features, using lean code, that quickly enhance your business. (Software factories can closely relate to SAFe environments, too.)

A software factory relies on reducing the amount of interaction from developers so they can focus on higher-level technical challenges within the organization, such as:

- Monitoring and maintaining the automated framework
- Ensuring that enterprise data is secured

Today, companies across all industries are trying to become more like these leading tech companies. Essential to this approach are:

- A true DevOps environment, where software development and IT operations collaborate harmoniously
- Right mindset and company culture
- Skills
- Creativity

With these components in place, you'll next look to automation. Automation is essential to creating a software development process that works much like an assembly line of software creation—hence, a software factory. Automation can apply to a number of dev practices, like continuous integration/delivery (CI/CD) and automated testing.

Typically, a software factory consists of proprietary tools, processes, and components packaged together. This package offers templates and code that you can easily arrange and process to create a program quickly—with little original code required. Of course, software engineers still must interact with the product to ensure it does what it is supposed to do and doesn't have bugs or other issues, but when you create or update an app quicker, you have time to shift your testing left.

Why businesses encourage "software factory" mentality

A well-functioning software factory implies a well-functioning internal development team that works hard on shared goals with operations team members, creating features that impact the entire business unit. This harmonious environment is conducive to:

- Higher levels of satisfaction and success
- Better technology utilization
- Fast communication of information, resulting in fast decision making

All and all, implementing a software factory with machine learning and artificial intelligence helps enterprise businesses achieve this goal.

Deepak Seth's article in CIO—How's the 'software factory' going?—calls software factory the "holy grail" of enterprise software development performance. Seth suggests that, with QA teams "squeezed to the breaking point," smart automation is the future because it reduces the burden on teams responsible for testing applications.

In addition to creating a better work environment for dev teams overwhelmed by high software production goals, creating a software factory is the most efficient use of enterprise resources because it ensures that:

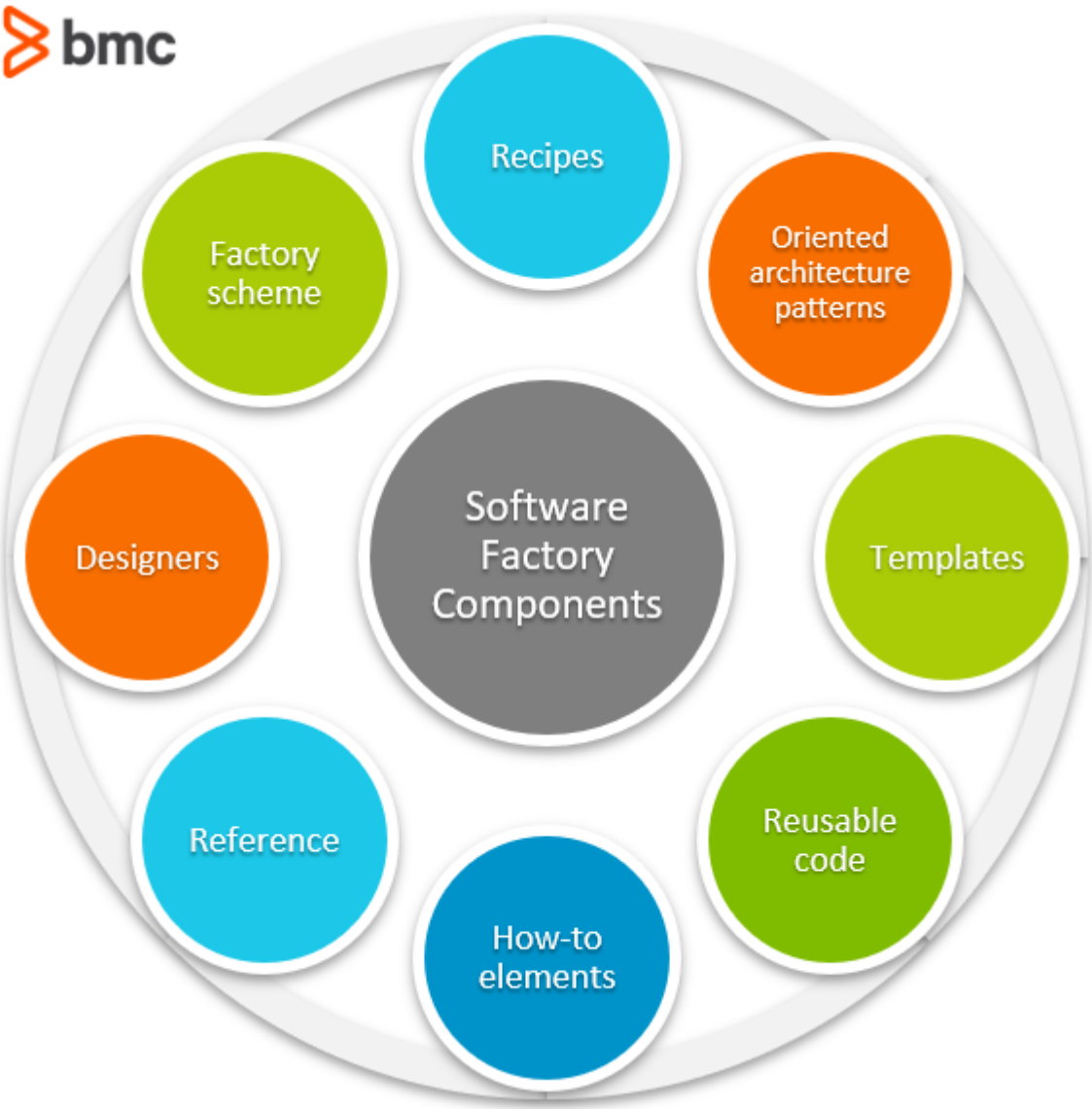
- Learning occurs before the next round of software.
- Any learned methods are applied in future builds.

Continuous improvement is a foundational pillar of DevOps. When you deploy smart automation to

accomplish software factory goals, teams deliver on a commitment to always improve and provide faster, more comprehensive services and upgrades to customers when and where they need them.

Components of software factories

These components comprise a software factory:

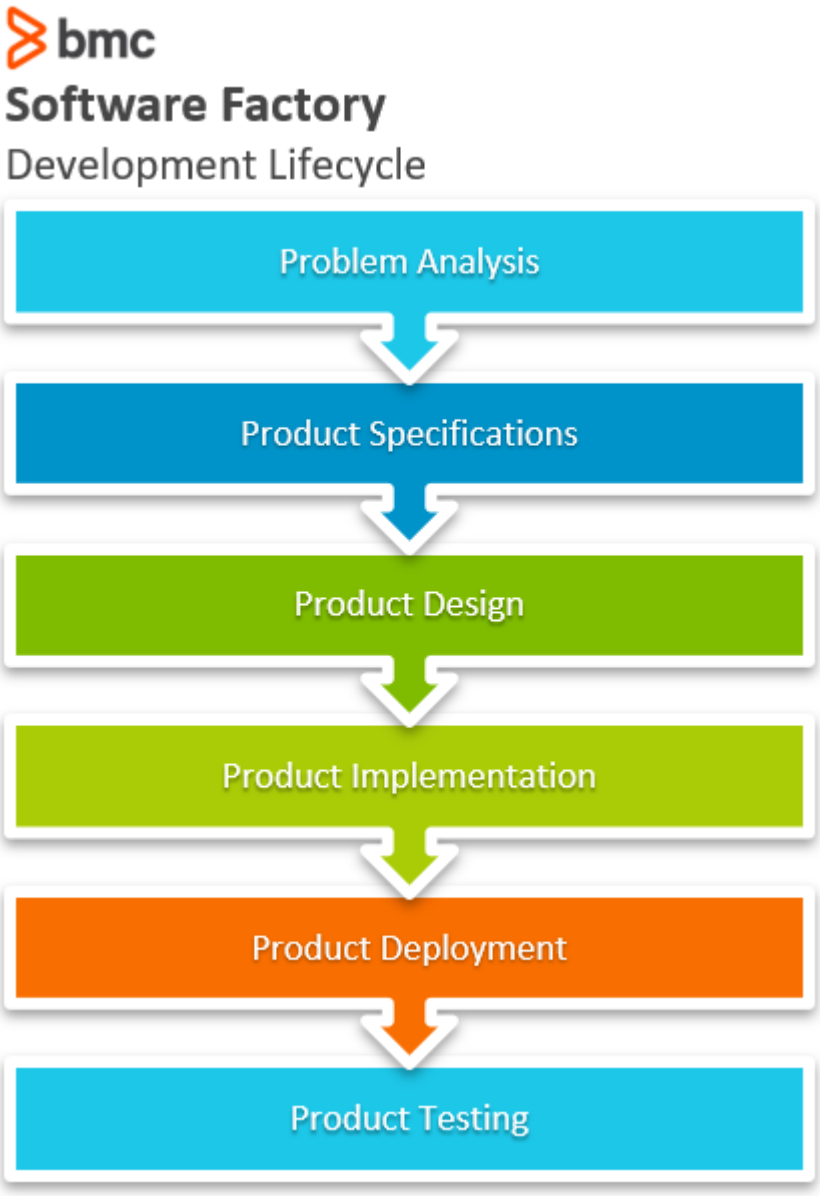


- **Recipes.** Automated processes to perform routine tasks with little or no regular interaction from the developer.
- **Oriented architecture patterns.** These patterns define how the design will be executed in the application and why those choices were made.
- **Templates.** Prefabricated application elements, code, and development features, including placeholders for arguments. Used early in a project, templates establish a consistent framework and configurations.
- **Reusable code.** Reusable components that implement common mechanisms or routine functions. These are used for creating elements throughout the project that would otherwise be wholly manual.
- **How-to elements.** These informational elements are a development resource for those getting started with the software factory.
- **Reference implementation.** AKA an example of realistic product completion.

- **Designers.** Tools that aid developers in more complex design hierarchies.
- **Factory scheme.** This map defies hierarchies and serves as a basis for project development.

Software factory development lifecycle

Below is the product development lifecycle:



1. Problem Analysis

First, determine whether the product scope makes sense for a software factory and can be implemented successfully. Understand two key components:

- Which parts of the product can be successfully implemented using automation
- Where manual development is required

2. Product Specifications

Next, define the scope. Using machine learning the software factory will compare product specifications against previous products to draw inferences about the most efficient development

strategies that can be automated.

3. Product Design

Then, automated programs can map differences between two designs and update based on changes in scope.

4. Product Implementation

The various mechanisms that can be used to develop the implementation depends on the extent of the differences in implementation between existing products.

5. Product Deployment

Deploy or reuse existing constraints for default deployment and configuration of the resources required to install and execute the product.

6. Product Testing

Finally, smart automation can create and reuse testing components (such as test cases, data sets, and scripts) and implement instrumentation and measurement tools that offer important data output.

Software Factory best practices

Follow these tips for more success, sooner, in your software factory.

- **Avoid tool redundancy.** With templates and tools available across agencies and departments, ensure that a sprawl of the same tool doesn't exist within the organization.
- **Ensure vendor management protocols are in place.** Using services that allow you to create and automate the creation of DevOps software often means working with more than one service provider. Comprehensive vendor management is required to ensure user access, security, billing considerations, service considerations, and other elements of the relationship meet your enterprise standards.
- **Diversify your dev skills.** With automation in place, developers can spend more time on the product itself, including exploring new languages or approaches. With time on their hands, they can look at the best way to get a feature or improvement done—instead of using only what's worked in the past. As such, look for devs who know a variety of languages. In a software factory, there's no need to commit to one programming language or framework only.

Related reading

- BMC DevOps Blog
- DevOps Feedback Loops: An Introduction
- How to Write Test Cases for Software
- Low Code vs No Code in the Enterprise
- SRE vs DevOps: What's The Difference?, part of our multi-part DevOps Guide