DGP Requirements

DGP's architecture and code have been designed and built using the following process:

1. Clearly Defined and Documented Requirements
    a. Two levels of requirements:
        i. Product and service functional requirements, which change frequently to meet the needs of competitive markets
        ii. Non-Functional Requirements (NFRs), which are relatively stable for a given software system over time
    b. Each requirement must specify the following:
        i. *What* feature, functionality or capability is needed by the system
        ii. *Why* the feature, functionality or capability is needed for the system (helps with analysis/design and as a filter)
        iii. *How* the delivered feature, functionality or capability will be tested/measured to prove that that the implementation meets the requirements (helps with testing, verification, and as a filter)
2. Analysis, Design and Implementation (detailed in other sections of the documentation)
3. Deployment and Testing of deliverables for every release to prove they meet all of the applicable requirements
    a. Test results must themselves be independently reproducible/verifiable in order to be *trusted as proof*
    b. This type of proof is used as the basis for decisions and choices between various alternatives over time

Many businesses typically do a poor job of defining NFR's (which are also sometimes referred to as system-level requirements). This is not an option for DGP systems, due to the scientific method-based methodology which tests and measures the deliverables produced by each development iteration to prove whether or not they are able to meet all of the documented requirements.

All functional requirements are tested and proven by the test files used by the API Tester test harness, along with the end-to-end and server-side performance of each method. The API Tester can also do load tests of the lower environments as well, measuring the performance and resource utilization (efficiency) during various simulated workloads. Testing and measuring the majority of the NFR's must be periodically performed independently of the normal development and testing processes.

SAFe (Scaled Agile Framework) terminology is used because it does a good job of covering NFR's and other requirements that are often neglected by other versions of the agile methodologies. However, no specific methodology is required for DGP systems.

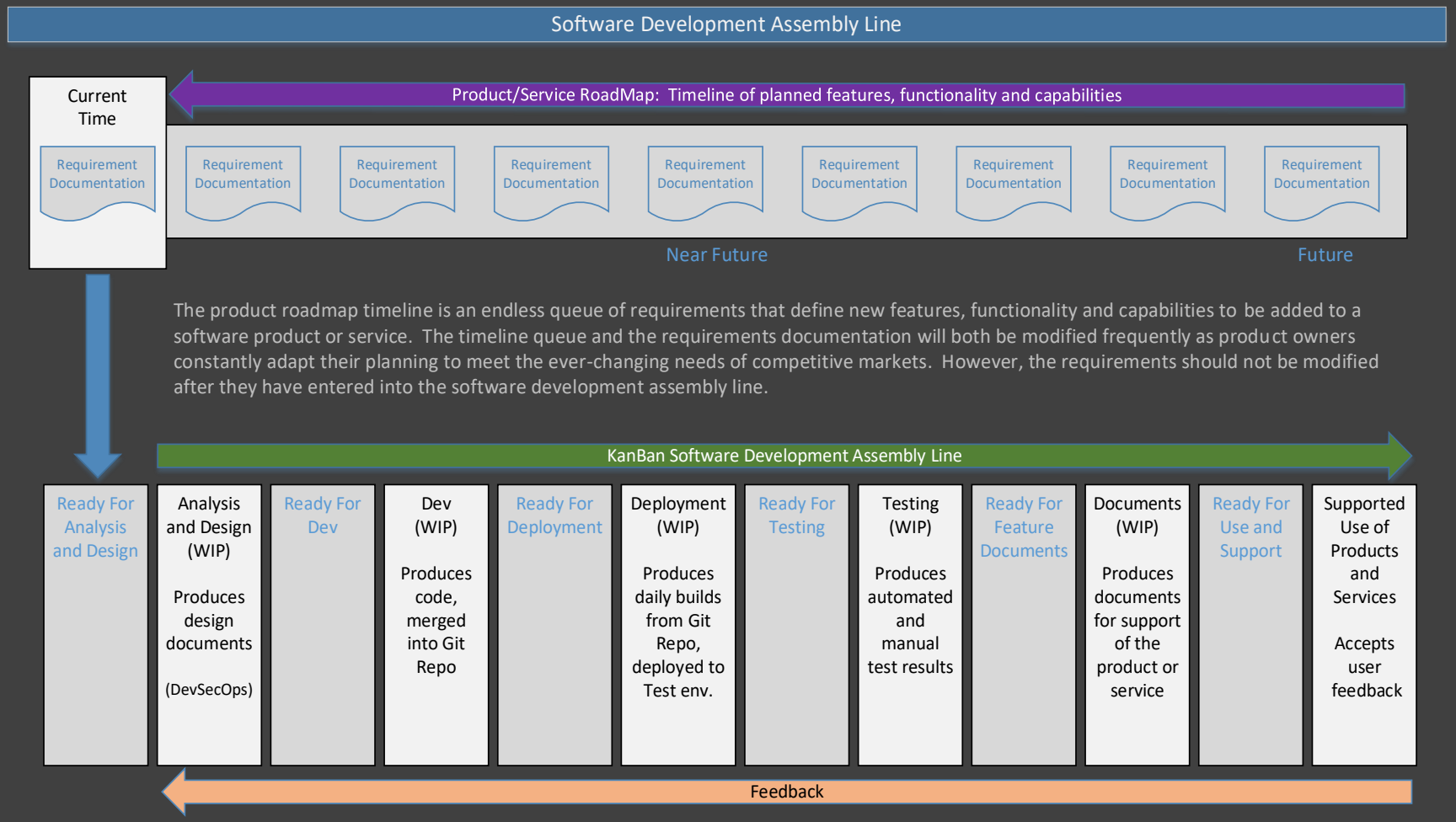
Software Assembly Line (SAFe Agile Release Train modified for DGP systems)

(Discover) → Define Requirements → Analysis and Design → Implement → Deploy → Test Results vs. Requirements

As a platform/framework, DGP focuses on the non-functional requirements (NFRs) and capabilities that are needed by almost all custom software systems (refer to SAFe documentation at https://www.scaledagileframework.com/nonfunctional-requirements/). They are all defined as sets of requirements from a business perspective, each of which are described in more detail in the specific NFR section documentation. The technical designs of the implementations which meet those NFR requirements are explained in their respective section of the design documentation. An example of a SAFe ART-compatible software assembly line is shown in the diagram below.

For DGP systems, the practice of DevSecOps is followed in the process diagramed below, which occurs during the requirements collection and design phases. DevSecOps is a term that describes the collection of requirements (mainly NFRs) from Developers, Testers, IT Ops, and InfoSec staff describing all the features, functionality and capabilities that they need from a system to make their respective jobs as easy and productive as possible. Their requirements are then merged with all the other requirements collected from the various parts of an organization so that the design of the system encompasses the sum total of all of the requirements from all of the different sources. The net result is that the system is designed and built from the ground up to give all of the various departments what they need in order to do their jobs better/faster/cheaper and more securely.

The integration of all of the requirements from all parts of an organization into a comprehensive design is an example of SAFe systems thinking in practice. The overall process is iterative, and continues to collect requirements from all of the different departments for each software development iteration, merging them into the incremental designs of new features, functionality and capabilities for the entire useful life of the software system.



The product roadmap timeline is an endless queue of requirements that define new features, functionality and capabilities to be added to a software product or service. The timeline queue and the requirements documentation will both be modified frequently as product owners constantly adapt their planning to meet the ever-changing needs of competitive markets. However, the requirements should not be modified after they have entered into the software development assembly line.

The software development is a series of KanBan queues linked together sequentially like a software assembly line. The output produced by each queue becomes the input for the next queue in the sequence. Feedback mechanisms provide a way for information about poor results to flow backwards to whatever stage of the assembly line is appropriate. Managers control the flow of work through the assembly line by modifying the prioritization of items in the various Work-In-Progress (WIP) queues, and also by insuring that there are adequate resources to handle the amount of work in each WIP queue to prevent them from becoming process bottlenecks.

Deploying a release to production must include documentation, training and updated tools needed to support the users of the product or service.