The development of successful business software systems does not stop until they are either replaced or decommissioned, which is a period of time that can last for many years. During their productive life, systems are constantly being extended, enhanced and improved in order to meet the ever-changing conditions of competitive markets. This process is referred to as system evolution, and serves two main purposes: first, it enables the constant addition of new features and functionality to a system, and second, it enables technical debt or other mistakes in either requirements or implementation to be incrementally removed from a system, while the system remains in constant use, without ever breaking backward compatibility.

Requirements

1. API's must be designed to follow an immutable append-only pattern.

What: *Following the immutable append-only pattern means that once API's and methods are used in production, they cannot be removed or modified. Only new APIs, API methods or versions of methods can be added.*

Why: *The platform must be designed and built to allow the constant extension and modification of the system without ever causing breaking changes for the applications and integrated systems that depend on the API functionality. For API's, this means that once an API method is in use in production, it becomes immutable. All API functionality is tested during every full regression test whenever new code is deployed to an environment. In practice, the API Tester tests and documentation pages also become "effectively" immutable once the methods they test are put into production.*

Testing: *The full regression tests of the API Tester test harness proves that all API methods in a system are working correctly, end-to-end every time new code is deployed to an environment (verifies no breaking changes in the API's).*

2. Database schemas must be designed to follow an immutable append-only pattern.

What: *Following the immutable append-only pattern once databases and tables are used in production, schema elements cannot be removed or modified. Only new elements can be added to the schemas.*

Why: *The API's provide a layer of abstraction between dependent applications and the underlying database schemas. This abstraction allows some changes to be made to database schemas without causing breaking changes for client applications or integrated systems.. However, significant changes to a database schema cannot be papered over by this abstraction. For this reason, once a database is in production the existing schemas should not be significantly modified. The easiest convention to maintain is to only allow the addition of new tables, new columns, new indexes, etc. allows for evolution while also guaranteeing that backward compatibility will not be broken.*

Testing: *The full regression tests of the DB Tester test harness proves that all data access methods in a system are working correctly, every time new code is deployed to an environment (verifies no breaking changes to the data access logic).*

3. The various tiers of a system must be modular and loosely coupled

What: *The tiers of a system must be loosely coupled, using RPC's for inter-process communication. The configuration of the RPC endpoints (URL's, ADO.NET connection strings, etc.) controls the inter-process communication between the tiers.*

Why: *The platform tiers must be modular and loosely coupled to allow them to evolve independently. Client applications only work with the platform API's via the URL endpoints of the web services. This enables duplicate functionality of methods, API's, or even tiers to coexist in a single system, allowing a gradual transition from old functionality to new.*

Testing: *Client applications can switch between different environments by changing the endpoint configuration of the web service URL. The endpoint configuration of ADO.NET connection strings control which databases are used by web services in a given environment. These configurable endpoints are the mechanism that allows multiple versions of functionality at each level to coexist, and provides a way to transition between those versions gradually while the system is in constant use.*

4. A mechanism to enable "feature toggles" must be available: depends on RBAC authorization

What: *A mechanism to control which accounts are authorized to use specific versions of an API method is necessary to allow for incremental evolution of a DGP system.*

Why: *Feature toggles are used for both continuous deployment (deploying code that is not fully tested but is "turned off" for most users) and also to allow service evolution. This same functionality enables the head-to-head champion/challenger style of experimentation to determine if new versions of functionality are truly superior to existing versions. It also allows a system to be incrementally rewritten to remove technical debt, etc. while it remains in use.*

Testing: *The full regression tests of the DB Tester test harness proves that all data access methods in a system are working correctly, every time new code is deployed to an environment (verifies no breaking changes to the data access logic).*

5. The combination of immutable append-only conventions must enable zero code maintenance

What: *The combination of the various immutable append-only conventions must allow existing methods, test files and documentation pages to become immutable once they are used in production. This translates to zero code maintenance work for the existing API methods, test files and documentation pages of a system going forward.*

Why: *The only work required for zero maintenance systems is to add new API methods and corresponding data storage, etc. to an existing system. As a result of this reduced workload, only small teams of developers and testers are ever needed, even for large systems – which greatly reduces the total costs of the system.*

Testing: *The full regression tests are used to insure that existing immutable methods continue to function correctly and perform well after every release.*