PRINCIPLE BASED   
ENTERPRISE ARCHITECTURE

TABLE OF CONTENTS

Contents

[Enterprise Architecture 6](#_Toc485204762)

[Elements of an Enterprise Architecture Practice 7](#_Toc485204763)

[Architecture Metrics 7](#_Toc485204764)

[Lifecycle 7](#_Toc485204765)

[Patterns and Best Practices 7](#_Toc485204766)

[Architecture Governance 8](#_Toc485204767)

[What Triggers Asset Governance 8](#_Toc485204768)

[Asset Governance: People 9](#_Toc485204769)

[Asset Governance: Policies 9](#_Toc485204770)

[Asset Governance: Process 9](#_Toc485204771)

[Principles 13](#_Toc485204772)

[Principle Overview 13](#_Toc485204773)

[Enterprise Architecture Objectives 14](#_Toc485204774)

[System Asset Principles 14](#_Toc485204775)

[Data asset Principles 15](#_Toc485204776)

[Principle steward 16](#_Toc485204777)

[Golden Rules 16](#_Toc485204778)

[Golden Rules for Systems 17](#_Toc485204779)

[Golden Rules for Data 17](#_Toc485204780)

[How the Golden Rules Evolve 17](#_Toc485204781)

[Body of Evidence 17](#_Toc485204782)

[Bad Outcomes 17](#_Toc485204783)

[Positive (Good) Outcomes 18](#_Toc485204784)

[Secure 18](#_Toc485204785)

[Secure Golden Rules 18](#_Toc485204786)

[End user Authentication secrets must be protected. 19](#_Toc485204787)

[Customer accounts must be difficult to attack. 21](#_Toc485204788)

[Web traffic must be kept private. 21](#_Toc485204789)

[Inputs from untrusted sources must be sanitized before use. 21](#_Toc485204790)

[Data must not become code. 24](#_Toc485204791)

[Transactions involving sensitive data must be auditable. 25](#_Toc485204792)

[Credit card payment account numbers must not be manipulated. 25](#_Toc485204793)

[Highly regulated data (SPII) must be protected. 26](#_Toc485204794)

[Sensitive data must not be placed in a URI.. 27](#_Toc485204795)

[Standard encryption implementations must be used where available. 27](#_Toc485204796)

[Production changes must be reviewed, approved and auditable. 28](#_Toc485204797)

[3rd party software must be used safely. 28](#_Toc485204798)

[Internet facing security exposures must be caught before being exploited. 30](#_Toc485204799)

[Important security related events must be recorded and reported. 31](#_Toc485204800)

[Production environments must be kept separate and secure. 31](#_Toc485204801)

[Standard authentication implementations must be used where available. 31](#_Toc485204802)

[Systems must degrade gracefully when attacked. 32](#_Toc485204803)

[Infrastructure environments must be safe. 32](#_Toc485204804)

[Access to important systems and data must be managed. 33](#_Toc485204805)

[Compliant 33](#_Toc485204806)

[Compliant Golden Rules 34](#_Toc485204807)

[Intellectual Property must be protected. 34](#_Toc485204808)

[3rd party IP must be used in accordance to its license. 35](#_Toc485204809)

[Source code must be stored in a secure and managed repository. 36](#_Toc485204810)

[Customer facing U/Is must be accessible by users with disabilities. 36](#_Toc485204811)

[Reliable 36](#_Toc485204812)

[Reliable Golden Rules 37](#_Toc485204813)

[Systems must meet performance and availability SLRs and Recovery Objectives. 38](#_Toc485204814)

[Systems must have an appropriate plan for functional testing. 39](#_Toc485204815)

[Scalable 40](#_Toc485204816)

[Scalable Golden Rules 40](#_Toc485204817)

[Systems must deliver acceptable performance under anticipated load. 40](#_Toc485204818)

[Systems must optimize purchase of capacity. 41](#_Toc485204819)

[Manageable 41](#_Toc485204820)

[Manageable Golden Rules 42](#_Toc485204821)

[Systems must respond to standard control commands. 42](#_Toc485204822)

[Systems must publish appropriate operational events. 43](#_Toc485204823)

[Systems must publish Performance and Capacity data. 43](#_Toc485204824)

[An inventory of all system hosts must be available. 44](#_Toc485204825)

[Simple 44](#_Toc485204826)

[Simple Golden Rules 45](#_Toc485204827)

[Legacy Assets and Deprecated Interfaces must not be used by Strategic assets. 45](#_Toc485204828)

[Code packaging must facilitate independent releases. 46](#_Toc485204829)

[Modular 47](#_Toc485204830)

[Modular Golden Rules 47](#_Toc485204831)

[Assets must expose and consume only well-defined External Interfaces. 48](#_Toc485204832)

[External Interfaces must be versioned and well managed. 49](#_Toc485204833)

[External Interfaces must be easily consumable. 50](#_Toc485204834)

[Systems must not be tightly coupled to their infrastructure / environment. 52](#_Toc485204835)

[External Interfaces must not be tightly coupled to implementation details. 53](#_Toc485204836)

[Maintainable 54](#_Toc485204837)

[Maintainable Golden Rules 55](#_Toc485204838)

[An interface must be callable directly and not require a proprietary library. 55](#_Toc485204839)

[Requests must be traceable from point of entry through all intermediaries. 55](#_Toc485204840)

[Code, Schemas, and APIs must be appropriately documented and commented. 56](#_Toc485204841)

[Mastered 57](#_Toc485204842)

[Mastered - Golden Rules 58](#_Toc485204843)

[Data Assets and their system host must be registered. 58](#_Toc485204844)

[Data Must be of high quality. 58](#_Toc485204845)

[Data must be encapsulated. 59](#_Toc485204846)

[Data must be traceable to its source. 59](#_Toc485204847)

[Data must be validated. 59](#_Toc485204848)

[Global 59](#_Toc485204849)

[Global - Golden Rules 60](#_Toc485204850)

[Systems must handle data in a globalized way. 61](#_Toc485204851)

[3rd party translations must be distinguished from company translations. 62](#_Toc485204852)

[Systems must to adapt to the user’s preferred locale. 64](#_Toc485204853)

[Data assets 66](#_Toc485204854)

[Compliant Data 67](#_Toc485204855)

[Compliant Data - Golden Rules 67](#_Toc485204856)

[Compliant Data - Information Value Classification 67](#_Toc485204857)

[Compliant Data - Regulated Data 67](#_Toc485204858)

[Compliant Data - Restricted Data 68](#_Toc485204859)

[Compliant Data - Retention and Purging 68](#_Toc485204860)

[Compliant Data - Asset Registration and Naming 68](#_Toc485204861)

[Reliable Data 68](#_Toc485204862)

[Reliable Data - Golden Rules 68](#_Toc485204863)

[Reliable Data - Indirection 69](#_Toc485204864)

[Reliable Data - Data Quality 70](#_Toc485204865)

[Reliable Data - Data Revisions 71](#_Toc485204866)

[Reliable Data - Data Curation 71](#_Toc485204867)

[Reliable Data - Data Asset Design 71](#_Toc485204868)

[Reliable Data - Data Accuracy and Implementation 72](#_Toc485204869)

[Mastered Data 72](#_Toc485204870)

[Mastered Data - Golden Rules 72](#_Toc485204871)

[Master Data - Mandatory Use 72](#_Toc485204872)

[Master Data - Data Models 73](#_Toc485204873)

[Master Data - Higher Standards 73](#_Toc485204874)

[Flexible Data 74](#_Toc485204875)

[Flexible Data - Golden Rules 74](#_Toc485204876)

[Flexible Data - Enrichment 74](#_Toc485204877)

[Flexible Data - Separate Presentation 75](#_Toc485204878)

[Flexible Data - Separate Product Content 75](#_Toc485204879)

[Flexible Data - Product-Neutral 75](#_Toc485204880)

[Flexible Data - Fidelity 76](#_Toc485204881)

[Flexible Data - Tables and Illustrations 76](#_Toc485204882)

[Flexible Data - Unit Preservation 76](#_Toc485204883)

[Flexible Data - Data Handling 77](#_Toc485204884)

[Global Data 77](#_Toc485204885)

[Global - Golden Rules 77](#_Toc485204886)

[Global Data - Text Handling 78](#_Toc485204887)

[Global Data - Number-Centric Data Handling 78](#_Toc485204888)

[APPENDIX 79](#_Toc485204889)

[Scorecard 79](#_Toc485204890)

[How to fill out a scorecard 80](#_Toc485204891)

[Golden Rules for Systems Quick Reference 85](#_Toc485204892)

[Golden Rules for Data Quick Reference 86](#_Toc485204893)

# Enterprise Architecture

The term **architecture** is used to define the structured description of a set of systems and how they relate, as well as the structure of a system itself. As we zoom in on the architecture of a system, we reveal greater and greater detail and at some point we are talking about design rather than architecture. When we refer to an Enterprise Architecture, we mean the highest level view of the technology landscape, that makes sense to describe.

But why do we need an Enterprise Architecture? Can’t we let the software engineers just get on with writing the code, without all this structure? It’s not just CEOs asking that question, it’s the architects as well. But before answering, let’s ask a related one:“Before the invention of the GPS, why did we need highway maps?”

Imagine trying to drive from New York to Los Angeles, possessing only the detailed street-by-street maps of all the cities in the United States. While you could certainly get there in the end, it would neither be easy nor quick. To solve the problem of driving from New York to L.A., you simply don’t need the detail provided by the street maps. What you need is a context map that abstracts away the unnecessary detail, providing just enough so that the entire picture can fit in your brain at once, and you can see how to get from Point-A to Point-B.

In an enterprise the size of the United States, at the most abstract level (the enterprise architecture), we understand that we have states, cities within states, and interstate highways. In addition, when driving through a state, one is much like another unless you want to stop and look around. To complete the mission (driving from NY to LA), all we really need to know is which states we want to go through, and what interstates we want to take to optimize our time. This is an architectural approach. At this level of abstraction, all interstates can be treated much the same (except for where they lead) and while we know that the road systems within each city are different, that level of detail is not immediately important. If we stop in a city for gas or to stay overnight, we can always refer to the detailed map for that city, knowing how it fits into the overall United States context map.

Using this analogy, the Enterprise Architecture is like the highway map of the U.S. It lays out the high level structure of how all the states (systems) and interstate highways (system-to-system interfaces) fit together, so that the more-detailed state maps (system level architectures) have context.

The important point to recognize is that architecture focuses on the whole to gain perspective, by abstracting away detail in order to draw higher level relationships. Enterprise Architecture patterns capture “what is the same” about groups of elements within the architecture. System-level architecture and design, on the other hand is about reducing scope and increasing detail. In order to get the necessary detail, you zoom in, allowing elements beyond your “design scope” to fall outside the field of view. To put these in perspective with other design elements, we can think of “zooming-out” to the architecture level, moving to a different part of the map and “zooming back in” again.

The root of the Enterprise Architecture Practice is its core objectives and principles. The Architecture Principles are the mission statements for the assets in the architecture. Each Principle is further elaborated as a set of golden rules, which define how to achieve the mission, and the consequences of deviation. Golden Rules read like the requirements statements for meeting the mission statement as defined by he architecture principles.

To achieve the goal of a governable architecture, and to avoid the dreaded “Death by Governance”, it is essential that the Golden Rules be limited to only the rules that matter, avoiding the clutter caused by trying to enforce every good idea.

The **Enterprise Architecture Practice** defines the key roles and processes for creating the Enterprise Technology Architecture.

## Elements of an Enterprise Architecture Practice

The elements of an Enterprise Architecture Practice are::

* Architecture Metrics
* Inventories - Such as Assets, Domains, Technology Standards, APIs
* Patterns and Best Practices
* Design Guidelines
* Communications
* Key Processes
  + Architecture Governance processes:
    - System Asset Governance
    - Data Asset Governance
    - API Governance
    - Technology Standards
  + Placement of Function
  + Change Management

## Architecture Metrics

Architecture Metrics are the informative facts created from governance processes, which measure the fitness of the technology investments that have been made. Enterprise Architecture measures compliance, convergence, and deviation from business strategy. Measures are aligned to business objectives through governance levels for systems and data; A dashboard presents these metrics in a way that can be understood by business decision makers.

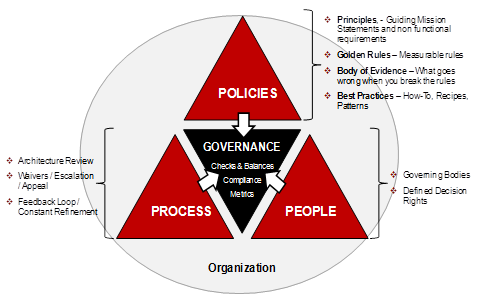
## Lifecycle

* Placement of Function - is the strategic planning activity whereit is determined which assets will be changed as part of a project, assuring consistent separation of concerns, and preserving the long term value of technology investments.
* Golden Rules - Global architecture standards and guidelines to assure our systems and data are Safe, Operable, Sustainable, and Global
* APIs - For those who implement services and the interfaces that provide access to them so that we can be wildly profitable, and responsive to new revenue and operational efficiency opportunities
* Asset retirement.

## Patterns and Best Practices

* Patterns - A pattern is a specific template for capturing a specific solution to a commonly recurring problem
* Best Practices - A Best Practice is a method or technique that has consistently shown results superior to those achieved with other means. Best practices are not mandatory. An Enterprise Best Practice is optional, but following it will ensure your path through architectural governance is a happy one
* Body of Evidence - This is a collection of real-life "stories" that describe what goes wrong when you break the rules.
* Architecture Diagramming - describes architecture diagramming in general, points to example diagrams, and links to answers to commonly-asked questions about producing architecture diagrams

# Architecture Governance

The business benefits of a Service Oriented Architecture (SOA) are derived primarily from the increased reuse it facilitates on system and data assets. The time-to-market, cost-of-ownership, integration, and quality benefits all derive in some way from increasing reuse. As such, achieving the business goals of SOA requires that we achieve reuse goals, which requires Governance. Architecture Governance is the process of measuring the adherence of the assets to the strategy and to the golden rules..

Asset governance begins with people, who have well-defined decision rights and a clear escalation path in the case of conflict. Policies (which we call principles and golden rules) define what we are trying to achieve in a measurable way. Golden rules do not define functional requirements of products, but rather the non-functional requirements. The **asset governance proces** measures adherence of assets to the strategy and the golden rules in order to produce metrics defining compliance or deviation. Given these metrics, business decisions can be made as to future investments in the assets. The process iterates so that, over time, we achieve the right level of compliance with policy and strategy.

Asset governance is distinct from program and project management. Projects are funding envelopes initiated to create, change or decommission assets. They have financial goals and delivery goals.

## What Triggers Asset Governance

As the business adapts to ever-changing market conditions, we not only deal with the ramifications of past decisions, but also try to make the best decisions for the future. The biggest decisions often involve allocation of limited resources.

Asset Governance is triggered by an investment in an asset or to create a new asset, changes in technology solutions, mergers and acquisitions. Generally asset and technology related triggers are the result of a standard GTO Process and integration is based on timing and cadence of the target or business unit.

Governance is a process with checks and balances implemented by people. Projects initiate **governance**, but **Architecture Governance** measures assets, not projects. **'Projects end, Assets have a life of their own'**. There are lots of projects, but not every project initiates an **architecture governance** action. Only when an asset undergoes a significant change, is **architecture governance** required. If any of the following criteria is true, then the asset is considered to have gone through a significant change and is therefore subject to formal review:

* The asset is new or was acquired through merger or acquisition
* The asset has undergone an architecturally significant change, for example:
  + Components have changed
  + Externally facing APIs have changed
  + Data models have changed
  + COTS software has changed.
  + The asset's scorecard has expired (18 months for strategic assets, never for non-strategic systems and)

**Asset governance applies** to all assets; the process varies by asset type and whether the asset is strategic, tactical or legacy.

## Asset Governance: People

Enterprise architects (EAs), solutions architects and technology architects work together in the **governance process** but each has a distinct and prescribed role.

## Asset Governance: Policies

The **policies** used for **Asset Governance** are the principles and golden rules.

## Asset Governance: Process

### Asset Inventory Management

Business as Usual Asset Governance requires the active management of the global asset inventory. In the end, we'll see where we have redundant systems and can begin to takes steps to rationalize them - choosing the best system to solve problems, while concurrently reducing costs. The Global Asset Inventory will not likely ever be 100% correct. But over time it gets better.

### Asset Scorecards

(EAs) review the scores assigned by SAs and TAs and roll-up the scores for golden rules to principles to business objectives and finally to Assets. Asset scores are then rolled up to domain. (EAs) work as a body to ensure that all of the various assets (systems, and data) being developed make sense as a whole and adjust the evolving technology strategy and business. (EAs) also harvest the of one group and evangelize them to other groups. These practices apply equally to system assets, and data assets. The principles and golden rules for each type of Asset are different and the people are different, but the process is essentially the same.

Data assets are scored against the golden rules for Data. A benchmark of the data quality is established through the review process. (EAs) work with (DAs) and others within the business units and development groups to establish operational governance over the data assets. Operational governance is carried out by the teams responsible for the development and maintenance of the data and the data management systems.

The purpose of asset governance is to create a view of the technology, and data landscape. The Architecture Debt associated with that Asset and the detailed breakdown of what has contributed to the debt feeds into the portfolio planning process so that debt may be paid off *in the fullness of time*.

Projects initiate governance actions. When the project is first initiated the total amount of Architecture Debt that this project will increase or decrease from the landscape is assessed during Placement of Function. During the course of the project, debt may be added or subtracted through change control. Every change request during a project is sized before it is approved. Some changes are relatively free and some can be quite expensive. When a change is sized, there are four figures are required:

1. The cost to build – Changes can imply an increase in scope or a decrease in scope. Scope increases usually imply additional funding
2. The cost to operate (OpEx) – A change may imply a delta to the operational expense (OpEx) for the asset. This is sized as well.
3. Architecture Debt – A change may add architecture debt, remove debt or leave it the same.

A project may impact a single Asset or a group of Assets. Placement of Function defines the impacted assets. For each Asset requiring **governance** the specific process appropriate for that Asset is invoked.

### System Asset Governance Process

System Assets proceed through a series of reviews as part of the **governance process**. This is done early in the product software development lifecycle so that rework is kept at a minimum. The first review is called a pre-implementation review. Ideally, this review occurs in the beginning of a project, after systems architecture has completed but before production coding begins. In reality, coding often begins first. There are also many systems developed long before the goverance process existed, and systems that we aquired from other companies. To address the need to govern these systems too, we simplified the process even more through Governance Levels for System Assets.

If the system asset exposes application programming interfaces (APIs), then an API review is scheduled sometime during the middle of the project after an API design completes.

Before the system asset delivers into the certification and production environments, the physical architecture should be completed and a post-implementation review is scheduled. The post-implementation review determines the operational characteristics of the asset, such as availability including disaster recovery, scalability, security and manageability.

The architecture review begins with the production of diagramming artifacts by the solution architect. The review process may begin as soon as the first set of diagrams are produced and need not wait until everything is complete. It is usually best to iterate through the review, especially for complex systems. After the review is complete and domain architects and enterprise architects are satisfied that that the architecture represents intent, a Scorecard is filled out for the asset and Architecture Debt is assessed.

### Data Asset Governance Process

Some of the benefits of a strong data governance program are:

* Reduced data redundancy
* Reduced development time and costs for application development tasks
* Clarity of data definitions and improved data accuracy;
* A single point of control for versioning of enterprise data models and other definition and knowledge resource assets
* Increased ability to share and re-combine data across systems
* Increased flexibility to meet the diverse needs of product teams
* A standardized, measurable and repeatable methodology
* Better information available for decision making.

### Architecture Debt

Both on an individual and a national scale, debt imprisons. Tom Hodgkinson

Architecture Debt is the key Metric that comes from the Asset Governance process. Architecture Debt is essentially the difference between what we have spent on technology and what we should have spent to be fully compliant with Policy and Strategy. Essentially it is an obligation for future investment.

Carrying architecture debt is not intrinsically bad. All assets carry around a bit of debt. We just need to make sure that we don't collapse under the burden of debt. As long as we manage the debt appropriately, mitigating the business risks associated with non-compliance of policy, carrying the debt and paying it off over time is a natural part of the technology lifecycle. The architecture governance dashboard gives a global view of the debt being carried by the assets under governance and is a tool for managing the debt. There are several reasons that debt may occur:

#### Commercial Decisions

Commercial decisions may result in architecture debt when the decision is made to build out legacy technology rather than wait for a strategic alternative. Generally the decision to do so is made as a trade-off by product and commercial decision makers with their IT counterparts.

When we choose to build out existing legacy technology assets with new money, we must factor into the decision:

* We generally also need to build the same functionality into the strategic asset (hence we will be spending twice for the same customer-facing feature)
* We will need to maintain the Legacy technology year on year (YoY) and that maintenance increases with age and amount of investment. We will also be maintaining the strategic asset, so will be double spending until the migration is complete.
* We will need to invest in migrating from the legacy asset to the strategic asset. Migration costs general increase as the number of connections to an asset increase and the number of users accessing the asset increase. They also increase exponentially with complexity.

#### IT Planning and Execution Decisions

IT Planning and execution decisions may result in architecture debt when an asset releases prior to complying with the Architecture Golden Rules. Generally this is the due to challenges or trade-offs in the software development lifecycle.

The primary driver for doing this is time to market, secondarily unfunded or delayed dependencies. As long as any issues are mitigated and the debt is eventually paid, this iterative compliance model is rarely a problem. The single largest compliance issue is generally provision of Disaster Recovery.

#### Financial Planning and Analysis Decisions

The Financial Planning and Analysis function informs leadership decisions for diversification of budgets for all global functions and business units. Thus architecture debt may result from underinvestment in annual maintenance and repair (M&R) post initial release of an asset. M&R is year on year (YoY) investment to fix bugs and address non-discretionary (e.g. compliance related) software changes. By not allocating enough ongoing funding to M&R live systems, **maintenance debt** increases.

An industry rule of thumb for an asset's M&R is ranges 4% and 8% on the low-range, to 11% - 15% on the high-range, of the original investment to develop the asset's complete feature function. It is important to note that these estimates must be inflation adjusted YoY.

#### Guidelines for Debt Calculation

The following steps should be followed in order to calculate architecture debt

1. Define the Debt Issue and identify where the issue impacts policy compliance in the Scorecard. Only a single Principle may be assigned to a Debt Reference. The Debt issue should be written in the form of a user story.
2. The Architecture Debt is essentially the cost (usually measured in time or money) that it would take to resolve the issue, determined in the same manner as the standard sizing process.
3. For data assets, the estimate should be based on how much effort it would take to correct the deviation as a one-time effort w/o on-going maintenance (assuming on-going maintenance is part of the operational process). In other words, one pass through the data to correct the anomaly.
4. “Placement of Debt” should be the same as our guiding principal for the architecture diagrams, which is “does it help tell the story?”. With that regard, the thing that needs a lot of money to solve the root problem should be the thing that carries the lions share of the debt.

#### See Also

* CAST - 7 Steps to paying down Technical Debt
* Gartner - Technical Debt
* Gartner - Total Lifecycle Cost

### Governance Levels for Systems Assets

It's important to know if systems are safe. We are frequently asked by prospects, customers, and of course regulators to explain and affirm our security controls and compliance to them. Through the Enterprise Architecture governance program, compliance and divergence with declared controls, standards, and technology best practices are measured and reported through the system asset dashboard. Architecture Governance identifies, assesses, and provides input to leadership to prioritize remediation of decisions inconsistent with RELX Group policies, and statutory and regulatory obligations. This is business as usual in an Internet-enabled economy.

This dashboard is used to communicate with participants in the governance process, and stakeholders. There are five levels of Governance for systems, which are defined based on achieving specific business objectives. All systems that we build, license - install - and operate, consume through Software as a Service (SaaS), or consume through Application Programming Interfaces (API), are subject to governance.

The governance levels are:

1. **Ungoverned** – No governance metrics.
2. **SAFE** – Metrics regarding wether the asset is secure, and compliant with policy/law
3. **RELIABLE** and **PERFORMANT** - Metrics regarding wether the asset is *Safe* + the system can run with a high level of availability and scalability. Operational problems will be resolved quickly
4. **FAST TIME TO MARKET**  - Metrics regarding wether the asset is Safe + Reliable & Performant + the system is designed to be upgraded easily and avoid entropy

### So what can make governance arduous?

To participate in system asset governance you must understand the Golden Rules and the Diagramming Guidelines Like anything worth knowing, learning these takes time to learn and master. The challenge of finding time is difficult in any business environment. Then discovering you've broken the rules *(that you didn't even know existed)* is quite frankly demoralizing. Let's face it, no one wakes up excited each day to do harm by wasting time, money, or creating security or compliance issues.

The challenge though, is that we can't afford to waste, and take unnecessary risks associated with breaking the Golden Rules. These Golden Rules and the policy, strategy, standards, and best practices they represent encapsulate decades of knowledge from hundreds of technologist across the industry.

Most of this knowledge is distilled into a scorecard that's the key deliverable of the governance process. The scorecard is a simple spreadsheet. It is organized around our Principles and Golden Rules, with Yes/No questions to identify compliance or divergence to the rules themselves. Answering these questions provides meaningful metrics about an Asset. The aggregate metrics are quite meaningful too. Examining a portfolio of scorecard metrics could identify patterns of exceptions that require funding to provide both training and remediation to our systems. Examples include a security issue; or quickly discovering if an asset is global, thus localization for a foreign territory limit revenue potential and/or time to market without additional unplanned investment (and that would be bad).

### The best way to make anything happen is to make it easy

Governance Levels are a practical strategy to govern most IT Investments by calibrating the metrics produced proportional to the business objective. Even systems we choose not to govern shgoudl still be identified in the asset inventory and tracked. Prioritizing which systems to deliver scorecards first does not need to be difficult.

### Golden Rule Change Management Process

The Golden Rules can be updated based on needs within the firm. They are not set in stone and can be changed to meet business directional changes or industry standard changes. A process exists for introducing, approving and deploying new changes to Golden Rules and the corresponding Scorecard items that would relate to testing the Golden Rule update via review. A new golden rule should only be defined to avoid a “bad outcome” whose impact and likelihood make it worth the extra effort of ensuring its avoided.

# Principles

A principle is a statement of architectural characteristics.



The root of the Enterprise Architecture Governance Practice is its core principles. Architecture Principles organize the non-functional (i.e. non-feature) requirements for the assets in the architecture. These **principles** are based on what we want to achieve as a company, and are not limited to a single division or product. In fact, the architecture principles and associated Golden Rules specifically capture the non-functional requirements that every product needs to adhere to, but which requirements specifications often ignore.

## Principle Overview

Each Principle is further elaborated as a set of golden rules, which define how to achieve the mission, and the consequences of deviation. Golden rules read like the requirements statements for meeting the mission statement of the principles. A Golden Rule exists to either promote one or more positive outcomes or prevent one or more bad outcomes. More often than not we are preventing bad outcomes rather than promoting good ones. These bad outcomes provide the Body of Evidence for the Golden Rules. Golden Rules are further elaborated into a set of *tests*, which are Yes/No questions whose answers determine whether the Golden Rule has been followed and the bad outcome avoided.

To achieve the goal of a Governable architecture, and to avoid the dreaded “Death by Governance”, it is essential that the Golden Rules be limited to the rules that matter, avoiding the clutter caused by trying to govern every good idea. In other words, the bad outcomes that a Golden Rule exists to prevent must be worth preventing. Each bad outcome is assigned an impact (i.e. how bad is the bad outcome) and a likelihood (i.e. if the rule is broken, what is the probability that the bad outcome will happen?). The combination of impact and likelihood define the severity of the bad outcome and thus define how important it is to follow the Golden Rule

As no two companies are precisely alike, no two Enterprise Architectures will be precisely alike and no two sets of core Golden Rules will be precisely alike, but you would be surprised (or maybe not) how often the principles are exactly the same in every company that has achieved the technological maturity level to which we aspire.

Companies who fail in this endeavor do so because they either think too narrowly or try to bite off too much at once. It is essential to marry the rigor of the process to the maturity of the organization and therefore to start off small (i.e. with a small set of Golden rules) and build out to larger set based on shared experiences. Through root cause analysis, we build the Body of Evidence (i.e. list of bad outcomes for the architecture / design errors that have impeded business progress or caused customer pain.

Adheramce to a principle ismeasured through a Scorecard and contains a set of Golden Rules and Golden Rule tests. There is a steward responsible for each principle.

Principles (and the Golden Rules derived from them) impose constraints on the technology infrastructure. Constraints properly applied are both the foundation of architecture governance and of technology innovation .

The more constraints one imposes, the more one frees one's self.

And the arbitrariness of the constraint serves only to obtain precision of execution.

– **Igor Stravinsky**

## Enterprise Architecture Objectives

An **Enterprise Architecture** is put in place to achieve five primary business objectives:

* **SAFE** - Ensure systems and data are secure (limiting the attack surface and protecting privacy) and compliant (with corporate policy and the law).
* **RELIABLE** and **PERFORMANT** - Ensure that systems and data are highly available, and responsive to the end-user.
* **FAST to MARKET** - Ensure that products can add new functionality quickly, so that the end-user may benefit from our investments as soon as possible. This requires that our systems and data be versatile and flexible
* **GLOBAL –** Ensure that systems may be easily localized for new geographies
* **COST EFFECTIVE** - Ensure systems and data minimize total-cost-of-ownership maximizing time-to-market

The *principles* are defined as the mission statements to achieve these objectives of the Enterprise Architecture. While the objectives apply equally to system assets and data assets, the *principles* and Golden Rules for systems assets and data assets are unique.

## System Asset Principles

### SAFE Principles

Safe systems protect the brand and reputation of the company. The impact of failing to be *SAFE* transcends any single product. A security breach in any product taints every product in the eyes of the customer. A compliance failure can result in large fines and in some cases jail. These principles apply not only to the code we write, but also to the technology we buy/license/use and the way we use it.

1. **Secure (#1)**: Secure solutions are consistent with the level of risk the company has elected to accept. They protect and preserve access to proprietary services and confidential information in the company’s systems. They transport information in a secure manner as necessary for the class of information.
2. **Compliant (#2):** Compliant solutions uphold the law and comply with regulations. They adhere to the provisions of the company's contractual obligations. They protect the Intellectual Property of the corporation and do not infringe the intellectual property of others.

### RELIABLE and PERFORMANT (RAP) Principles

Reliable and Performant systems meet Service Level Agreements (SLAs) . They facilitate identification and resolution of problems quickly and long before customers complain. This includes the ability to react to demand by increasing (or decreasing) capacity in a manner that meets utilization needs (measured by performance SLAs). Having dramatically more capacity than is required may meet SLAs, but is also not cost-effective.

1. **Reliable (#3)**: Reliable solutions provide measurable service in terms of responsiveness, availability and dependability during normal operation, as well as in failure scenarios and in the event of a disaster.
2. **Scalable (#4)**: Scalable solutions support load increases via a proportional increase in resources in a cost-effective manner.
3. **Manageable (#5)**: Manageable solutions have hooks to monitor, measure, and modify operational behavior and adopt, operational . These systems know what is happening before customers do. Issues are resolved before they become issues.

### FAST to MARKET Principles

Minimizing Time-to-Market requires that systems combat the tendency toward entropy and complexity and facilitate the production of future software releases in an agile, affordable, and predictable manner. Software tends to follow the Law of Entropy. It becomes more disordered and therefore more complex over time. As software increases in complexity, it becomes harder and harder to change, which drags the release cycle and increases time-to-market (which is bad).

1. **Simple (#6)**: Simple solutions have clear responsibilities with little or no overlap. They [tessellate](https://en.wikipedia.org/wiki/Tessellation). They are "as Simple as Possible and no simpler". They have proper Placement of Function, duplicate as little as possible based on organizational reality and follow well-defined patterns and blue-prints.
2. **Modular (#7)**: Modular solutions employ [Separation of Concerns](https://en.wikipedia.org/wiki/Separation_of_concerns). They divide labor among encapsulated components that are loosely-coupled via well-defined interfaces, entities, and data models. They do not share infrastructure.
3. **Maintainable (#8)**: Maintainable solutions are easily supported and easily modified. They are able to be extended into adjacent functional areas with minimal surgery.
4. **Mastered (#9)**: Data (including, but not limited to business data, content, and operational data) is handled rigorously. Data items are mastered, and their transit through systems and way-points are carefully managed. Values, names, mnemonics, descriptors, classifiers, and other descriptors (Metadata) are handled with the utmost rigor.

### GLOBAL Principles

1. **Global (#10)**: Global solutions have the ability to be easily localized for use in a specific geography, including, but not limited to: language, script, culture, currency, color conventions, holidays, and sort-order

### COST EFFECTIVE Solutions

There are two aspects to cost-effectiveness. Cost effective solutions minimize the one-time cost of bringing new features to market as well as minimizing the year-on-year cost of running and maintaining these solutions. In a normal technology life-cycle, only 2% - 15% of the total cost of ownership of a solution comes from the one-time investment cost. The rest comes from the ongoing cost of operating and maintaining the solution.

There are no principles specially associated with COST EFFECTIVENESS. Instead the principles that facilitate cost-effective operations come from being **RELIABLE and PERFORMANT**, while the principles that facilitate cost-effective investments come from being **FAST to MARKET**

## Data asset Principles

### SAFE Data Principles

Safe data is managed in compliance with laws, regulations, corporate policies, and contractual obligations. Data is Intellectual Property.

1. **Compliant Data (#1)**: Access to data is controlled in order to protect its value for generating revenue and profit and to avoid improper use and exposure to liability. For example regulated data is only present in specific systems and accessible only to authorized individuals. Data is managed and protected in compliance with laws, regulations, and corporate policies

### RELIABLE Data Principles

Reliable data is maintained in a well-controlled manner, so that users can rely on the data’s structure and values. Data value is protected from its point of origin onward by preserving its accuracy, identity, completeness, and currentness.

1. **Reliable Data (#2)**: Reliable is consistent everywhere it is presented so that the values can be trusted.
2. **Mastered Data (#3)**: Mastered data has an authoritative "single source of truth" that avoids data redundancy. Since it is meant to displace lower quality sources of the same data and can be used in many places, standards for its definition are higher.

### FAST to MARKET Data Principles

Minimizing Time-to-Market requires that data be versatile.

1. **Flexible Data (#4)**: Data represent a significant investment of editorial, operations, fabrication, and storage resources. Flexible data is structured so that it can be used in many products, repeatedly enriched for ever-increasing value, and reformatted for multiple uses in online and print products at the lowest possible cost.

### GLOBAL Data

1. **Global Data (#5):** Global companies operate in many countries. Each country has its own languages, scripts, time zones, currencies, and customs for formatting text. Global Data supports this rich diversity by following the globalization/localization rules.

### COST EFFECTIVE Data

Cost-effective data is versatile, reliable, well-managed and well-structrured so that it may be used in as many products as possible without duplication.

## Principle steward

A **principle steward** is an Enterprise Architect who is responsible for maintaining one of the numbered principles on an Enterprise Architecture scorecard, and all subordinate golden rules and scorecard rules.

This responsibility extends to maintaining supporting guidelines on related reference materials.

In general, changes to the wording of principles, golden rules, and scorecard rules require approval through the Architecture Review board. However, minor wording changes can be made by a principle steward outside of this process.

# Golden Rules

What is an architecture golden rule? It’s a rule that when broken is likely to lead to a bad outcome for the company. But nothing is perfect and rules will be broken. The scorecard process measures and tracks these deviations so that risk, cost and time may be balanced. Breaking a rule is not intrinsicly bad as long as it is transparent, measured and tracked.

Adherance to the architectural golden rules lead to higher long-term value for the platform. The **Golden Rules** and the principles that organize them are not limited to a single division or product. In fact, the architecture principles and associated **golden rules** specifically capture the non-functional requirements that apply to every product, but which business requirements often assume rather than specify.

Given a Body of Evidence, which are the bad outcomes that have happened through poor technology choices and through Root Cause Analysis, **golden rules** are defined to avoid these bad outcomes in the future. Adherence to Golden Rules have a direct correlation to customer satisfaction and NPS, even though they are not features.

**Golden rules** and the asset governance / scorecard process are designed to catch these errors early and share the key learning across the architecture, development and business communities.

## Golden Rules for Systems

The Architecture Principles for Systems (and the **Golden Rules for Systems** derived from them) impose constraints on the system assets and their use of technology infrastructure. Constraints properly applied are both the foundation of Architecture Governance and of technology innovation.

There are eleven Architecture Principles applying to system assets. The principles are organized into four levels based on their alignment to business objectives.

## Golden Rules for Data

The Architecture Principles for Data and **Golden Rules for Data** are distinct from those that apply to system assets, even though there is an obvious relationship between the two.

## How the Golden Rules Evolve

Golden Rules evolve for a number of reasons:

* We start off with a small set of **Golden Rules** and then add more as we find issues and perform Root Cause Analysis
* Through feedback from customers, architecture reviews, and code reviews we learn that rules need clarifying, elaborating or eliminating
* Technology and the business change and the rules must change with them.

# Body of Evidence

A major component of the work of Enterprise Architecture is to govern the evolving architecture of system and data assets to maximize their value to the enterprise. This governance is not based on arbitrary rules, but rather on rules derived from long experience. Architects have collected stories of things that went wrong in the past, that cost this company (or other companies) money in redundant investment and/or lost revenue, that reduced competitiveness, upset customers, and/or increased various kinds of risk.

We call this collection of stories the Body of Evidence that justifies Enterprise Architecture's principles, golden rules, and scorecard rules.

## Bad Outcomes

For the most part the *stories* proving the **body of evidence** for the Golden Rules are bad outcomes. In other words, bad things that have happened to technology in the past, which we would not want to accidentally do again.

Once a **Bad Outcome** is identified, it is assigned an impact (i.e. how bad is the bad outcome) and a likelihood (i.e. if the rule is broken, what is the probability that the bad outcome will happen?). The combination of impact and likelihood define the severity of the Bad Outcome and thus define how important it is to follow the Golden Rule.

## Positive (Good) Outcomes

While Golden Rules can exist to promote *good outcomes* as well as prevent bad ones, almost invariably we write the *body of evidence* to avoid a bad outcome rather than promote a good one. In other words, instead of writing the rule to promote a simple architecture, we write the rule to avoid a complex one.

# Secure

Trust, but verify. – Ronald Reagan

Security is mostly a superstition. It does not exist in nature, nor do the children of men as a whole experience it. - Helen Keller

A Flexible Standard is an Oxymoron – Ian Koenig

SAFE solutions are **secure**. Throughout the entire product lifecycle careful consideration must be given to securing the application from accidental and malicious operations (both from within and without). In the architecture, key functions and checkpoints should be identified to serve as secure boundaries. When crossing these secure boundaries only identified (authenticated) and allowed (authorized) access is permitted.

Information which crosses a secure boundary should be secured according to the rules associated with the classification of said information and the change in security level crossing from one side of the boundary to the other.

## Secure Golden Rules

Secure solutions are consistent with the level of risk the company has elected to accept. They protect and preserve access to proprietary services and confidential information in the company’s systems. They transport information in a secure manner as necessary for the class of information.

|  |  |
| --- | --- |
| **Number** | **Golden Rule** |
| 1.1 | End user Authentication secrets must be protected. |
| 1.2 | Customer accounts must be difficult to attack. |
| 1.3 | Web traffic must be kept private. |
| 1.4 | Inputs from untrusted sources must be sanitized before use. |
| 1.5 | Data must not become code. |
| 1.6 | Transactions involving sensitive data must be auditable. |
| 1.7 | Credit card payment account numbers must not be manipulated. |
| 1.8 | Highly regulated data (SPII) must be protected. |
| 1.9 | Sensitive data must not be placed in a URI. |
| 1.10 | Standard encryption implementations must be used where available. |
| 1.11 | Production changes must be reviewed, approved and auditable. |
| 1.12 | 3rd party software must be used safely. |
| 1.13 | Internet facing security exposures must be caught before they are exploited. |
| 1.14 | Important security related events must be recorded and reported. |
| 1.15 | Production environments must be kept separate and secure. |
| 1.16 | Standard authentication implementations must be used where available. |
| 1.17 | Systems must degrade gracefully when attacked. |
| 1.18 | Infrastructure environments must be safe. |
| 1.19 | Access to important systems and data must be managed. |

If you are moving Customer personal Information from your core network (most secure zone) to your DMZ (less secure zone) across a core firewall, Golden Rules define whether, for that class of content, encryption is required (or not) and the level of encryption

Different classes of information may have different rules applied when the information is at rest, such as in a repository, or in motion, such as when it is being sent in a message and crossing a secure boundary from a zone at Security level x to a zone at security level y.

For example: Billing Information (and other information that affect the Financials of a company) needs to be secured against malicious intrusion and all but authorized access, according to the rules associated with that classification of the data. Different classes of data have different requirements for confidentiality and trust and access when in motion and at rest.

System designers must ensure that their implementations guard against attack scenarios, and also provide necessary access controls that are required for that type of application. Reducing the “attack surface” is of paramount importance. It is difficult enough to guard the expected access points, it is almost impossible to guard against wide open access or unnecessary access points. By default all firewall ports should be closed unless they need to be open. All operating systems for production systems should be installed in a minimal locked-down configuration. All known vulnerabilities should be checked regularly. A mechanism should exist for applying security patches to all production systems automatically.

All inputs to systems (via APIs) must be sanitized and inspected to ensure the data coming in is appropriate. Failing to sanitize inputs is the root cause of SQL injection attacks.

There should be a complete inventory of every production system and its configuration (i.e. every 3rd party software title installed, every process that is supposed to be running, and how every system operates under normal circumstances).

*Where security is concerned, information really is power.*

## End user Authentication secrets must be protected.

Data in transit is highly vulnerable. Data in transit over a network can be trivially “sniffed” by anyone with physical access to the sub-network (including desktops compromised by malware). Data on media being physically transported can be stolen or misplaced. There are simple and straightforward methods to protecting data in transport, including TLS (the replacement for SSL, what we require here is HTTPS), static encryption, or physical hardening of isolated production networks.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.1.1 | Scorecard Rule 242 | All authenticated transactions must be encrypted (i.e. HTTPS) |
| 1.1.2 | Scorecard Rule 243 | All end user selected secrets (like passwords and security answers) must be encrypted when transmitted. |
| 1.1.3 | Scorecard Rule 1802 | All end user selected secrets (like passwords and security answers) must be encrypted or hashed when stored. |
| 1.1.4 | Scorecard Rule 1803 | Authentication secrets should not be captured to insecure log repositories (mask the secret in the log entry instead). |

### Examples

* Legally protected data such as un-redacted Social Security numbers or Drivers License numbers that are transmitted by HTTPS / SSL.
* Sensitive data moving across an enterprise service bus that runs in a physically isolated and secured “Purple” production network.
* Sensitive data going from service to service via a VLAN.
* Contractually protected data such as Credit Card Primary Account Numbers (PANS) that are transmitted by HTTPS / SSL.
* Transactions that are authenticated are done only via SSL (now TLS).
* Policy protected data such as customer passwords which are hashed or encrypted within a browser cookie.
* Passwords stored encrypted at rest.
* Session based authentication tokens that are not usable more than 10 minutes from the time they are first transmitted are currently not considered a functional equivalent to an authentication credential. This was tried, but made many products unusable on remote dial up networks.
* Queries protected by contracts with customers that are required to be transmitted only via SSL.
* Note that expired or one time use passwords going through email and sent by approved credential management systems are approved exceptions to this rule.

### Consequences

* Violation of law, contract, or policy.
* Incur security risks beyond what company executives have chosen to accept.

### Body of Evidence

#### Story 1

* An "internal use only" feature of a product deeply links into the infrastructure and was added for testing and kept “secret”. It would allow people who knew the secret (presumably internal users) to put ID's and Passwords directly in a visible plaintext parameter on a URI. The product would accept that ID and Password (which will only be sent internally) and authenticate a user and drop them into an a running ession. Almost immediately upon being released into production, it got widespread non internal use. Close to 10 years later, it is now possible to scan one week of log files, and intercept about 10,000 valid customer ID s and Passwords. The security group will have to spend over significantg time and effort to try and find the applications using this feature, and get them to move from GET to POST and require them to use HTTPS instead of HTTP. The cost to resolve the issue is roughly 10x – 100x the cost/time saved by hacking in the feature in the first place.

#### Story 2

* A network sniffer sitting on an internal network doing a penetration test was able to intercept several hundred valid (and indefinitely replayable) session cookies, and could have successfully used the session cookies to completely impersonate the end user. An attacker on a customer network or any node upstream of a customer could have done the exact same thing, and it would be very difficult to detect.

## Customer accounts must be difficult to attack.

|  |  |  |
| --- | --- | --- |
| Number | **Scorecard Rule** | Rule Text |
| 1.2.1 | Scorecard Rule 1804 | The customer password rules implementation must meet the described minimum scores on the Password Score Worksheet 6f. |
| 1.2.2 | Scorecard Rule 253 | Publicly accessible administrative interfaces MUST use two-factor authentication (e.g. password plus IP address). |

### Body of Evidence

#### Story 1

If a customer account is compromised via credential attacks on the customer, the Customer's privacy is violated and we must spend significant time and energy detecting and responding to the problem.

#### Story 2

If sensitive customer information or regulated data is stolen while in transit, the Customer's privacy is violated and customer satisfaction suffers.

#### Story 3

One company using the cloud neglected to protect its administrative account with 2-factor authentication. As a result, the account was compromised, and all of the company's content and backups were destroyed. The 'small' company elected to exit the business rather than rebuild due to recovery costs.

## Web traffic must be kept private.

|  |  |  |
| --- | --- | --- |
| Number | **Scorecard Rule** | Rule Text |
| 1.3.1 | Scorecard Rule 1805 | All authenticated web traffic on uncontrolled networks must only be transmitted via HTTPS (TLS). |

### Body of Evidence

#### Story 1

A lawyer using her laptop in a Starbucks in preparation for an acquisition has her traffic intercepted by a malicious party who deduces the merger based on her searches, and uses that nonpublic knowledge to commit securities fraud and sour the merger.

## Inputs from untrusted sources must be sanitized before use.

Data sanitization is not data validation. Data validation simply ensures that inputs are reasonable. For example to check to make sure a month is a number from 1 to 12. The goal of data validation is to get correct data, and to improve user experience. Data sanitization has a different intent… it desires to strip content that may have been inserted by a hostile attacker that could cause downstream operations to do undesirable things. The goal of data sanitization is to remove content that could be dangerous, not to determine if the data is correct or even coherent.

|  |  |  |
| --- | --- | --- |
| Number | **Scorecard Rule** | Rule Text |
| 1.4.1 | Scorecard Rule 246 | Untrusted inputs must be sanitized before being further used. |

Incoming data can ingress via many different paths and forms… for example cookies, url params, headers, xml content, etc. All must be sanitized before use.

Note data that is cryptographically tunneled does not necessarily have to be sanitized. For example, if we properly encrypt the contents of a cookie in our servers, send that cookie without any keys to be stored by the users browser, then decrypt that cookie later back within our networks, this data can be said to have not come from an uncontrolled network.

Also note that input sanitization is not just removal of malicious changes to real input variables, it is also the protection of state that should not be changed as the result of a request. For example, if a form is not asking for a field, somebody deliberately injecting a name value pair in a GET or POST request should not result in that field being updated in the application. It also means that if a field requires some kind of bump up authentication in order to be changed, that additional authentication challenge must be satisfied before the field can be changed.

One generally useful strategy for data sanitation is to "HTML Encode" your inputs as soon as you get them and before further using them in any way. All web platforms include some kind of HTML Encode native function. This call converts dangerous characters (such as a double quote or angle bracket) into safe sequences of characters that will be displayed correctly but not interpreted as HTML.

For example, the string <”foo**”**> when HTML encoded, is changed to &lt;'&amp;quot;foo&amp;quot;'&amp;gt;. But when displayed by the browser, it appears as <”foo**”**> as opposed to being interpreted as HTML. This doesn't solve every possible problem, but it solves most of them.

### Examples

* A web application accepts a "year" field in a submit form, and runs a regular expression against the user supplied value removing anything that is not a 0,1,2,3,4,5,6,7,8, or 9.
* A service that communicates with a client deployed Flash component that accepts and XML string that specifies a “research topic” label and validates that XML against a narrowly defined schema rule that enforces that the actual tag contains nothing besides "A-Z, a-z, 0-9, ',', '-', and '.'".
* A service that accepts what is by necessity a fairly complex search string, and removes anything not in a whitelist of approved characters (either by a regular expression or schema) that is not part of the approved character set (which will need to be fairly broad). The service would t hen proceed to also further apply a blacklist operation, and find and remove valid javascript constructs (which would otherwise be allowed by the white-listed characters).
* A application that takes a freeform text field “HTML Encodes” the entire string before further processing.
* We receive a document to publish from a fabrication partner, and search it to remove any attachments, HTML, Javascript, etc.

### Body of Evidence

#### Story 1

A web site (fortunately in this case one that sold non regulated data) (a) failed to sanitize inputs and (b) failed to use parameterized stored procedures. As a result, it was breached by hostile externals via SQL injection. The eventual outcome was that the entire site was ordered taken off line until a risk analysis could be performed and vulnerabilities remediated. The site remained offline for over a month while investigations and cleanup were completed. This made it very difficult for the application to meet promised revenue goals. Neither the product nor the product’s VP remained with the company.

#### Story 2

Community servers sites that allow self registration and posting of shared content are finding themselves advertizing a lot of pornography websites. This is inconsistent with company branding and content guidelines, and contradicts the companies published ethics goals.

#### Story 3

(Summarized from information posted at: (<http://thenextweb.com/socialmedia/2010/09/21/twitter-security-flaw-allows-third-party-sites-via-mouseover/> ))

If you missed it on the news, there was a twitter worm going around. Rumor has it that it gave lots of unsuspecting people following tweets from Robert Gibbs (Whitehouse Press Secretary) their first opportunity to promote hardcore Pornography. As usual, it was a a result of the exploitation of a collection of half insecure features. In a nutshell, this is what happened (which includes some simplification of details):

1. Twitter recognizes URL's and turns them into actual clickable hyperlinks (just like Word tries to do) when they are redistributed.
2. Unfortunately for Twitter, they did not have and follow the security golden rule captured: *All data coming from a client side application (which includes browsers, Silverlight, flash, java applets, web parts, or desktop applications) is sanitized before being further used within the application.*
3. Therefore, it was possible to create a carefully crafted tweet that would cause javascript to be executed when anyone read a tweet.

|  |  |  |
| --- | --- | --- |
|  | **User Input** | **HTML Twitter sends to everyone** |
| How Twitter meant for it to work. | Woo Hoo! contains text that Twitter used to create a link to "http://www.company.com/" | Woo Hoo! § |
| How attackers exploited it. | Woo Hoo! contains text that, when twitter tries to use it, creates not just a link but also an instruction to the browser to execute some malicious javascript | Woo Hoo! § |

1. The net result was that Twitter sent out a tweet that would actually result in a javascript mouseover event if you simply let your mouse roll over the link. Because this malicious command originated from Twitter itself, the javascript (which is bound in the browser sandbox to only reach back to the server it came from) was free to manipulate the viewers Twitter account.

In reality it was a little more complicated and required the native twitter short URL facility to bypass sanitization, but the net effect was the same.

This was a situation where individual pieces were half secure alone, it was when they got combined they could be exploited. Attacks are getting much more sophisticated as defenses get stronger, and this kind of "thread the needle" attack is now pretty much par for the course.

So anytime you are getting user input that you may display back to anyone, make sure it is filtered down to the narrowest possible subset of valid characters. If you know it is a number, strip anything not a number, and throw some regular expressions at it after the fact to try and kill any attempts at HTML injection (like stripping all quotes and angle brackets).

Taking the input and passing it through a HTML Encoding library is even easier, and a very effective control (provided there aren't bugs in the URL encoding library). An attacker can try and inject things like <script>, but after HTML encoding and reflection, it won't be <script> anymore, it will be &lt;script&gt;, which won't be interpreted as a command by a web browser.

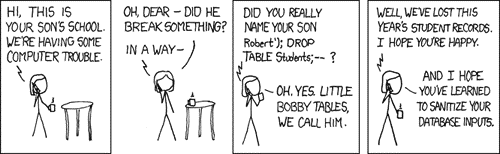
So sanitize those inputs! Even harmless pranks could carry significant reputational risks.

#### Story 4

A requirement was added that forced two factor controls to be applied to some customers before they could update their email address. It was found to be possible for an authenticated user to update the email address without doing the two factor control by injecting the value into a state that was read from the client and that was subject to tampering before being returned to the server. This new injected value was not sanitized out of the request, nor cryptographically validated to detect tampering, and the two factor control was not applied before using it to set a new email. The solution was not to "read back" the email address and insert it into the database (the form never requested it in the first place).

## Data must not become code.

|  |  |  |
| --- | --- | --- |
| Number | **Scorecard Rule** | Rule Text |
| 1.5.1 | Scorecard Rule 247 | All interactions with databases MUST be via parameterized queries. |
| 1.5.2 | Scorecard Rule 248 | All interactions with the operating system MUST be via parameterized calls. |



* All application and enterprise service interactions with databases MUST use parameterized queries that prevent user provided input from being interpreted as commands.
* Parameterized procedures programmatically enforce separation between the “logic” of the operation and the “data” associated with the query. This prevents manipulations of the data from tampering with the query logic (a class of attacks that include “SQL Injection Attacks”).
* Any interactions with an operating system command line interface (such as interpreter or shell) MUST use parameterized calls to the operating system that prevent user provided input from being interpreted as commands.
* Operating system feature invocation must programmatically enforce separation between “logic” of the operation and the data in the query. This prevents manipulations of the data from changing the query logic

### Body of Evidence

#### Story 1

In the early days of one product, the web UI used dynamic SQL in many of the queries that were run. A QA Engineer found that he could perform a SQL Injection attack against the login page. Luckily it was a QA engineer working for the company and not a “bad guy”.

#### Story 2

Heartland Payment Systems was a major midwest processor of credit card payments. An internet exposed back end system failed to (a) sanitize input and (b) use paramaterized procedures to execute SQL commands.

An attack on this interface used SQL injection to introduce "network sniffing" software deep into Heartlands infrastructure. As a result, it is estimated that 130,000,000 (!) credit card numbers were compromised. Even not accounting for actual fraud, simply re-issuing a credit card with a new number costs between $5 and $30 per card.

Heartlands stock price, which was trading as high as $30 per share, dropped to as low as $4 per share after the breach was made public.

## Transactions involving sensitive data must be auditable.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.6.1 | Scorecard Rule 249 | All transactions involving sensitive data MUST generate an auditable record. |

### Body of Evidence

#### Story 1

During a high profile breach involving an acquired company, an investigation was launched to try to identify potentially fraudulent use. Tens of thousands of potential fraudulent actions were initially identified. Detailed logging on the systems with detailed log information allowed all but %.002 of these potential fraudulent actions to be be correctly classified and moved from a "high risk" category to a "low risk" category.

The "real world" implications of this reclassification included significant reductions in risks of litigation, and significant reductions in remediation costs (high risk remediations were $60 per instance, low risk remediations were much less, resulting in over $1M savings.

#### Story 2

A flagship customer facing web application suddenly started experiencing very slow performance and total outages. Detailed log records allowed investigators to quickly realize that the system was undergoing a distributed dictionary attack (a collection of compromised hosts at many different IP addresses were making wild guesses at valid ID's and Passwords for system accounts, submitting millions of requests hoping to find an random correct guess).

The sheer volume of these requests (coupled with the fact that authentication and authorization were not separated, and that a failed authentication request consumed significant database resources attempting to retrieve authorization information) meant that the dictionary attack turned into an all out denial of service attack.

Analysis of the log file entries allowed the InfoSec group o reactively start blocking IP addresses that were part of the attack, preventing the traffic from hurting the web application. Quickly, the denial of service aspect of the attack was resolved (as the worst offending IP's were quickly blocked).

Eventually, the dictionary attack aspect of the problem was resolved as well. It is likely the attackers grew tired of trying to recruit new botnet systems.

## Credit card payment account numbers must not be manipulated.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.7.1 | Scorecard Rule 250 | The system MUST NOT store, transmit, or process PAN's (i.e. Credit Card Primary Account Number). |
| 1.7.2 | Scorecard Rule 1806 | If your system accepts credit cards or payment account numbers, you must be using (or be) one of the approved and tracked systems for doing so. |

### Body of Evidence

#### Story 1

Needing to follow highly invasive, ever changing, and always expensive to follow PCI rules, will result in a high cost and bad customer experience. So access to that type of data must be constrained and if possible eliminated altogether.

#### Story 1

Company A was not compliant with the PCI rules (that apply when you touch a credit card number). They were breached, and were taken to court. As a result they were held responsible for all expenses relating to the breach, including unauthorized charges, and costs to reissue consumer cards. For example, Target agreed to a $39M settlement with the credit card companies because they did not feel they could prove they were fully PCI complaint at the time of the 2013 breach (even though before the breach, they thought they were cmpliant).

## Highly regulated data (SPII) must be protected.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.8.1 | Scorecard Rule 1807 | The system must not use any highly-regulated (SPII) data. |
| 1.8.2 | Scorecard Rule 255 | The system must mask any highly-regulated (SPII) data wherever possible. |
| 1.8.3 | Scorecard Rule 1808 | The system must encrypt highly regulated (SPII) data when it is transmitted or stored. |

### Examples

* A UserID does not cross the “sensitive” threshold and should be logged in order to facilitate security and problem resolution.
* Production user passwords, Social Security numbers, Drivers License Numbers, and Credit Card numbers are all current examples of sensitive and should be masked before logging.
* A web application cuts a log record into a non secure repository for a credit card transaction. The record includes (among other things) the first 12 digits of the credit card number replaced with the "\*" character, and the last four digits intact.

### Body of Evidence

#### Story 1

Various applications were capturing customer search strings in poorly controlled logging repositories. This is inconsistent with various contractual obligations and stated policy controls. A project to find and remediate these repositories is estimated to cost 10-100x the original cost of the project.

#### Story 2

Credit card companies created and issued (in a vacuum) a new set of rules for anyone accepting credit card payments. One of these rules included precluding any plaintext storage, display, or transmission of Credit Card numbers. A Company compying with the previous set of requirtements (and no more) were forced into a choice:

* Spend $1.2 Million remediating internal systems.
* Risk being assessed fines of up to $40,000 per day for non compliance.
* Walk away from $120 Million per year (!) in automated customer payments.

#### Story 3

For the data breach story du jour see (<http://breach.scmagazineblogs.com/> ) or (<http://catless.ncl.ac.uk/Risks> )

## Sensitive data must not be placed in a URI..

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.9.1 | Scorecard Rule 256 | Sensitive user data must not appear in a URI. |
| 1.9.2 | Scorecard Rule 604 | HTTP(S) links to third party sites must not disclose sensitive information (for example session or authentication information due to GET parameters in a "referrer" field). |

### Body of Evidence

#### Story 1

If a referrer field, bookmark, or analytics element contains sensitive data, it could inadvertently leak all information necessary to impersonate one of our customers.

#### Story 2

If a referrer field, bookmark, or analytics element contains sensitive data, it could inadvertently leaks a customer’s private information.

## Standard encryption implementations must be used where available.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.10.1 | Scorecard Rule 257 | Encryption implementations must use approved standard solutions 6e. |

Encryption implementations are notoriously difficult to get right and very expensive to build. It is therefore best to do them once and do them right.

Encryption is only obfuscation unless it includes all of the following:

* Strong Keys (random with sufficient entropy).
* Rotating Keys.
* Securely managed Keys.
* Revocable Keys.
* Tamper Resistant Keys (i.e. keyfiles carry checksums validated off box).
* Auditable Key Lifecycles (generation, distribution, access, revocation).
* Secured Key Storage and Access (i.e. Etrust or isolated networks).

### Body of Evidence

#### Story 1

A Product initially used a locally-written encryption implementation. When that developer left the resulting intellectual property confusion delayed a release until the encryption could be replaced with a $70k licensed package.

## Production changes must be reviewed, approved and auditable.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.11.1 | Scorecard Rule 560 | All production changes must be reviewed, approved, and the approvals documented and maintained for at least 2 years |
| 1.11.2 | Scorecard Rule 747 | All production destined code changes must be reviewed by another qualified party before being put in production, and are those review results documented and maintained in a way an auditor could review |
| 1.11.3 | Scorecard Rule 561 | All source code must be submitted for source code security defect analysis at least yearly |

All source code additions or modification must undergo independent review before being integrated into a production release. It is important to review code changes to ensure security defects are not introduced. The individual reviewing source code changes must be familiar with the product, knowledgeable about the technology in use, and abreast of secure coding practices. The review must be independent, it may not be only the individual who made the changes being reviewed.

All source code must be submitted for source code security defect analysis at least yearly. In addition to manual code change reviews for security defects, automated code defect scanners may find security defects. These tools are quick and simple to use, and have very broad coverage in terms of the kinds of defects they find and the amount of code they can cover. Full and detailed analysis of the results can be time consuming, but a quick review is simple and it has a reasonable chance of revealing any glaring security problems. There may be source code technologies that are not capable of being scanning. These source packages should be submitted for scanning anyway, and that will satisfy this golden rule. It will be up to Security Architecture and Information Security to decide where and how to proceed in these cases, and that is also outside the scope of this golden rule.

### Body of Evidence

#### Story 1

Many audits requires evidence of a change review. If we can’t produce it, we fail the audit resulting in findings that may be expensive to remediate

#### Story 2

A malicious or disgruntled employee introduces a deliberate defect into the code, which makes it into production, and the result is a violation of customer privacy or regulated data.

## 3rd party software must be used safely.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.12.1 | Scorecard Rule 562 | COTS components of the asset must be registered. |
| 1.12.2 | Scorecard Rule 597 | COTS components of the asset musdt not be “prohibited”. |
| 1.12.3 | Scorecard Rule 1809 | All licenses for COTS components of the asset must be tracked |
| 1.12.4 | Scorecard Rule 799 | All hosts associated with the asset must report the software installed on it (and any usage related to licensing) in an automated fashion |

### Understanding controlled sources

Once reviewed, code must be controlled and managed. This precludes implementations where code we send to our users browser causes them to dynamically load content (for example JavaScript routines) from unapproved site. For example, Google.com, Googleapis.com, Webink.com, Webtrendslive.com, Cloudfront.net, Google-analytics.com, Optify.net, Cetrk.com, etc).

Causing users to dynamically load code or content from untrusted sources opens up significant privacy and security risks. The untrusted source, even if OK today, could change in the future to include a variety of straightforward attacks (for example causing the user to be prompted to download and install malware).

Causing a server to dynamically load code or content from untrusted sources is even worse, as we are then directly delivering that potentially malicious content directly to a customer. For example, java.sun.com/jsf/facelets, java.sun.com/xml/ns/j2ee, etc.

### Products and services must only load code from trusted and controlled sources.

For example, if we wish to use a third party open source Javascript library on a page, we should host that library and serve it from our systems, not cause the user’s browser to load the library directly from the third party repository. Or if a server wants to load a DTD, it should be loaded from the same domain as the application, or from some other approved source.

### Frequently Asked Questions

#### Does this mean that if I am using a JavaScript library such as JQuery and loading it from Google's CDN that I broke the rule?

YES!!! This is because Google and sites like it are considered an abstract risk. At any time, Google could be compromised, and at that moment, the next request we get to pull this content would contain something bad. This could be an attacker compromising Google, Google deciding to be a little more evil, or someone like the NSA exploiting that avenue to compromise us. So we shouldn’t do that. We should serve up our own code if we are including that code in our products.

If there is a reason that simply isn’t practical, we can do a security assessment on the third party and decide if they are trustworthy enough to take on this risk. So we would try and declare Google a “controlled source” by doing an NDA and a Security Assessment. That is a much less desirable path though, as it has to be renewed yearly (eating time and resources), and it doesn’t make risk go away, it just tries to measure it to see if the bleeding would be arterial. Much better not to bleed at all if possible.

#### Did you really just call me stupid?

Not technically. It just raises the question, not answer it.

Maybe there is a good reason to do this. The statement is there because in at least one case, it could be stupid. Say hypothetically a group built a product for customer use that added dynamically loaded javascript for three different sites because it "did really cool stuff". Then say, again hypothetically, it turned out it did some not so cool stuff also... and by the time we would have finished running down the rabbit hole, say we found more than 15 different domains serving up various pieces of javascript, web beacons, and tracking cookies to every customer that came to that system. What if some of those domains included yahoo adds, doubleclick, and half a dozen sites we knew nothing about? That would be pretty stupid.

### Body of Evidence

#### Story 1

A third party javascript library used in an application is compromised, and ends up installing cryptolocker malware on customer networks

#### Story 2

A third party analaytics package tracks private customer information, and resells it to other parties, and customers object

#### Story 3

A piece of software that is free for nocommercial use is integrated into for-sale products, but has extreme license terms for commercial use, and we are forced to settle with them for a substantial penalty

#### Story 4

If we use an Oracle product, and they make unreasonable and predatory license interpretation changes (again), an audit could result in significant fines or legal action

If a piece of software we use later has a published serious vulnerability, and we don’t recognize our exposure and patch it, we could get breached

## Internet facing security exposures must be caught before being exploited.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.13.1 | Scorecard Rule 753 | Assets must undergo appropriate penetration testing |
| 1.13.2 | Scorecard Rule 752 | Assets must be subject to network vulnerability scans at least quarterly |

### Body of Evidence

#### Story 1

If an application security defect lingers in an Internet facing product, it could be exploited and result in a security breach

#### Story 2

A JBoss console that was not requested by developers, and that developers did not know existed, gets unknowingly installed by a system administrator when installing other requested software (because it was added to the standard image). It has a default and generally known password. Since the developer did not specifically request JBoss they were unaware of its existance and therefore never changed the default password. As a result, an attacker discovers it, and uses it to install malicious software that gives them console access to our production network.

## Important security related events must be recorded and reported.

|  |  |  |
| --- | --- | --- |
| Number | **Scorecard Rule** | Rule Text |
| 1.14.1 | Scorecard Rule 754 | Successful and failed authentication events must be reliably captured and retained |
| 1.14.2 | Scorecard Rule 755 | Authentication credential management events must be reliably captured and retained |
| 1.14.3 | Scorecard Rule 796 | Sensitive transactions (privacy sensitive, regulated data sensitive, or cost sensitive) must be reliably captured and retained |
| 1.14.4 | Scorecard Rule 797 | Do the records include a reliable timestamp, originating IP address, user identity, and specific information about the activity wherever possible? |
| 1.14.5 | Scorecard Rule 765 | Are records retained at least 90 days online and for one year total? |

### Body of Evidence

#### Story 1

An ID is found to be compromised, but because there is insufficient logging, it and several other similarly compromised ID’s are undetected, and the compromise lingers.

#### Story 2

An attack against the application is successful after a long drawn out campaign because of a lack of sufficient instrumentation to detect and respond to it proactively.

#### Story 3

Many audits requires evidence that security related events are logged. Failure to produce such logs could result in failing the security audit (customer or FTC)resulting in increased cost and negative impact to reputation

## Production environments must be kept separate and secure.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.15.1 | Scorecard Rule 1794 | Any machine used for Internet web browsing may not have immediate and transparent write access to data masters. |

### Body of Evidence

#### Story 1

A malware infected desktop with write access to a production environment causes a production disruption (for example a cryptolocker infected desktop encrypts and locks a production file share).

## Standard authentication implementations must be used where available.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.16.1 | Scorecard Rule 228 | Authentication for customer facing strategic applications and services must be strategic authincation solution(s). Authentication for Internal / Enterprise applications and services must be via default user network identity. (NOTE: Customer-facing applications MAY be non-authenticated.) |

### Body of Evidence

#### Story 1

A new asset builds its own new authentication system, requiring authentication credentials to be manually maintained. Disgruntled ex-employees whose access credentials are not revoked after termination take malicious action and NPS suffers.

#### Story 2

A new asset builds a new authentication correctly, and no users on it will use any other LN product, but we have now doubled the number of opportunities for coding defects for major security vulnerabilities.

#### Story 3

Company XYZ has a collection of 1 million identities in one master identity repository and another 2 million identities in a second master repository. Half of these identities are the same humans, the rest are unique to one system or the other. Th company lost 6% market share year over year and urveys indicated that a key reason is because a competitor has an equivalent product offering, but a much less painful seamless single signon experience.

## Systems must degrade gracefully when attacked.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.17.1 | Scorecard Rule 609 | Internet facing elements must be resistant to denial of service attacks, either by use of a protective service, or by making sure the vulnerable parts of the site are protected (perhaps by authentication, or perhaps by making the site highly and rapidly scalable). |

### Body of Evidence

#### Story 1

An authentication system, in order to validate if a password that a user entered is correct or not, does a very computationally and I/O expensive operation to retrieve all data about the customer who is trying to authenticate (in addition to just the password check).  As a result, a fairly minor nuisance level dictionary attack that requires few attacker resources ends up causing a complete system outage.

#### Story 2

A single server is responsible for Domain Name Resolution, and it goes down due to a minor denial of service attack.  The entire product becomes unavailable as a result, not because it isn’t working, but only because it suddenly  becomes impossible for customers to know how to reach it.

## Infrastructure environments must be safe.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.18.1 | Scorecard Rule 742 | The infrastructure for the asset must have a current Security Assessment, it must have scored either Corporate Trust or Trusted Partner, and the \*SAQ must be published and available on request. |

### Body of Evidence

#### Story 1

A development manager, frustrated with the time and process overhead of getting vetted infrastructure resources, uses their corporate credit card to secure their own hosting.  Two years later, as a result of poor physical security practices or a failure of the development team to do patching, that environment and all of its data is completely compromised by an exploitation of a known and common vulnerability.

## Access to important systems and data must be managed.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 1.19.1 | Scorecard Rule 732 | If an employee separates from the company, is their access to important production systems automatically? |
| 1.19.2 | Scorecard Rule 575 | Is there a process in place to distinguish which users need access to important production systems, and grant it only to them? |
| 1.19.3 | Scorecard Rule 1810 | Are human access rights to important production systems reviewed at least yearly by managers? |
| 1.19.4 | Scorecard Rule 573 | Are changes to important production systems reliably traceable to the human who made them? |

### Body of Evidence

#### Story 1

A developer who needs access to sensitive resources works on 6 different products over an 8 year period. Then their PC is compromised by undetected malware, a reverse proxy to a root account is installed, and authentication credentials are captured.  That reverse proxy is then used to launch internal attacks at all 6 products (when the developer was only working on one of those systems at the time of the compromise).  The breach was 6 times larger than necessary had there been better managed access.

# Compliant

Intellectual Property has the shelf life of a banana – Bill Gates

It's a major milestone. … I am personally committed to full compliance. We are committed to being a responsible industry leader. – Bill Gates

Resistance is futile – The Borg

Whether you believe the current state of affairs regarding Intellectual property protection in the software industry is a good thing or a bad thing, it would be naive to ignore it and every company should have a clearly stated position, part of which is best governed through the architecture process.

Even seeming altruistic efforts like “Open Source”, have subversive elements and there are more than one case of company’s paying “undisclosed sums” out of court to settle cases where unsuspecting developers used Open Source in a manner inconsistent with its license.

While enterprise architects are certainly not lawyers, we unfortunately find ourselves in the position of needing to understand the difference between: patents, copyrights, trademarks, company secrets and what constitutes an invention and what does not.

Some companies choose to patent everything, even those things they license for free. Other companies file no patents at all. Still others act more subtly and subversively.

Managing the Intellectual property coming from the technology base of the company and ensuring that other company’s Intellectual property and licensing restrictions are not infringed is part of any well-governed enterprise architecture.

Most architectures / infrastructures incorporate data and technology from 3rd parties. These will always come with a contract of sorts which specify the rights and obligations associated with usage. It is a corporations duty and responsibility to comply with these obligations as well as to comply with applicable laws and/or consent decrees (e.g. Sarbanes Oxley or SOX)

## Compliant Golden Rules

Compliant solutions uphold the law and comply with regulations. They adhere to the provisions of the company's contractual obligations. They protect the Intellectual Property of the corporation and do not infringe the intellectual property of others.

|  |  |  |
| --- | --- | --- |
| **Number** | **Golden Rule** | **Description** |
| 2.1 | Intellectual Property must be protected. | Intellectual Property must be protected. |
| 2.2 | 3rd party IP must be used in accordance to its license. | 3rd party IP must be used in accordance to its license. |
| 2.3 | Source code must be stored in a secure and managed repository. | Source code must be stored in a secure and managed repository. |
| 2.4 | Customer facing U/Is must be accessible by users with disabilities. | Customer facing U/Is must be accessible by users with disabilities. |

## Intellectual Property must be protected.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 2.1.1 | Scorecard Rule 230 | Aggregations of data must not lose the intellectual property or copyright ownership information from the original data and content. |
| 2.1.2 | Scorecard Rule 231 | Company-owned source modules must contain an appropriate copyright notice, whether the source module was created by company employees, contractors, or external vendors. This includes client side executable components, such as plug-ins, web parts, JavaScript, Flash, and Silverlight. The following copyright notice is sufficient: "Copyright LexisNexis, NNNN". (where NNNN is the current year). |
| 2.1.3 | Scorecard Rule 233 | LN Produced documentation for any public API MUST include a copyright notice: "Copyright LexisNexis, NNNN". (where NNNN is the current year). |

### Examples

* Every document that is not in the public domain must be marked to reflect the copyright owner of that document.
* If a company handled document contains an attachment – such as a table, photograph or audio file – with a different intellectual property owner than the document itself, any presentation of that attachment must be accompanied by information identifying the intellectual property owner of that attachment.
* If a company derived document contains materials from more than one copyrighted IP owner than the materials from each IP owner must be marked to reflect that ownership.
* Each module of source code must bear a copyright notice.
* Any executable source code delivered as part of an HTML page – such as Javascript routines –must be marked to reflect intellectual property ownership.

### Body of Evidence

#### Story 1

Intellectual property that is not marked can be copied and claimed by anyone. This pertains to content, source code, and Javascript files as all of these are protected by patents, copyrights, and trademarks.

## 3rd party IP must be used in accordance to its license.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 2.1.1 | Scorecard Rule 230 | Aggregations of data must not lose the intellectual property or copyright ownership information from the original data and content. |
| 2.1.2 | Scorecard Rule 231 | Company-owned source modules must contain an appropriate copyright notice, whether the source module was created by company employees, contractors, or external vendors. This includes client side executable components, such as plug-ins, web parts, JavaScript, Flash, and Silverlight. |
| 2.1.3 | Scorecard Rule 233 | Documentation for any public API MUST include a copyright notice. |

This compliance must include, but is not limited to, aspects such as time-based or market-based embargoes, royalty payments, contract-mandated presentation behaviors, copyright rules, and time limitations on retention of copies by customers.

### Examples

* The content fabrication system might receive its copy of a certain publication’s issues before the publisher releases that issue for sale to the general public, while our license for that publication might specify that we must wait until three days after that date before making that issue available on our services. Such agreements must be honored.
* If our license for a publication restricts us from selling it in academic markets, then we must ensure that the publication is not included in content offerings to our academic customers.
* If we are obliged to pay a percentage of revenue associated with customer use of a given publication based on user accesses to the full text of the document, then instrumentation mechanisms must be used to record – and provide – user accesses to that publication’s documents sufficient for royalty calculations to be made.

### Body of Evidence

#### Story 1

Insufficient records for royalty accounting can result in large penalties, fines, and/or loss of access to vendor content.

## Source code must be stored in a secure and managed repository.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 2.3.1 | Scorecard Rule 260 | All source code MUST be maintained in an approved Source Code Management system and follow approved maintenance processes as defined by Source Code Maintenance Policy. |

Improper control over source and deployment undermines the integrity of many other controls.

### Body of Evidence

#### Story 1

The Industry has many incidents where source code was not in a managed repository and not backed up. As a result, the company in question lost the source code and were unable to rebuild the software

## Customer facing U/Is must be accessible by users with disabilities.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 2.4.1 | Scorecard Rule 434 | Applications must pass accessibility testing with favorable results. |

The Government has defined a set of techniques for [compliance with accessibility mandates](https://lngdevelopment.lexisnexis.com/sites/GSDNL/Shared%20Documents/Development/New%20Lexis%20application/New%20Lexis%20core%20application%20-%20M1/Reference%20documentations/Accessibility) when using various presentation technologies.

### Body of Evidence

#### Story 1

If companies do not comply with 508C accessibility guidelines, they will not be able to sell to US Government markets. Some geographies require that software comply with WCAG standards to be sold in their government markets. Failure to comply with this rule limits the markets that software can be sold

# Reliable

I value self-discipline, but creating systems that make it next to impossible to misbehave is more reliable than self-control. - Tim Ferriss

Simplicity is prerequisite for reliability. - Edger Dijkstra

Technology does not always rhyme with perfection and reliability. Far from it in reality! - Jean-Michel Jarre

A reliable system is one designed to avoid single points of failure. A system can only be as reliable as its weakest link. Hence in a highly modular and distributed systems, each component unit of that system must itself be highly reliable – or it becomes the weak link.

Core to a highly reliable architecture is catering for the fact that even under normal conditions; systems fail, so in order to achieve high reliability, there can be no single points of failure. A single point of failure is defined as a case where a single system failure can cause a loss of service (loss of service defined according to the business requirements of the service).

The most common technique for achieving reliable systems involves redundant components. Common techniques for managing redundant components are: load balancing, clustering and parallelism. For example: Two components may be in a clustered A/B configuration where the “B” component automatically picks up if the “A” component fails or they may be in a load-balanced A/B configuration, where “A” and “B” are both live and processing and the load balancer acts as a director such that if one fails, all requests are vectored to the live component.

Reliable services not only have physically resilient infrastructure, they contain software of high quality and all the human elements required for the service to operate have appropriate back-up.

Each system should define its availability target commensurate with the business problem it is solving and not fall into the trap of always requiring 24x7 operation with 99.999% (Five nines) availability. If the business model will support 24x6 and three nines than that’s a pretty good target since each additional “nine” required generally means a big uptick in cost and time to implement.

A disaster ready system is designed as reliable within a single site plus designed to be resilient to a site failure. This is not to say that a disaster ready system is automatically resilient to every natural or unnatural disaster. Deciding the set of disasters that a corporation can afford to be resilient to vs. the associated risk of not doing so is an important business decision.

Every system should define its recovery point objective (amount of data loss deemed to be acceptable in the event of a disaster) and a recovery time objective (amount of down time deemed to be acceptable in the event of a disaster) commensurate with the business model for the service

Disaster Readiness (D/R) provides the technical foundation for Business Continuity. Business Continuity adds the additional organization and process (on top of the technology availability) required to keep the business running in the event of those disasters to which the business has chosen to be resilient.

## Reliable Golden Rules

Reliable solutions provide measurable service in terms of responsiveness, availability and dependability during normal operation, as well as in failure scenarios and in the event of a disaster.

|  |  |
| --- | --- |
| **Number** | **Golden Rule** |
| 3.1 | Systems must meet performance and availability SLRs and Recovery Objectives. |
| 3.2 | Systems must have an appropriate plan for functional testing. |

Under extreme circumstances, where there may be multiple independent system failures, a service may remain reliable by treating the case as a site failure

## Systems must meet performance and availability SLRs and Recovery Objectives.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 3.1.1 | Scorecard Rule 504 | Applications and Services MUST document Recovery, Performance, and Availability SLRs for the asset and critical interfaces, including financial constraints for assets deployed to elastic computing Clouds, using a numeric form that can be measured, confirmed accurately, precisely, and consistently. |
| 3.1.2 | Scorecard Rule 507 | Applications and Services MUST define an Operational Plan that achieves minimally acceptable Recovery, Performance, and Availability SLRs, including data backups and testing cadence to assure the plan works within the specified financial constraints for assets deployed to elastic computing clouds. |
| 3.1.3 | Scorecard Rule 548 | Recovery, Performance, and Availability testing results MUST demonstrate the application or service meets minimally acceptable SLRs, within financial constraints for assets deployed to elastic computing Clouds, in both pre-production and production environments and the results must be retained and available for review. |
| 3.1.4 | Scorecard Rule 554 | Dependency analysis results MUST demonstrate the Application or Service will introduce no significant Recovery, Performance, and Availability impact on other Applications and Services. Dependent parties MUST be notified of SLR and Recovery Point Objective (RPO) changes. |

In order to increase confidence in Applications and Services, they must be tested to verify they meet expected Service Levels, and that they account for effects on depending and dependent Applications and Services.

### Applications and Services MUST test Performance SLRs

Tests MUST be performed in an environment which simulates the actual production environment, including all Applications and Services which could reasonably affect performance. Tests MUST collect and document performance measures compatible with Performance SLRs.

Performance testing validates the actual performance Service Level an Application or Service can provide.

### Applications and Services MUST be test Availability SLRs

Analyses MUST be conducted on Applications and Services as they are intended to be deployed in a production environment including all Applications and Services which could reasonably affect availability. Analyses MUST document availability measures compatible with Availability SLRs.

Availability analysis validates the expected availability Service Level an Application or Service can provide.

### The Recovery Plan MUST be analyzed for conformance with Recovery Objectives

Analyses MUST be conducted on Applications and Services as they are intended to be deployed in a production environment including all Applications and Services which could reasonably affect recoverability. Analyses MUST document recoverability measures compatible with Recovery Objectives.

Recoverability analysis validates the actual Recovery Objectives an Application or Service can meet.

### Applications and Services MUST be tested for the performance impact they will have on other Applications and Services they depend on

An Application or Service may depend on other Applications and Services. When that occurs, this Application or Service may introduce additional load on the Applications and Services it consumes, which may in turn have a negative impact on those Applications and Services.

Conversely, an Application or Service may undergo changes that affect its ability to support Applications and Services which depend upon it.

#### Example

A new Application depends on an existing Service. The Service is currently servicing 50,000 transactions per day. The new Application is expected to add an additional 75,000 transactions per day. The dependent Service must now handle 125,000 transactions per day.

#### Example

An existing Service has an average response time of 1.2 seconds. With a new version, the Service will now take an average of 1.5 seconds to respond.

#### Consequences

Failure to analyze the upstream and downstream effects an Application or Service may have on other Applications and Services may result in unexpected failures to meet other performance, availability, and recoverability Service Levels.

### Body of Evidence

#### Story 1

Failure to agree and document Performance SLRs for Critical Interfaces may result in a disconnect between the expectation of development / operations and the business sponsor. In these cases, Development and Operations are always "wrong".

#### Story 2

Failure to agree and specify Recovery Objectives greatly increases the risk that an Application or Service will remain unavailable for long periods of time. Data loss outside of defined RPO is likely to incur Customer dissatisfaction, lower NPS, financial loss and potential penalties.

#### Story 3

Having no plan to recover data, applications or a service will, without doubt, lead to financial penalties and bad reputation.

#### Story 4

Company ABC fully documented its performance, availability and Recovery Point Objectve (RPO) values, but did not test for them. As a result, they did not actually meet the documented RPO. This is akin to performing backups, and then not testing them….and then when you come to restore data, you find you cannot….because you did not test.

## Systems must have an appropriate plan for functional testing.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 3.2.1 | Scorecard Rule 767 | Any new and/or change to an application, component, service MUST be documented in a test plan, verified before deployment to production, and the results retained for independent review. |

### Body of Evidence

#### Story 1

Failure to document an Operational Plan may result in not being able to meet SLRs or RPOs. Not being able to deliver on SLRs and RPOs will impact customer satisfaction and have potential financial impact

# Scalable

Great businesses can be built on scale. I think Amazon has built a phenomenal commerce business largely on scale. Their network effect isn't obvious to me, but boy, have they used scale effectively. Jeff Jordan

The way to really scale a venture firm is with software. Sam Altman

A scalable architecture is one built to adapt cost-effectively to increases (and possibly decreases) in load. Load can come from multiple directions: from users, from more content or from more changes / transactions. Scalability is not a technology per se, it’s a model / plan. The act of being scalable is the act of defining, for each “scalability unit” in the system, the appropriate action to take, as the use / content of the system grows and the unit approaches its capacity limit.

If a module needs to scale proportional to user growth, then horizontal scaling is desirable (i.e. scale out – or adding more modules of the same type). Vertical scaling (scale up – or keeping the same number of systems, but making them bigger) is often adopted as an option for infrastructure growth that is independent of user growth when processing must be state-full, so that spitting across multiple instances is inconvenient or uneconomical. . Note that vertical scaling indicates that cost growth may become disproportionate with revenue growth and thus must be monitored (see Manageable).

## Scalable Golden Rules

Scalable solutions support load increases via a proportional increase in resources in a cost-effective manner.

|  |  |
| --- | --- |
| **Number** | **Golden Rule** |
| 4.1 | Systems must deliver acceptable performance under anticipated load. |
| 4.2 | Systems must optimize purchase of capacity. |

## Systems must deliver acceptable performance under anticipated load.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 4.1.1 | Scorecard Rule 1811 | Applications and services must efficiently balance load across available resources. |
| 4.1.2 | Scorecard Rule 1812 | Strategic applications and services must successfully complete a load test validating scalability and responsiveness under anticipated load. |
| 4.1.3 | Scorecard Rule 1813 | Application and service components must scale either horizontally (preferred) and/or vertically while minimizing disruption to service. |
| 4.1.4 | Scorecard Rule 1814 | Applications and services must degrade gracefully once load exceeds available capacity |

### Body of Evidence

#### Story 1

Users (customers or employees) experience degraded performance or failure as load increases adversely impacting revenue, customer satisfaction, productivity, and company reputation.

## Systems must optimize purchase of capacity.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 4.2.1 | Scorecard Rule 1815 | Applications and services must be implemented to add or remove capacity with minimal engineering costs and delays. |
| 4.2.2 | Scorecard Rule 1816 | Applications and services must minimize the size of each incremental increase or decrease in capacity or maintain efficient density. |
| 4.2.3 | Scorecard Rule 1817 | Applications and services must optimize the use of purchased capacity over the purchase period. 1) Applications and services with predictable fixed workloads should use low cost options based on volume discounts and/or long term commitments. 2) Applications and services with unpredictable workloads should purchase capacity to minimize long term commitments. |

### Body of Evidence

#### Story 1

We are unable to effectively match changing (growing/shrinking) demand to capacity on a timely basis resulting in general over-provisioning of systems and increased cost.

# Manageable

Bad software lives forever. Good software gets updated until it goes bad, in which form it lives forever - Casey Schaufler

Deploying Manageable systems generally involves the monitoring of the health of production systems, procedures to modify the behavior and functionality of production systems at run time and tracking any problems / incidents through resolution.

Generally speaking, manageability includes designing for resolving issues as an integral part of the design process and not an afterthought. This is often implies adding instrumentation and logging. It is only when manageability is designed in that we get the most effective use from our systems and the support personnel who administer them.

It is essential for implementations to provide quantitative, as well as qualitative measures of availability, responsiveness, capacity and reliability. These qualitative and quantitative measures cover a wide range of items including: Usage statistics, Error events, Exception events, Performance metrics, Availability metrics, Security Intrusions, Capacity thresholds, and Accounting events.

Even in properly designed and tested applications, failures do happen and these are often diagnosed at run-time. It is essential that applications take special care to log any and all exceptional events in a manner that allows harvesting for diagnostic use. It is also vital that the volume of logged information not overwhelm either the operator interface or available disk space such that important events are not lost in a fire-hose of non-actionable messages.

## Manageable Golden Rules

Manageable solutions have hooks to monitor, measure, and modify operational behavior and adopt, operational. These systems know what is happening before customers do. Issues are resolved before they become issues.

|  |  |
| --- | --- |
| **Number** | **Golden Rule** |
| 5.1 | Systems must respond to standard control commands. |
| 5.2 | Systems must publish appropriate operational events. |
| 5.3 | Systems must publish Performance and Capacity data. |
| 5.4 | An inventory of all system hosts must be available. |

The ability to disable an account, raise or lower the level of diagnostic reporting, etc. are all common practices in reliable run time management systems.

Where a service contract has specified Service Level Agreements (SLAs) or Operation Level Agreements (OLAs), these should be specifically monitored.

## Systems must respond to standard control commands.

|  |  |  |
| --- | --- | --- |
| Number | **Scorecard Rule** | Rule Text |
| 5.1.1 | Scorecard Rule 525 | Applications and Services MUST respond to start/stop commands. |
| 5.1.2 | Scorecard Rule 526 | When high availability and graceful shutdown is a priority for an Application or Service to avoid or minimize customer impact they MUST respond to commands to stop/start a single node with minimal or no customer interruption. |

### Bad Outcomes

#### Story 1

System ABC had increased application incident recovery times due to lack of standard control commands in instances where system issues required an application bounce.

#### Story 2

Infrastructure resource maintenance (such as patching hosts) or code releases impact customers in the form of reduced availability and potentially the loss of some customer data.

#### Story 3

A Mobile app gets released to an app store, Apple or Google Play, and it has a security issue, or some other issue where it is necessary to discontinue use of the app and/or force the user to update the app. Neither Apple, nor Google (Android), provide a mechanism to do this beyond removing the app from the app store and preventing new users from installing the app.

#### Story 3

The initial release of a mobile app was improperly configured for a production release allowing network requests to be made in the clear potentially exposing sensitive data. The issue was quickly discovered and an update was created and deployed to the app store; however, there was no way to force users, who previously installed the app, to update their version.

## Systems must publish appropriate operational events.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 5.2.1 | Scorecard Rule 528 | Applications and Services MUST publish operational events to support troubleshooting during the certification and production stages. |
| 5.2.2 | Scorecard Rule 527 | Cloud hosted Applications and Services MUST provide a way to correlate standard tags to operational events. |

### Bad Outcomes

#### Story 1

Without operational events, there is a risk of an outage occurring wherein the Incident Response teams cannot track back to a root cause. This situation can lead to long times to recover, meaning that KPIs are not met. In certain circumstances these can translate into both financial damages and reputational risk as well as unsatisfied customers.

#### Story 2

Security incident response, financial management, and application performance analysis depend on operational events containing the association between the Infrastructure and the asset. If operational events don’t contain standard tags these functions will be impaired or cannot be performed.

## Systems must publish Performance and Capacity data.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 5.3.1 | Scorecard Rule 530 | Applications and Services MUST publish performance and capacity data needed to support KPIs and verify SLRs. |
| 5.3.2 | Scorecard Rule 532 | Applications and Services MUST publish performance and capacity data needed for troubleshooting and root cause analysis of performance and capacity issues. |
| 5.3.3 | Scorecard Rule 531 | Applications and Services MUST publish performance and capacity data to support capacity planning and elastic scaling. |
| 8.3.1 | Scorecard Rule 529 | Applications and Services MUST publish instrumentation data using methods compatible with the environment in which they will be deployed. |

### Bad Outcomes

#### Story 1

Without publishing performance and capacity data there is the risk of a performance issue occurring wherein the teams cannot track back to a root cause. This can lead to KPIs are not met. In certain circumstances these can translate into both financial damages and reputational risk as well as unsatisfied customers.

## An inventory of all system hosts must be available.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** | |
| 5.4.1 | Scorecard Rule 798 | | The hosts MUST be part of an approved Inventory per the description in the Technology Owner responsibilities. |
| 5.4.2 | Scorecard Rule 1818 | | All physical/virtual systems must be inventoried and traceable back to the assets they support and to E2E team that manages them. |

### Bad Outcomes

#### Story 1

Without a clean host inventory, there is a risk that software installed which has security vulnerabilities, does not get patched. This creates a security risk. The risk is extended when software goes end of life and patches are no longer available.

#### Story 2

Without a clean host inventory, there is a risk that licensed software deployed is in excess of what is owned, since many system software titles require licenses allocated based on the number of processors or cores. If these are discovered in an audit (which are becoming increasingly likely), the damages often run into the $millions. For example: During an internal license verification, it was discovered that MS SQL Server had been inadvertantly overdeployed because there was a desire to move from physical servers with 1-5 SQL Server instances per physical server to 1 SQL Server instance per virtual server instance.

#### Story 3

Without a clean inventory, there is a risk that an outage cannot be tracked back to a root cause. This can lead to long times to recover, meaning that KPIs are not met. In certain circumstances these can translate into both financial damages and reputational risk as well as unsatisfied customers.

# Simple

Make things as simple as possible, but not simpler. – Albert Einsten

Simplicity is a core tenet of every architecture but measuring simplicity can be highly subjective and, a matter of significant debate. Simple architectures lead to simple designs and (hopefully) simple implementations.

Simple architectures begin with well-defined Assets that have clearly defined roles to play and clear boundaries. Lack of clarity is a driver of complexity. Once roles are clearly defined, keeping to those roles and avoiding the technological sprawl that comes from overlapping or partially overlapping roles is another driver of complexity. Often this situation occurs from acquisitions followed by the failure to integrate.

Given clearly defined roles for Assets in the infrastructure / architecture, proper Placement of Function maintains this clarity over time and is the number #1 weapon in the fight against entropy. . In other words, doing the right thing in the right place in the right way avoids the tendency to place functions in a system just because the people who build it have time on their hands or happen to own the budget.

Legacy systems may never go away, so while we have them we need to manage them. No sooner do you sunset one legacy system than you classify another one as legacy. Managing to simplicity implies that strategic systems cannot depend on legacy systems, but legacy systems can depend on strategic systems. If you don’t avoid the complicated string of linkage dependencies that occur from connecting new strategic systems to legacy systems you intend to shut off, you will never shut anything off (because something you need will depend on it)

Systems should adopt patterns / blueprints. The larger the system, the more critical the definition of a pattern / blueprint governing its design and the design of like systems. For example: Content systems may adopt the Content Distribution Pattern. Architecture Patterns define the system layers and the interfaces between those system layers. Patterns define the responsibilities of each layer in terms of what it is responsible for and each interface in terms of the contract and rules for crossing the encapsulation boundary. Complex architectural problems need not be designed from “first principles” each and every time.

## Simple Golden Rules

Simple solutions have clear responsibilities with little or no overlap. They tessellate. They are "as Simple as Possible and no simpler". They have proper Placement of Function, duplicate as little as possible based on organizational reality and follow well-defined patterns and blue-prints.

|  |  |
| --- | --- |
| **Number** | **Golden Rule** |
| 6.1 | Legacy Assets and Deprecated Interfaces must not be used by Strategic assets. |
| 6.2 | Code packaging must facilitate independent releases. |

## Legacy Assets and Deprecated Interfaces must not be used by Strategic assets.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 6.1.1 | Scorecard Rule 217 | Strategic assets and infrastructure MUST NOT be dependent upon legacy assets. |
| 6.1.2 | Scorecard Rule 218 | Use of legacy assets MUST be identified on architecture diagrams. |
| 6.1.3 | Scorecard Rule 546 | Strategic assets MUST NOT be dependent on non-strategic or deprecated interfaces. |

* Each domain shall maintain and publish a list of components it owns that are deprecated.
* A component must have a viable functional equivalent identified and in service before it can be deprecated.
* Keep the system simple by not carrying the old code forward as part of the future code base.

### Consequences

You will never be able to retire old systems because the new systems cannot run without it. That leads to inefficiencies as older systems are unable to efficiently be extended, maintained, and evolve. SLAs may be violated and new development may be fettered.

### Bad Outcome

#### Story 1

Despite multiple attempts spanning decades, many companies never retire classic systems that exists on old and expensive mainframes.

#### Story 2

[http://www.technewsworld.com/story/Cleaning-Out-the-Closet-What-to-Do-With-Those-Worn-Out-Legacy-Systems-66355.html wlc=1236592728](http://www.technewsworld.com/story/Cleaning-Out-the-Closet-What-to-Do-With-Those-Worn-Out-Legacy-Systems-66355.html?wlc=1236592728)

This article talks about how legacy systems, intertwined in the infrastructure of companies operation become the root causes of regulation issues, security issues, performance issues and just plain lack of agility, speaking to a number of other Golden Rules in the code.

#### Story 3

When building a new strategic system, in order to meet aggressiive timeframes, it was decided to pull data from an existing legacy system rather than go directly to the strategic source. This resulted in a prolonged and complex project to retire the legacy system, given the complex spider-web of dependencies. Additionally, as there was no investment in the legacy systems, they became a source of compliance, performance, and reliability problems for the strategic systems that depended on them.

## Code packaging must facilitate independent releases.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 3.4.3 | Scorecard Rule 795 | The release and deployment of a system asset must be handled by a single release team and a single end-to-end team, respectively. |

Physical packaging and deployment boundaries enable loose coupling that simplify lifecycle and release management. Use of dynamic libraries and services enable clients and providers to evolve independent of one another, so long as compatibility is maintained. This, in turn, avoids the need to perform costly wholesale deployment of all artifacts.

### Consequences

Poor physical separation of concerns increases overall system coupling and often results in monolithic solutions that are costly to adapt to varying degrees of change.

### Bad Outcome

#### Story 1

In order to avoid the cost of buying new hardware, a new asset was deployed onto the same hardware as an existing asset, because it seemed to have spare capacity. Instead of abstracting away the hardware through virtualization, it was felt to be quicker to just share the box. Over time, the two systems outgrew the capacity of the box, but as they were not logically separated, upscaling carried forward the joint deployment. Coupling between the assets hindered agility by forcing monolithic release of changes. Scaling flexibility was compromized because shared infrastructure scaling had to be negotiated across all assets sharing resources. Cost allocation and decommission planning was overcomplicated as well.

# Modular

I intend to describe for your illumination the most common cases in which the “average” computing scientist fails to separate the various concerns. In doing so, I hope and trust that my colleagues in the profession do interpret this as an effort to help them rather than insult them. – E.W. Dijkstra

A modular system is one that falls apart easily! - E.L. (Ted) Glaser

Decomposition into smaller pieces is a fundamental approach to mastering complexity. The trick is to decompose a system in such a way that the globally important decisions can be made at the abstract level, and the pieces can be implemented separately with confidence that they will collectively achieve the intended result. – Jim Horning

Every technology problem we seem to face today is too large to solve all at once and all in one system or by one team. It used to be the case that Computer systems were bigger than computer problems. So we ran multiple computer problems on a single system . Now we run a single computer problem across grids or clouds of computers. The bottom line is that the problems we are solving with computers have gotten so large that most effective way to solve them is to first decompose them into smaller discrete problems.

This decomposition is hierarchical. We have domains. Within domains we have assets within assets we have asset areas. Within asset areas we have modules. An asset area is generally implemented as a micro-service. Modules can be further decomposed into sub-modules as necessary. It is critical that when a problem is decomposed according to this model, it must be done based on the problem being solved not on any pre-conceived notion of what the solution should look like. How far do we go in this seemingly endless decomposition? This is like asking: “How long is a piece of string?” You decompose far enough so that the size of the largest problem is small enough to fit into one brain.

Modularity has many benefits. Modules have well-defined boundaries providing encapsulation. Crossing the boundary can only happen via pre-defined interfaces. Well-defined interfaces, that adhere to the rigors of proper versioning and strong type checking support a type of interoperability we call loose-coupling.

Loose coupling is defined as the manner of integrating two modules (across an interface boundary) in such a way that each can upgrade independently of the other and either side can upgrade first without a dependency on the other making a change. In a very real sense, a loosely coupled interface cements a contractual relationship between a service provider and a service consumer. This is often called the interface contract.

Loosely coupled modules can be developed and tested independently. This is called unit testing. It is also called bottom-up testing because by testing the functional units independently and then re-combining them into larger units, we are proceeding from the bottom up in the structure chart of the design.

Modularity is also a key enabling factor for reusability, extensibility and scalability. Component units are aggregated to provide a complete service or solution.

Modularity extends beyond functionality or business logic to content as well. In other words, maintain proper encapsulation of content and treating content modules almost the way you treat software modules, with regard to loose-coupling, and re-use leads to another important concept called master data management

## Modular Golden Rules

Modular solutions employ Separation of Concerns. They divide labor among encapsulated components that are loosely-coupled via well-defined interfaces, entities, and data models. They do not share infrastructure.

|  |  |
| --- | --- |
| **Number** | **Golden Rule** |
| 7.1 | Assets must expose and consume only well-defined External Interfaces. |
| 7.2 | External Interfaces must be versioned and well managed. |
| 7.3 | External Interfaces must be easily consumable. |
| 7.4 | Systems must not be tightly coupled to their infrastructure / environment. |
| 7.5 | External Interfaces must not be tightly coupled to implementation details. |

## Assets must expose and consume only well-defined External Interfaces.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.1.1 | Scorecard Rule 348 | Each System Asset MUST define its logical structure using the company UML diagramming Standards and maintained accordingly. |
| 7.1.2 | Scorecard Rule 466 | Consumers of services / web applications MUST only use documented external interfaces designed and registered for that purpose. |
| 7.1.3 | Scorecard Rule 784 | System assets MUST NOT expose external service interfaces of type “Q” (SQL). |
| 7.1.4 | Scorecard Rule 783 | External service interfaces running atop HTTP(S) must be REST services. |

Modularity forms the basis for loose coupling which makes complex systems simpler to understand, maintain and change. This can be likened to the [Single Responsibility Principle](http://en.wikipedia.org/wiki/Single_responsibility_principle) for OO design. Modularity also minimizes redundancy and increases the likelihood of reuse at different levels of abstraction.

### Consequences

Failure to enforce modularity leads to monolithic systems that are costly to maintain and brittle to extend. Even relatively simple changes to monolithic systems become increasingly complex and error prone over time. These systems are often described as being a [Big Ball of Mud](http://en.wikipedia.org/wiki/Big_ball_of_mud).

### Examples

Java packages that treated all classes as ‘public’ lack well defined interfaces that encapsulate implementation details. Applications that fail to follow some form of [MVC Pattern](http://en.wikipedia.org/wiki/Model%E2%80%93View%E2%80%93Controller).

**Bad Outcome**  - Likelihood=High, Impact=High

System X decided it was easiest to integrate with System Y via backend database to get the information it required. System Y alters its database (unaware of System X usage) and System X breaks. This results in availability issues, lost time (increased support effort) and missing key milestones (as resource pulled out of other efforts to fix).

## External Interfaces must be versioned and well managed.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.2.1 | Scorecard Rule 725 | All external interfaces exposed and called by an asset MUST be registered in the Asset Interface Exposed & Called registries. |
| 7.2.2 | Scorecard Rule 159 | Major external interface versions MUST remain available for a minimum of three years following availability of the next major version or until all client dependencies have been removed. |
| 7.2.3 | Scorecard Rule 161 | Services MUST be deployed so that incompatible versions of their interfaces don't interfere with one another. |
| 7.2.4 | Scorecard Rule 162 | Services MUST be deployed so that compatible versions of their interfaces replace one another. |

Agility necessitates change. Proper upfront design can minimize the risks, complexity and costs of making likely changes, but not eliminate them. Such designs accommodate change by extension or other means without breaking existing clients.

### Consequences

Failure to anticipate the impact of probable changes to a system as it evolves often leads to frequent piecemeal changes that maintain compatibility but degrade overall design quality ([Big Ball of Mud](http://en.wikipedia.org/wiki/Big_ball_of_mud)) or introduce breaking changes that increase complexity and costs.

### Examples

Interfaces that exchange messages using XML that fail to make use of explicit schemas, fail to version the schemas or define rigid schemas that fail to incorporate extensibility points using mechanisms like xs:any or xs:anyAttribute.

### Details

* Applications, Services and the Modules that make them up MUST follow naming standards that convey their boundaries and capabilities. Consistent use of common nomenclature and naming conventions make names meaningful within each level of abstraction, without them names become nothing more than a [Tower of Babel](http://en.wikipedia.org/wiki/Tower_of_babel).
* Services MUST avoid coupling their consumers to the code used to implement them. Like all modules, services should shield their consumers from the details of their implementation using proper encapsulation.
* Services and consumers MUST be loosely coupled to the operational hosting environment. Services and consumers are often deployed across multiple hosting environments (development, cert/test, production) over the course of their lifecycle. As a result, they must be configurable to their environment. For Example: Consumers that hardcode the absolute locations of services are tightly coupled to a single operating environment.

### Bad Outcomes

**Bad Outcome 1** - Likelihood=High, Impact=High

System A exposes an un-versioned interface which System B,C,D and E utilise. System B requires more information (in request and response) than System A interface currently supplies so System A modifies the interface method to cater. All calls from system C, D and E break as they are not adhering to the interface specification. Initially resulting in availability issues for system C,D & E and/or future problem of lockstep changes to all systems - which impacts speed of delivery.

**Bad Outcome 2** - Likelihood=High, Impact=High

System A exposes an external public interface that is later exposed to customers as part of an API product. The customer builds the API into its systems (as desired) and is now dependent on the API provider (a good thing). But it turns out that the design of the interface did not completely hide the implementation and new requirements forced changes to the API which could not easily be done in a backwards compatible manner. Changing the interface forced changes by the customer, who simply refused to make the change. “They say, if you make me make this change, I might as well go to your competition”. The company and the system was stuck with the old, poorly designed interface (and the underlying implementation) forever.

**Bad Outcome 3** - Likelihood=High, Impact=High

System A exposed an interface that was used by multiple other systems. New requirements came in requiring changes to the exposed interface. It was a lot of work to do these changes in a backwards compatible , so instead the developer chose to take a shortcut in order to make the sprint goal and just assume the API consumer would make the corresponding change since they sat right next door and were making changes in the same sprint. But as time went on, the exposed interface got used more and more and it was no longer possible to get everybody to change in lock step. So either the build breoke and the sprint goals got missed, or the API would have to go through a major redesign and the PI goals would get missed. “Pay me now or pay me later”.

## External Interfaces must be easily consumable.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.3.1 | Scorecard Rule 350 | External interfaces MUST document their specifications using a standard solution. |
| 7.3.2 | Scorecard Rule 727 | An interface must conform to the use of custom headers and media types that were previously defined by other interfaces and register new custom HTTP headers and media types for the benefit of others. |
| 7.3.3 | Scorecard Rule 464 | External service interfaces MUST NOT allow distributed transactions across interface boundaries. |

Interfaces should be based upon key abstractions (discrete capabilities) from the subject matter domain that make them intuitive and durable to change. These abstractions should appeal to a broad collection of heterogeneous consumers who may make use of them in ways not originally intended. Such serendipitous reuse is often the genesis of innovation.

### Consequences

Poor choice of abstraction often lies at the root of complex interfaces that impede adoption and are error prone to use. In some cases, early abstractions give way under the weight of increasingly complex requirements. A common pitfall is to continually extend the early abstractions until they break, when for example adding new (broader / higher order) abstractions may have better addressed the growing complexity. In other cases, starting over is the right answer.

### Examples

Interfaces that attempt to be all things to all parties by continually adding optional parameters that fundamentally change their behavior. Interfaces that require inputs that can only be found deeply embedded in the response of some prior call.

### Details

* Interfaces MUST describe discrete capabilities modeled upon domain specific abstractions that are loosely coupled to implementation. When each interface describes a discrete domain specific abstraction its surface area shrinks making it more readily understood. Such interfaces become minimal, but complete. One can liken this to the notion expressed by the [Interface Segregation Principle](http://en.wikipedia.org/wiki/Interface_segregation_principle).

Furthermore, interfaces modeled on key domain abstractions tend to be more durable to change and naturally fit into a larger programming model which makes it possible to combine interfaces to solve larger problems.

Finally, use of abstractions also lowers coupling, since abstractions naturally hide implementation.

* Interfaces at the same level of abstraction MUST be consistent by adhering to common conventions, patterns and standards. Common naming, nomenclature, conventions, patterns and standards lessen the burden of learning what capabilities are available and how to make use of them. Consistency makes it possible to apply lessons learned in one interface to the next, thus flattening the learning curve. Within an interface the concept of [Progressive Disclosure](http://en.wikipedia.org/wiki/Progressive_disclosure), makes it possible to "show a small number of features to the less experienced user to lower the hurdle of getting started and yet have a larger number of features available for the expert to call up".

The combination of these design factors reduces the effort needed to consume an interface.

* Interfaces MUST be closed to change and SHOULD be open to extension. Interfaces should be explicit about what cannot change as well as how and where extensions may be added. By constraining extensions, interfaces maintain control while expanding reuse opportunities. One can liken this rule in some sense to the [Open/Closed Principle](http://en.wikipedia.org/wiki/Open/closed_principle) in Object Oriented Design (OOD), although in this case it is not achieved using inheritance.
* External interfaces MUST be interoperable and adaptable to a variety of clients, devices, platforms. The enterprise is heterogeneous and there shouldn't be any second class citizens with regards to interoperability. Interfaces must also adapt to new classes of client devices who make use of them over a variety of protocols that may impose network latency and bandwidth constraints.
* Public (External:Open) interfaces MUST provide consistent support. To be successful public APIs require a large commitment to supporting them in a consistent and predictable way. It is unreasonable to expect our customers to navigate different support systems just because we lack discipline. For Example: An API was released that failed to account for customer support. Instead they relied upon development staff that lacked customer support training, tools and were dedicated to the development of the next release.

### Body of Evidence

#### Story 1

When building a product, the development team was forced to learn and integrate numerous inconsistent services. It took much longer and cost mor for both development and QA than it would have if the interfaces had all been consistent and standardized. They also had to write individual test harnesses for each unique API in order to ensure that they were working properly again, very expensive rather than at worst one parameterized driver.

#### Story 2

In one product they used information out of a second product. Their original uses employed a link that had a very temporary liefespan -- it was not intended for use as a permanent URL. Later, the product tried to use the URL only to find that it failed.

#### Story 3

In one product, there was a process for capturing URLs from a second product but they did not use Permalinks, they just captured the URLs as they saw them. Later, when customers attempted to reuse the persisted URLs they wouldn t work and customers called Customer Support to complain.

#### Story 5

Google and Amazon and others started with SOAP-based web services and got very low adoption rates. They switched to REST interfaces, which could then be accessed equally well from programs in Javascript as from other languages and suddenly they got huge increases in adoption rates. Since AJAX style of applications rely on Javascript, Google and others were able to build more sophisticated applications providing a richer user experience based on leverage of RESTful interfaces. This example shows the value of RESTful interfaces as a standard approach to service access.

#### Story 6

In a Product, there were no specific standards for schemas. When they expanded the system, it was a nightmare to upgrade the schemas because there were no consistencies to guide the upgrades and migrations. No data was lost, but some data was inaccessible because the new schema was not compatible with the older one and didn t pick up some of the data. They also wound up inconsistently naming one of the fields resulting in other inaccessible data. Very expensive to change and retest, too.

#### Story 7

When moving into the web realm from a purely mainframe-based system back in the 90s, some business data structures were defined by the online development teams. When they later transferred control of these systems to the business systems organization, that organization changed the names of some elements which broke the applications which depended on the data, causing them to have to do additional rework -- all because there were no standards for naming.

**Story 8**  - Likelihood=Medium, Impact=Medium

System A calls Interface B of System C to apply a change to some data. Programmatic mechanisms do not exist to allow System A to know when System C has successfully fully processes the activity (long running). Therefore occurrences exist where either changes aren't applied (failure / fault unseen), or its felt to take too long and System A retries again (multiple times) where final result is not as desired as the change has been applied multiple times. This results in incorrect / corrupt information being visible to our end customers.

## Systems must not be tightly coupled to their infrastructure / environment.

|  |  |  |  |
| --- | --- | --- | --- |
| Number | **Scorecard Rule** | Rule Text | ID |
| 7.4.1 | Scorecard Rule 537 | Shared services, web application URL APIs, and their consumers MUST make parameters that change between operational environments (DEV, CERT, PROD, . . . ) configurable. | 537 |
| 7.4.2 | Scorecard Rule 786 | Any external interfaces, including user interfaces, must be exposed via an infrastructure-agnostic address scheme, such as ESB & DNS aliases, not IP addresses or machine names. | 786 |

### Bad Outcomes

**Bad Outcome 1** - Likelihood=Medium, Impact=High

System A wished to use functionality implemented in System B. System B did not expose a clean and simple interface to consume this (requiring access via System C & D) and that the data was already transformed into an alternative representation. This resulted in additional initial cost to utilise via C & D, negative system performance impact (and support costs) due to extra complexity.

**Bad Outcome 2** - Likelihood=Medium, Impact=High

System A wished to use functionality implemented in System B. System B did not expose a clean and simple interface to consume this (requiring access via System C & D)and that the content was already transformed into an alternative representation. System A decided to re-implement functionality. Future evolution meant that the duplicate implementations diverged, resulting in inconsistent behaviour (depending upon which route taken) and higher total cost of ownership to support both implementations.

**Bad Outcome 3** - Likelihood=Medium, Impact=Catastrophic

A Hardware failure occured and system X’s interface was explicitly tied to specific machine name / IP. Resolution required manual machine rebuilds / reconfiguring. With external interface addresses coupled to the hardware the effect was compounded as all consumers were exposed requireingrepointing to replacement machine / IP. The result was an availability impact (possibly customer visible) and increased operational expense to resolve.

**Bad Outcome 4** - Likelihood=Medium, Impact=Catastrophic

System X’s code contained hardwired machine names (for multiple internal and external system dependencies), to rollout the system. Development / certification / production required all of these settings to be tweaked and recompiled and then deployed. During this process a setting was missed (left as dev vs production) and another wasn't changed to reflect the new machine which system Y then used. The end result was a culmination of failed rollouts and wrong environment functionality and incorrect data / offerings.

**Bad Outcome 5** - Likelihood=Medium, Impact=Catastrophic

System X’s code contained hardwired machine names and when recompiling from the certification environment to the production environment. Either the wrong version of the code was retrieved and/or compilation activity didn't go 100% successfully. Things that worked in Cert no longer did in production and code analysis in the development environment could not determine the issues. This resulted in increased operational costs and negative customer impact of failed rollout attempts.

## External Interfaces must not be tightly coupled to implementation details.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.5.1 | Scorecard Rule 583 | An external interface must be built on a layer of indirection so that the asset’s internal data structures may change without necessarily forcing changes in the interface. |
| 7.5.2 | Scorecard Rule 408 | External service interfaces MUST NOT make use of 3rd party proprietary, commercial, and platform-specific elements that promote coupling. |

Messages form the basis for exchanging information via interfaces. As such, the messages should reflect the durable abstractions chosen by interfaces. At each level of abstraction messages must form a common canonical data model that can then be expressed in a variety of formats/representations.

### Consequences

The lack of a common data model creates a morass of integration complexity when attempting to exchange, master or mash-up data spanning different sources. It often requires the use of costly and complex transforms that lack fidelity (are lossful) and that become increasingly difficult to maintain and add little true business value.

### Details

* Message implementations MUST avoid [leaky abstractions](http://en.wikipedia.org/wiki/Leaky_abstraction) that couple them to implementation. Since messages are a key element of interfaces used by services it is just as important not to allow implementation details to leak into them. A common example of a leaky abstraction is the direct serialization of an implementation object graph into a message format. Serializations like these change as the underlying object model changes. They also often require consumers to navigate complex graphs for information that should otherwise be readily accessible given the abstraction in use.
* Messages at the same level of abstraction MUST minimize overlap, redundancy, or the need for complex transformations when used between requests. Messages should streamline and simplify conversations between consumers and providers not impede them.
* Messages MUST be consistent by adhering to common conventions, patterns and standards for a given format. Consistent use of common nomenclature, naming conventions and patterns increases opportunities for reuse and simplifies consumption.

### Bad Outcomes

**Bad Outcome**  - Likelihood=Medium, Impact=Medium

Consumer A wished to use many interfaces exposed from many systems of Company X. Each interface was exposed over different protocols, using different terminology and varying level of documentation / referencing materials. Consumer A saw this as a real pain as it took them much longer to use everything they wished. The End result was that Consumer A had a low opinion of Company X, which resulted in lower adoption and lost customers.

# Maintainable

The study of the art of motorcycle maintenance is really a miniature study of the art of rationality itself. Working on a motorcycle, working well, caring, is to become part of a process, to achieve an inner peace of mind. The motorcycle is primarily a mental phenomenon. Robert M. Pirsig

Maintenance is a function of every software system, once it goes into production. Therefore a part of the real cost of developing any software or any system is the ongoing maintenance required to keep it operational. This is not meant to imply that the system has bugs that escaped testing and need to be fixed. It may. But even disregarding that, there is always change to the surrounding infrastructure and eventually maintenance is required. In many real systems, over the course of their lifetime, the cost of maintenance can exceed the cost of initial development by quite a bit. Proper encapsulation, modularity, simplicity, adoption of standards and good, clean simple documentation all lead to maintainable systems.

A Maintainable system is by definition testable.

With all the methodologies for producing software more efficiently and more cost effectively, by far the best is to produce less of it! This is why re-use has achieved “holy grail” status in the software development industry.

## Maintainable Golden Rules

Maintainable solutions are easily supported and easily modified. They are able to be extended into adjacent functional areas with minimal surgery.

|  |  |
| --- | --- |
| **Number** | **Golden Rule** |
| 8.1 | An interface must be callable directly and not require a proprietary library. |
| 8.2 | Requests must be traceable from point of entry through all intermediaries. |
| 8.3 | Code, Schemas, and APIs must be appropriately documented and commented. |

## An interface must be callable directly and not require a proprietary library.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** | **ID** |
| 8.1.1 | Scorecard Rule 311 | An asset MUST NOT require its clients to use a proprietary library in order to access the asset. | 311 |

This is intended to eliminate the need for client artifacts required to access a system or its services. Any required stub code that must be used by the caller should be generated through a standard.

### Consequences

Client side artifacts and code increase coupling and complicate release coordination.

1. Coupling between assets hinders agility by forcing monolithic releases of changes
2. Increased cost to create and maintain client libraries in all needed programming languages and environments.

### Bad Outcomes

#### Story 1

If the access to a service is through an API that is only exposed through a hand-built client library that is provided by the service then this limits the use of the service. The service is forced to build, maintain, and test every OS/programming language combination in which all of the clients of the service are deployed. This leads to increase cost to create and maintain the client libraries in all the needed programming languages and environments. In order to create a loose coupling between clients and services, a hand-built client library must not be required to access a service.

## Requests must be traceable from point of entry through all intermediaries.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 8.2.1 | Scorecard Rule 312 | Transaction ID MUST be automatically assigned early in the request processing and passed through all service requests. |
| 8.2.2 | Scorecard Rule 314 | Failure events MUST generate log entries that include Transaction ID and User Perm id (if present) |

Failure explanations returned to the client should have as much information as the server log. This includes code module, version, and line number where the condition was detected.

### Consequences

Solving a customer problem and performing root cause analysis on a problem becomes nearly impossible without this type of instrumentation.

### Experiences

Violation results in endless questions from application and client developers, inability to reproduce errors, and inability to track and budget response time.

### Bad Outcomes

#### Story 1

A customer reported periodic slow response on the product s/he was using, expressing that it was a major pain point. Improvements had been made by extending the cache of the product and turning on SSL for all customer sites, but response time still did not meeting customer expectations. Continued investigation pointed to one of three possible causes: capacity/performance of servers, application code, or source database size. Product Management, Development & Operations, and Sales all had to meet multiple times to find the problem and assuage the customer.

#### Story 2

When a customer request is first processed, it is logged in the front-end server. The request may then wind its way through multiple systems as system makes an external call to another system to perform some function. If the user's request fails completely or fails an SLA, we can see this in the log file of the front-end server, but the actual root cause of the failure might have been six levels deep or possibly even a double failure where two systems in the call chain partially fail, but neither one fully fails. If the individual calls between systems are not threaded together in a way that allows a team to quickly identify all the calls in the chain through multiple log files, the problem may be difficult or impossible to identify and solve.

## Code, Schemas, and APIs must be appropriately documented and commented.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 8.3.1 | Scorecard Rule 315 | Schemas MUST be documented by references in comments to the documentation or embedded as comments |

Proper documentation of an XML Schema is required to understand the content or interface information that flows through it. XML Schemas are not self documenting, especially when complex structures or semantics are involved.

### Consequences

Lack of understanding of the XML Schema often leads to errors in the content or the interface it is used for leading to failures. These errors may be visible or silent. Often the silent failures lead to LN Data corruption or semantic interface errors leading to wrong answers.

1. Increased rates of errors in content and/or services
2. Increased development cost due to unavailable information regarding schema semantics

### Bad Outcomes

#### Story 1

As part of a project, customer account information was being extracted and exported from their products store and imported another into another system. The data Schemas for the products were known only through tribal knowledge (and at times, the knowledge was conflicting). As a result, the amount of effort required to extract the data and map it took 4 times longer than estimated..

# Mastered

With a tad more data modeling, engineers at NASA and Lockheed Martin might have settled on metric (used by NASA) or English (used by Lockheed) units of measurement and not lost the $125 million Mars climate orbiter in 1999 - Joe Maguire

The architecture must ensure that the data (all data) is correct and understandable to support customer decision-making. This commonly involves modeling and naming the data correctly, transforming supplied data consistently to its defined (normalized) form, preserving and protecting the data and providing it in a context that supports its understanding. The ability to break down the data, normalize and capture the meaning, relate it to important core entities and topics are among the key practices for data.

Data often passes through many way-points on its long and sometimes arduous path from the source to the end user. It is stored, transformed, transmitted, combined with other data, re-stored, re-transmitted, etc. But no matter how long and complicated a path that data has taken to reach its goal, the customer looking at the data has the right to demand and a data provider has the obligation to affirm the truth of that data. In order to do that,the principles of master data management must be embraced and every item of data must be transparently tracked through the systems it transits. In addition, a single data item can have only one source of truth.

Many disciplines come together in order to make the most of data. Subject matter experts, data modelers, linguists, software developers and data operation support are all involved. When working across such a diverse team clear naming and definitions are essential!

Even this may not be sufficient to keep the integrity of the data. Data reuse is a powerful force for creating value. However, in an environment where data is duplicated, preserving which instance is the master (the source of truth) is critical for preserving data integrity. This is the point of control for creation, maintenance, and deactivation.

The data world is not perfect! Using statistical sampling techniques to feed the quality evaluation of the data over time is a useful line of defense against hidden data corruption. Further, the use of Statistical Process Control (SPC) techniques can position the organization to further understand and continuously improve the data.

The days of making money from simply having a lot of data are gone! To play in this space in the 21st century, you have to add significant value to the data, by making it easier to find initially (Search), easier to find related information (Navigation) and the management of core Entities (i.e. specific items, not groups of things – like a person or a company or a place). These more advanced capabilities are significantly enhanced by a base of high quality data.

## Mastered - Golden Rules

Data (including, but not limited to business data, content, and operational data) is handled rigorously. Data items are mastered, and their transit through systems and way-points are carefully managed. Values, names, mnemonics, descriptors, classifiers, and other descriptors (Metadata) are handled with the utmost rigor.

|  |  |
| --- | --- |
| **Number** | **Golden Rule** |
| 9.1 | Data Assets and their system host must be registered. |
| 9.2 | Data Must be of high quality. |
| 9.3 | Data must be encapsulated. |
| 9.4 | Data must be traceable to its source. |
| 9.5 | Data must be validated. |

## Data Assets and their system host must be registered.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 9.1.1 | Scorecard Rule 563 | A newly identified data asset MUST be registered to its mastering system asset. |
| 9.1.2 | Scorecard Rule 1797 | A system asset that is a master for some type of data must be registered as such in the official registry of master data. |

### Body of Evidence

#### Story 1

Experience with products that derive data suggests that the lack of a single owner leads to inconsistencies and sub-optimal user experience. Recognizing these as data assets enables data governance to make these problems visible so that they may be prioritized and addressed.

Recognizing these as data assets enables us to use data governance to make these problems visible so that management can address them.

## Data Must be of high quality.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 9.2.1 | Scorecard Rule 566 | A system asset must obtain its data only from authorized sources. |
| 9.2.2 | Scorecard Rule 564 | A system asset must keep its data in sync with its sources. |
| 9.2.3 | Scorecard Rule 568 | A system asset must provide synchronization mechanisms for data it distributes. |
| 9.2.4 | Scorecard Rule 565 | A system asset that redistributes data must provide a faithful copy of the original. |
| 9.2.5 | Scorecard Rule 567 | A system asset must only populate data attributes/elements with data that is consistent with the design/definitions of those attributes/elements. |

## Data must be encapsulated.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 9.3.1 | Scorecard Rule 1 | A data asset may only be updated by one system asset. |
| 9.3.2 | Scorecard Rule 1798 | A system asset shall wrap its persistent data in a layer of indirection so that its data’s structures, keys, enumerations, and other internal details may change without forcing changes in the rest of the asset or its clients. |

## Data must be traceable to its source.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 9.4.1 | Scorecard Rule 331 | Data must be traceable back to the system which distributed it. |
| 9.4.2 | Scorecard Rule 458 | A system must identify data using an identifier designated as a preferred identifier by the data's master. |
| 9.4.3 | Scorecard Rule 24 | Content derived from other content items must be traceable back to the content items from which it was derived. |

## Data must be validated.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 9.5.1 | Scorecard Rule 1799 | An editorial system MUST ensure that content sets under its control adhere to the structure, schema, and/or validity constraints defined for the content sets. |
| 9.5.2 | Scorecard Rule 1800 | A master system asset MUST validate any master data before it stores it, unless the data comes from a trusted source. |

# Global

If you want to make peace with your enemy, you have to work with your enemy. Then he becomes your partner – Nelson Mandella

Never before has information been so important, to governments and businesses alike. …. Globalization means that the 'butterfly effect' is everywhere at work. – Jacques Chirac

A Global system is one designed to be capable of being localized for different languages, scripts, cultures, currencies, color conventions, rules and regulations without needing to rewrite / redesign the code. It is considered best common practice to do this via resource and script files that separate the language and culture dependent strings and functions from the language and culture agnostic business logic of the service / application.

Globalizing is distinct from localizing. That is, the implementation for a specific locale. For example: adoption of Unicode for string encoding and separation of strings presented to the user, from code, into resource files are examples of globalizing. Translating English strings into Japanese is an example of Localizing. Localizing is more than translating. Cultural issues are also important. For example, in western cultures the color green normally means “up” or “positive”, while red means down or negative. In some Asian cultures, these meanings are reversed. Iconography is another poster child for care when creating global systems. Any hand position and many facial expressions, no matter what it means in your culture is likely to be a curse in at least one other culture around the world.

Globalizing an application can be an expensive process, so don’t take it on lightly. But going back into a non-globalized system to globalize it, can be major surgery, so many companies will develop with globalization in mind, if they have even an inkling they might globalize. In this case, globalization becomes a core principle of the architecture, while localizing an application becomes a business decision taken on a case-by-case basis.

Systems / Applications are Global if they can support an appropriate subset of: Language Preference; Time Format Preference; Date Format Preference; Time Zone Preference; Text Data in Multiple Languages; Number Format; Text Direction

## Global - Golden Rules

Global solutions have the ability to be easily localized for use in a specific geography, including, but not limited to: language, script, culture, currency, color conventions, holidays, and sort-order.

|  |  |
| --- | --- |
| **Number** | **Golden Rule** |
| 10.1 | Systems must handle data in a globalized way. |
| 10.2 | 3rd party translations must be distinguished from company translations. |
| 10.3 | Systems must to adapt to the user's preferred locale. |

### Single-Locale Assets

If an asset is truly and forever intended to serve only the needs of a single locale, the Global Principle should not be scored. For example, it may be company policy for all employees to speak English. In such an instance a system that only faced employees does not need to be global.

## Systems must handle data in a globalized way.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 10.1.1 | Scorecard Rule 288 | System Assets that handle data fields representing text MUST represent such text using UTF-8, UTF-16, or UTF-32. |
| 10.1.2 | Scorecard Rule 290 | All text in UTF-8, UTF-16 or UTF-32 encoding MUST be normalized to a consistent Unicode Normal Form prior to any comparison or sorting operations. |
| 10.1.3 | Scorecard Rule 293 | System Assets MUST preserve language, region and script identification appearing in natural language text they process. |
| 10.1.4 | Scorecard Rule 294 | System Assets MUST preserve writing direction hints/instructions appearing in natural language text they process. |
| 10.1.5 | Scorecard Rule 295 | System Assets MUST support the exchange of locale-independent representations of date, time, currency and other numbers (but may also optionally support the exchange of locale-specific representations) |
| 10.1.6 | Scorecard Rule 610 | If a System Asset supports the exchange of locale-specific representations of date, time, currency or other numbers, such representations must be accompanied by a locale identifier compliant with the company's Technology Standard for Locales. |
| 10.1.7 | Scorecard Rule 429 | System assets MUST represent money in their external service interfaces as both a numeric value (representing the amount of money) plus a specification of the currency in question using an identifier compliant with the company's Technology Standard for the Representation of Currency. |
| 10.1.8 | Scorecard Rule 430 | System assets that convert money values across different currencies MUST perform and document such conversions in compliance with the company's Technology Standard for Currency Conversion. |
| 10.1.9 | Scorecard Rule 431 | The exchange of date and/or time values via a system asset's external service interfaces MUST comply with the company's Technology Standard for the Representation of Date and Time Values. |
| 10.1.10 | Scorecard Rule 297 | Services MUST support the exchange of time zones through their external service interfaces in a form compliant with the company's Technology Standard for the Representation of Time Zones. |
| 10.1.11 | Scorecard Rule 612 | Services MUST support the exchange of decimal numbers through their external service interfaces in a format compatible with the company's Technology Standard for the Representation of Decimal Numbers. |
| 10.1.12 | Scorecard Rule 613 | Services MUST support the exchange of telephone numbers through their external service interfaces in a form compliant with the company's Technology Standard for the Representation of Telephone Numbers. |

### Body of Evidence

#### Story 1

As the first release of CRM was being architected, designed, and implemented, the project team needed to decide whethe rhe system would be built for global use or just the US. The incremental cost at the original time of implementation was around $800K for addition disk space in the Database and Analytics environment to handle Unicode versus Latin 1 character sets. The decision was made to develop for US only as the other regios had not commited to using the system..

One year later, the Asia Pacific group began researching CRM tools for their sales team. They were willing to adopt the sales processes that have been implemented for the US and were wanted to move quickly and cheaply but they needed a solution that provided local language support for Japan, China, and Malaysia. The cost of upgrading the existing system to support Asian langauges came in at a whopping $12 million due to building additional capacity in the environment, migrating to Unicode, providing system availability (24x7) during Asia Pacific business hours, and implementing locale based screens, data, and analytics

#### Story 2

A product serving US courts was unable to capitalize on a possible opportunity to become the sole government-sponsored supplier for all UK courts due to inadequate internationalization of its software.

## 3rd party translations must be distinguished from company translations.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 10.2.1 | Scorecard Rule 427 | Any Application that enables third-party language translations of content provided by our company MUST ensure that such translations are unambiguously identified in the user experience as not having come from our company despite being based on content provided by us, using wording that has been approved by the corporate legal department. |
| 10.2.2 | Scorecard Rule 428 | Any Application that enables third-party language translations of content provided by our company MUST ensure that such translations are accompanied by prominent access to the original content we provide on which the translation was based. |

A content provider has the obligation to reflect data correctly. This includes support for official translations of documents into multiple human languages. For example, many Canadian laws are published by the Canadian government itself in both an official English and official French versions.

Sometimes, however, users might not understand very well (or at all) the language in which documents are published and applications might provide additional language translations. Such translations could even be machine-generated. If the user is not fully informed of the nature of this third-party translation, s/he may mistake the translation (inclusing any embarrassing errors) as official.

It is therefore absolutely critical that any translations of a document through a third-party mechanism -- regardless of whether machine-generated or human-generated -- be clearly and unambiguously differentiated from the "official" publication of the content.

### Body of Evidence

#### Story 1

A 3-million dollar lawsuit was brought against the city of Portland, Oregon, and a list of other defendants due to a faulty Spanish-to-English translation of a 911 call by the third-party translation service it uses that sent help to the wrong address as a woman was gasping for breath. By the time the rescue team was able to determine the true location of the person she was already brain dead, not having taken a breath for over 14 minutes. The story was documented in an article at URL (<http://www.oregonlive.com/portland/index.ssf/2014/04/spanish_interpreter_botched_9-.html> ).

#### Story 2

A panic in the world's foreign exchange market led the U.S. dollar to plunge in value after a poor English translation of an article by Guan Xiangdong of the China News Service zoomed around the Internet. The original article was a casual, speculative overview of some financial reports, but the English translation sounded much more authoritative and concrete.

#### Story 3

In 1840, the British government made a deal with the Maori chiefs in New Zealand. The Maori wanted protection from marauding convicts, sailors, and traders running roughshod through their villages, and the British wanted to expand their colonial holdings. The Treaty of Waitangi was drawn up and both sides signed it. But they were signing different documents. In the English version, the Maori were to "cede to Her Majesty the Queen of England absolutely and without reservation all the rights and powers of Sovereignty." In the Maori translation, composed by a British missionary, they were not to give up sovereignty, but governance. They thought they were getting a legal system, but keeping their right to rule themselves. That's not how it turned out, and generations later the issues around the meaning of this treaty are still being worked out

#### Story 4

In a paper in issue 7 of the Journal of Specialized Translation, Dr. Jody Byrne documents the serious negative consequences that can result from imprecise language translations. The abstract of the article reads as follows:

"At the very heart of translation studies is the issue of translation quality. Yet, while there are numerous methods for assessing the quality of translations, little is known about what happens when a translator produces a bad translation. This paper will show that translation error, as a whole, can have significant consequences for both translator and client and by examining a number of case studies gathered from official reports and communications, court records, newspaper articles and books it will illustrate the diversity of situations which can arise as a result of translation errors. The paper will then examine the issues of liability and negligence to illustrate the legal means by which translators can be held accountable for the quality of their work. By understanding how liability for faulty translations arises, it will be possible to see the implications of laws and directives governing technical translations which are subsequently examined. This paper examines specific legal requirements relating to technical translation and discusses the consequences of translation errors using specific case studies relating to technical translation."

This article can be found at [Caveat Translator: Understanding the Legal Consequences of Errors in Professional Translation](http://www.jostrans.org/issue07/art_byrne.php|).

## Systems must to adapt to the user’s preferred locale.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 10.3.1 | Scorecard Rule 615 | System assets that use locales to modify their behaviors MUST support the use of two locale values: -- One for control of language-specific behaviors -- One for the control of numeric data handling behaviors |
| 10.3.2 | Scorecard Rule 303 | Applications MUST use resource files/bundles to supply all text that is implemented as part of the user interface, including (but not limited to) messages, instructions, help information, field headers, button labels, and text-based navigation aids/links. |
| 10.3.3 | Scorecard Rule 304 | Applications MUST use resource files/bundles to supply all icons and images that are implemented as part of the application’s user interface. |
| 10.3.4 | Scorecard Rule 306 | Applications MUST use resource files/bundles to supply all client-side code (e.g. JavaScript) that contains natural language text, time/date values, currency values, or numeric data that those scripts can manifest to the user. |
| 10.3.5 | Scorecard Rule 307 | Applications MUST use resource files/bundles to supply all CSS styling that is part of the user interface. |
| 10.3.6 | Scorecard Rule 616 | Customer-facing user interface applications MUST provide a user interface that complies with the company’s Global Product Design Standard. |
| 10.3.7 | Scorecard Rule 300 | System assets with customer-facing interactions MUST support locale-appropriate representations of numeric data (date, time, currency, etc.) for those interactions. |
| 10.3.8 | Scorecard Rule 303 | Applications MUST use resource files/bundles to supply all text that is implemented as part of the user interface, including (but not limited to) messages, instructions, help information, field headers, button labels, and text-based navigation aids/links |

#### Examples

Applications should use resource bundles for each locale to identify things like:

* the names that appear in buttons on web pages or desktop application screens that identifies what those buttons do
* the descriptive labels associated with form fields that identify to the user what information is to be entered in that field
* messages that can appear on the screen to inform the user, describe errors that have occurred, etc.
* icons that appear within the user interface (so that culturally-appropriate icons can be provided in each locale the application supports)
* JavaScript, Silverlight, Flash or other client-side logic that carries text that can be displayed to the user
* CSS style sheets (so that styling can be appropriate to the needs of individual locales)

### Good Development Practices

The following are some beneficial development practices to consider when developing an application that will need to support multiple locales:

* **Honor the locale choices of the user -** If a system asset manifests locale-specific behaviors, it is usually best to base those behaviors on the user's locale settings. And since there are TWO such settings -- one for language and one for numeric data handling -- you need to ensure that whatever locale-specific behaviors your asset needs to manifests that are able to deal with the appropriate locale setting for the behavior at hand.
* **Leverage development platform-provided classes and methods to set the locale -**  Any system asset that needs to modify its behavior based on locale will generally benefit from leveraging the libraries and other facilities made available as part of their software development environments to implement such localization behaviors.
* **Avoid the use of icons involving hands or faces -** there is literllly no hand or face representation that is NOT offensive or at the very least laden with undesired cultural overtones to at least some culture!
  + For an introduction to how differently the same hand gesture can be interpreted in various parts of the world, for example, see Huffington Post article "The Global Guide to Hand Gestures".
  + For a more general overview of gestures and how they are likely to be interpreted around the world, there is a very nice wikipedia page on the topic.
  + As Microsoft warns in its msdn web page about creating icons: "Consider the cultural impact of your graphics. Avoid using letters, words, hands, or faces in icons. When you must represent people or users, depict them as generically as possible"
* **Applications should consider the use off-the-shelf internationalized UI components -** One of the challenges in providing a culturally-friendly user interface is accommodating cultural variations that appear around the world. In some cases, this is reasonably straightforward to accommodate -- like having message text accessed through resource bundles that are tied to a user's locale. But some aspects -- like having a calendar or date selector as part of your user interface -- are more challenging to produce. Fortunately, a number of vendors provide libraries of globalized web and native UI widgets/components that can be fairly easily incorporated into your user interface.
* **Employ user interface components that provide flexibility in accommodating the space demands of text of varying languages -** Applications that can support a variety of languages must often deal with variations across locales in the width needed to accommodate text inputs (and outputs!) that will work well for different languages. Techniques that support flexibility in text width are generally likely to make adapting to new locales easier. Some example include the following:
  + Expandable drop-down lists whose width is not limited by the display width of the text field to which they belong
  + Buttons that contract/expand to accommodate labels of varying lengths
  + Text entry boxes whose length can be substituted with an externally-supplied value (rather than hard-coded into the user interface template)
  + More sophisticated approaches exist that respond dynamically to the amount of text entered: (1) http://www.whatstyle.net/articles/11/vfields (2) http://stackoverflow.com/questions/3392493/adjust-width-of-input-field-to-its-input

# Data assets

A **data asset** is a collection of one or more data sets managed as an asset.

There are several axes along which data assets may be categorized. The categorization above primarily reflects the nature of the data. Recall, however, that almost all data can be categorized in two or more ways, even along a single axis, so the categories should not be seen as necessarily mutually exclusive.

### Storage Structure

What we usually refer to as storage structure is really storage *interface* structure; that is, the structure of data as it's presented to a data management system or as presented by that data management system to its users. A data management system may change the structure of the data internally in a variety of ways, some of which might involve dramatic re-structuring (e.g., "shredding"). However, we wouldn't classify data by how a data management system stores it internally, since that is really an attribute of the data management system rather than of the data.

**Structured data** refers to data that is stored in a data management system that requires that the data's structure be stipulated at or before the time of storage. Data that is stored in classic relational database management systems (RDBMSs) are the primary representatives of structured data. Most *business data* (see next section) is treated as structured data.

**Unstructured data** refers to data that is stored in a data management system that does not require that the data's structure be stipulated at or before the time of storage. The term is a bit of a misnomer, since almost all data has structure. Most *content* (see next section) is treated as unstructured data.

### Nature of the Data

* **Content** is defined as "multi-media phenomena that are intended for direct human consumption or analytical processing and include some representations of information". Content may be human-readable text, with supporting graphics, video, and audio or or structured data such as tables.
* **Business data** is defined as data that is used to operate the business. This includes transactional data (e.g., orders, purchases, sales) and reference data supporting such transactional data (e.g., customers, products, accounts). Most business data is handled as *structured data* (see above). However, business data also includes textual data such as contracts.
* **Operational Data** is defined as data that is used to support High availability, performance, incident response and problem resolution. The most common form of operational data are log files.
* **Reference data** is defined as data that is used in a read-only fashion in support of some other data. Unfortunately, this is a relative definition.
  + **Entity master data** is data about real-world entities, including: judges, lawyers, law firms, government entities, geographic entities, expert witnesses, and commercial organizations.
    - Much of entity master data is structured (e.g., name, address, date of graduation), but includes some content as well (e.g., biographical narrative).
    - Since we serve the legal industry, our customer master data will include many lawyers and law firms. Therefore, our customer data and our attorney and law firm master data overlap greatly, and are used to support both content fabrication and business operations.
  + **Relationship data** is data copied from various master data assets that define the relationships between entities that are stored in that data. Relationship data is collected in one place in order to discover relationships that are not evident in a single data asset. A common approach to storing relationship data is using n-tuples in a graph database.

By definition, every data asset is *managed by* exactly one system asset, but is often *used by* many other assets. One can then categorize data assets indirectly by some categorization of the system assets that manage them. For instance, one could have Product Delivery Systems data assets, Content Fabrication data assets, Editorial data assets, etc.

# Compliant Data

Data must be managed and protected in compliance with laws, regulations, and corporate policies.

## Compliant Data - Golden Rules

|  |  |  |
| --- | --- | --- |
| **Number** | **Golden Rule** | **Description** |
| 2.1 | Compliant Data - Information Value Classification | Data must be classified according to the Information Value Classification Instructions. |
| 2.2 | Compliant Data - Regulated Data | Regulated data must be managed appropriately. |
| 2.3 | Compliant Data - Restricted Data | Data whose usage is restricted by its provider(s) must be managed appropriately. |
| 2.4 | Compliant Data - Retention and Purging | Data must be retained as required by the business and by legal and regulatory requirements, and destroyed thereafter. |
| 2.5 | Compliant Data - Asset Registration and Naming | Assets must be registered in a central repository using unique and meaningful names. |

## Compliant Data - Information Value Classification

Data must be classified according to the Information Value Classification Instructions.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 2.1.1 | Scorecard Rule 618 | A data asset must be classified according to the Information Value Classification Instructions from the Corporate Compliance Department Record Management site. This classification must be recorded in the Data Specification for the data asset. |

## Compliant Data - Regulated Data

Regulated data must be managed appropriately.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 2.2.1 | Scorecard Rule 619 | If the data asset contains any data subject to oversight by any governmental regulatory body, such data must be managed with respect to applicable regulations. |

## Compliant Data - Restricted Data

Data whose usage is restricted by its provider(s) must be managed appropriately.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 2.3.1 | Scorecard Rule 620 | If the data asset contains any data whose usage is restricted by its provider(s) with regard to intellectual property rights, privacy, or any other aspect, such data must be managed and shared with respect to these restrictions. |

## Compliant Data - Retention and Purging

Data must be retained as required by the business and by legal and regulatory requirements, and destroyed thereafter.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 2.4.1 | Scorecard Rule 769 | Data in a data asset must be removed when necessary to comply with legal and regulatory requirements, corporate data retention standards, and business requirements. Data assets must document the retention requirements in a data specification that follows the Data Asset Specification Template. |

## Compliant Data - Asset Registration and Naming

Assets must be registered in a central repository using unique and meaningful names.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 2.5.1 | Scorecard Rule 770 | All assets (system assets and data assets) MUST be registered in the Enterprise Architecture Governance asset (3d) using unique and meaningful names. |

# Reliable Data

We must ensure that data is maintained in a well-controlled manner, and that users can rely on data’s structure and values to be confined to those promised by its stewards.

## Reliable Data - Golden Rules

|  |  |  |
| --- | --- | --- |
| **Number** | **Golden Rule** | **Description** |
| 3.1 | Reliable Data - Indirection | Data must be shared through a layer of indirection. |
| 3.2 | Reliable Data - Data Quality | A data asset must have a well-established set of operational data quality control processes to ensure the data’s value. |
| 3.3 | Controlled Data - Data Revisions | There must be well-established processes for managing changes to the data of a data asset. |
| 3.4 | Reliable Data - Data Curation | There must be well-established processes for curating data. |
| 3.5 | Reliable Data - Data Asset Design | In order to ensure that our data remains high quality and meets business requirements, the design of a data asset must follow company standards. |
| 3.6 | Reliable Data - Data Accuracy and Implementation | In order to ensure that our data remains high quality and meets business requirements, the implementation of a data asset must follow its design. |

## Reliable Data - Indirection

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 3.1.1 | Scorecard Rule 621 | There must be an intermediate interface layer between a shared data asset and its data consumers, so that the shared data asset’s schema can be modified independently of the data schema exposed through the interface layer. This must enable access to the data asset in a form valid to the data interface schema, either through a dynamic interface (such as an SQL view) or by accessing a persistent copy of the data that is valid to the data interface schema. |
| 3.1.2 | Scorecard Rule 622 | The document defining the data interface schema (e.g., an XML Schema, DTD, SQL DBMS DDL, or equivalent) must be managed in conformance to the Core Versioning Policy for Data Artifacts. |
| 3.1.3 | Scorecard Rule 623 | Three-Year Rule: If a data asset changes its data interface schema in a way that is not backwards-compatible (which would be reflected in a change to the schema’s major revision number), the superseded data interface schema must continue to be supported alongside the updated data interface schema until all data consumers have decoupled themselves from the superseded data interface schema, but not for longer than three years after the updated data interface schema was placed into production. A superseded data interface schema must be supported during this transition time as follows: • All data accessible through the updated data interface schema must also be accessible through the superseded data interface schema. • Backwards-compatible changes to the updated data interface schema must also be applied to the superseded data interface schema, if applying them to the superseded data interface schema provides the same benefits as does applying them to the updated data interface schema. |
| 3.1.4 | Scorecard Rule 790 | If a data asset's external interface is in XML and described by an XML schema or schemas, the creation and maintenance of its schema(s) must follow the Namespace Standard for XML Content Schemas. |
| 3.1.5 | Scorecard Rule 791 | If a data asset's external interface is in XML, the creation and maintenance of its schemas and/or DTDs must follow LexisNexis Markup Design and Implementation Best Practices 3c. |
| 3.1.6 | Scorecard Rule 792 | A data asset's external interface must have a physical data model, schema, DTD, or other design artifact, and its implementation must be in strict conformance with it. |
| 3.1.7 | Scorecard Rule 793 | If a data asset's external interface is in XML, then the XML must provide the locations of XML schemas that define all of the XML elements and attributes used in the asset. |

A data asset's schema is the internal representation by which the software that maintains the data asset understands the structure of the data asset. In contrast, the data schema seen by data consumers, known as the data interface schema (note: not the same as the interface itself), is the contract between the data supplier and the data consumer.

**NOTE:** The interface itself might have an interface schema used for requesting and returning data; for example, a message schema, a REST API schema, etc. That schema is not in scope here. The data interface schema in scope here may also be known as the payload schema, and is typically a subset of the interface schema. If the data is exposed through a database-level interface (e.g., JDBC, ODBC), then the data interface schema is typically an SQL view or other DBMS-type view. It is permissible to expose database-level interfaces only if other steps are taken to decouple systems from the data source; for example, by using a database specifically designed for data distribution; however, topics of system decoupling are outside the scope of the Golden Rules for Data. See Golden Rules for Systems.

## Reliable Data - Data Quality

A data asset must have a well-established set of operational data quality control processes to ensure the data’s value.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 3.2.1 | Scorecard Rule 624 | Operational data quality processes must include measurements of: timeliness, completeness, and accuracy of data; and turnaround time on problem resolution. |
| 3.2.2 | Scorecard Rule 625 | Statistical information on data quality must be made readily available to the data steward on a regular basis, at a rate appropriate for the rate of update of the data asset and its business significance. |
| 3.2.3 | Scorecard Rule 626 | Trending of quality metrics over time must be supported. |
| 3.2.4 | Scorecard Rule 788 | Structured data fields must be defined with reference to a type, foreign key, or list of values that limits the possible values as much as possible to legitimate values only. |

## Reliable Data - Data Revisions

There must be well-established processes for managing changes to the data of a data asset.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 3.3.1 | Scorecard Rule 627 | There must be well-established processes for managing changes to the data of a data asset. |

## Reliable Data - Data Curation

There must be well-established processes for curating data.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 3.3.1 | Scorecard Rule 628 | There must be current, documented editorial policies associated with a content asset. |
| 3.3.2 | Scorecard Rule 629 | There must be well-established processes for managing changes to the data of a data asset. |
| 3.3.3 | Scorecard Rule 630 | There must be evidence that the editorial policies are being followed. |

## Reliable Data - Data Asset Design

In order to ensure that our data remains high quality and meets business requirements, the design of a data asset must follow company standards.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 3.4.1 | Scorecard Rule 771 | A content asset must be stored in XML, or in a structure that preserves the same degree of information as would be preserved by properly tagged XML. |
| 3.4.2 | Scorecard Rule 631 | If the data asset is in XML, then it must have a corresponding schema, DTD or equivalent, and the creation and maintenance of its schemas and/or DTDs must follow the Content Architecture Core Versioning Policy For Data Artifacts. |
| 3.4.3 | Scorecard Rule 632 | If the data asset is in XML and described by an XML schema or schemas, the creation and maintenance of its schema(s) must follow the Content Architecture XML Namespace Policies. |
| 3.4.4 | Scorecard Rule 633 | If the data asset is in XML, the creation and maintenance of its schemas and/or DTDs must follow Markup Design and Implementation Best Practices. |

## Reliable Data - Data Accuracy and Implementation

In order to ensure that our data remains high quality and meets business requirements, the implementation of a data asset must follow its design.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 3.5.1 | Scorecard Rule 634 | To ensure that data is accurate, a data asset must have a physical data model, schema, DTD, or other design artifact, provided and maintained by an authorized data or content architect, and its implementation must be in strict conformance with it. |
| 3.5.2 | Scorecard Rule 635 | If a data asset is in XML, it must provide the locations of XML schemas that define all of the XML elements and attributes used in the asset. |

# Mastered Data

Master data declares an authoritative "single source of truth" that avoids data redundancy. Since it is meant to displace lower quality sources of the same data and can be used in many places, standards for its definition are higher.

## Mastered Data - Golden Rules

|  |  |  |
| --- | --- | --- |
| Number | **Golden Rule** | Description |
| 4.1 | Master Data - Mandatory Use | Master data assets must be used. |
| 4.2 | Master Data - Data Models | Master data assets must be modeled. |
| 4.3 | Master Data - Higher Standards | Master data assets must meet higher standards. |

## Master Data - Mandatory Use

Master data assets must be used.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 4.1.1 | Scorecard Rule 636 | Data of a given type must be mastered in only one data asset. |
| 4.1.2 | Scorecard Rule 772 | There must be a plan to eliminate exceptions to rule 4.1.1 that completes within three years of when the exceptional conditions began. |
| 4.1.3 | Scorecard Rule 773 | A data asset must manage a given type of data in a single system asset and in a single format. |
| 4.1.4 | Scorecard Rule 637 | If a data asset is partly comprised of data from another asset, then it must be brought into the data asset either directly from the master data asset, or from an approved distribution of the master data. |

## Master Data - Data Models

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 4.2.1 | Scorecard Rule 638 | A master data asset must be described by a conceptual data model. |
| 4.2.2 | Scorecard Rule 639 | A master content asset must be described by a logical data model unless (a) the business indicates that there is an expectation of minimal investment in the quality of the content asset; and (b) Enterprise Architecture agrees that it is prudent to proceed without a logical data model for the content asset. |
| 4.2.3 | Scorecard Rule 640 | A master structured data asset must be described by a logical model. |
| 4.2.4 | Scorecard Rule 641 | If there are any models, they must be maintained in an approved data model repository. |
| 4.2.5 | Scorecard Rule 642 | If there are any models, they must be managed and approved according to the Data Model Review Process. |
| 4.2.6 | Scorecard Rule 643 | If there are any models, they must be managed according to the Core Versioning Policy For Data Artifacts. |

## Master Data - Higher Standards

Master data assets must meet higher standards.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 4.3.1 | Scorecard Rule 644 | The implementation of a master data asset must be in strict conformance with the intent of its Data Specification 4c, and with its conceptual, logical, and physical data models/schemas or other controlling design artifact, as provided and maintained by an authorized data or content architect. |
| 4.3.2 | Scorecard Rule 645 | If a master data asset is in XML, then, in addition to the Golden Rules for Data herein, the creation and maintenance of master data assets must follow Content Architecture Base XML Naming and Design Rules. 4a |

# Flexible Data

Our content assets represent a significant investment of editorial, operations, and fabrication resources. To maximize the value of this investment, content should be structured so that it can be used in many products, repeatedly enriched for ever-increasing value, and reformatted for multiple uses in online and print products and in different localizations, at the lowest possible cost.

## Flexible Data - Golden Rules

|  |  |  |
| --- | --- | --- |
| **Number** | **Golden Rule** | **Description** |
| 7.1 | Flexible Data - Enrichment | Content must be evaluated for possible enrichment. |
| 7.2 | Flexible Data - Separate Presentation | Meaning and presentation must be kept separate in markup. |
| 7.3 | Flexible Data - Separate Product Content | Master content and product content must be able to evolve separately. |
| 7.4 | Flexible Data - Product-Neutral | Content must be published in a product-neutral schema. |
| 7.5 | Flexible Data - Fidelity | A content asset must be evaluated to determine the degree of fidelity to the original that must be preserved. |
| 7.6 | Flexible Data - Tables and Illustrations | Tables and illustrations must be rendered. |
| 7.7 | Flexible Data - Unit Preservation | Numbers must preserve units. |
| 7.8 | Flexible Data - Data Handling | Data must comply with specified minimum data handling standards to support low cost of use and a high degree of interoperability across systems. |

## Flexible Data - Enrichment

Content must be evaluated for possible enrichment.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.1.1 | Scorecard Rule 668 | Content must be evaluated for possible semantic enrichment based on the nature of the content and the kinds of Entities, Topics, and Citations in the content. A conscious decision must be made whether to enrich a content set or not. The enrichment decision and its rationale must be recorded in the Data Specification. |
| 7.1.2 | Scorecard Rule 669 | If content enrichments are applied, then they must be applied within the mastering boundary, and not to product content assets or source content sets. |
| 7.1.3 | Scorecard Rule 1795 | If enrichments (entities, citations, etc.) are considered high value for this content (as documented in the data specification or elsewhere by product or editorial), then metrics must be provided to demonstrate the quality of the enrichments and the trend in that quality over time. |
| 7.1.4 | Scorecard Rule 1796 | There must be committed resources responsible for maintaining enrichment to the quality level required for a positive customer experience, as evidenced by a positive enrichment quality metric trend. |

## Flexible Data - Separate Presentation

Meaning and presentation must be kept separate in markup.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.2.1 | Scorecard Rule 670 | Presentation markup must not be used to express semantics. (Some presentation markup (e.g., line breaks, emphasis) is needed to mimic the original presentation without communicating any semantics.) |
| 7.2.2 | Scorecard Rule 671 | Conversely, semantic markup must not be used for any presentational characteristics typically associated with it when the content being marked up has a different semantic. |

## Flexible Data - Separate Product Content

Master content and product content must be able to evolve separately.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.3.1 | Scorecard Rule 672 | Product content must be separate from master content, and must be able to evolve independently. |
| 7.3.2 | Scorecard Rule 673 | The master and product schemas must be able to evolve independently. |

## Flexible Data - Product-Neutral

Content must be published in a product-neutral schema.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.4.1 | Scorecard Rule 674 | When content is made available for publication, and that publication is potentially to more than one product, where each product might have its own schema, the content must be made available in a neutral schema (for instance, the master interface schema) that will not in general need to be updated as requirements for multiple independent products evolve. |
| 7.4.2 | Scorecard Rule 675 | The product-neutral and product-specific representations of data must be able to evolve independently. |

## Flexible Data - Fidelity

A content asset must be evaluated to determine the degree of fidelity to the original that must be preserved.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.5.1 | Scorecard Rule 676 | A content asset must be evaluated to determine the degree of textual fidelity, style fidelity, and page fidelity with the original that is required of its final product-specific form. A conscious decision must be made to decide the degree to which textual fidelity, style fidelity, and page fidelity with the original are to be preserved in the content asset. The decision must be recorded in the Data Specification, together with the rationale. |

## Flexible Data - Tables and Illustrations

Tables and illustrations must be rendered.

|  |  |  |
| --- | --- | --- |
| Number | **Scorecard Rule** | Rule Text |
| 7.6.1 | Scorecard Rule 677 | Where tables and illustrations naturally occur in the text, their representation on screen or in print must include all the data and graphics in the original text. In cases where the limitations of HTML do not permit the entire item to be displayed, the data must be offered to the customer in alternate formats such as .pdf or image files with metadata describing the content that was omitted from the HTML display. |

## Flexible Data - Unit Preservation

Numbers must preserve units.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.7.1 | Scorecard Rule 678 | If the data contains numbers that can be converted or rounded (such as inches or feet, grams or kilograms) that are or might be used as the basis for calculations, then they must preserve their original units of measure. |

## Flexible Data - Data Handling

Data must comply with specified minimum data handling standards to support low cost of use and a high degree of interoperability across systems.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 7.8.1 | Scorecard Rule 658 | Text must be represented using UTF-8, UTF-16 or UTF-32 when persisted to a repository. |
| 7.8.2 | Scorecard Rule 659 | If data is in XML, then XML data items must always provide an xml:lang attribute in a consistent location within that document, and on each element where the language of the content changes, as a means for specifying the language in which it is written. The xml:lang attribute value must conform to the current XML specification. 6a |
| 7.8.3 | Scorecard Rule 664 | If the data contains date values (optionally including a time of day) that are or might be used as the basis for calculations, then they must be stored either in a DBMS-specific date type field, if available to the data asset, or in a string format compliant with the company's Technology Standard for the Representation of Date and Time Values 6c. In any case, the date must be with respect to Coordinated Universal Time (UTC). |

# Global Data

Global companies, with operations in many countries must recognize that each country has its own languages, scripts, time zones, currencies, and customs for formatting text. In order to support this rich diversity, all data sets must follow the globalization/localization rules.

## Global - Golden Rules

|  |  |  |
| --- | --- | --- |
| **Number** | **Golden Rule** | **Description** |
| 6.1 | Global Data - Text Handling | Text must be stored in a globalized way. |
| 6.2 | Global Data - Number-Centric Data Handling | Number-centric data -- time, date, currency, and other numbers -- must be stored in a globalized way. |

## Global Data - Text Handling

Text must be stored in a globalized way.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 6.1.1 | Scorecard Rule 660 | If data is in XML, then XML data items that include text rendered using script other than Latin or text with other than left-to-right directionality must follow the Markup Design and Implementation Best Practices for directionalit. |

## Global Data - Number-Centric Data Handling

Number-centric data -- time, date, currency, and other numbers -- must be stored in a globalized way.

|  |  |  |
| --- | --- | --- |
| **Number** | **Scorecard Rule** | **Rule Text** |
| 6.2.1 | Scorecard Rule 661 | If the data contains date, time, currency, and/or other numbers that are or might be used as the basis for calculations, then they must be stored in locale-independent representations, and may optionally be stored in locale-specific representations as well. |
| 6.2.2 | Scorecard Rule 662 | If the data contains money values that are or might be used as the basis for calculations, then they must be stored both as a numeric value (representing the amount of money) plus a specification of the currency in question using an identifier compliant with the company's Technology Standard for the Representation of Currency. |
| 6.2.3 | Scorecard Rule 663 | When money values are converted to different currencies, the original money representation must be preserved. |
| 6.2.4 | Scorecard Rule 665 | Time of day values that are not accompanied by a date must be stored either in a DBMS-specific time type field, if available to the data asset, or in a string format compliant with the company's Technology Standard for the Representation of Date and Time Values. |
| 6.2.5 | Scorecard Rule 666 | Time-zone-specific date/times may optionally be stored but then MUST be accompanied by a time zone identifier that is either DBMS-specific or is compliant with the company's Technology Standard for the Representation of Time Zones. |
| 6.2.6 | Scorecard Rule 667 | Locale-specific date/time representations may optionally be stored but then must be accompanied by a locale specifier that is either DBMS-specific or is compliant with the company's Technology Standard for Locales. |

# APPENDIX

## Scorecard

Scorecards are a fundamental tool of Architecture governance. They are the means whereby EA assesses the condition of an asset relative to principles and golden rules.

The templates are implemented in Excel. Excel is the master for the text of the principles, golden rules, and scorecard rules.

### Internal Scorecard Structure

A scorecard template is composed of a hierarchy of:

* One or more business objectives on the scorecard template
  + One or more principles under each business objective
  + One or more golden rules under each principle
    - One or more scorecard rules under each golden rule

Additionally, a scorecard template has a number of profiles. A profile represents a set of common behaviors for assets of the same class. The use of profiles allows greater customization of the scorecard to the asset so that only the most relevant questions are asked and fewer answers are “N/A”. Each profile is related to every scorecard rule with one of the following relationship types:

* not related: The rule is excluded from the profile.
* pre-implementation: The rule is to be considered in a scorecard that is filled out before implementation of the asset being scored.
* post-implementation: The rule is to be considered in a scorecard that is filled out after implementation of the asset being scored.
* pre- and post-implementation: The rule is to be considered in a scorecard regardless of whether the asset being scored has been implemented.

### How to score and asset

A scorecard template is a tool, created as an Excel spreadsheet, that is used to measure the degree to which an asset has been implemented in accordance with the most important architectural principles and golden rules for that type of asset. For example, Golden Rules that apply to a web server may or may not alsop apply to a database server or a mobile app.

### How do I create a new scorecard?

You make a local copy of the Excel scorecard template, and update it with your answers to the questions it contains.

### What is the impact of scorecard version changes?

Scorecard templates change over time, as architectural principles, golden rules, and scorecard rules are improved, new rules are added and less useful rules are deprecated and deleted. This means that the questions asked of an asset by an older scorecard may be different than the questions asked of an asset by a newer scorecard.

When starting a new scorecard, it can be initialized from a pre-existing scorecard so that oinly changes to the asset or changes to the scorecard need be answered in the new scorecard. Each rule is given a GUID, so that even ifg the rule text changes, as long as the meaning is reasonably preserved, the pre-existing answer can be carried forward into the new version using the "Initialize from Previous Scorecard" button on the Instructions tab.

### Recommended Process for Creating a Scorecard

1. Architect creates a new asset version
2. Architect creates or updates the asset's diagrams.
3. Architect reviews the asset's diagrams with EA.
4. Architect initializes the asset's scorecard.
5. Architect fills out the scorecard.
6. Architect fills in corrective actions – These are the resolutions to the Debt items (written in the form of user stories)
7. Architect reviews the asset's scorecard with the EA.
8. EA prepares summary.
9. EA reviews summary with the architect.
10. EA and architect present the scorecard at an Architecture Review Board (ARB) Meeting.
11. EA finalized scorecard.

## How to fill out a scorecard

An architectural scorecard is a measurement tool, created in an Excel spreadsheet, that is used to measure the degree to which an asset has been implemented in accordance with the most important Architectural principles and golden rules.

### Answering Questions for Architects

The Excel scorecard template has an Instructions tab, which should be read before filling out a scorecard. Some generic Frequently Asked Questions are below. Every principle, golden rule, and scorecard rule may have explanatory text, examples, and answers to frequently asked questions, to help while filling out a scorecard.

Your task in filling out a scorecard as an Architect is to supply an answer, to the best of your knowledge and with reasonable effort to discover, whether an asset fully complies or does not comply (in whole or part) with the statement made by each scorecard rule. In addition to the answers Yes and No, other possible answers are detailed below.

### Answers to Individual Scorecard Rules

#### A tip on writing explanations, particularly for questions answered No

It's always helpful to provide an explanation, any time an asset fails to comply with the rule. The Explanation on the rule should clearly explain in what way the asset fails to meet the intent of the rule. The key word is intent, the scorecard rules identify reasonable and anticipated ways to comply with the golden rule, but are far from the only reasonable and acceptable approaches.

Avoid talking about the cure in the rule explanation, the Issue Description on a Corrective Action (debt item) should clearly explain what has to be done for the asset to meet the rule, or rules for which the issue is intended to resolve.

#### Answer Yes

If an asset materially meets with the spirit a scorecard rule, answer Yes. No explanation is necessary, and no debt item is necessary. However, it is always helpful to future architects to provide an explanation if there is anything special about the asset's compliance.

#### Answer No

If a scorecard rule logically applies to this type of asset and the asset does not materially comply with the spirit of the scorecard rule, then mark the scorecard rule as “No”.

A No answer:

* MUST have an explanation of the nature of the non-compliance.
* MUST be related to a debt item / corrective action .

The explanation for a No answer:

* MUST explain the exact nature of the non-compliance--in other words, what's wrong.
  + The explanation should not describe what should be done to correct the non-compliance. The corrective action required should be explained in the corresponding debt item.
* MAY also provide information that the EA can use to assess the business impact of the non-compliance. Examples:
  + "The fix is already being incorporated into the next release."
  + "The non-compliance has not caused problems in operations."
  + "This rule violation is related to significant operational expense that could be avoided if the asset were brought into compliance."

A debt item MUST describe the corrective action to be taken to bring the asset into compliance with the rule. If there is no commercial benefit to remediate the asset to bring it into compliance, then the debt item should have zero debt associated with it, and the debt item must describe why the asset will not be remediated. Remember, it's important to understand where we have deviated from policy, strategy, and standards, even when looking retrospectively at legacy systems; however, only estimate debt for problems we MUST fix to prevent bad outcomes.

#### Answer T-WVR (temporary waiver)

If a scorecard rule logically applies to this type of asset and the asset doesn’t comply, but some other asset(s) and/or infrastructure must be modified before this asset can be brought into compliance, then the rule should be answered with a “T-WVR”. Debt MUST be estimated for this scorecard rule The debt estimate should represent the work to be done to bring this asset into compliance once the work done in other assets and/or the infrastructure is completed.

A T-WVR answer:

* MUST have an explanation.
* MUST be related to a debt item.

Note that as soon as the asset or infrastructure is modified on which compliance depends, the waiver is rescinded, and the T-WVR answer should be changed to a No answer.

#### Answer P-WVR (permanent waiver)

*"A permanent waiver (P-WVR) indicates that the related rule is permanently waived for this asset for the foreseeable future."*

Enterprise Architecture must approve any P-WVR compliance answer to a scorecard rule.

There are a number of reasons why a P-WVR might be granted:

* EA might be planning to change or eliminate a rule from the scorecard. Until the change is implemented in the scorecard template, a P-WVR answer is appropriate.
* A permanent exemption from the rule may be granted when the rule is valid for this type of asset, but there is a valid architecture and/or business reasons that the rule does not make sense for this particular asset. *For example, sharing databases is bad in every way. However, sharing database infrastructure is understandable when insane licensing rules (like Oracle's) make doing the right thing financially impossible.*

If the rule makes sense for the asset but there is no commercial benefit to remediating, the answer should be "No", and the debt item associated with the exception should record zero debt. This may represent a situation where the business is willing to accept the risk with not remediating the exception. See Answer: No.

A P-WVR answer:

* MUST have an explanation as to why bringing this asset into compliance is not in the business's best interests.
* MUST NOT be related to a debt item.

#### Answer Yes-N/A

There are two situations in which the "Yes-N/A" answer should be used.

If a scorecard rule logically should not apply to assets *of this type* but the scorecard insists on raising it in one of the asset’s legitimate profiles, then the scorecard should ideally be changed to not expose this rule for assets having this profile, or perhaps a new profile should be introduced. In a situation like this, “Yes-N/A” is the proper scoring. In essence, this should be used as a trigger to suggest a possible need to change the scorecard itself. No debt would have to be calculated for this scorecard rule.

Or, if a scorecard rule logically applies to this type of asset but the asset doesn’t meet the rule's preconditions that would allow it even to have the potential for non-compliance, then mark the scorecard rule as “Yes-N/A” (compliant) and the DA should note this reason in the Comments column for that row. No debt would have to be calculated for this scorecard rule. We call this the "Yes, we have no bananas" answer.

A Yes-N/A answer:

* MUST have an explanation as to why the rule is not applicable to this *type* of asset, or a statement that the precondition of the rule is not met (the "no bananas" answer).
* MUST NOT be related to a debt item
  + It is important to clarify when a rule's precondition is not present, resulting in the "Yes-N/A" answer, because answering Yes" answer could imply that the condition the rule addresses is in fact present in the asset

Example:

* the precondition is not met
  + Suppose a customer-facing application is expected to comply with the security principle rule which reads as follows: “All transmission of sensitive personal identifiers such as National Identifiers (Social Security Numbers), Drivers License Numbers, and Credit Card Numbers traversing untrusted networks MUST be protected via SSL.” But if the application does not deal with any of these sensitive data types, it should simply mark this row as being “Yes-N/A” (gray) and say in the Comments column something like “No sensitive data handled by this application”. Without the explanation, it could be incorrectly assumed that the application handles sensitive data (and handles it correctly).

### When to Calculate Debt

Debt is required on mandatory rules for which the answer is No or T-WVR. However, for rules that are indicated to be operational by the issue type column in the SAFE business objective roll up guidance, in general, the architecture debt should be zero. And if there is no tangible business value to remediating the exception, the answer should still be "No", but the debt item associated with the exception should explain why this is so, and should record zero debt.

A rule is categorized as "operational" if compliance to the rule can be obtained merely by changing operational processes, without requiring investment. For example, if code reviews have not been a standard practice, management can issue a directive instituting code reviews on all code changes going forward, and no investment is needed to make this happen, so this is an operational rule. Exceptions should be shown to operational rules, and debt items should be added, but the debt should be zero.

The Issue Description on a debt item should clearly explain what has to be done for the asset to meet the rule, or rules for which the issue is intended to resolve. Everything becomes clearer, if you avoid talking about the shortcoming in the debt item, because that should be described in the rule explanation.

### Answering Questions for Enterprise Architects

The EA’s task in filling out a scorecard is to roll up the architects answers from scorecard rules to golden rules to principles to business objectives. At each stage of this task, judgment is required. At each stage of this task, the judgment required is somewhat different.

|  |  |
| --- | --- |
| **Rollup Level** | **Ask yourself . . .** |
| golden rule | To what extent do the non-compliant issues noted at the scorecard rule level cause this asset to fail to comply with the letter and spirit of this golden rule, and does that deviation have a high (red) or merely moderate (yellow) impact on the business? |
| principle | To what extent do the non-compliant issues noted at the golden rule level cause this asset to fail to comply with the letter and spirit of this principle, and does that deviation have a high (red) or merely moderate (yellow) impact on the business? |
| business objective | To what extent do the non-compliant issues noted at the principle level contribute a high net risk (red) or merely a moderate net risk (yellow) that this asset will fail to meet this business objective? |
| scorecard | To what extent do the risks of failing to meet our business objectives with this asset indicate that the asset requires significant (red) or minor (yellow) remediation? |

In answering these questions, take all possible factors into account, including but not limited to:

* the actual current impact of the deviation on the business;
* the cost of correcting the deviation, indicated by the golden rule debt assessed;
* the operational cost of the deviation remaining, indicated by the infrastructure debt assessed;
* the risks of potential loss of revenue, potential loss of reputation, and other risks and potential losses posed by the deviation.

### Rolling up Scorecard Rule Answers to Golden Rules, and Golden Rule Answers to Principles (answered by EA)

There must be an answer and an explanation provided for every golden rule in a scorecard that is relevant to the profile(s) and governance level selected. There must be an answer and an explanation provided for every principle in a scorecard that is relevant to the governance level selected.

Every principle relevant to the level of governance chosen for the scorecard must have an answer which is R, Y, or G. The management of the application of principles to assets is done through business objectives and governance levels. See Rolling up Principle Scores into Business Objective Scores and Governance Levels for Systems Assets.

#### Answer G (Green Light)

This value means that the asset in question is substantially compliant with this golden rule, either because there are no violations or the violations are inconsequential. Selecting this value will cause the fill color of this cell to turn Green to reflect this summarization choice.

The EA may roll up to G any combination of Yes, Yes-N/A, P-WVR, and T-WVR answers to scorecard rules, although it is expected that the majority of scorecard rules or golden rules that are rolled up to G have Yes answers or G answers, respectively. An EA MUST NOT roll up scorecard rule exceptions (No answers) to G unless the exception is very minor and has minimal business impact. If exceptions to rules are to be waived, they must be granted P-WVRs or T-WVRs at the rule level. Doing so preserves fine-grained metrics on decisions for waivers.

#### Answer Y (Yellow Light)

This value means that there are one or more areas of concern representing consequential impacts from non-compliance. This value should be selected only if there are one or more "No" values in the Comply column of a subordinate scorecard item. Selecting this value will cause the fill color of this cell to turn yellow to reflect this summarization choice.

#### Answer R (Red Light)

This value means that the asset in question suffers from serious impacts due to non-compliance -- impacts that require reinvestment for remediation imminently or acceleration of retirement/replacement. Selecting this value will cause the fill color of this cell to turn red to reflect this summarization choice.

#### Answer N/A

The N/A answer may be used at the golden rule level but not at the principle level.

The N/A answer indicates either that none of the scorecard rules below the golden rule apply to *this type of* asset, or that none of the preconditions were met for any of the scorecard rules below this golden rule—in other words, all of the scorecard rules below this golden rule were answered Yes-N/A. Rules marked as N/A will be excluded from the scoring calculations. Selecting this value will cause the fill color of this cell to become grey.

### Rolling up Principle Scores into Business Objective Scores

At the business objective level, the possible answers are red, yellow, and green, indicating a high level of risk, a moderate level of risk, or a low level of risk that the asset will fail to meet the business objective as a result of deviation from architectural principles. For each business objective answer, the scorecard template provides an Action cell, where a summary of corrective actions needed must be listed, and a Commentary cell. The Commentary cell must explain the rationale for choosing the level of risk of failing to meet the business objective, based on the answers under the principles that roll up to that business objective.

### Counting Exceptions and Waivers

The Exception Count on the Scorecard tab counts the number of "No" answers to *mandatory* scorecard rules across all tabs of the Excel spreadsheet. It does not count "No" answers to non-mandatory (so-called "*Should*") rules. Similarly, the Permanent Waiver Count and Temporary Waiver only count waivers to *mandatory* scorecard rules.

## Golden Rules for Systems Quick Reference

Below is a complete list of golden rules for system assets.

|  |  |  |
| --- | --- | --- |
| **Number** | **Golden Rule** | **Principle** |
| 1.1 | End user Authentication secrets must be protected. | Secure |
| 1.2 | Customer accounts must be difficult to attack. | Secure |
| 1.3 | Web traffic must be kept private. | Secure |
| 1.4 | Inputs from untrusted sources must be sanitized before use. | Secure |
| 1.5 | Data must not become code. | Secure |
| 1.6 | Transactions involving sensitive data must be auditable. | Secure |
| 1.7 | Credit card payment account numbers must not be manipulated. | Secure |
| 1.8 | Highly regulated data (SPII) must be protected. | Secure |
| 1.9 | Sensitive data must not be placed in a URI.. | Secure |
| 1.10 | Standard encryption implementations must be used where available. | Secure |
| 1.11 | Production changes must be reviewed, approved and auditable. | Secure |
| 1.12 | 3rd party software must be used safely. | Secure |
| 1.13 | Internet facing security exposures must be caught before they are exploited. | Secure |
| 1.14 | Important security related events must be recorded and reported. | Secure |
| 1.15 | Production environments must be kept separate and secure. | Secure |
| 1.16 | Standard authentication implementations must be used where available. | Secure |
| 1.17 | Systems must degrade gracefully when attacked. | Secure |
| 1.18 | Infrastructure environments must be safe. | Secure |
| 1.19 | Access to important systems and data must be managed. | Secure |
| 2.1 | Intellectual Property must be protected. | Compliant |
| 2.2 | 3rd party IP must be used in accordance to its license. | Compliant |
| 2.3 | Source code must be stored in a secure and managed repository. | Compliant |
| 2.4 | Customer facing U/Is must be accessible by users with disabilities. | Compliant |
| 3.1 | Systems must meet performance and availability SLRs and Recovery Objectives. | Reliable |
| 3.2 | Systems must have an appropriate plan for functional testing. | Reliable |
| 4.1 | Systems must deliver acceptable performance under anticipated load. | Scalable |
| 4.2 | Systems must optimize purchase of capacity. | Scalable |
| 5.1 | Systems must respond to standard control commands. | Manageable |
| 5.2 | Systems must publish appropriate operational events. | Manageable |
| 5.3 | Systems must publish Performance and Capacity data. | Manageable |
| 5.4 | An inventory of all system hosts must be available. | Manageable |
| 6.1 | Legacy Assets and Deprecated Interfaces must not be used by Strategic assets. | Simple |
| 6.2 | Code packaging must facilitate independent releases. | Simple |
| 7.1 | Assets must expose and consume only well-defined External Interfaces. | Modular |
| 7.2 | External Interfaces must be versioned and well managed. | Modular |
| 7.3 | External Interfaces must be easily consumable. | Modular |
| 7.4 | Systems must not be tightly coupled to their infrastructure / environment. | Modular |
| 7.5 | External Interfaces must not be tightly coupled to implementation details. | Modular |
| 8.1 | An interface must be callable directly not require a proprietary library. | Maintainable |
| 8.2 | Requests must be traceable from point of entry through all intermediaries. | Maintainable |
| 8.3 | Code, Schemas, and APIs must be appropriately documented and commented. | Maintainable |
| 9.1 | Data Assets and their system host must be registered. | Mastered |
| 9.2 | Data Must be of high quality. | Mastered |
| 9.3 | Data must be encapsulated. | Mastered |
| 9.4 | Data must be traceable to its source. | Mastered |
| 9.5 | Data must be validated. | Mastered |
| 10.1 | Systems must handle data in a globalized way. | Global |
| 10.2 | 3rd party translations must be distinguished from company translations. | Global |
| 10.3 | Systems must to adapt to the user's preferred locale. | Global |

## Golden Rules for Data Quick Reference

Below is a complete list of golden rules for data assets.

|  |  |  |
| --- | --- | --- |
| **Number** | **Golden Rule** | **Principle** |
| 1.1 | Compliant Data - Authorized Users | Compliant Data |
| 1.2 | Compliant Data - Information Value Classification | Compliant Data |
| 1.3 | Compliant Data - Regulated Data | Compliant Data |
| 1.4 | Compliant Data - Restricted Data | Compliant Data |
| 1.5 | Compliant Data - Retention and Purging | Compliant Data |
| 1.6 | Compliant Data - Asset Registration and Naming | Compliant Data |
| 2.1 | Reliable Data - Indirection | Reliable Data |
| 2.2 | Reliable Data - Data Quality | Reliable Data |
| 2.3 | Reliable Data - Data Curation | Reliable Data |
| 2.4 | Reliable Data - Data Revisions | Reliable Data |
| 2.5 | Reliable Data - Data Asset Design | Reliable Data |
| 2.6 | Reliable Data - Data Accuracy and Implementation | Reliable Data |
| 3.1 | Mastered Data - Mandatory Use | Mastered Data |
| 3.2 | Mastered Data - Data Models | Mastered Data |
| 3.3 | Mastered Data - Higher Standards | Mastered Data |
| 4.1 | Flexible Data - Enrichment | Flexible Data |
| 4.2 | Flexible Data - Separate Presentation | Flexible Data |
| 4.3 | Flexible Data - Separate Product Content | Flexible Data |
| 4.4 | Flexible Data - Product-Neutral | Flexible Data |
| 4.5 | Flexible Data - Fidelity | Flexible Data |
| 4.6 | Flexible Data - Tables and Illustrations | Flexible Data |
| 4.7 | Flexible Data - Unit Preservation | Flexible Data |
| 4.8 | Flexible Data - Data Handling | Flexible Data |
| 5.1 | Global Data - Text Handling | Global Data |
| 5.2 | Global Data - Number-Centric Data Handling | Global Data |