What is DGP (Distributed Grid Platform)?

DGP is a free, open-source, API-centric software development platform that is intended to help people quickly and easily build their own custom software systems while reusing all of the advanced capabilities of the base platform. In this context, the term "platform" and "framework" are basically interchangeable. DGP is an architecture pattern combined with a collection of reusable code libraries, and can be built using a variety of standard development tools and programming languages. DGPDrive is a reference system that provides some useful file sharing and collaboration functionality, similar to a self-hosted version of OneDrive or Dropbox but with very fine-grained control over which accounts are able to access folders and files.

DGP's simple hub-and-spoke architecture evolved to meet the unique requirements of the R&D groups in several large companies, plus the extreme NFR's specified for some of the prototype systems that were built using the architecture (refer to the Requirements documentation for detailed information regarding all of the NFR's used to design and build DGP). In order to meet all the combined NFR's and functional requirements of the many prototypes, a general-purpose architecture was needed as a platform that could be used to build virtually any type of business software quickly and inexpensively.

DGP's architecture and its extensive use of the immutable append-only convention significantly changes the way software systems are designed and built. First, a base system that is able to meet all NFR's acts as a reusable core foundation (DGPDrive for example), after which new software systems are created by appending a combination of new application and data storage "spokes" plus new APIs to the base system, while reusing all of its security systems, admin UIs, testing, logging, monitoring, and utility applications, etc. This technique along with the immutable append-only convention allows new systems to be built very quickly while "inheriting" the strong security, high performance, high scalability, 100% uptime, high quality, etc. from the base system.

In this context, the immutable append-only convention is used in DGP's architecture to ensure that the constant extensions and enhancements added to a system never cause any breaking changes for the applications and integrated systems that depend on its API functionality. The data-driven RBAC security provides fine-grained control over access to every API method in a system, which ensures that the constant additions never break backward compatibility. This effectively provides all of the same benefits as low-code

development while using standard tools and programming languages, eliminating the need for expensive, ineffective low-code tools. Initially, all new API methods basically start out as experimental prototypes that only testers are authorized to call (per environment).

In addition, in recent years it has become obvious that current software development methodologies and architectures contain a number of serious flaws that cause them to deliver low-quality software systems with poor security. In order to verify that these problems are 1) real, and 2) the cause of significant negative consequences, a collection of reports by CISQ, OWASP and other sources have been included as part of the DGP documentation. For example, according to the CISQ 2022 report, the total cost to US businesses of low-quality software with poor security for 2022 has increased to $2.41 Trillion, with an additional $1.52 Trillion in technical debt. The combined costs of these problems represent almost 17% of the $23.3 Trillion US GDP for 2022.

In addition to the copies of the reports included in the DGPDrive documentation, links to the original reports are listed below:

- CISQ "The Cost of Poor Software Quality in the US - 2022": https://www.it-cisq.org/technical-reports/
- CISQ "The Cost of Poor Software Quality in the US - 2020": https://www.it-cisq.org/technical-reports/
- OWASP Top 10: https://owasp.org/www-project-top-ten/
- OWASP API Top 10 https://owasp.org/www-project-api-security/
- Synopsis "Open-Source Security and Risk Analysis - 2023": https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2023.pdf
- Cyentia "State of Application Exploits in Security Incidents: https://www.f5.com/content/dam/f5-labs-v2/article/articles/reports/20210720_soso/The-State-of-the-State-of-Application-Exploits-in-Security-Incident-F5Labs.pdf

Most of the problems cited in the reports have to do with software system capabilities that are defined by Non-Functional Requirements (NFR's) – such as security, performance, efficiency, availability, scalability, reliability, etc. One of the main reasons that these problems occur is that the current development methodologies typically do a poor job defining the NFR's for software systems. As a result, many of the NFR's needed to deliver strong security and high-quality systems are neglected or ignored. This situation has allowed software architectures and designs that deliver low-quality results with poor security to become commonplace as widely-used "best practices", which explains why the annual costs to US businesses are so high (and why they continue to increase every year).

In summary, DGP's software development methodology and universal architecture have evolved over many years in the R&D groups of several large companies. The methodology is based on the scientific method, experimentation, testing and measuring different technical solutions in order to ***prove*** which alternatives are able to deliver the best results that meet all of the requirements, at the lowest cost, and in the shortest amount of time. The proofs (experimental data) themselves must also be independently verifiable in some meaningful way.

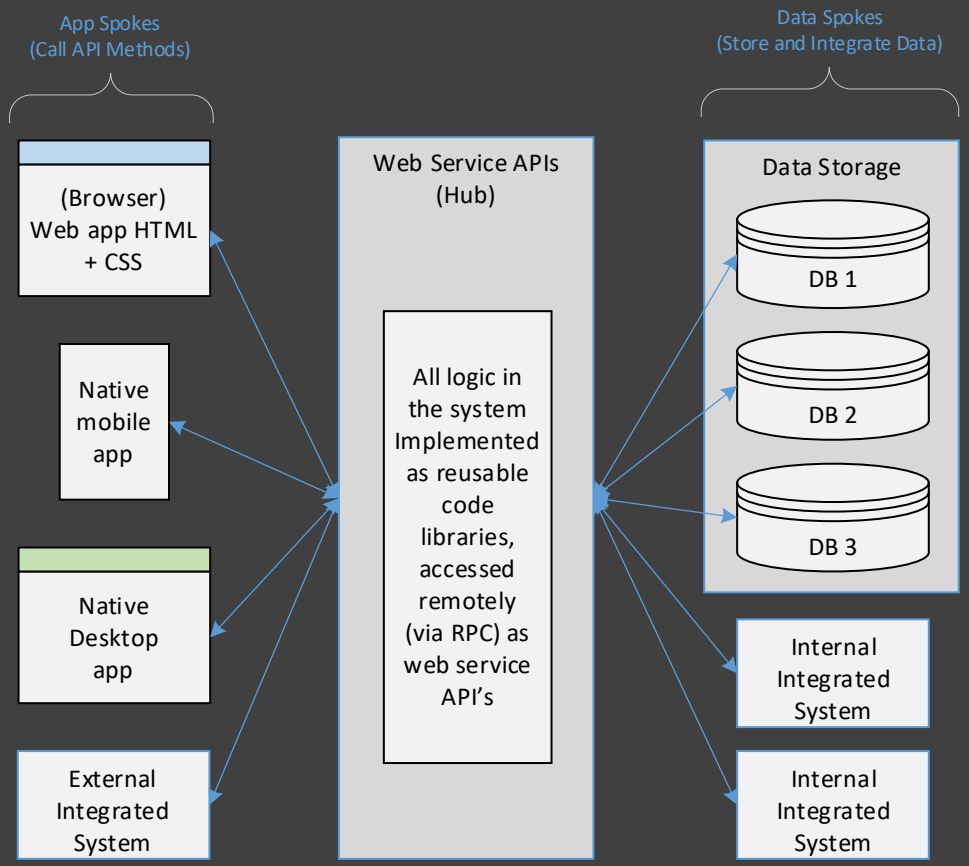
What is the Purpose of DGP (Distributed Grid Platform)?

DGP was originally created to meet the need for rapid prototyping and rapid application development in the R&D groups of several large companies. In addition, it also had to meet all of the various extreme NFR's of the many prototype systems. Later, it was determined that DGP's non-standard architecture and methodologies effectively solved almost all of the problems with current software development best practices.

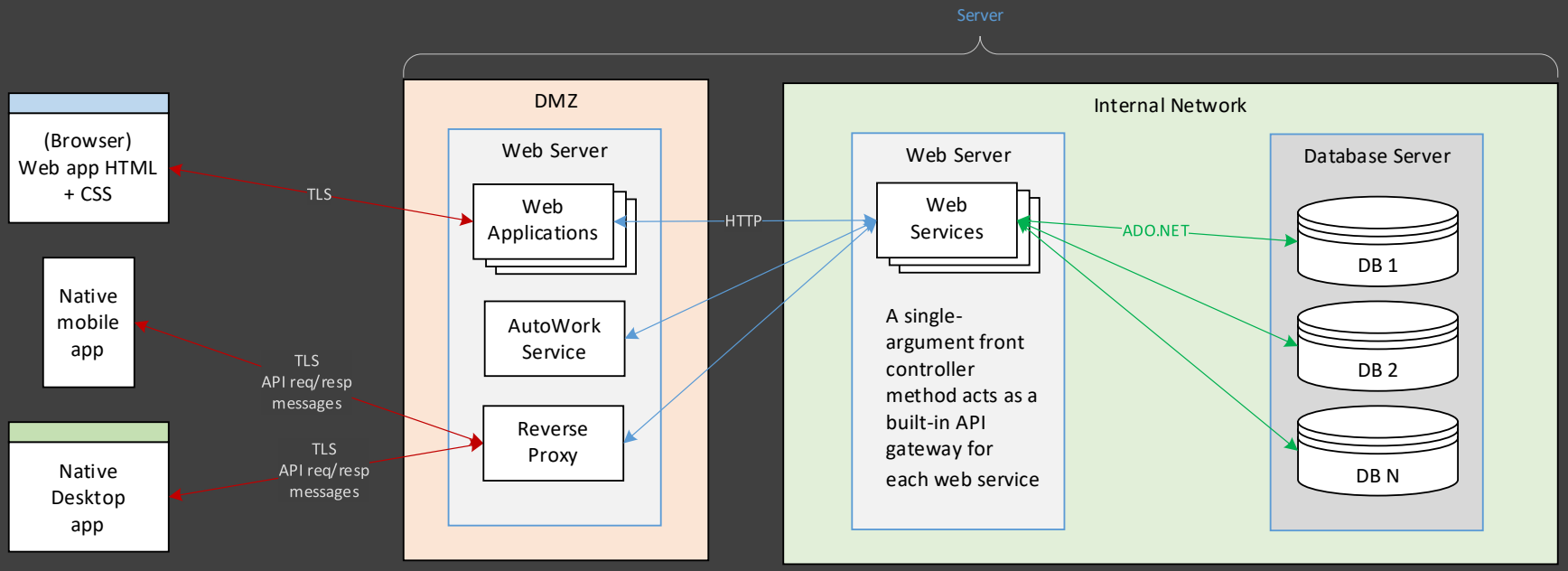
- DGP's original purpose was to provide the same advanced capabilities (NFR's) as the big tech public cloud infrastructures for privately owned custom software systems, using a much simpler architecture and code. These NFR's include:
    - Strong, effective security (able to pass PCI audits if necessary)
    - High performance and efficiency
    - True 100% system uptime, with no maintenance windows (no periods of planned system downtime for patching, etc.)
    - Easy scalability, from very small to very large systems
    - Easy system evolution (the ability to incrementally rewrite a software system while it remains in constant use)
    - Zero code maintenance
    - Continuous testing, with easy full regression testing
    - Low total cost and short delivery timelines
- In addition, DGP has been adapted to fix/correct the problems of low-quality software systems with poor security documented in the CISQ and other reports, which cost US businesses $2.41 trillion in 2022. This is accomplished by defining NFR's designed to correct the various problems, and then testing each deliverable release to prove that it is able to meet all such requirements.

- Provide the same advantages (reduced development costs and shorter delivery times) promised by low-code development tools and platforms while using standard programming languages and development tools that are already known and in use.

DGP Architecture and Design

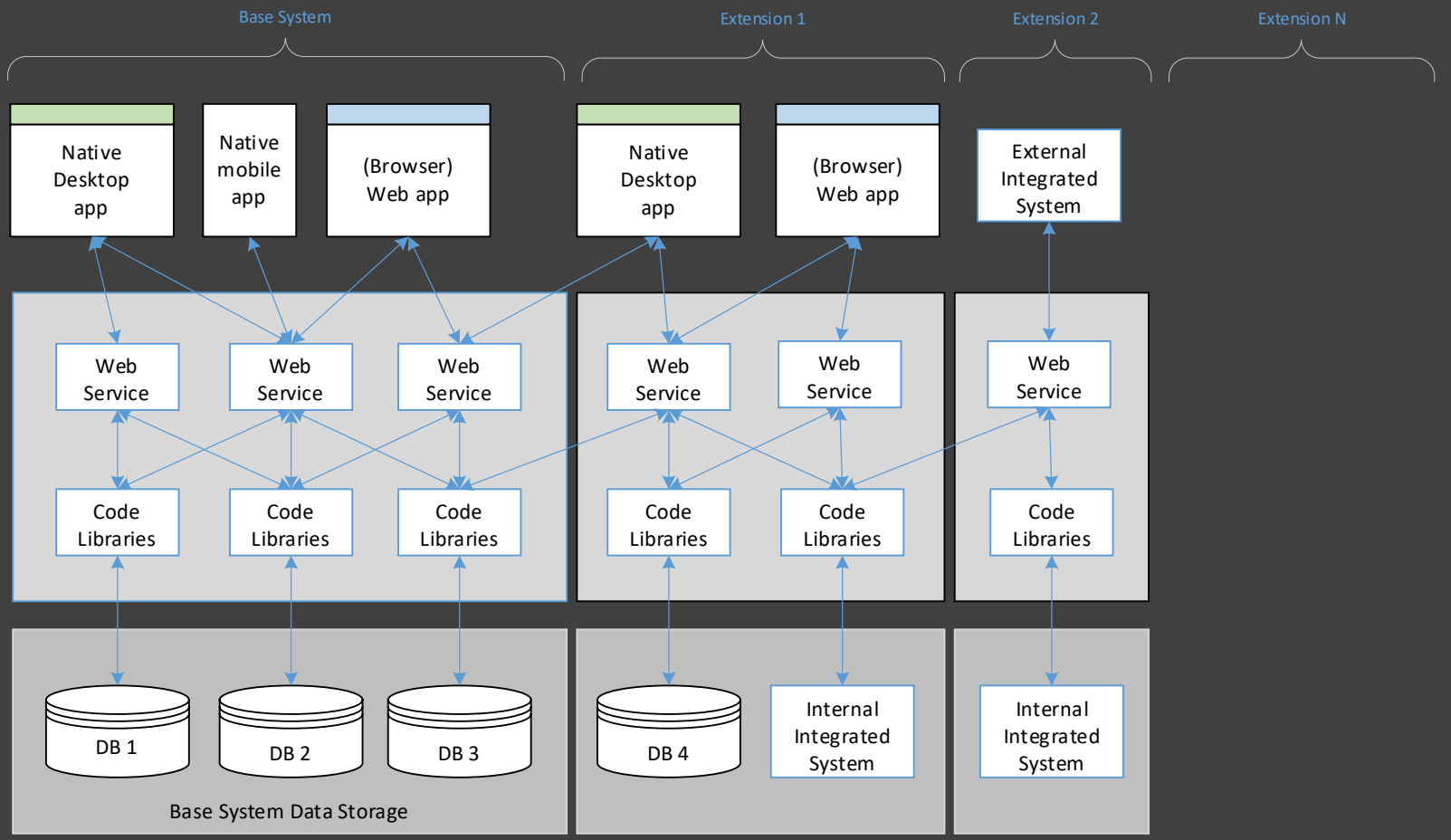


At a high level, DGP's universal architecture is based on a simple hub-and-spoke design (shown below), in which all logic in a system is consolidated into a centralized hub, and are built as web service APIs.



The logical tiers of the hub-and-spoke design are then mapped to a two-tiered thin client/fat server topology, with both the logic and the data spokes grouped together within the server tier. DGP's architecture has been proven to meet all of the NFR's listed above for the production systems that were built at those businesses at the time. The DGPDrive reference implementation is basically an open-source rewrite of those earlier systems. This centralized architecture provides the best overall security for business software systems, and also greatly simplifies their maintenance. The client applications and data storage spokes are designed by default to contain no logic, but this restriction can be relaxed as needed. This simple API-centric architecture has been proven to deliver good results as long as the end-to-end performance of the web service API methods are very fast, efficient, scalable and able to provide strong,

effective security. The internal network consisting of the logic hub (APIs) and data spokes is itself a modular multi-tier architecture that easily accommodates the ever-growing collection of APIs appended as extensions to a base system over time. The diagram below illustrates how a hub-and-spoke base system is extended with the functionality and data storage to implement new systems, etc.



A base system is extended by following immutable append-only conventions, adding new APIs, code libraries, data storage, and applications while reusing the security, admin UIs, utilities, testing, logging, etc. from the base system. The combination of hub-and-spoke architecture, message-based APIs, the integrated data-driven RBAC security system, single-argument front controller web services, and the immutable append-only convention all work together to enable easy system evolution, competitive experimentation (prototypes), continuous testing, zero code maintenance and true 100% system uptime.

- System Evolution

For successful software systems, development continues throughout the entire useful life of the system, which can last for many years. Constantly appending new functionality and applications to a system without ever breaking backward compatibility is the core of DGP's system evolution, experimentation and zero code maintenance capabilities. At a high level, DGP's centralized APIs behave much like a server-based version of the .NET Framework, whose collection of APIs is shared by all the various "spokes" of the hub-and-spoke architecture. Each client app or integrated system only uses the specific API methods that they need from the ever-growing collection of APIs in a DGP system, in much the same way applications only use the desired namespaces in the .NET Framework. The data-driven security system controls which accounts are authorized to call which API methods. This not only enables experimentation, but also the incremental rewriting of a system to eliminate problems and technical debt, while it remains in constant use.

- Continuous Testing

Every deliverable must be tested and measured to verify that it is able to meet all of the documented requirements of a system. This change in emphasis from focusing on best practices and the development process to instead focusing on testing and measuring the deliverables produced by software development helps to significantly improve the quality of software systems. In DGP systems, this capability is based on the simple unit tests and performance measurements that are permanently built into every API method. The results of the unit test and server-side performance measurement are returned as part of every API response message, and can then be displayed to the users of the client applications. Plus, users that are members of the RemoteMonitor role automatically save these remote end-to-end test results and performance measurements to whichever location they are connected to at the time. This functionality is able to collect very useful real-time metrics from production systems without adversely affecting those systems due to

the testing and monitoring workload. In addition, the API Tester test harness application is able to remotely run full end-to-end regression tests of all of the API methods in a location. The test files used by the API Tester application contain a series of both positive and negative tests for each individual API method, and can be run by any account that is a member of a test role in a given environment (but only for authorized methods). The results of these end-to-end full regression tests can optionally be saved to the SysMetrics database for further analysis as desired. Finally, the API Tester application is also able to run load tests (synthetic workload) of a DGP location. Allowing many people to easily run the tests is part of the effort to ensure that the tests and measurements are independently verifiable by both the users and testers of each DGP system.

- Experimentation (functional prototypes)

The process of building APIs and/or API methods as prototypes (experiments) to either prove or disprove how well they are able to meet all of the requirements is really nothing more than using the normal system evolution capabilities to append new API methods to the code base of a system. The fine-grained authorization of the data-driven RBAC security system ensures that all new API methods automatically start out as prototypes which are only available to testers. The API Tester test harness and test files can then be used to fully test and measure both the functionality and performance of the prototype API methods in the standard way.

- Zero Code Maintenance

Another advantage of following the immutable append-only convention is the fact that no maintenance of existing DGP code is required. Once an API method is used in production it becomes immutable, along with its corresponding API Tester test file and documentation web page. From that point forward, no modifications to the API method (other than bug fixes) are permitted. The test files and documentation page for each API method are not strictly immutable, but once they have been completed to an acceptable level, they no longer require any modifications going forward since their respective API methods do not change. Ongoing enhancements, extensions and modifications to a system are implemented as new API methods or versions of methods that are appended to the code base instead of merging changes into any existing methods. Authorized access to these new methods is controlled by the RBAC security system in each environment as described above. Older methods can be deprecated once it is proven that no applications are using them. This mechanism is not only for system evolution, but is also used to eliminate technical debt, as

new improved methods can be built to replace older methods that contain various types of technical limitations. The same type of immutable append-only convention also applies to database schemas, which are maintained using the DBSetup utility.

A very important side effect of zero code maintenance is that small teams of developers and testers are all that is ever needed for a software system, no matter how large it may grow over time. The team only needs to be large enough to keep up with the workload of adding new functionality to the system and testing it using the API Tester test harness application. All existing API methods are constantly tested as part of each end-to-end full regression to verify their continued correct functionality, and are never modified (or broken) by new development. The same is true for their associated test files and documentation pages. The combination of the zero maintenance and continuous testing innovations are responsible for the largest reductions in the total cost of software development for DGP systems, mainly due to the small size of development teams.

- 100% System Uptime and Automatic Recovery

The stateless web server farms used to host the web service APIs in each location of standard distributed systems provide very good fault-tolerance as long as the server farms contain some extra servers to handle the workload of any servers that go offline. Standard storage systems (RDBMS, NoSQL, etc.) used in distributed systems are able to replicate data between the nodes of a cluster within a single location. However, they are generally not able to replicate data reliably between separate locations. However, the ideal functionality desired by businesses for mission-critical systems would allow multiple separate locations to be active and writeable at the same time in order to provide true 100% system uptime, which also effectively eliminates the need for Disaster Recovery and/or Business Continuity processes. The following URL provides a good overview of distributed systems, grid systems, their similarities and differences: https://www.toolbox.com/tech/cloud/articles/distributed-vs-grid/

The primary feature that differentiates a distributed grid from the more common distributed systems is the real-time data replication between distributed locations that enables each location to be active and writeable at the same time, along with automatic recovery of the system from failures of individual servers (including their data). These capabilities are the key to delivering true 100% system uptime, but the final critical requirement is the capability to patch and maintain all parts of a distributed system while the system remains in constant use (in other words, no maintenance windows of planned system downtime). Periods of planned downtime are

simply not compatible with products and services that must be up and running 7 x 24 x 365. Delivering this functionality for the stateless web server farms hosting the web service APIs is relatively easy. Implementing that same functionality for data storage subsystems is the problem. Currently there are only 2 known options available that are able to do so (aside from the proprietary grid systems used internally by the big tech companies).

The first and currently the best option to deliver 100% system uptime is provided by SQL Server Availability Groups that have been configured to span across 2 separate datacenters. This technique has been used very successfully in production since 2014 by a credit card payment gateway vendor. This NFR for 100% system uptime is necessary because merchants with ecommerce websites need to be able to process credit card transactions 7 x 24 x 365, so any type of downtime for the payment gateway system is unacceptable.

The second option is DGP's omnidirectional data replication subsystem that has been specifically designed to continuously replicate data between multiple active, writable locations. By default, DGP's omnidirectional replication provides eventual data consistency between the multiple locations. However, strong data consistency can be achieved by designating one DGP location as the primary, used for all reads and writes in a system. This is essentially the same concept of a single writeable node borrowed from database clusters (including SQL Server Availability Groups) and applied to DGP locations.

DGP's data replication also supports a type of horizontal data scalability referred to as database shards. However, not all software systems need 100% uptime and all the redundant hardware and software required to deliver that level of availability, which is why the functionality to deliver 100% uptime are optional. Standard database or NoSQL clusters, etc. can be used for data storage whenever that is the best fit for a given set of requirements.

DGP/DGPDrive Beta SBOM (Software Bill of Materials)

- [closed source] Windows operating system.
  - Windows 10 (or later) for software development and Windows Server 2016 (or later) Standard Edition for production.
  - Windows IIS 10.x web server (built into Windows operating systems).
  - .NET Framework 4.8.1 (built into Windows operating systems).

- [closed source] SQL Server RDBMS.
    - SQL Server Express 2016 (or later) for Dev, Test environments
    - SQL Server Standard Edition 2016 (or later) for QA and Production environments.

Neither DGP nor DGPDrive use any open-source libraries, NuGet packages, tools or frameworks other than the dependencies in the SBOM list above. This means that all parts of DGP use functionality that is built into the Windows operating system. Since no open-source tools are used, no scans of their respective trees of dependencies for malicious code are necessary.

100% Self Support

The objective for users of DGP/DGPDrive's free open-source code and open documentation is to become 100% fully self-supporting, which effectively eliminates the need for any sort of external support for DGPDrive. The fact that a single individual has been able to design, build, test, document and maintain DGP/DGPDrive is the best possible proof of the feasibility for small teams to be able to handle a subset of that same workload in order to become fully self-supporting in reality.

100% self-support is a realistic objective for the following reasons:

- The simplicity of the architecture and code. An entire low-level DGP system diagram can be shown on a single page.
- The small amount of simple code. All of the API functionality in a DGPDrive system totals approximately 12,000 lines of code. In addition, a majority of this code is the repetition of a few basic templates (mapper classes, data classes, and so on).
- The full documentation of DGP's architecture, design, configuration, deployment, testing and all of the other parts of DGP have all been built into the open-source code as a documentation web app included in every release.
- The immutable append-only conventions followed in all tiers of DGP which never alters any existing code, releases, or core data. This results in zero maintenance for all existing immutable APIs, API test files, and method documentation pages in each system.
- The testing and performance measurement permanently built into every DGP API method provide continuous testing.
- The full regression tests run on all API methods using the built-in API Tester test harness, which can also be used for load tests.

Summary

The CISQ 2022 report estimates the annual cost of poor-quality software to US business to be $2.41 trillion for 2022 in the US alone, with another $1.52 trillion in technical debt. Combined, that is almost 17% of the estimated 2022 US GDP of $23.35 trillion. It is also evident by comparison to a similar report from 2020 that these costs continue to increase at a significant pace, especially cybercrime, OSS supply chain security problems, and technical debt.

In order to fix all of these problems, while also reducing total costs and delivery timelines, DGP uses a development methodology that was created and refined over many years in the R&D groups of several large companies. It is based on the scientific method and focuses on the rigorous testing and measurement of the deliverables produced by each iteration of development in order to prove that they are able to meet all of the documented requirements. Finally, the data from the various tests and measurements must themselves be independently verifiable in order to catch any mistakes that may have been made during the overall process.

Most importantly, capabilities such as zero code maintenance + system evolution + continuous testing + the RBAC security subsystem significantly improves the way software systems are designed and built. The fastest way to build a new DGP software system is to append new functionality as extensions to an existing system. In terms of code maintenance, one of the biggest improvements is that it enables software systems to be incrementally rewritten in order to constantly add new functionality, correct mistakes and remove technical debt, all performed without breaking backward compatibility, while the system itself remains in constant use. These capabilities protect the investments made in custom software systems, plus this high level of adaptability can extend the useful productive lives of those systems.

Finally, the combination of open-source code, the simplicity of the code, the small amount of code and the detailed documentation of DGP's architecture, design, etc. included in each release all help to enable organizations to become truly 100% self-supporting in practice. The emphasis on self-support is intended to eliminate the risks of using a system that has been designed, built, tested, documented and maintained by a single individual (Alan Rahn).

"To bear and not to own; to act and not lay claim; to do the work and let it go: for just letting it go is what makes it stay." — Lao Tzu