Service Evolution

Software development does not end for a custom software system until it is replaced and/or decommissioned. A successful software system will be used for many years, and will constantly be enhanced with new features and functionality during those years. In other words, there really is no such thing as "maintenance mode" for a software system, it is really "ongoing software development mode" with a bug fix occasionally added to the mix. This is especially true when following immutable append-only conventions.

Service evolution is the practice of designing a system's web service architecture and code to make the constant addition of enhancements as fast, easy, reliable and inexpensive as possible. The modular multi-tier architecture, with the tiers isolated from each other, only communicating via RPC's is a large part of that. Each tier can use alternate functionality just by changing the configuration of the RPC endpoints. At a lower level, DGP's architecture supports this type of experimentation by allowing current functionality to "peacefully coexist" alongside experimental functionality within the same system, at the same time. This is made possible by using the data-driven RBAC security system as a feature-toggle mechanism to control which sets of user have access to which sets of API methods.

This means that additions and enhancements to a system are first implemented as experiments (prototypes), usually at the API method level, which are used to test and measure the results of the experiment to prove how well each one works in practice. Overall, the concept of "Champion / Challenger" head-to-head competition between existing and experimental alternatives is used to ensure that whichever functionality is best for the organization will gradually rise to the top over time and become the "champion" functionality that is used in a system.

The underlying premise of these competitions is to allow decisions to be made based on facts, data and verifiable proof – rather than on subjective opinions, group-think, the static continuation of current practices, etc.

- Assuming anything new is automatically better than current practices is frequently wrong.
- Assuming current practices are automatically better than anything new is also frequently wrong.

Proving which alternative (new or current) is best able to meet the requirements of an organization, at the lowest cost and in the shortest amount of time, produces far superior results compared to all other mechanisms used for decision making.

Evolution Verification

1. *Almost all aspects of a DGP system follow an immutable append-only convention, and this includes all API methods as well as the test harness files that test those API methods. In practice this means that API methods cannot be changed once they have been deployed to production environments (excluding bug fixes), and that also means that the test files which test those API methods never need to be maintained or changed from that point forward either. As a result, a DGP system becomes a growing collection of immutable API methods and test files that require no maintenance work. All development work is focused on building new API methods and test files to add to the collection.*
2. *Since all possible logic in a system is consolidated in the middle tier as web service API methods, the full regression which runs all test files for every API method as part of each deployment is the mechanism used to verify that no breaking changes to any existing API methods have occurred as part of each new deployment.*
3. *The immutable append-only conventions also apply to all of the database schemas and in some cases the data itself.*