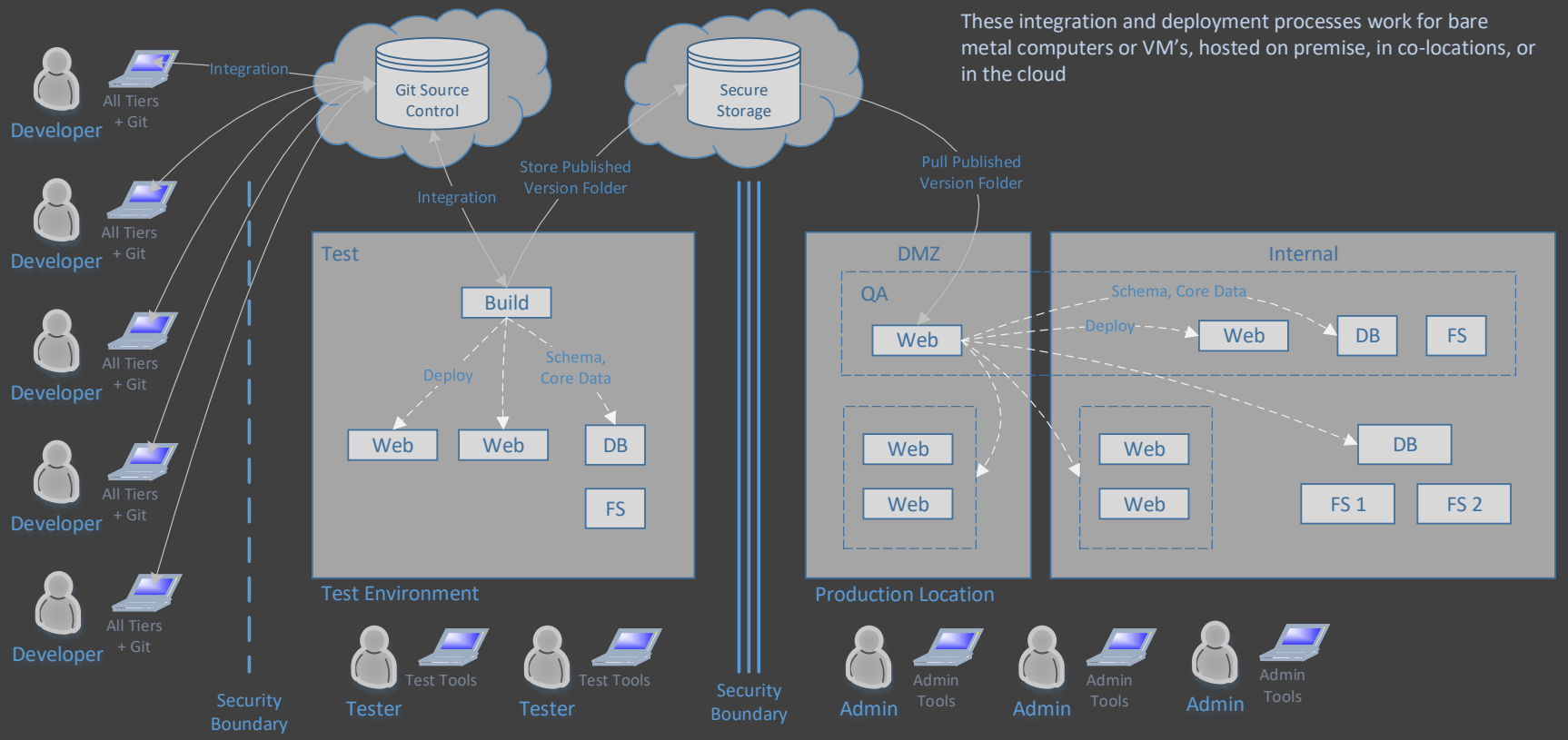
CI/CD/CT

Software development for successful software systems does not end until the system is ready to be replaced or decommissioned. That period of active software development can last for many years. The purpose of CI/CD/CT processes is to help minimize the time, effort and cost of the constant software development by simplifying the code integration, builds, deployments, testing and rollbacks of the daily software versions produced for each environment and location in a distributed system. DGP's architecture and code has been designed to improve CI/CD/CT processes by taking advantage of some of the features of the .NET framework to simplify deployments and rollbacks, as well as simplifying and automating testing by using the integrated test harness applications built into DGP systems.

Continuous Integration (CI) is managed using Git source code repositories and trunk-based development. Each developer workstation has its own local Git repository, which the developer uses to constantly stage and commit local changes. Developers will then periodically pull changes from the master Git repository while also pushing changes from their local repositories to the master (usually at least once per day). Trunk-based development means no branches are used in any of the Git repositories. The equivalent of a branch is when a developer pulls changes from the master to their local repository periodically, but does not push their changes up to the master until their local code is ready (able to successfully build).

Continuous Deployment (CD) is simplified using .NET XCOPY deployments for all tiers of the system, which also allows for extremely simple and fast rollbacks when needed. These deployments follow an immutable append-only pattern. Native apps create a release build, and web apps/web services publish a release to a folder. These folders are copied and renamed using the date they were built as their version number. These date folders are then copied under a parent folder on a client computer or web server. The values of various keys in the .config files must be edited for each location. Then the path to the executable is edited for a client app shortcut, or for an IIS web application to point to the executable in the new date folder. If any problems are found, rollback is performed by pointing the executable path back to the previous date folder, and deleting the new date folder.

Continuous Testing (CT) is performed using the various test harness applications built into the DGP system. The daily deployment to multiple distributed locations per environment is effectively impossible without a good mechanism for automated testing, and the testing itself must provide very close to 100% coverage of the code base. Another advantage of the consolidation of all functionality in a DGP system as web service API methods is that all of that functionality can be tested using the API Tester test harness.



These integration and deployment processes work for bare metal computers or VM's, hosted on premise, in co-locations, or in the cloud

Each developer workstation has all tiers plus a Git repository

By default, development environments are VM's for consistency plus easy recovery from any problems that may occur

The build machine in the test environment pulls source code from the central Git repository into an empty working directory to create each daily build. The published build is saved in its own daily build folder, named for the date (YYYY-MM-DD). Optionally, the source code can be scanned as part of this process as needed.

Once the testing in the Test environment is successful, the build folders are saved to secure cloud storage, which is accessible to both testers and the system admins for all production locations. This is the mechanism that allows assemblies to cross the security boundary.

Production admins pull the build folders down from the secure cloud storage to a QA web server in the DMZ, which is then used to deploy to the other QA servers. If testing in the QA environment is successful, then the build folders are deployed to the production servers.

CI/CD/CT Verification

1. *The trunk-based development and daily integrations of all the local developer Git repositories with the central master repository is tested and verified by the build server's daily build, which performs a clean pull from the master repository (no merges with the code from previous pulls). Each daily build uses a standardized date value as its version number. Each build produces a folder containing the published or released assemblies necessary to run the app or service.*
2. *Deployments consist of copying the new daily build folder below the parent folder that represents an application or service. Data in the configuration file then needs to be edited to match the specific location (usually by copying the configuration file from a previous build folder). Finally, the path to the app or service executable is edited to point to the assemblies in the new build folder. Rollbacks consist of reversing that final step and pointing the path back to the previous app or service executable and then deleting the new failed build folder. Also, if any additions have been made to the database schemas or core data, the DB Setup utility must be run in each location as port of the deployment process.*
3. *Once a deployment is complete, the API Tester test harness is used to run a full regression of all end-to-end test files for all of the API methods in the system. If any problems are found in the test results, the deployment is rolled back.*
4. *Role-based feature toggles are tested manually by logging into a system using accounts that have different roles.*
5. *Web service version feature toggles are tested by changing the executable path for a web service from older versions to newer versions, and manually verifying the correct toggle functionality in an application by connecting to those different versions.*

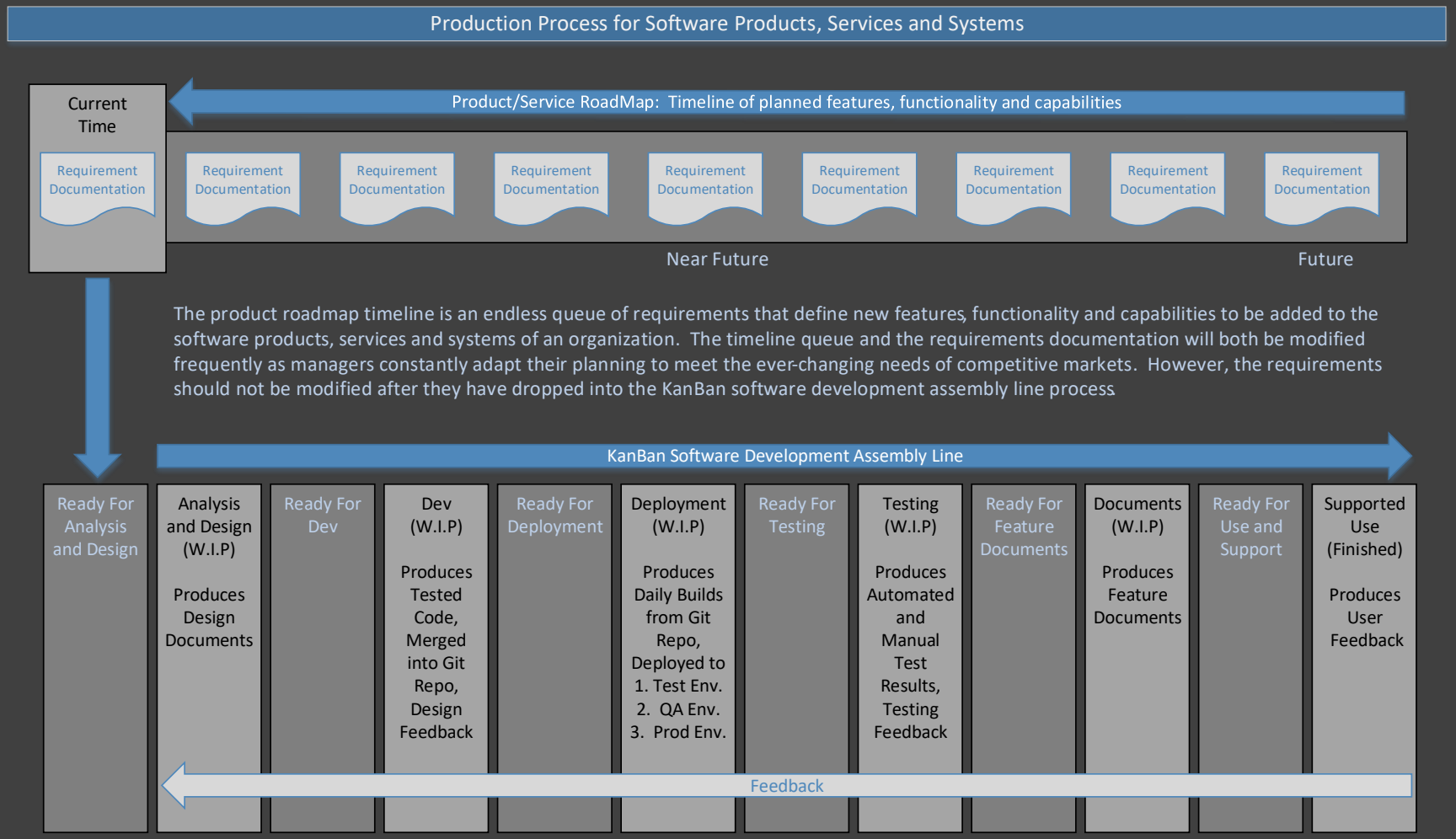
It is a challenge to keep both database schemas and security data consistent when developing distributed software systems across multiple environments and locations. DGP systems use a data driven role-based access control security system that must contain a set of core security data in order for the API methods in any location to be used. In addition, enhancing database schemas over the life of a software system without ever breaking backward compatibility requires that those schemas be maintained using an immutable append-only convention.

The DB Setup utility is used to manage the immutable append-only conventions for both the database schemas and the core security data in the SysInfo database. The logic in the utility is the equivalent of DDL scripts for the schemas, and DML scripts for the security data. The data is maintained as if it had been replicated from a "virtual" master database that only exists in the code.

The utility does not enforce any static naming conventions or default account names, etc. Admin users must provide the name and password of the super admin user account, and the encryption key to be used for the zero-knowledge encryption for the environment. Running the utility in each location of each environment makes sure the schemas and security data are consistent without compromising security. The names, passwords and encryption keys must be maintained securely by the system admins.

Schema and Core Data Verification

1. *DGP databases contain no logic themselves, so verifying that constant enhancements to a system do not cause any breaking changes is proven as part of the full regression tests run on all API methods after each deployment.*
2. *Several automated processes calculate counts in matching source and destination tables, checking to insure that those counts are the same, which verifies that the results of merge replication are correct.*
3. *Other automated processes work to incrementally repair problems in the data in-place. A verification version of the merge replication process runs in parallel to fix any gaps found in the destination data, while any "extra" records detected could mean records lost from the source or duplicate records in the destination, both of which have to be fixed by admin users.*



The software development process is a series of KanBan queues linked together sequentially like an 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 queues, and also by insuring that there are adequate resources to handle the amount of Work in Progress in each production queue to prevent those queues from becoming process bottlenecks.