Here are the other key elements of our SW Factory.

**Application lifecycle management**

Application lifecycle management (ALM) is at the core of our SW Factory. On the one hand, the ALM system serves as the main system of record that interrelates and provides the traceability and context across such lifecycle artifacts as backlog items to requirements, tests, defects, code commits, builds, and so on.

On the other hand, the ALM system leverages that information to provide insights we need to manage and optimize the lifecycle (e.g., to optimize backlog prioritization, realize which tests are required to cover a code change, determine where the risks and hot spots are, and so on). With the variety of disparate technologies and supporting tools we had in our portfolio, we needed an open, extensible ALM system that could easily integrate with our broader tool chain and support all of our tools.

To be agile at scale, we selected [the Scaled Agile Framework (SAFe)](https://techbeacon.com/app-dev-testing/4-biggest-challenges-moving-scaled-agile-framework-safe) as our overall operating framework. Different teams needed the flexibility to use their own flavors of agile, such as water-Scrum-fall, Scrum, or Kanban. So we needed a scalable, flexible ALM system that would enable operating SAFe with standard workflows for portfolio alignment (e.g., standard defect lifecycle workflow) while providing the flexibility each team needed to operate within its own optimal agile flavor.

Lastly, because quality is key to our business and customers, we looked for a tool that would let us balance agility and quality as we accelerated the pace of delivery and that would provide us with the visibility and agile quality management needed to enable that.

**The build factory**

Our build factory spans the application development lifecycle from when the developer pushes a code change from the integrated development environment (IDE) to the source-code management (SCM) system, which in turn triggers the continuous integration (CI) to run, executes the required tests, and pushes the build results into an artifact repository.

**The integrated development environment**

Our developers use an IDE as their primary interface, so we wanted all relevant developer use cases to be done directly within the IDE. You can achieve this by using IDE plugins that enable integration with your tool chain. In this way, your developers can commit code directly through the IDE (integrated with your SCM), manage the their backlog (ALM integration), validate their code quality and security while coding (integration with testing tools), and so on.

Our developers prefer various IDEs, depending on the programming languages they use and their own experience. Due to the relative strong attachment our developers have to their IDEs and the wide variety in use, we decided not to standardize on one, but to support the three most popular ones: Eclipse, Visual Studio, and IntelliJ.

**Source-code management system**

SCM is a key service because it holds your company's intellectual property. That's why it's critical that any SCM product meet your enterprise readiness (scalability, backup and restore, disaster recovery) and security requirements (ability to integrate with the corporate IDM, robust authorization management, audit trail, and so on).

We were looking for a tool that would also foster collaboration and inner sourcing across the portfolio to power agility, productivity, and innovation. With that in mind we adopted a Git solution, based on GitHub Enterprise, to serve as our foundation. We have adopted common Git practices for managing pull requests, code reviews, and code merges.

As part of our Git adoption we moved from SVN to Git for version control, which in turn triggered our transition from the monolithic repositories we had in SVN to smaller, more agile, microservices-oriented repositories in Git. This tool complements the SCM that we have in place.

**Continuous integration**

Our CI system ensures that our main code stream is continuously functional. While our teams have different policies, any code commit triggers a CI cycle that runs a short test cycle that checks for core issues with code changes while controlling the time required so as not to hold up the pipeline.

For us, it's all about balancing agility and risk. On top of the short test cycle, we also run a longer period cycle—a nightly run—that provides increased coverage of our tests. The combination of the two enables us to run fast, with "good enough" validation on the individual code commit level, while still protecting quality in our continuous testing practice.

CI is at the core of the build execution, and has many integrations: with the SCM system so that code commits trigger CI execution, with various testing tools to run the tests, with the ALM system to report all build and test results, and with the artifact repository to push build artifacts.

We standardized on Jenkins as our primary CI system. Many teams already used it, and its relatively rich plugin ecosystem was key to integrating it with our tool chain.

**Binary repository**

The binary repository is the place where all of your build intermediary artifacts and build artifacts become accessible. Because of the rich technology portfolio we have and the globally distributed organization in which we operate, we needed a system that would support rich artifact types such as Maven, NPM, and the Docker Registry; the ability to deliver a global deployment in a relative simple manner; and flexibility in delegating permissions to enable self-service to our consumers as much as possible and to avoid bottlenecks.

**Testing**

As part of our motivation to improve agility and quality, we extended testing across the entire application development lifecycle. We are shifting testing left in order to detect issues as early as possible, and we are doing continuous testing to balance agility and quality.

This means that as the code commit progresses in the delivery pipeline from software development to testing, staging, and production, validation is extended and risk reduced. We integrate functional, performance, and security testing with our CI to provide early detection, and with our ALM system to enable quality management and insights. By consolidating all of our data into the ALM system, we can detect risky areas that require more attention, which we address through manual tests and additional validation cycles.

**Functional testing**

We require a functional testing system capable of covering a broad set of technologies in our portfolio, including web, web services, mobile, Java, and .NET. We also aim to increase test automation coverage for our applications, but with increased test automation coverage comes a commensurate increase in the effort required to keep the tests up to date with application changes.

That makes it difficult to also increase the pace of application change delivery,as we extend test coverage. To overcome that, we need a tool that uses artificial intelligence and other techniques to help us keep our tests resilient while minimizing maintenance overhead due to application changes.

**Performance testing**

Also known as nonfunctional testing, [performance testing](https://techbeacon.com/app-dev-testing/web-performance-testing-top-12-free-open-source-tools-consider) includes concurrency testing, longevity testing, stress testing, and load testing. We needed a tool that supported the broad set of technologies. The system had to be flexible, able to shift left, and able to execute performance tests at scale, from tens to hundreds of users, as part of our CI. And we needed to load test with millions of concurrent users as the commit progressed in our delivery pipeline. Finally, we needed a tool that would quickly and accurately detect performance bottlenecks and their respective root causes so we could quickly fix them.

**Security testing**

We implemented a set of security testing practices that give us complete security coverage, including static code analysis, dynamic code analysis, third-party dependency checks using [OWASP Dependency Check](https://www.owasp.org/index.php/OWASP_Dependency_Check), container scanning with [Clair,](https://coreos.com/clair/)and infrastructure scanning.

We integrated all of these technologies as part of our CI in order to shift left and provide earlier detection. The results are being audited and governed through our security intelligence tool, and qualified security defects are integrated with our ALM system. This enables us to shorten security validation cycle time, reduce risk, and manage security as an integral part of the delivery cycle.

**Release management**

[Release management](https://techbeacon.com/enterprise-it/3-devops-techniques-stress-free-release-management) is the process by which our products and services become available to the market. Each of our product teams releases according to its own schedule and has its own form factor (product installation, virtual appliances, containers, etc.) and delivery model (product and SaaS). And in some cases our products span multiple form factors and delivery models.

We must coordinate each release across the portfolio with business stakeholders, such as the support group, which should be trained up front; the professional services group, which offers services on top; and so on.

We also defined a standard portfolio release management process that we call Product to Market (P2M). The process is lean in nature: It focuses on validating that the information and deliverables that each business stakeholder requires for a given release are ready and that our product teams have the flexibility to deliver in whatever way works best for them.

Our release management system is layered in the sense that it provides visibility, at the portfolio level into the readiness of each release, and at the product team level into release progress across our build, test, staging, and production environments.

**Infrastructure services**

Our infrastructure services, which power the various use cases in our SW Factory, include more than just basic networking, compute, and storage elements. The continuous integration and build processes have spikes that require high compute power.

We have an elastic infrastructure that allows us to meet that need cost-effectively while remaining agile. As a software vendor, our products should support our customers’ environments, which means supporting a variety of infrastructure types and form factors that span physical, virtual, and containerized environments. The software must run in both on-premises systems and a variety of cloud environments, and it must integrate middleware technologies such as databases, application servers, and web servers.

To enable testing and certifying of our software products across this complex support matrix, we use a hybrid IT service. Specifically, we delivered an infrastructure-on-demand (IoD) service that provides a self-service experience to our engineering teams. They can quickly access on-demand services that cover our entire matrix of supported environments and form factors. We built this system based on a shared poll of infrastructure resources that span our data centers as well as public clouds.

**Service portal**

This is a one-stop shop that provides access, support, and knowledge for all of the SW Factory services we offer.

**Service catalog**

We modeled all of our services and made them available in our catalog. That's where engineers can get access to the source-code management system, learn which security scanners are available, and get support to integrate the engineer's CI to the ALM system. We aspire to provide a self-service experience, and we achieve this by having a rich, continuously updated knowledge management system with a smart search mechanism, as well as offering simple, automated fulfillment processes.

**Collaboration**

We are actively promoting collaboration within and between teams through a shared wiki service that we use for planning activities such as requirements gathering and definition, design and architecture, and release planning.

We also implemented a ChatOps service that enables collaboration around topics such as Java, UI/UX, and Linux. And we create ad hoc channels on specific issues such as cross-functional team collaboration to solve customer problems or resolve defects.

We also create DevOps channels where product teams can collaborate among themselves as well as with the SW Factory bot. The system provides progress updates and informs teams when we push out a code change, when a build finishes, or when they are assigned a new backlog item. This has become more and more popular over time and now serves as the main interface for engineers to collaborate and interact with the SW Factory.

**Insights**

In order to continuously improve, we needed a set of KPIs that could help us capture our current baseline with regard to agility, productivity, and quality so that we could measure our progress, bottlenecks, and areas in need of improvement.

Shortened delivery cycles have meant more frequent deliveries, with more code changes, more tests to execute, and so on—and that means you'll have much more data to manage. In addition, moving from manual testing to automation enables faster testing and more tests, which also increases the amount of data under management.

With the large and ever-increasing amount of data we have across the SW Factory, we needed to set up a big-data system that could gather all of the data, analyze it, and generate the required reports.

**Closed-loop incident process**

CLIP is a process and integration between the SW Factory and the support system that enables our product engineers to effectively collaborate and share knowledge. This includes:

* Empowering customers and support engineers with product knowledge so they can self-support. We push the latest product information, including documentation, how-to guides, known problems, and resolutions, FAQs, and so on, from the product teams into our support systems.
* Providing customers and support engineers with the latest status of an enhancement request or issue by providing visibility into the engineering backlog status.
* Accelerating customer problem resolution by enabling support and product engineering teams to exchange information about and collaborate on customer issues.
* Enabling product groups to get insights into customer issues, read feedback, and reflect that through a unified backlog in the ALM system.

Moving forward we plan to extend integrations within our SW Factory services to streamline our four value streams and support new ones. Our backlog is agile: It dynamically changes based on the continuous feedback we receive from customers, the product teams, and our business portfolio priorities.

**Getting started: Four steps you can take now**

To successfully roll out an SW Factory, you need to take four steps:

1. Take an outside-in view of your operation to get a holistic view of the value streams required to power the business and processes and integrations with external functions.
2. Take an inside-out view to capture the main engineering services and integrations you'll need for those value streams.
3. Operate in an agile fashion, deliver an MVP, and continuously evolve it based on consumer demand and feedback.
4. Use this as an opportunity to transform: Re-evaluate your tool choices, modernize, consolidate, and integrate your tool chain to deliver faster and at a higher level of quality.

*That's how we built our SW Factory. How will you build yours? Post your questions and comments below.*