Table of Contents

Reference Architecture (RA)

Return to Public Area **OR** View this document in the Wiki

DISTRIBUTED IMMUTABLE DATA OBJECT

(DIDO)

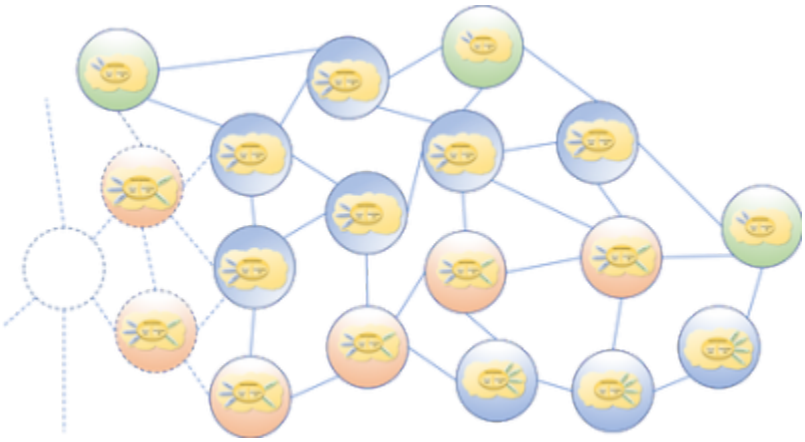
Reference Architecture (RA)

for

Cryptocurrencies, Blockchains, Distributed Ledgers and Tangles

Version 3.0

June 2021



R. W. “Nick” Stavros
Ian T. Stavros
Bryan Turek
Char Wales
Ian Murphy
Jackrabbit Consulting

Abstract

The excitement, expansion, and advancement in Distributed Computing resulting from Satoshi Nakamoto’s paper “Bitcoin: A Peer-to-Peer Electronic Cash System”[1] highlights the need for a Reference Architecture (RA) focused on Distributed Immutable Data Objects (DIDOs), where it is understood that DIDOs include, but are not limited to cryptocurrencies, the original blockchain, and distributed ledger technologies. This paper, an OMG Discussion paper, presents an RA for DIDOs.

Book Contents

PDFs

The following are provided for those readers who would like to download a printable albeit **static** copy of this RA, current as of the month/date shown.

- DIDO-RA 2.0 PDF (June 2020)
- DIDO Data Model (DIDO-DM) 1.0 PDF (June 2020)
- DIDO-RA 3.0 PDF (Slim Version - June 2021) This is a “slimmed down” version of the RA: it only contains the first pages of Appendices A through F. Any links to entries in these six Appendices will direct the reader back to the DIDO Wiki.
- DIDO-RA 3.0 PDF (Compact Version - June 2021) This version is much larger but still not the whole. It contains a truncated version of Appendix B, i.e., three pages only: “Appendix B”, “Technical Standards Bodies”, and “de facto Standards Bodies”. Any links to pages in Appendix B *other than these three pages* will direct the reader back to the DIDO Wiki.

1)

S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 24 May 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

Front Matter

return to Front of Book

- Cover Page
- OMG Disclaimer
- Change Log
- Abstract
- Copyright
- Contents
- Preface

a. Cover Page

return to Reference Architecture (RA)

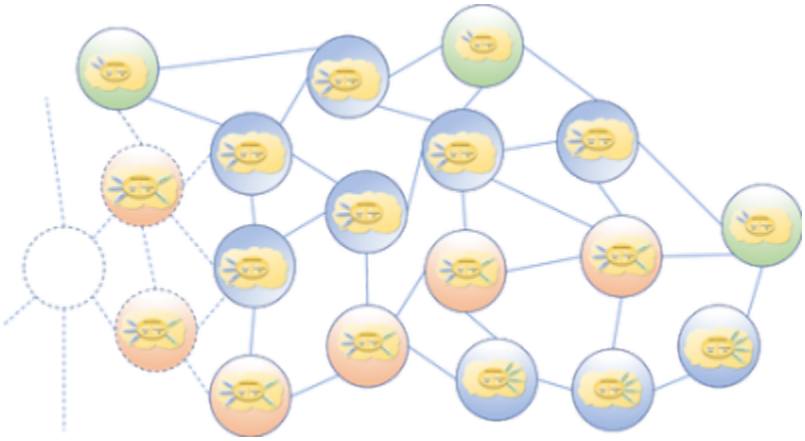
DISTRIBUTED IMMUTABLE DATA OBJECT
(DIDO)
Reference Architecture (RA)

for

Cryptocurrencies, Blockchains, Distributed Ledgers and Tangles

Version 2.0

June 2020



R. W. “Nick” Stavros
Ian T. Stavros
Bryan Turek
Char Wales
Ian Murphy
Jackrabbit Consulting

This material is based upon work supported by the
U.S. Army Small Business Innovation Research Program Office,
Defense Advanced Research Projects Agency (DARPA) and the
Army Research Office under Contract Number: W911NF-17-P-0029.
The views and conclusions contained in this document or website
are those of the authors and
should NOT be interpreted as representing
the Official Policies, either expressly or implied,
of the
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY (DARPA)
or the
U.S. GOVERNMENT

OMG Discussion Paper Disclaimer

return to Front Matter

In accordance with the OMG's Policies & Procedures (P&P), to be issued as an OMG Discussion Paper, the DIDO RA 3.0 must include the following Disclaimer:

> *This paper presents a discussion of technology issues considered in a Subgroup of the Object Management Group. The contents of this paper are presented to foster wider discussion on this topic; the content of this paper is not an adopted standard of any kind. This paper does not represent the official position of the Object Management Group.*

b. Change Log

return to Front Matter

Changes made between version 2.0 and 1.0

The major change was moving the content from a Microsoft Word document to a Wiki. This transition allowed for content to be accessed on a section by section basis, easy cross-referencing of each section, and the ability to easily determine not only how many times a section is referenced, but also where. Specifically:

- Technical section
  o Reorganized into a series of technical views, citing relevant technical & de facto standards for each
  o As appropriate, tools to support technical views: added
- Taxonomy of data and nodes: added to Technical Views
- Governance for managing the DIDO Architecture: added
- Glossary of Terms (Appendix A): added
- A new section on technical and de facto standards (Appendix B): added
  o Data sheets for technical standards: added
  o Data sheets for de facto standards: added
- A new section for tools (Appendix C): added

Changes made between version 3.0 and 2.0

- Architectural Views & Governance (Sections 2 & 3): General clean-up and small additions, e.g., a new Technical View (Decentralized Finance (DeFi) Layers)
- Requirements (Section 4): added
- Glossary (Appendix A): expanded with additional terms
- Standards Organizations (Appendix B) - standards added and/or updated
- Hardware Architectures (Appendix C) - added
- Operating Systems (Appendix D) - added
- Tools (Appendix E) - no change
- DDS Quality of Service (Appendix F) - added, to be updated in 3.x
- Tests (Appendix G) - added
- Acronyms (Appendix H): expanded with additional entries
- Cognitive Model (Appendix I) - added
- Governance Model (Appendix J) - added

c. Abstract

return to Front Matter

The excitement, expansion, and advancement in Distributed Computing resulting from Satoshi Nakamoto's paper "Bitcoin: A Peer-to-Peer Electronic Cash System"[2] highlights the need for a Reference Architecture (RA) focused on Distributed Immutable Data Objects (DIDOs), where it is understood that DIDOs include, but are not limited to cryptocurrencies, the original blockchain, and distributed ledger technologies. This paper, an OMG Discussion paper, presents an RA for DIDOs.

2)

S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 24 May 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

d. Copyright Notice

return to Front Matter

f. Preface

return to Front Matter

The Distributed Immutable Data Objects (DIDO) Reference Architecture (RA) is meant to be used as a resource to guide the design, use, or selection of Blockchain, Distributed Ledger Technology (DLT), or other Distributed Computing solutions such as InterPlanetary File System (IPFS) and Data Distribution Service (DDS).

The purpose of DIDO RA 1.0 was to create a better understanding of the Blockchain and DLT, which were exploding in the Information Technology (IT) world after the publication of Satoshi Nakamoto's paper "Bitcoin: A Peer-to-Peer Electronic Cash System"[3] and the subsequent success of the Bitcoin. Since the publication of Nakamoto's paper, this excitement has grown way beyond the original Bitcoin. It has led to the promise/emergence of many other new cryptocurrencies, as well as the application of the well known and established concepts of distributed, peer-to-peer applications to supply chains, the Industrial Internet of Things (IIoT), natural resources, environmental sciences, and even the monetization of data.

In DIDO RA 2.0, the goal was to focus less on cryptocurrencies and more on generalizing peer-to-peer, distributed computing. As a parallel effort to the publication of DIDO RA 2.0, several products have been developed to work in parallel with and complement this paper:

- **DIDO Data Model (DIDO-DM)**: captures the conceptual data constructs described in the DIDO-RA including the Community of Interest (CoI) and testing.
- **DIDO Testing Environment (DIDO-TE)**: creates an environment that allows for virtualized testing of a Distributed Application (DApp) before it can be released into the "wild" using real hardware and networks.
- **DIDO Command Line Interface (CLI)**: defines a high level command language with which to send commands to each node covering the configuration, definition, and manipulation of data on distributed nodes.
- **DIDO Reference Implementation (DIDO-RI)**: provides a working interface to the DIDO-DM, DIDO-TE and DIDO-CLI.

The major enhancement of the DIDO RA 3.0 is the addition of a section on Requirements for DIDO Implementations from the "End User" perspective. Even though these requirements are for DIDO platform developers (e.g., Ethereum, Hyperledger, etc.) the intent is to enable a broad range of "End Users", e.g., financial, retail, supply chains, etc., to employ one or more DIDO platforms. This new section defines a Governance Requirements Model, which is comprised of three processes: Regulations (i.e., requirements specification), Execution (i.e., lifecycle from design to maintenance), and Compliance (i.e., oversight of a product or service, as well as, the processes of Regulation and Execution). This model ensures DIDO implementations are well-governed and moves DIDO Governance from its current "Execution Centric" state (i.e, DIDO platform developers) to the overall DIDO Communities of Interest.

[3]
S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 24 May 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

1 Introduction

Return to Reference Architecture (RA)

The term Distributed Immutable Data Objects (DIDO) refers to the underlying technologies supporting distributed data and computation across a distributed network of peers using consensus algorithms to maintain integrity and consistency across the network. After the publication of Satoshi Nakamoto's paper *Bitcoin: A Peer-to-Peer Electronic Cash System*[2)] and the exponential growth of other cryptocurrencies, there is a need to understand in general terms the underlying DIDO architectures and provide a framework to enable engineering due diligence of DIDOs. DIDOs are not limited to cryptocurrencies, the original blockchain, or distributed ledger technologies. DIDOs are also applicable to other non-cryptocurrency domains such as supply chain, registries for births, deaths etc., and lists of acceptable Identification and Authentication (IA) acceptable certificates, including those that have been revoked. The DIDO concepts captured within a Reference Architecture (RA) are intended to represent any architecture relying on distributed networks of peers that store data and allow parallel computation. The DIDO RA is not intended as a physical "must-have" requirements list, but more as a "what-can-be" conceptual catalog.

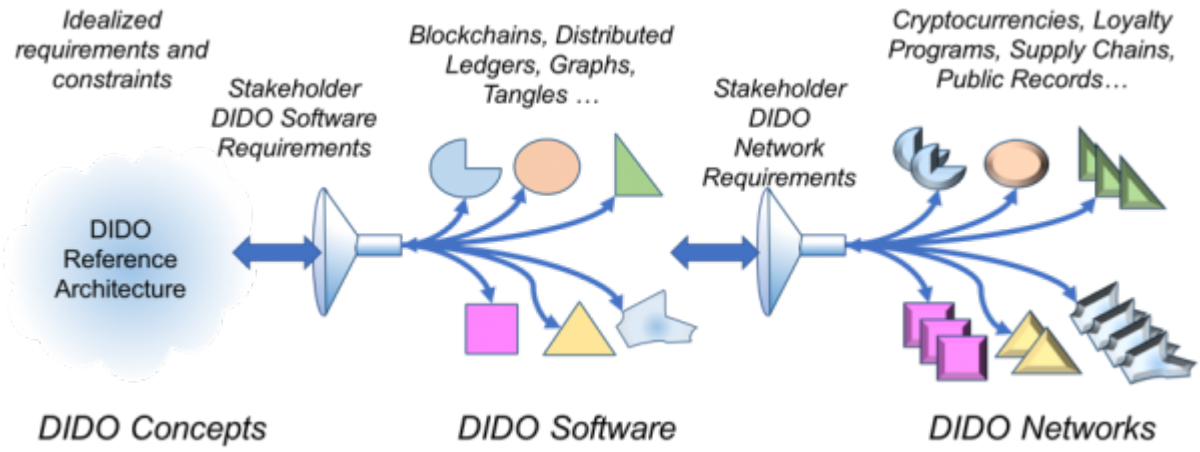


Figure 1: The relationship of DIDO Reference Architecture to incarnations such as Bitcoins, distributed ledgers, and tangles.

As illustrated in the figure, the DIDO RA is an idealized general set of requirements and constraints. Each incarnation of the DIDO software (e.g., cryptocurrency or distributed ledger) uses a set of stakeholder requirements to filter the DIDO RA and tailor it to suit the unique needs and desires of its community of stakeholders. For example, the DIDO RA provides standards for logging, even though the logging requirements driven by DIDO software stakeholders may lead to a decision to opt not to include them. There are a plethora of DIDO software incarnations available, starting with the original Blockchain software that drives Bitcoin, and moving onto IBM's Distributed Ledger, Ethereum, and Iota.

The DIDO software incarnations are adopted or used by another community of stakeholders that wish to leverage the DIDO software into a DIDO Network of Nodes to address requirements and needs of a specific domain. For example, one DIDO network community wants to provide a cryptocurrency and another public records. The DIDO software selected by the DIDO network stakeholders might be different depending on its intended purpose. In Figure 1, the bi-directional arrows communicate the notion that

the DIDO RA influences not only the DIDO software and networks, but also that evolution of DIDO software and networks feeds back into subsequent versions of the DIDO RA.

---

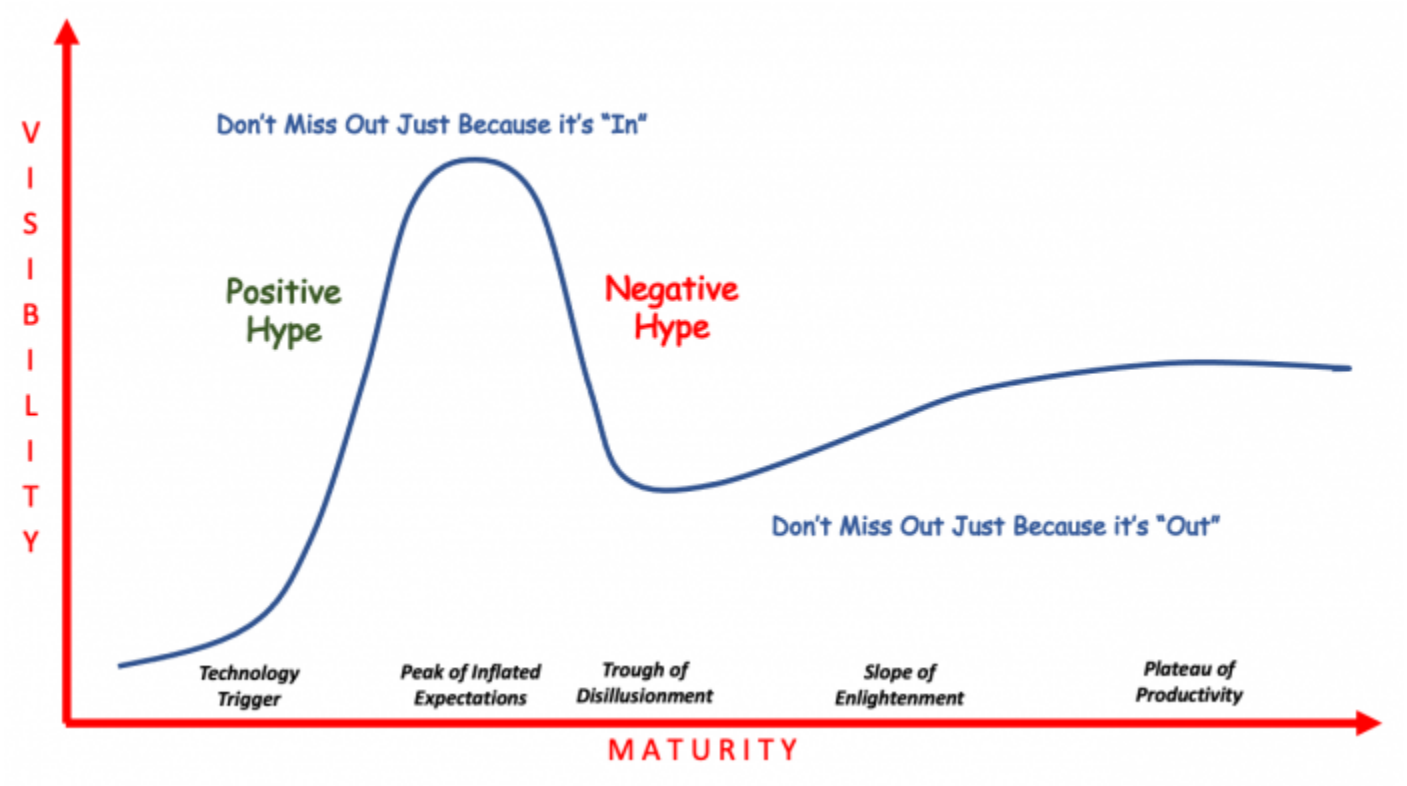
2)

S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 24 May 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

1.1 Problem

return to Introduction

As of June 12, 2018, there were 1,628 Cryptocurrencies available, with a total market cap around $300 Billion[5),6)]. DIDO implementations and interest in them are currently in the Positive Hype part of the technology Hype-Cycle as defined by Gartner.[7)]



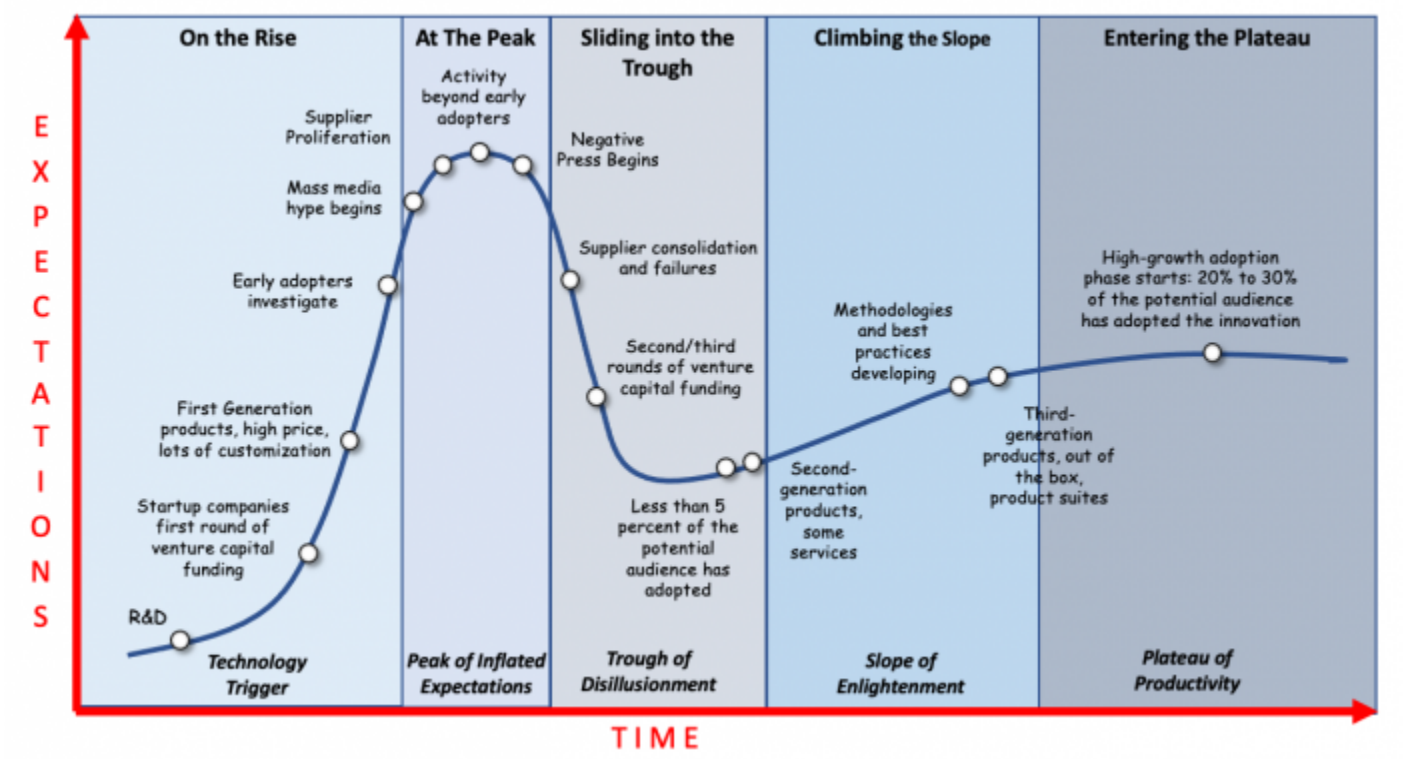


Figure 2: Gartner Group Hype Cycle

A major concern and part of the motivation behind the creation of a Reference Architecture (RA) is the lack of a comprehensive mechanism to evaluate all the risks associated with DIDO implementations (especially cryptocurrencies). Many of the risks to DIDOs are common to distributed computing and have already been identified and addressed using risk mitigators in existing processes, procedures, and standards. An example is vulnerabilities in source code. unfortunately, no statistics are available on how rigorously these risk mitigators are applied to DIDOs, if at all. Specifying a Reference Architecture and cross-referencing its components to a set of existing standards establishes the following:

- An actuarial framework for assessing risks associated with existing products
- Metrics for evaluating, comparing, and selecting existing products
- A roadmap for future standard implementation, enhancement, and development

5)

Coin Market Cap, “Cryptocurrency Market Capitalizations,” 3 December 2017. [Online]. Available: https://coinmarketcap.com/all/views/all/. [Accessed 3 December 2017].

6)

D. Palmer, “Market Cap Hits All-Time High,” 23 August 2017. [Online]. Available: https://www.coindesk.com/150-billion-total-cryptocurrency-market-cap-hits-new-time-high/. [Accessed 20 November 2017].

7)

Understanding Gartner's Hype Cycles, 30 May 2003, Alexander Linden, Jackie Fenn, https://www.bus.umich.edu/KresgePublic/Journals/Gartner/research/115200/115274/115274.html

1.2 Purpose

return to Introduction

The goals of a Reference Architecture (RA) as expressed in this paper are derived from the Office of the Assistant Secretary of Defense for Networks and Information Integration (OASD/NII)[8)] , which are:

- Provide a common language for the various stakeholders
- Provide consistent implementation of technology to solve problems
- Support validation and comparison of implementations
- Encourage adherence to common standards, specifications, and patterns

Achieving these goals will enhance the likelihood that DIDOs and networks will be engineered correctly. DIDOs are intended to cover the entirety of distributed computing, including improvements in data storage and computing that are now used to support DIDO processes, but were not considered in Nakamoto's paper[9)] and the consequent highly successful launch of Bitcoin cryptocurrency. Nakamoto's paper proposed a "ledger" for storing data and a collection of transactions, captured in a block. Blocks of transactions are verified and validated using a consensus algorithm. Bitcoin relies on Proof-of-Work (PoW) consensus. This solution, although usable, has proven to be expensive, making widespread ubiquitous adoption difficult[10)]. Other implementations such as Ethereum were built upon the Bitcoin blockchain concept, which replaced the costly PoW with Proof of Stake (PoS). The DIDO RA is extensible to evolving and emerging blockchain technologies. For example, Boyen[11)] notes that:

> *Our blockchain-free proposal shifts onto the transactions themselves the task of affirming prior transactions. Verification no longer results in a chain of transactions blocks, but in a lean graph comprised only of transactions...* [12)]

The DIDO RA concerns distributed, Peer to Peer (P2P) computing including blockchains and cryptocurrencies, but also covers domains that have little or nothing to do with currencies. For example, DIDOs can be blockchains, distributed ledgers, or graphs. Blockchains may use any consensus algorithm such as PoW or PoS. Cryptocurrencies are used as a placeholder for any of the other domains that can be supported using DIDO: supply chains, government records, scientific data, medical records, escrows, swaps, etc.

The explosion in cryptocurrencies is well known and documented. An example is presented extremely well in the animation provided by Jeff Desjardins[13)].

> **Note** "ICO funds raised" went from zero in 2014 to about $6.5 billion in November 2017.

Figure 3: The explosive growth of Initial Coin Offerings (ICOs) over four years

To provide a common language for stakeholders, the DIDO RA describes the components of a distributed network of peers supporting distributed data and computation. It is comprised of a collection of peer nodes that operate within a virtual distributed network. Each node in the architecture selects the set of architectural components, and the relationships between the components, required by their stakeholders to solve the specific requirements (e.g., blockchains, cryptocurrencies, distributed ledgers, graph databases). The individual nodes synchronize via distributed software communicating over secure messaging infrastructure. All computations and operations could be executed redundantly on all the nodes within the DIDO network of peers. DIDO components are virtual representations of functionality found in DIDO products. In addition to identifying and defining the components and their inter-relationships, the RA associates each component with existing standards (refer to Section 2.1.7).

8)

G. Doyle and B. Wilezynski, “Reference Architecture Description,” U. S. DoD, Washington, DC, 2010.

9)

S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 24 May 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

10)

S. Lee, “Bitcoin's Energy Consumption Can Power An Entire Country – But EOS Is Trying To Fix That,” 2018 Aoril 2018. [Online]. Available: https://www.forbes.com/sites/shermanlee/2018/04/19/bitcoins-energy-consumption-can-power-an-entire-country-but-eos-is-trying-to-fix-that/#7a0603931bc8. [Accessed 11 June 2018].

11)

Gartner, “Gartner Hype Cycle,” [Online]. Available: https://www.gartner.com/technology/research/methodologies/hype-cycle.jsp#. [Accessed April 2018].

12)

C. C. T. H. Xavier Boyen, “Blockchain-Free Cryptocurrencies, A Framework for Truly Decentralized Fast Transactions,” December 2017. [Online]. Available: https://eprint.iacr.org/2016/871.pdf. [Accessed 17 December 2017].

13)

J. Desjardins, “Business Insider - Visual Capitalist,” 13 December 2017. [Online]. Available: http://www.businessinsider.com/animation-shows-the-explosion-in-ico-funding-over-the-last-four-years-2017-12. [Accessed 17 December 2017].

1.3 Content Organization

return to Introduction

Following this Introduction, this discussion paper is organized as follows:

Section 2, Architectural Views: Presents and describes in detail the DIDO Architecture as a set of components collected into complementary Architectural Views: Stakeholder, Technical, and Taxonomic.

- **The Stakeholder View** is organized in accordance with the communities considered to be the "customers" of the DIDO RA, e.g., Platform, Domain, and Ecosystem.
- **The Technical View** is organized into two basic views: Fundamental (e.g., System of Systems, Assurance, Tools) and Node Network.
- **The Taxonomic View** is organized into four taxonomies: network topology, network access control, nodes, and data.

Standards (technical and *de facto*) relevant to or associated with the components supporting each view are listed.

Section 3, Governance: Proposes and describes the elements of the governance structure to manage the DIDO Architecture as an Open Source program, i.e., Charter (for the effort as a whole plus each of the DIDO COIs), Policies & Procedures (P&P), and Guide (plain English version of the P&P).

Section 4, Requirements: Lays the groundwork (models and methods) with which to specify, assess, manage, implement, and govern DIDO requirements to ensure well governed DIDO Implementations. Describes in detail the functional and non-functional requirements one must consider during the requirements specification process for DIDO platforms and then how to assess the resulting requirements specifications.

Appendices

- A: Glossary of Terms
- B: Standards Organizations: Complete listing and description of the Standards Organizations associated with the technical and *de facto* standards cited in the DIDO RA Architectural Views.
- C: Hardware Architectures : Describes Hardware Architectures to consider when designing distributed systems
- D: Operating Systems : Glossary of Operating Systems
- E: Tools : Tools to support the development, management, and operation of DIDOs
- F: DDS QoS: Quality of Service (QoS) parameters defined in the DDS specification
- G: Tests
- H: Acronyms
- I: Cognitive Model
- J: Governance Model

2 Architectural Views

return to Reference Architecture (RA)

The DIDO Reference Architecture (RA) is presented as a series of components collected into different views. The components are then associated with a set of standards relevant to each component. There are two kinds of standards provided, each produced by different kinds of standards bodies:

- Technical Standards Bodies
- de facto Standards Bodies

There are three main categories of Architectural Views:

- 2.1 Stakeholder Views
- 2.2 Technical Views
- 2.3 Taxonomic Views

2.1 Stakeholder Views

return to Architectural Views

Within this context, the following definition of Stakeholder is used:

> *A **stakeholder** is a person, group or organization that has interest or concern in a* [target] *organization. Stakeholders can affect or be affected by the* [target] *organization's actions, objectives and policies. http://www.businessdictionary.com/definition/stakeholder.html*
> **Note**: [target] added for clarification purposes

In a centralized or decentralized topography, this definition is adequate; however, in a distributed topography the understanding of what the *[target]* organization is becomes important. In a DIDO, this is by design and intent. There is no centralized authority or centralized cluster for the data, the processing of which is considered a major feature of the distributed architecture. It is a network of peers working together in parallel and simultaneously to solve problems. In other words, no single organization owns:

- all the computer resources or control them
- the definition of, or has control over, the processes
- the definition of data structures or data being distributed

In contrast to distributed systems, centralized systems (i.e., mainframes) are the authority for computation and data. In essence, the only reality is the processes and data that reside in the centralized system.

This is also in contrast to decentralized systems (i.e., traditional cloud servers), which rely on well-orchestrated and coordinated efforts of a few well connected and synchronized systems. Collective servers are the authority for the computation and data. In essence, the only reality is that which can be found on the decentralized servers; the infrastructure is expected to keep the software and data consistent and synchronized. However, this only needs to be concerned with servers that, although they are decentralized, still fall under a single authority thereby making the requirements, architecture, design, implementation, and maintenance relatively easy.

Both centralized and decentralized systems often have extensive data models and functionality, which adds to the complexity of managing them. This generally requires a single governing body (i.e., enterprise) to be ultimately responsible for the entire ecosystem and the lifecycle of the systems and the integration of components including hardware, operating systems, database management systems, web servers, application servers, software languages, networking, and other protocols. These “stacks” often result in stovepipe solutions.

In distributed systems, much of the ecosystem and governance of the components is handled by various Communities of Interest (CoIs): each has a responsibility for different aspects of the distributed system. The traditional role of a corporation or enterprise is to participate in these communities. The following graphic illustrates the various communities considered to be “customers” of the DIDO RA.

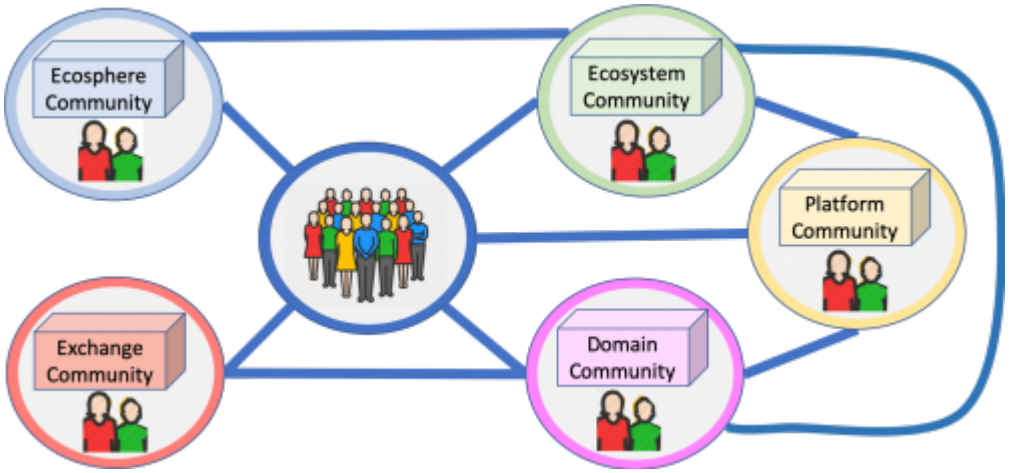


Figure 4: DIDO RA Stakeholder Communities

Platform

is responsible for the software used to distribute and control data within the Node Network, for example Bitcoin, Ethereum, Iota, DDS, and IPFS.

Domain

is responsible for the use of the data distributed on the Node Network, for example currency, rewards programs, or certificates.

Ecosystem

is responsible for a collection of domains associated with a particular ares of interest, such as green groceries, interest rate swaps, a particular tank, or class of automobiles.

Ecosphere

is responsible for a collection of domains and ecosystems associated with a common governance, which crosses over multiple areas of interest, such as military, government, automotive, or finance.

Exchange

is responsible for the exchange of data (tokens) from one domain or ecosystem with data in another domain or ecosystem, for example exchanging Bitcoins for U.S. dollars, or strawberries for jars of jam.

Enterprise

is responsible for being the systems integrator of all the domains, ecosystems, and ecospheres needed to fulfill the mission and goals of a corporation or organization, such as

an auto company, a chain of retail stores, or a bank.

Each of these areas is explained in more detail in the following views, concluding with a list of standards applicable to these Stakeholder Views:

2.1.1 Platform View

return to Stakeholder Views

A Platform has a community of interest (CoI) that is responsible for the DIDO infrastructure that resides on each Node in the Node Network. Direct involvement within a platform covers the gamut from observers, to passive users of the software, and significant contributors to the community or de facto standards (see: 2.3.3 Node Taxonomy. As a general rule, domains, ecosystems, and ecospheres participate in platform communities rather than sponsor them. For the most part, platforms are already existing, self-contained communities with well established governance. Common CoI activities are bug tracking, collaboration and voting on resolutions of problems and future directions.

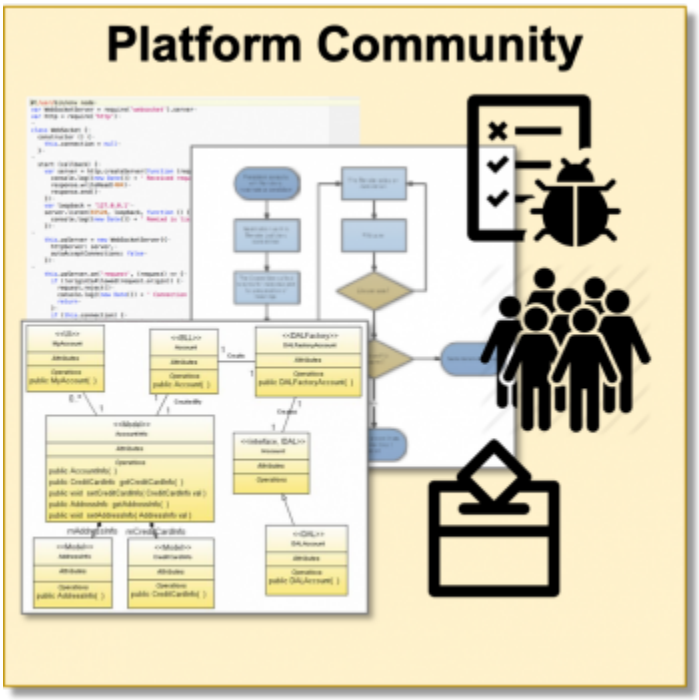


Figure 5: Platform Community

Some examples of platforms (see: 4.2.1 Platforms) are:

- Bitcoin
- DDS Foundation
- Ethereum
- Hyperledger
- Iota
- IPFS
- Multichain

Standards

Technical Standards

- RFC2026 - The Internet Standards Process
- OMG: Data Distribution Service (DDS)
- OMG: DDS Interoperability Wire Protocol (DDSI-RTPS)
- OMG: ISO/IEC C++ 2003 Language DDS PSM (DDS-PSM-Cxx)
- OMG: Java 5 Language PSM for DDS (DDS-Java)
- OMG: OPC-UA/DDS Gateway (DDS-OPCUA)
- OMG: RPC Over DDS (DDS-RPC)
- OMG: DDS Security (DDS-SECURITY)
- OMG: Web-Enabled DDS (DDS-WEB)
- OMG: DDS Consolidated XML Syntax (DDS-XML)
- OMG: DDS For Extremely Resource Constrained Environments (DDS-XRCE)
- OMG: Extensible and Dynamic Topic Types for DDS (DDS-XTypes)
- OMG: Interface Definition Language (IDL)

de facto Standards

- InterPlanetary File System (IPFS)
- Bitcoin
- Ethereum
- Linux Foundation: Hyperledger
- IOTA

2.1.2 Domain View

return to Stakeholder Views

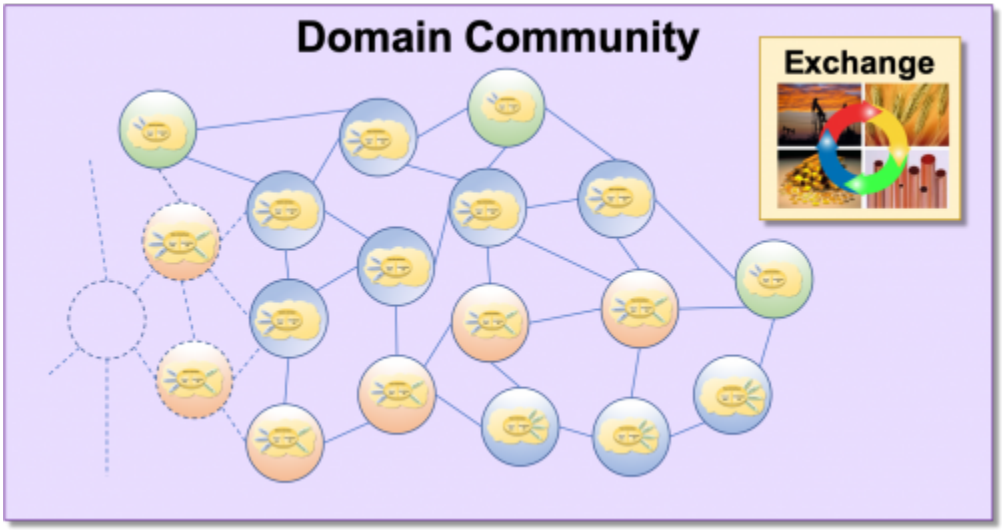


Figure 6: A Domain Community

The concepts of a **Domain View** are captured within the DIDO Domain Community.

A DIDO network consists of a system of nodes usually organized by a community of interest (CoI) connected together by a common, secure protocol – usually Transmission Control Protocol (TCP) and Internet Protocol (IP). Typically, each node executes its own copy of software that securely distributes data between the nodes. Data are generally contained within a transaction each node receives, interprets, and processes independently of the other nodes. Data can be as simple as base types such as integer, double, or string, or as complex as an entire document. All the data required to process the transaction may or may not be contained within the transaction or even the DIDO Network.

In most networks there is a one-to-one relationship between the system of nodes and fungible data managed by the network. For example, within the Bitcoin network, the *coin* is the Bitcoin. The system is solely dedicated to managing Bitcoin. In contrast, although Ether is the basic coin that drives the community and the system of nodes, Ethereum's software allows for multiple kinds of fungible data to be maintained within one network. For example, a single Ethereum network could manage Ether, customer loyalty reward points, and a registry of births and deaths.

Sometimes there is a need for multiple networks, each comprised of a different system of nodes. For example, there may be public networks or private (restricted) networks. Each network is centered on a particular kind of fungible data. Alternatively, the network might be aggregated into broader classifications such as those that have to do with currency (Ether, Bitcoin, etc.) and those that are based around customer loyalty reward programs (e.g., frequent flyer, guest, buyer programs). Still other networks might be based on public records (e.g., births, deaths, divorces), or commodities such as grains or metals. Governments or large corporations may decide to create private networks since the base of nodes they own and control is large enough for redundancy and security and can run on a private intranet. See: 2.3.2 Network Access Control Taxonomy

NOTE: Refer to 2.1.7 Relevant Community Standards for this view.

2.1.3 Ecosystem View

return to Stakeholder Views

The concepts of a **Ecosystem View** are captured within the DIDO Ecosystem Community.

The perspectives of Domain and Ecosystem Communities of Interest (CoIs) are different. Whereas a DIDO Domain Communityfocuses on a specific, real world solution or implementation to a specific concrete problem (i.e., a thing), an Ecosystem CoI focuses on the interactions and links between subordinate Ecosystems (i.e., sub-Ecosystems) and Domain Community. Governance of the overall enterprise Data Model (DM) and individual datastores for the enterprise should remain unchanged from the pre-DIDO or non-DIDO state; it is still hierarchical in nature with a definite chain of command and specific responsibility attributed to individuals within the organization. The primary focus of the DIDO Ecosystem is coordination of various other CoIs. For example, one Ecosystem CoI could coordinate other sub-Ecosystem and Domain Communities to ensure consistent perspective and context across for interoperability across all the CoIs.

A well run DIDO Community should have a charter that documents:

- the rules of engagement between DIDO CoI members
- the approval processes for changes to the charter and for releases, such as software and configuration
- trouble report procedures (capturing, distributing, resolving, disseminating afterwards)
- maintenance procedures (requesting, frequency, dissemination of maintenance reports)

- **Note** for the Ecosystem, the Charter and Bylaws are inherited from the encapsulating 2.1.3 Ecosystem View. Often, the Ecosphere is Charter, Bylaws and Policies and Procedures are subsets of the Ecosphere.

Ideally, a DIDO CoI should have a chairperson and a board of directors. Some DIDO CoIs may add a technical board or architectural board to oversee technical changes in the product and dedicate the board of directors to oversee governance of the DIDO CoI.

The following diagram represents the various kinds of DIDO CoIs and their relationship to each other.

- DIDO Ecosystem is comprised of 1 or more DIDO Platforms
- DIDO Ecosystem is comprised of 1 or more DIDO Domains
- DIDO Domain community is comprised of of 1 or more immutable data objects
- DIDO Exchange can bridge 1 or more immutable data objects within a DIDO community
- DIDO Exchange can bridge 1 or more DIDO communities
- DIDO Exchange can bridge 1 or more DIDO Ecosystems

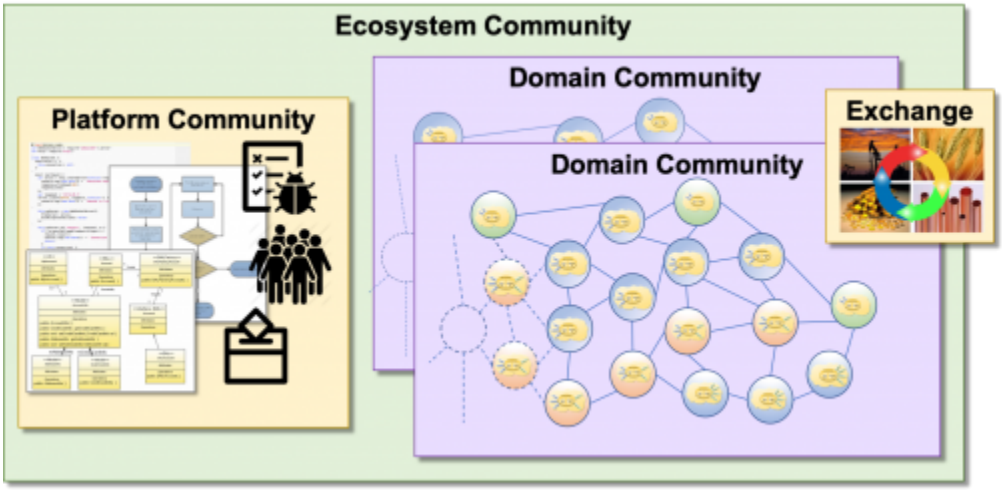


Figure 7: DIDO Ecosystem

NOTE: Refer to 2.1.7 Relevant Community Standards for this view.

2.1.4 Ecosphere View

return to Stakeholder Views

The concepts of a **Ecosphere View** are captured within the DIDO Ecosphere Community. The Ecosphere has legal requirements, see 3.2 Legal Documents.

The Ecosphere View comprises the set of all known Ecosystems within the Ecosphere and the interactions of the e=Ecosystems. Outside of a limited set of use cases, a DIDO cannot function independently, especially when a DIDO is ultimately meant to be part of an enterprise. The description of the DIDO Ecosystem Community works well when everything is defined as a DIDO using greenfield development. However, greenfield development for an enterprise probably will not happen, nor should it happen, given the large amount of legacy data information and processes held outside of the Ecosystem. Often, this type of development is referred to as brownfield development. This external data, referred to as *ancillary data*, **is** the immutable data object.

Figure 8: The high-level Ecosphere context

The Ecosphere concept is a way to encapsulate the Ecosystem and external ancillary data required to make the individual Domains within the Ecosystem functional. An individual DIDO Domain Community can access other data sources outside its Domain or, for that matter, even within its Ecosystem.

- **NOTE:** Refer to 2.1.7 Relevant Community Standards for this view.

2.1.5 Exchange View

return to Stakeholder Views

An exchange is responsible for the exchange of information between immutable data objects. An exchange can be wholly contained within a domain, act as a bridge between domains, or even span across ecosystems or ecospheres. Obviously, the complexity of an exchange increases as the range of what is exchanged moves from just internally within a domain and crosses ecosystem or ecosphere boundaries.

An exchange is a process that exchanges one kind of immutable data object for another. Bitcoin would not be worth much if there were no way to exchange it for existing currencies such as euro, dollar, etc. Exchanges may be completely contained within a DIDO community network or act as the bridge between networks within the community; or be used to transfer immutable data objects between communities (i.e., Bitcoins to Ethereum, Iotas, or commodities such as gold, silver and grains).

An example of an exchange that could operate completely within a community might be for customer rewards programs that include customer loyalty reward programs.

Standards

Technical Standards

- W3C: OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (second Edition)
- W3C: RDF 1.1 Concepts and Abstract Syntax (RDF)
- W3C: SPARQL 1.1 Overview (SPARQL)
- OMG: Ontology Definition Metamodel (ODM)
- W3C: RDF 1.1 Terse RDF Triple Language (Turtle)

de facto Standards

- None at this time

Tools

- None at this time

2.1.6 Enterprise View

return to Stakeholder Views

Referring to the figure in Section 2.1 Stakeholder Views, one can observe that enterprises roughly map to an ecosphere; however, the enterprise does not necessarily control the ecosystems, platforms, or domains it now encompasses. From a DIDO perspective, all the ancillary data (i.e., external data) is accessed through the use of an *oracle*. There can be any number of oracles available to the node within a domain; these oracles can access data from centralized, decentralized, or other DIDOs. Some examples of oracles[14]:

- Another domain within the same ecosystem
- Documents
- Relational databases
- NoSQL databases
- Flat files
- Web services
- Application services
- Event logs
- Retail transactions
- Bank account records
- Stock market streams

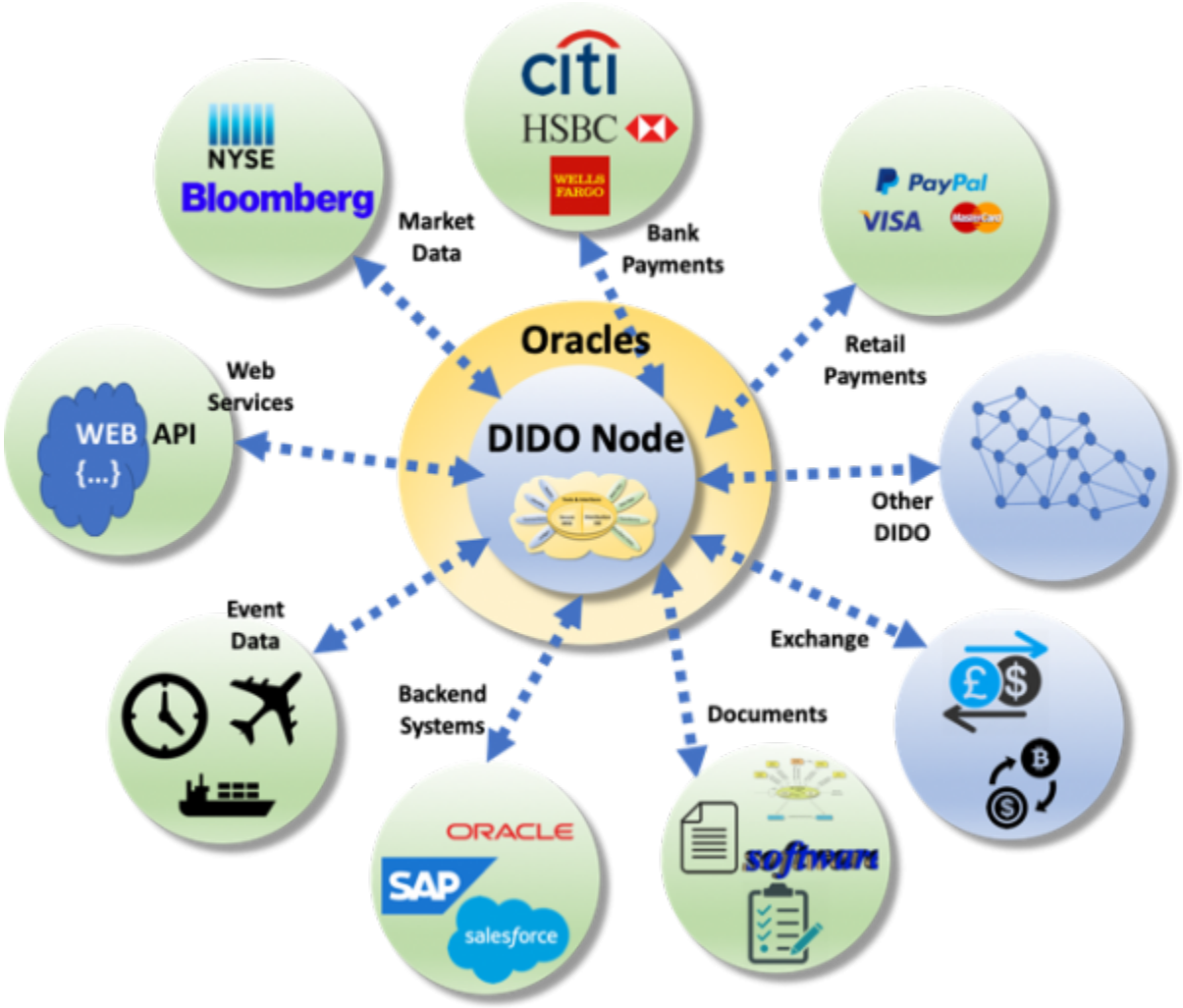


Figure 9: Oracles

One very important aspect of an enterprise with DIDO implementations is the heavy reliance on computer networks. These networks span all kinds of devices and operate in all kinds of environments, many of which rely on Disadvantaged Intermittent Links (DILs). Some examples are: nodes on mobile devices such as cell phones; submarines that are out of contact for long periods of time; nodes hit by natural and/or man-made disasters such as storms, earthquakes, or fire; devices that sleep to save power.

14)

This term should not be confused with the DBMS product from the Oracle Corporation; it is merely a term referring to something that is a good source of information.

2.1.7 Relevant Community Standards

return to Stakeholder Views

The following standards are applicable to all stakeholder views.

Technical Standards

- RFC2026 - The Internet Standards Process
- ISO 9001:2015 Quality management
- ISO/IEC/IEEE 90003:2018 Software engineering – Guidelines for the application of ISO 9001:2015 to computer software
- ISO/IEC/IEEE 25000:2014 SQuaRE -- Guide to SQuaRE
- ISO/IEC 25001:2014 SQuaRE -- Planning and Management
- ISO/IEC 25010:2011 SQuaRE -- System and Software Quality Models
- ISO/IEC 25012:2008 SQuaRE -- Data Quality Model
- ISO/IEC 25020:2007 SQuaRE -- Measurement Reference Model and Guide
- ISO/IEC 25021:2012 SQuaRE -- Quality Measure Elements
- ISO/IEC 25022:2016 SQuaRE -- Measurement of Quality in Use
- ISO/IEC 25023:2016 SQuaRE -- Measurement of System and Software Product Quality
- ISO/IEC 25024:2015 SQuaRE -- Measurement of Data Quality
- ISO/IEC 25030:2007 SQuaRE -- Quality Requirements
- ISO/IEC 25040:2011 SQuaRE -- Evaluation Process
- ISO/IEC 25041:2012 SQuaRE -- Evaluation Guide for Developers, Acquirers and Independent Evaluators
- ISO/IEC 25045:2010 SQuaRE -- Evaluation Module for Recoverability
- ISO/IEC/IEEE 15288:2015 Systems and software engineering -- System life cycle processes
- OMG: Test Information Interchange Format (TestIF)

de facto Standards

- TODO: How to create an open source program
- TODO: Measuring your open source program's success
- TODO: Tools for managing open source programs
- TODO: Using open source code
- TODO: Participating in open source communities
- TODO: Recruiting open source developers
- TODO: Starting an open source project
- TODO: Improve your open source development impact
- TODO: Shutting down an open source project
- TODO: Building leadership in an open source community
- TODO: Setting an Open Source Strategy

2.2 Technical Views

return to Technical Views

Technical Views are intended to define at a high level the various components of larger DIDO systems. They outline generic parts, subparts, and interconnections of those parts and subparts for a DIDO system. For each part or subpart, a list of potential standards, best practices, and tools is provided. There are multiple views, subdivided into two main areas:

2.2.1 Fundamental Views

return to Technical Views

Fundamental Views are sets of standards that apply throughout all DIDO Ecospheres and are not tied to any specific DIDO component; they relate simply to good system and software engineering. These fundamental views are geared not just to software, but refer to other aspects of a DIDO lifecycle such as requirements, software quality, system and software assurance, and the specification of interfaces to the outside world.

There is no intended hierarchy in the following list of fundamental views.

2.2.1.1 Interfaces

return to Fundamental Views

Interfaces exist between the various independent components of the DIDO and are therefore fundamental to the integration of DIDO components into a system. Bidirectional interfaces are defined between:

- hardware-to-hardware
- hardware-to-software
- hardware-to-human
- software-to-human

Hardware

Hardware interfaces are how the outside world connects to physical devices such as disk drives, monitors, keyboards, internal buses, batteries, internet ports, cameras, microphones, and sensors. These interfaces are described by the mechanical, electrical, and logical signals at the interface and the protocol for sequencing them.[15)] Fortunately, most of these interfaces are integrated into computer nodes and do not need to be addressed here. For example, the data from the sensor can be considered an Immutable Data Object and thus sent through the node network, but the actual acquisition of that data is a detail for nodes to implement.

Refer to Section 2.2.1.1.1 Platform Interface for more detail on Hardware interfaces.

Software

Software interfaces isolate the underlying complexity of functionality provided by a software product or package through the use of an Application Programming Interface (API). The end result of this is to treat software products or packages as opaque blackboxes.

Refer to Section 2.2.1.1.2 Software Interfaces for more details on this subject.

Human

Human interfaces within DIDOs are bidirectional and primarily cover the software-to-human components. A software Application Programming Interface (API), although defining interactions between people and software, is classified as a software interface and not a human interface. Human interfaces cover the human interaction between the humans and an operational system; examples include mouse, keyboard, trackpad, and windows presented to users of the system.

Refer to Section 2.2.1.1.3 Human Interfaces for more details on this subject.

15)

Govindarajalu, B. (2008). “3.15 Peripheral Interfaces and Controllers - OG”. IBM PC And Clones: Hardware, Troubleshooting And Maintenance. Tata McGraw-Hill Publishing Co. Ltd. pp. 142–144. ISBN 9780070483118. Retrieved 15 June 2018.

2.2.1.1.1 Platform Interface

return to Interfaces

A **Platform** is the computing environment that software resides-in and executes-on. In the DIDO context, the software is the Distributed Application (ĐApp or DApp) that runs on a particular DIDO Platform (i.e., Bitcoin, Ethereum, IOTA, IPFS, DDS, etc). The DIDO Platform is comprised of the following abstractions:

1. the computer hardware (real or virtual)
2. the operating system (OS)
3. other layers of software used to support the application (DIDO Platform)

The first layer, the Operating System, abstracts access to the actual hardware such as memory, disks, networks etc. by providing Application Programming Interfaces (APIs) for applications to use. In other words, the application is tied to the OS rather than a particular hardware configuration. However, this still ties an application to an OS.

The second layer of abstraction occurs when the interface to the OS becomes standardized, as it is with the IEEE Portable Operating System Interface (POSIX) 1003.1-2016[16]. The POSIX standard supports applications portability at the source code level and includes provisions for a standard operating system interface and environment, including a command interpreter (or “shell”), and common utility programs.[17]

The third layer (DIDO Platform) provides an API for a Distributed Application (DAPP) to reside-in and execute-on. Access to the other DIDO components is tightly controlled for security reasons. Unfortunately at this time, there is no standardized abstraction for DIDO Platforms.

Note: A DIDO Node within the DIDO Network provides a platform or subsetted platform so that DApps can be run on each node.

Standards

Technical Standards

- None at this time

de facto Standards

- Bitcoin: Developer's Guidance
- Bitcoin: Bitcoinj Developer's Documentation
- Ethereum: cpp Project
- Ethereum: Ethereumh Project
- Ethereum: Ethereumjs-lib Project
- Ethereum: Ethereum_j Project

- Ethereum: Go-ethereum Project
- Ethereum: Parity Project
- Ethereum: Pyethapp Project
- Ethereum: Ruby-ethereum Project

Tools

- None at this time

16)

IEEE Std 1003.1, 2016 Edition, IEEE Standard for Information Technology - Portable Operating System Interface (POSIX), includes IEEE Std 1003.1-2008, IEEE Std 1003.1-2008/Cor 1-2013, IEEE Std 1003.1-2008/Cor 2-2016.

17)

POSIX Product Standards, 1003.1-2016 Base, http://get.posixcertified.ieee.org/docs/base-2016.html

2.2.1.1.2 Software Interfaces

return to Interfaces

Defining formal, well-formed, consistent, and uniform interfaces between nodes within the network and the interfaces between the components within a node is essential for providing stable products that can evolve over time. For example, the distributed nature of a DIDO means the system must be designed to account for inherent mismatches between versions of software running on each node within the network, especially since it takes time for updates to flow through the system (sometimes referred to as “reboot the world problem”).

Software interfaces are the most sensitive to component changes within the DIDO. The problem is similar to the logging problem, requiring cross platform, cross programming language requirements, and cross version support; not everything can be written in Java, Python, or even PHP.

Some other tools that might be useful in a distributed environment are debuggers, network analyzers, and discovery aids. Furthermore, given that the system supports all kinds of operating environments and multiple implementations, clear, unambiguous definitions of software interfaces are essential.

Formally defined and maintained Application Programming Interfaces (APIs) are necessary for all the DIDO parts to work together seamlessly.

Standards

Technical Standards

- OMG: Interface Definition Language (IDL)

de facto Standards

- None at this time

Tools

- None at this time

2.2.1.1.3 Human Interfaces

return to Interfaces

Until humans evolve to have one culture and speak with one language, there is no standard way to interface with humans. In many ways human interfaces are cultural and are dependent on the human experience. Human interfaces have evolved, and are evolving, so that the effectiveness of the interface is in many ways fickle, trendy, and highly subject to bias. However, this does not mean that there are no guiding principles for human interfaces.

Projects should have a Graphical User Interface (GUI) Style Guide and projects should remain consistent within those guidelines.

Note: Because of the cultural nature of user Interfaces, only *guidance* is listed, not standards.

Standards

Technical Guidance

- ADA Website Accessibility Under Title II of the Americans with Disability Act (ADA) - ADA Best Practices Tool Kit for State and Local Governments, Website Accessibility Under Title II of the ADA, Chapter 5, https://www.ada.gov/pcatoolkit/chap5toolkit.htm
- Accessibility of State and Local Government Websites to People with Disabilities, U.S. Department of Justice, Civil Rights Division, Disability Rights Section https://www.ada.gov/websites2.htm

de facto Guidance

Every project and program usually develops its own guidelines for user interfaces (UI). Often, these are modeled after some of the guidance offered by major corporations known for having “good” products.

- Android Developer, User Interface and Design, https://developer.android.com/guide/topics/ui
- Apple, Mac OS Developer, https://developer.apple.com/design/human-interface-guidelines/macos/overview/visual-index/
- Apple IOS Developer, https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/
- GNOME Human Interface Guidelines, https://developer.gnome.org/hig/stable/
- KDE Human Interface Guidelines, https://hig.kde.org/
- Microsoft Designing a User Interface, https://docs.microsoft.com/en-us/windows/win32/appuistart/designing-a-user-interface

Tools

There are many tools and frameworks available. The best advice is to remember that DIDO deployments are not as agile as stand-alone applications and require significant effort to keep the distributed applications in sync. Choose tools and frameworks keeping that in mind.

2.2.1.2 Tools

return to Fundamental Views

Tools are used to create, debug, maintain, or otherwise support other programs, applications, and communities. Within the DIDO architecture the application covers the DIDO platform, domain, exchange, node, and node network.

2.2.1.2.1 Logging

return to Tools

One of the most critical requirements for distributed systems is the need to support common logging of events, information, and errors as well as support a multitude of programming environments (e.g., Java, C/C++, C#, JavaScript, Windows, UNIX, Linux).

> *Data logging is the process of collecting and storing data over a period of time in order to analyze specific trends or record the data-based events/actions of a system, network or IT environment. It enables the tracking of all interactions through which data, files or applications are stored, accessed or modified on a storage device or application.* [18)]

The protocol used for logging needs to be a redundant parallel system that reduces the dependency on the same infrastructure as the nodes and the node network to communicate. For example, if the node and node network use the same messaging software as the logging system, when a problem occurs, critical logging may also be down.

Standards

Technical Standards

- ISO 10003:2018 Quality management — Customer satisfaction — Guidelines for dispute resolution external to organizations
- ISO 10004:2018 Quality management — Customer satisfaction — Guidelines for monitoring and measuring
- RFC5424 - The Syslog Protocol (SYSLOG)

de facto Standards

- Oracle: Java logger API
- Apache: Log4j
- Apache: Log4cxx
- Apache: log4php
- Apache: log4net
- Apache: log4jscala

Tools

- Tools: Logging Tools

18)

Techopedia, “Techopedia Data Logging,” 2 December 2017. https://www.techopedia.com/definition/596/data-logging

2.2.1.2.2 Semantic Web

return to Tools

The Semantic Web extends the World Wide Web (www) through standards developed by the World Wide Web Consortium (W3C).[19] These standards provide common data formats and exchange protocols on the web, typically using Resource Description Framework (RDF). RDF facilitates data merging even when underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all data consumers to be updated or modified.[20] "The Semantic Web provides a common framework allowing data sharing and reuse across applications, enterprises, and community boundaries".[21] The Semantic Web is therefore regarded as an integrator across different content, information applications, and systems.

Standards

Technical Standards

- W3C: RDF 1.1 Terse RDF Triple Language (Turtle)
- W3C: OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (second Edition)
- W3C: RDF 1.1 Concepts and Abstract Syntax (RDF)
- W3C: SPARQL 1.1 Overview (SPARQL)
- OMG: Ontology Definition Metamodel (ODM)

de facto Standards

- None at this time

Tools

- None at this time

19)
"XML and Semantic Web W3C Standards Timeline" (PDF). 2012-02-04. http://www.dblab.ntua.gr/~bikakis/XML%20and%20Semantic%20Web%20W3C%20Standards%20Timeline-History.pdf

20)
Resource Description Framework, https://www.w3.org/RDF/

21)
"W3C Semantic Web Activity". World Wide Web Consortium (W3C). November 7, 2011. http://www.w3.org/2001/sw/

2.2.1.2.3 Open Source Communities

return to Tools

An important tool used within DIDOs is the use of Open Source Software (OSS). The use of OSS is not simply publishing software and allowing people to use it, but requires the establishment of an entire community with carefully developed rules and procedures to guide the design, development, release, maintenance, and sunsetting of the software.

Standards

Technical Standards

- None at this time

de facto Standards

- None at this time

Tools

- Community tools

2.2.1.3 Case Management

return to Fundamental Views

A major problem confronting DIDO Communities is the development of customer support to deal with issues encountered with either DIDO Transactions, Smart Contracts (especially those written externally by third parties), or DIDO OSS. For example, people have come to expect that when they use financial services, there will be recourse if transactions have unintended consequences. Recently, Coinbase, which provides an easy-to-use service for trading cryptocurrencies such as Bitcoin, Litecoin, and Ethereum, was hit with class action lawsuits alleging insider trading and also unlawful and unfair business practices.[22)] In traditional banking, lack of recourse was part of the motivation behind the Dodd-Frank Wall Street Reform and Consumer Protection Act [23)] , especially Section 1034 “Response to Consumer Complaints and Inquiries.” [24)] Case Management must be applied to both of the organizational parts of DIDO Communities: software and fungible data (i.e. currency) management.

There are several categories of problems that can arise in a distributed system:

- Those involving distributed data
- Those involving the software used to distribute the data
- Those on a local node (machine)

Problems with Distributed Values (Domain Issues)

Problems that arise on the node network with the values stored on a node or set of nodes are generally domain issues. These cases generally have to do with the implementation of a DIDO Domain (i.e., Bitcoin cryptocurrency versus the Bitcoin platform). Therefore, the case is reported to the DIDO Domain. If the problem can be resolved at the domain level that's as far as the case needs to go. However, sometimes these cases need to be resolved at the domain and platform levels, thereby requiring two cases.

Problems with Distributed Software (Platform Issues)

Problems that arise on the node network having to do with conflicts in valid values stored on a node or set of nodes are generally platform issues. Generally, there should be no conflict with the values on any of the nodes since the DIDO implementations employ consensus methodologies, which form a large part of the added value of the individual DIDO platforms. For example, Bitcoin uses a Proof of Work (POW) methodology, whereas Ethereum and others use a Proof of State (PoS) methodology.

Problems with Node

Sometimes a node within the node network has problems. In a DIDO that has built-in redundancy, validation, and verification, this is generally not a problem and should be handled by the the original

design. However, if nodes with a particular configuration (i.e., hardware, operating system, patches, security software, network cards, etc.) have issues, this could have consequences on the overall health of the domain.

Summary

Regardless of the location of the source for the case, most domains or platforms use Open Source Software and a bug tracking process based on a particular bug tracking tool such as Bugzilla, Jira, or Git-bug.

Generally, when consensus is reached for the correct value and requires a software upgrade, these cases are resolved using a Soft Fork. When consensus cannot be reached, a Hard Fork can occur within the domain.

Standards

Technical Standards

- ISO 10001:2018 Quality management — Customer satisfaction — Guidelines for codes of conduct for organizations
- ISO 10002:2018 Quality management — Customer satisfaction — Guidelines for complaints handling in organizations
- ISO 10003:2018 Quality management — Customer satisfaction — Guidelines for dispute resolution external to organizations
- ISO 10004:2018 Quality management — Customer satisfaction — Guidelines for monitoring and measuring
- OMG: Case Management Model and Notation (CMMN)
- OMG: Test Information Interchange Format (TestIF)

de facto Standards

- Bitcoin: Bitcoin Improvement Proposals (BIPs)
- Ethereum: Ethereum Improvement Proposals (EIPs)

Tools

- Tools: Bug and Issue Tracking

22)

S. Chang, “Coinbase Hit with 2 Class Action Lawsuits: Accused of Insider Bitcoin Cash Trading,” 4 March 2018. [Online]. https://www.investopedia.com/news/coinbase-hit-2-class-action-lawsuits-accused-insider-bitcoin-cash-tra

ding/
23)
Investopedia, “Dodd-Frank Wall Street Reform and Consumer Protection Act,” http://www.investopedia.com/terms/d/dodd-frank-financial-regulatory-reform-bill.asp.
24)
International Association of Risk and Compliance Professionals (IARCP), “Dodd Frank Act Text Section 1034,” 2010. http://www.dodd-frank-act.us/Dodd_Frank_Act_Text_Section_1034.html.

2.2.1.4 System of Systems (SoS)

return to Fundamental Views

To go beyond simply filling the roles traditionally fulfilled by cash, DIDOs (e.g., cryptocurrencies) need to become part of a fully distributed System-of-Systems (SoS) rather than a single monolithic product offering from a single source. Although the use of OSS helps mitigate the multiple source issue, it is not the panacea some envision because the specification of the system is by definition the current source code of the OSS.[25)] The migration from a single monolithic product offering to an SoS means there should be multiple implementations available for most of the systems (or components) that comprise the DIDO Ecosystem (or Universe). Multiple DIDO implementations greatly reduce the risk of any part of the system being compromised by a single system, subsystem, or component failure, resulting in an even more robust network.

Furthermore, each component must be deterministic in its behavior, meaning that given the same set of inputs, the outputs will always be the same. In other words, not only will all the same implementations of a node produce the same output, but multiple implementations of a component within a node will provide the same results given the same inputs. In OSS systems, the OSS implementation **is** the reference implementation and provides the baseline definition of behavior resulting in a set of specific outputs for specific inputs. Selecting a component should be left to the individual participants in the network and treated as a business decision based on trust, mutual interests, and history. This means the ideal for all components is to have at least two different implementations, with each implementation acting to provide independent validation and verification of the other. This is not so different from current systems, where each node within the blockchain running the same code and getting the same results validates the transaction; differences in the results indicate a potentially compromised node.

For example, a successful DIDO could have worldwide usage (i.e., Bitcoin, Ethereum, etc.). With an SoS approach, each country in which a transaction is executed can have its own implementations of the various components and these implementations can reflect the rules and regulations on reporting and logging of the governing organization. For example, the Swiss might not want to have their transactions reported to the USA, China, Russia or even EU members owing to their unique privacy laws and Data Residency issues. [26)] They would therefore select components that meet the needs of the Swiss rather than the world at large.

Technical Standards

- OMG: Systems Modeling Language (SysML)
- OMG: Unified Architecture Framework (UAF)
- ISO/IEC/IEEE 15288:2015 Systems and software engineering -- System life cycle processes
- ISO/IEC 25023:2016 SQuaRE -- Measurement of System and Software Product Quality

de facto Standards

- None at this time

25)

Section 2.2.1.2 covers the Open Source Paradigm in more detail.

26)

Object Management Group (OMG), “Data Residency Challenges and Opportunities for Standardization,” March 2017. [Online]. http://www.omg.org/cgi-bin/doc?mars/17-03-22.pdf.

2.2.1.5 Quality

return to Fundamental Views

The ISO/IEC 25010 standard provides consistent terminology for “specifying, measuring and evaluating system and software product quality”.[27)] The Consortium for Information & Software Quality (CISQ) [28)] provides the following diagram, highlighting eight ISO defined software quality characteristics and their associated sub-characteristics.[29)] These ISO defined characteristics must be applied to both DIDO software and coinage organizations (e.g., Bitcoin, Ethereum, IOTA), especially Reliability, Performance Efficiency, Security, and Maintainability, since these are candidates for automation.

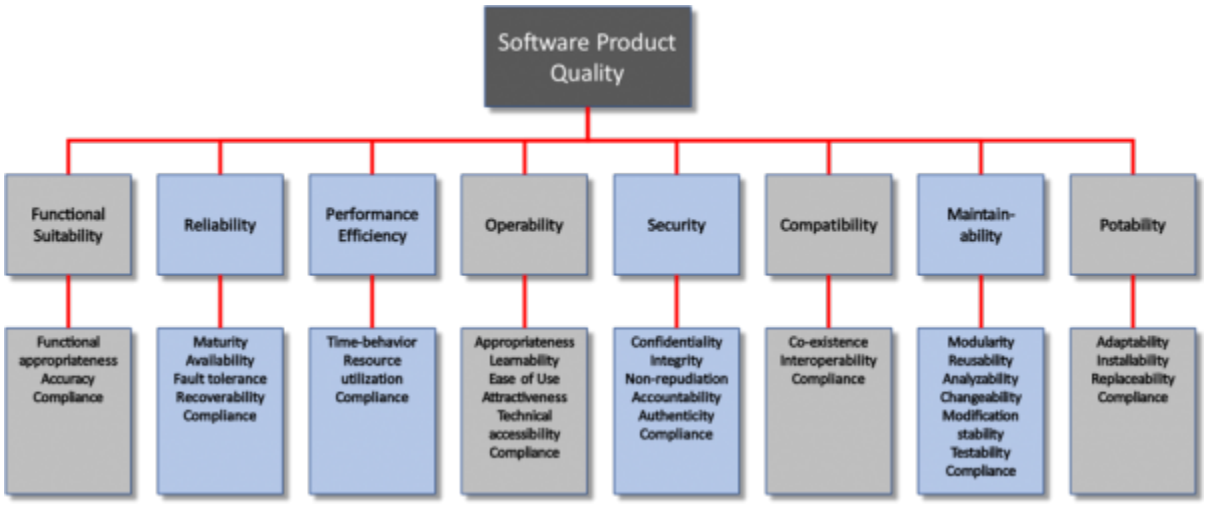


Figure 10: Quality Characteristics and Measures Specifications

Management

Management is part of SQuaRE and defines all common models, terms, and definitions referenced by all other standards from the SQuaRE series.

- ISO/IEC/IEEE 25000:2014 SQuaRE -- Guide to SQuaRE
- ISO/IEC 25001:2014 SQuaRE -- Planning and Management
- ISO/IEC/IEEE 90003:2018 Software engineering – Guidelines for the application of ISO 9001:2015 to computer software
- ISO 9001:2015 Quality management

Modelling

Modelling is part of SQuaRE and provides detailed quality models for computer systems and software products, quality in use, and data.

- ISO/IEC 25010:2011 SQuaRE -- System and Software Quality Models
- ISO/IEC 25012:2008 SQuaRE -- Data Quality Model

Measurement

Measurement is part of SQuaRE and includes a software product quality measurement reference model, mathematical definitions of quality measures, and practical guidance for their application.

- ISO/IEC 25020:2007 SQuaRE -- Measurement Reference Model and Guide
- ISO/IEC 25021:2012 SQuaRE -- Quality Measure Elements
- ISO/IEC 25022:2016 SQuaRE -- Measurement of Quality in Use
- ISO/IEC 25023:2016 SQuaRE -- Measurement of System and Software Product Quality
- ISO/IEC 25024:2015 SQuaRE -- Measurement of Data Quality

Requirements

Requirements are part of SQuaRE and help specify quality requirements. These quality requirements can be used in the process of quality requirements elicitation for a software product to be developed, or as input for an evaluation process.

- ISO/IEC 25030:2007 SQuaRE -- Quality Requirements

Evaluation

Evaluation is part of SQuaRE and provides requirements, recommendations, and guidelines for software product evaluation.

- ISO/IEC 25040:2011 SQuaRE -- Evaluation Process
- ISO/IEC 25041:2012 SQuaRE -- Evaluation Guide for Developers, Acquirers and Independent Evaluators
- ISO/IEC 25045:2010 SQuaRE -- Evaluation Module for Recoverability
- OMG: Test Information Interchange Format (TestIF)

27)
International Association of Risk and Compliance Professionals (IARCP), “Dodd Frank Act Text Section 1034,” 2010. http://www.dodd-frank-act.us/Dodd_Frank_Act_Text_Section_1034.html.
28)
International Standards Organization (ISO), “The ISO/IEC 25000 series of standards,” http://iso25000.com/index.php/en/iso-25000-standards?limit=4&start=4.
29)
Consortium for Information & Software Quality (CISQ) http://it-cisq.org/.

2.2.1.6 Open Source Paradigm

return to Fundamental Views

Using a DIDO is not just a simple shift in policies, procedures, and practices. It is a change in the entire architectural paradigm. Moving away from centralized control to distributed requires a complete change in how the system is normalized into systems, subsystems, components, etc. It also requires a shift in the basic underlying principles of the system. DIDOs are generally:

- Comprised of thousands if not millions of independent nodes
- Outside the control of any one individual or corporation
- Lacking any centralized authority; decisions are made by consensus

The DIDO architecture does not represent a single unified enterprise, but rather a loosely defined confederation of domains that requires systems integration (SI)[30]. Although SI is not new to enterprises, the granularity and types of components require a rethink. Within the DIDO environment, the definition of a platform shifts from hardware, operating system, software languages, and services (e.g., web, app, database) components to the DIDO Platform components. It is the responsibility of the DIDO Platform to isolate the enterprise from traditional platform concerns.

The granularity of the data elements within an enterprise can also shift to smaller, more isolated objects, which represent only a portion of the traditional Data Model (DM). In other words, the enterprise's data model is not going to be deployed into a single DIDO, nor should it. Enterprise data stores will continue to be needed but will be augmented and complemented by the DIDO. Some data will reside completely within the enterprise data stores, some data will reside completely within the DIDO, and some data will straddle both. Data that straddles both will require the definition of policies and procedures to ensure their data integrity.

Relevant Open Source Standards

The cultural shift from a stove-piped corporate or enterprise culture with almost complete control, to being a systems integrator participating in numerous distributed communities covering a wide range of domains, requires committed leadership and concerted effort by all the players.

Technical Standards

- None at this time.

de facto Standards

- There are none at this time but the Open Source Communities often rely heavily on the products of both Technical Standards Bodies and de facto Standards Bodies in building their projects.

- There are many guides available for participating in Open Source initiatives. **Talk Openly Develop Openly** (TODO) provides an extensive reading list.[31]. TODO also provides the following excellent guide as a place to start: TODO: Participating in open source communities.

30)

System Integrator - An individual or organization that builds systems from a variety of diverse components. With increasing complexity of technology, more customers want complete solutions to information problems, requiring hardware, software and networking expertise in a multi-vendor environment. https://www.pcmag.com/encyclopedia/term/52450/systems-integrator

31)

TODO Open Source Reading List, https://todogroup.org/guides/open-source-reading-list/

2.2.1.7 Assurance

return to Fundamental Views

The existing strategy for software and system assurance is already defined by the Systems and software Quality Requirements and Evaluation (SQuaRE). It establishes a common framework for analysis and exchange of information related to system assurance and trustworthiness, and defines the following kinds of assurance that need to be addressed: Information Assurance (IA), Safety Assurance (SfA), Software Assurance (SwA), Mission Assurance (MA) and System Assurance (SysA).

Assurance does not yield binary true / false answers. Assurance is a measure of risk which is a probability or threat of damage, injury, liability, loss, or any other negative occurrence that is caused by external or internal vulnerabilities, and that may be avoided through preemptive action.[32)]. Assurance is best handled using Structured Assurance Case Metamodels (SACMs) for each of the assurances detailed above. A DIDO community of interest (CoI) best interest is to provide assurance measurements of their software, especially those CoIs that are offering “coinage” products to provide formal SACM results.

32)

Business Dictionary, Accessed 1 June 2020, http://www.businessdictionary.com/definition/risk.html

2.2.2 Node Network View

return to Technical Views

There are slightly different variations for the definitions of a Node Network, Node, Full Node, Light Node, and Miner Node depending on the DIDO Platform.[33),34),35),36)]

- The **Node Network** is a collection of interconnected computers communicating over a network of equally privileged computers (i.e., there are no central authoritative computers). The node network is sometimes referred to as a **peer-to-peer (P2P) network**.
- A **Node** is an individual computer that participates as an equal within the node network. A node is sometimes referred to as a **peer**. Nodes receive input, perform one or more operations on those inputs, and returns an output.

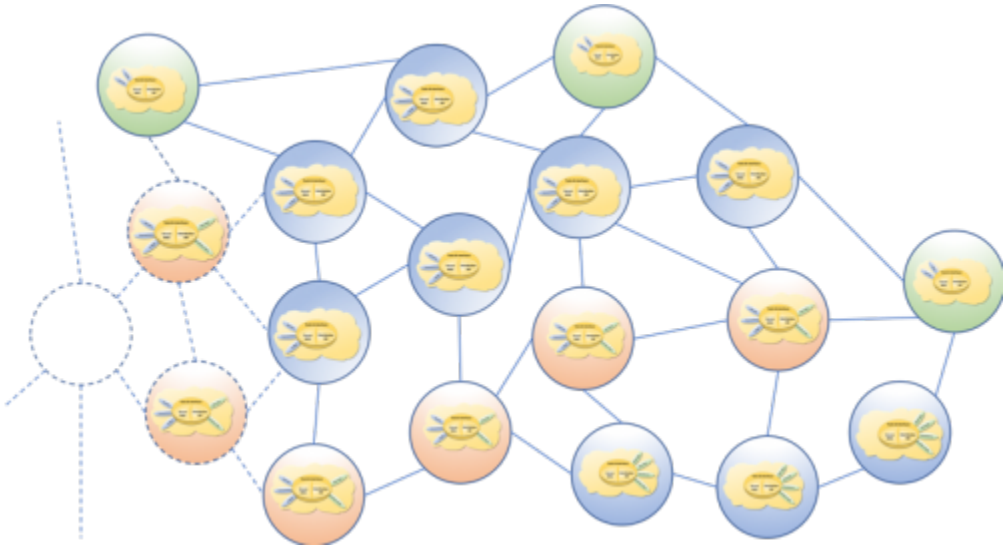


Figure 11: Node Network of Nodes.

33)

"What is a Bitcoin node?", Nate Eldredge, 14, December 2013, https://bitcoin.stackexchange.com/questions/18736/what-is-a-bitcoin-node

34)

"What are Ethereum Nodes And Sharding?", Ameer Rosic, 2017, https://blockgeeks.com/guides/what-are-ethereum-nodes-and-sharding/

35)

"IOTA Full Node—That's Why a Full Node Is Important for IOTA!", Marko Vidrih, February 2019, https://medium.com/altcoin-magazine/iota-full-node-thats-why-a-full-node-is-important-for-iota-1c46280d7712

36)

“Introduction to IPFS: Run Nodes on Your Network, with HTTP Gateways”, Ross Bulat, 3 December 2018, https://medium.com/@rossbulat/introduction-to-ipfs-set-up-nodes-on-your-network-with-http-gateways-10e21ea689a4

2.2.2.1 Network View

return to Node Network View

The Network View looks at the DIDO as a single entity. Even though the network is comprised of a collection of individual nodes, individual nodes work as one because they:

- Interact as a peer in community of peers (i.e., Peer to Peer (P2P))
- Use a single Data Object (i.e., base coinage)
- Process transactions according to the rules of the community

For example, there is a system of nodes involved in the lifecycle of Bitcoins. However, it makes no sense for public records such as births, deaths, marriages, and divorces to be in the Bitcoin system of nodes. Thus, the public records could form their own system of nodes using the Bitcoin software but restrict this network to storage of pubic records.

Obviously, Bitcoin's main purpose is to transfer assets around the world at high speed and with low overhead. However, these are not the capabilities motivating the use of DIDO networks for low volatility public records such as those which reflect the natural rates of births, deaths, marriages, and divorces. Such applications are usually under the jurisdiction of a single country or countries that have reciprocity agreements or treaties.

Similarly, the need to establish a private system of nodes might exist for internal use only users (e.g., government enclaves, large corporations) that have enough distributed resources to support the network. Although the current cryptographic protocols provide "security" to a DIDO, with the advent of quantum computing [37], a reliance on these algorithms in the future for highly classified or private data may not be acceptable. These risks provide even more justification for the development of private DIDO networks.

As a general rule, the larger the system of nodes, the more secure and tamper-proof the data held within the network becomes, which means that for networks to be viable from a security perspective, the number of nodes in the collection might have a minimum.

- 2.2.2.1.1 Secure Messaging
- 2.2.2.1.2 Transport
- 2.2.2.1.3 Security
- 2.2.2.1.4 Protocol
- 2.2.2.1.5 Distribution Software

37)
MIT Technology Review, "Quantum Computers Pose Imminent Threat to Bitcoin Security," 8 November 2017.
https://www.technologyreview.com/s/609408/quantum-computers-pose-imminent-threat-to-bitcoin-security/

2.2.2.1.1 Secure Messaging

return to Network View

The lowest functional layer of a node component is secure messaging. It is essential that a distributed system be able to have trusted, reliable, tamper resistant, and anti-snoop data flow between nodes in the network. Secure messaging must also include a standardized, consistent, and predictable way to marshal data between nodes. For example, the Endianness of each node may be different. This requires a solid transport mechanism, an excellent security infrastructure, and a standardized way to distribute data quickly and efficiently.

Standards

Technical Standards

- RFC7235 - Hypertext Transfer Protocol (HTTP/1.1): Authentication
- RFC2818 - HTTP Over TLS (HTTPS)
- RFC6749 - The OAuth 2.0 Authorization Framework
- RFC6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage
- RFC2315 - Cryptographic Message Syntax
- RFC3447 - PKCS #1: RSA Cryptography Specifications
- RFC7061 - eXtensible Access Control Markup Language (XACML) XML Media Type
- RFC2818 - HTTP Over TLS (HTTPS)
- RFC6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0
- RFC2104 - Keyed-Hashing for Message Authentication (HMAC)
- OMG: Data Distribution Service (DDS)
- OMG: DDS Security (DDS-SECURITY)

de facto Standards

- ZeroMQ Distributed Messaging
- Google: gRPC
- Google: Protocol Buffers
- Linux Foundation: Open Middleware Agnostic Messaging API (OpenMAMA)
- Linux Foundation: Open Messaging
- BIP 0070 - Payment Protocol

2.2.2.1.2 Transport

return to Network View

A cornerstone of DIDO networks is the transport component, enabling nodes to communicate with a very high degree of trust and reliability. Although it is possible to create a network of nodes that do not use Internet Protocol (IP), the breadth and acceptance of such an endeavor is questionable.

> *DIDO relies on the transport layer as defined by the Open Systems Interconnection (OSI) model. The transport layer is the layer in the open system interconnection (OSI) model responsible for end-to-end communication over a network. It provides logical communication between application processes running on different hosts within a layered architecture of protocols and other network components. The transport layer is also responsible for the management of error correction, providing quality and reliability to the end user. This layer enables the host to send and receive error corrected data, packets or messages over a network and is the network component that allows multiplexing.*[3)].

Standards

Technical Standards

- RFC0147 - The Definition of a Socket
- RFC0791 - Internet Protocol (IPv4)
- RFC6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0
- RFC2246 - The TLS Protocol
- RFC2460 - Internet Protocol, Version 6 (IPv6) Specification
- OMG: DDS Interoperability Wire Protocol (DDSI-RTPS)

de facto Standards

- Google: Protocol Buffers
- ZeroMQ Message Transport Protocol (ZMTP)

Tools

- None at this time

3)

Techopedia, “Transport Layer,” 30 November 2017.https://www.techopedia.com/definition/9760/transport-layer

2.2.2.1.3 Security

return to Network View

A DIDO is focused on two elements of security:

Internet Security: intended to protect the data flowing through the network. A major part of the functionality of a DIDO is its distributed nature over the internet and trust that it processes valid and accurate transactions.

> *Internet Security is essential in establishing trust to its success. Internet security is a catch-all term for a very broad issue covering security for transactions made over the Internet. Generally, Internet security encompasses browser security, the security of data entered through a Web form, and overall authentication and protection of data sent via Internet Protocol.*[39)]

Information Security: intended to protect the confidentiality, integrity, and availability of the information contained within the DIDO network.

> *Information security (IS) is designed to protect the confidentiality, integrity and availability of computer system data from those with malicious intentions. Confidentiality, integrity and availability are sometimes referred to as the CIA Triad of information security. This triad has evolved into what is commonly termed the Parkerian hexad, which includes confidentiality, possession (or control), integrity, authenticity, availability and utility.*[40)]

Standards

Technical Standards

- OMG: DDS Security (DDS-SECURITY)

de facto Standards

- None at this time

Tools

- None at this time

39)
Techopedia, “Techopedia Internet Security,” 30 November 2017. https://www.techopedia.com/definition/23548/internet-security

40)

Techopedia, “Information Security (IS),” 30 November 2017. https://www.techopedia.com/definition/10282/information-security-is.

2.2.2.1.4 Protocol

return to Network View

At the most elementary level, a communication protocol is comprised of a set of rules and guidelines that allow two or more nodes to communicate successfully over a network. DIDO networks must have a standardized robust protocol as the basis for processing transactions and to provide other communications between nodes.

> *A protocol is a set of rules and guidelines for communicating data. Rules are defined for each step and process during communication between two or more computers. Networks have to follow these rules to successfully transmit data.*[41)]

Standards

Technical Standards

- RFC7235 - Hypertext Transfer Protocol (HTTP/1.1): Authentication
- RFC2818 - HTTP Over TLS (HTTPS)
- RFC0791 - Internet Protocol (IPv4)
- RFC2460 - Internet Protocol, Version 6 (IPv6) Specification
- RFC0768 - User Datagram Protocol (UDP)
- RFC1112 - Host Extensions for IP Multicasting
- RFC3339 - Date and Time on the Internet: Timestamps
- RFC8259 - The JavaScript Object Notation (JSON) Data Interchange Format
- OMG: Data Distribution Service (DDS)
- OMG: RPC Over DDS (DDS-RPC)
- OMG: Java 5 Language PSM for DDS (DDS-Java)
- OMG: ISO/IEC C++ 2003 Language DDS PSM (DDS-PSM-Cxx)
- OMG: Web-Enabled DDS (DDS-WEB)

de facto Standards

- Bitcoin: Bitcoinj Developer's Documentation
- EIP 1474: Remote Procedure Call (RPC) specification (DRAFT)
- EIP 234: `blockHash` to JSON-RPC filter options (DRAFT)
- EIP 1898: ERC-NN Add `blockHash` to JSON-RPC methods which accept a default block parameter (DRAFT)
- EIP 1898: ERC-NN Add `blockHash` to JSON-RPC methods which accept a default block parameter (DRAFT)
- EIP 1193: Ethereum Provider JavaScript API (DRAFT)
- Ethereum: cpp Project

- Ethereum: cpp Project
- Ethereum: Ethereumh Project
- Ethereum: Ethereumjs-lib Project
- Ethereum: Ethereum_j Project
- Ethereum: Go-ethereum Project
- Ethereum: Parity Project
- Ethereum: Pyethapp Project
- Ethereum: Ruby-ethereum Project
- Google: gRPC

Tools

- None at this time

41)

Techopedia, “Technopedia Protocol,” 30 November 2017. https://www.techopedia.com/definition/4528/protocol.

2.2.2.1.5 Distribution Software

return to Network View

Each node has a distribution software component. It is responsible for distributing and coordinating data and software throughout a DIDO network. Probably one of the best-known implementations of distribution software is the Domain Name System (DNS), which forms the backbone of the Internet:

> *Domain name system (DNS) is a hierarchical naming system built on a distributed database. This system transforms domain names to IP addresses and makes it possible to assign domain names to groups of Internet resources and users, regardless of the entities' physical location.*[42)]

However, with the advent of the Blockchain papers by Satoshi Nakamoto [43)] another major player, Bitcoin, has emerged in the distribution software space.

Standards

Technical Standards

- RFC1034 - Domain Names - Concepts and Facilities
- RFC1035 - Domain Names - Implementation and Specification
- RFC3596 - DNS Extension to support IP Version 6
- RFC5011 - Automated Updates of DNS Security (DNSSEC) Trust Anchors
- RFC6376 - DomainKeys Identified Mail (DKIM) Signatures
- RFC6891 - Extension Mechanisms for DNS (EDNS(0))

de facto Standards

- None at this time

Tools

- None at this time

42)
Techopedia, “Techopedia Domain Name Service (DNS),” 30 November 2017. https://www.techopedia.com/definition/24201/domain-name-system-dns.

43)
S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 24 May 2009. https://bitcoin.org/bitcoin.pdf.

2.2.2.2 Node View

return to Node Network View

The Node View represents the internals of any particular node within the node network. As represented in the figure below, it is comprised of five layers; however, the layers are not set in stone – each implementation of a node within a domain may be organized differently.

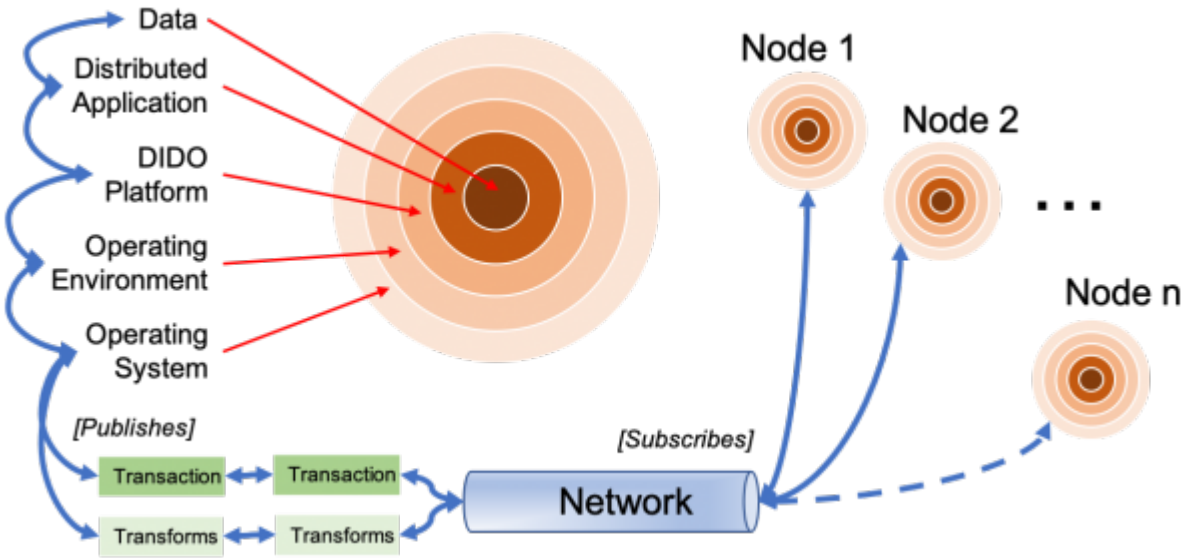


Figure 12: The idealized five layers of a Node.

At the boundary between each layer there is an interstitial layer formed by an Application Programming Interface (API), which is usually defined by one or more technical or de facto standards.

As each transaction, transform, or stream of data flows to or from a node over the network, it must travel through and/or interact with the:

- 2.2.2.2.1 Operating System (OS)
- 2.2.2.2.2 Operating Environment
- 2.2.2.2.3 DIDO Platform
- 2.2.2.2.4 Distributed Applications

2.2.2.2.1 Operating System (OS)

return to Node View

An operating system (OS), in its most general sense, is software that allows a user to run other applications in a computing device, as well as Virtual Machine applications, which emulate another computer. While it is possible for a software application to interface directly with hardware, it is not advisable from a portability or lifecycle perspective. Software applications that access hardware resources or other computer components directly pose a security risk.

Operating systems provide a common, well documented, and tested set of libraries, which abstract the idiosyncrasies of the host computer away from its applications.

An OS primarily manages a computer's hardware resources, including:

- Input devices such as a keyboard, mouse, track pad, touch screens, camera, microphone, scanners, or sensors
- Output devices such as display monitors, speakers, printers, or faxes
- Network devices such as modems, router, wired and wireless Internet Protocol network connections, and Bluetooth
- Storage devices such as internal and external disks
- Memory devices

The OS also manages a computer's:

- CPU
- Processes
- Privileges
- Cache
- Energy, i.e., power management

Standards

Technical Standards

- IEEE 1003.1-2017 - IEEE Standard for Information Technology--Portable Operating System Interface (POSIX(R)) Base Specifications (**NOTE:** See UNIX)
- ISO/IEC 23360-1:2006 Linux Standard Base (LSB) core specification 3.1 -- Part 1: Generic specification
- ISO/IEC The Linux Standard Base 5 Specification Series (LSB 5)

de facto Standards

- Apple: Darwin

- Apple: iOS
- Apple: MacOS
- Google: Android
- Microsoft: Windows API

Tools

- None at this time

2.2.2.2.2 Operating Environment

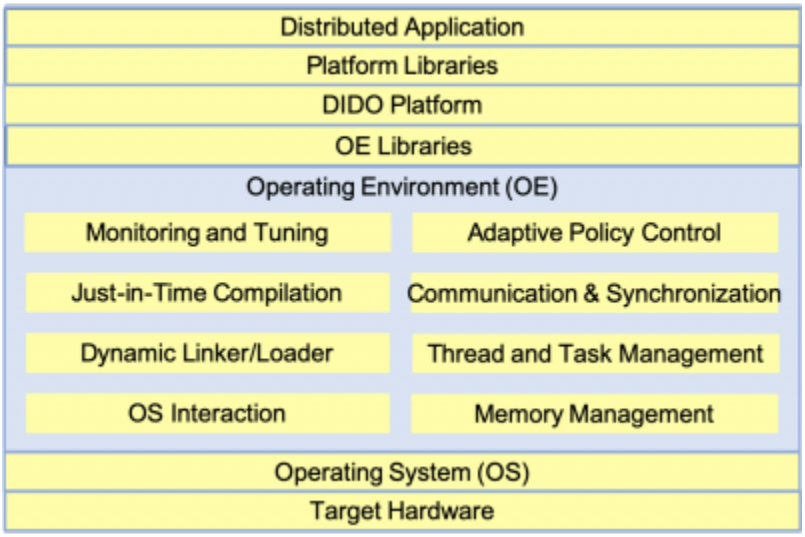
return to Node View

The concept of an Operating Environment (Run Time System) is generally traced back to *Ada*, which defined: an abstract interface to the underlying operating system, thread and task management, as well as mechanisms for monitoring, tuning, and performing dynamic memory management (all to protect against access to unallocated memory, buffer overflow errors, range violations, off-by-one errors, array access errors, and other detectable poor coding practices)[44].

Another major advancement came with Java and the Java Virtual Machine (JVM), which added a Just-In-Time (JIT) compiler, virtual processor, and an interpreter[45].

Microsoft later introduced the .NET framework, which runs on the Windows operating system. .Net includes Web Services, Web Forms, and Windows Forms in the Operating Environment[46]. Subsequently, Mono was introduced and is now sponsored by Microsoft in order to allow cross-operating system development and deployment of .Net applications. The Mono Framework is based on ECMA Standards for C# and the Common Language Runtime[47];

Figure 13: Generalized Operating Environment (OE) components[48]



Standards

Technical Standards

- ISO/IEC 9899:2018 Programming languages -- C
- ISO/IEC 14882:2017 Programming languages -- C++

- ECMA: Standard ECMA-262 - ECMAScript® 2018 Language Specification (Javascript)
- ECMA: Standard ECMA-334 - C# Language Specification
- ECMA: Standard ECMA-335 - Common Language Infrastructure (CLI)

- ECMA: Technical Report TR/84 - Common Language Infrastructure (CLI) - Information Derived from Partition IV XML File
- ECMA: Technical Report TR/89 - Common Language Infrastructure (CLI) - Common Generics
- RFC5424 - The Syslog Protocol (SYSLOG)
- W3C: RDF 1.1 Terse RDF Triple Language (Turtle)
- W3C: OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (second Edition)
- W3C: RDF 1.1 Concepts and Abstract Syntax (RDF)
- W3C: SPARQL 1.1 Overview (SPARQL)
- W3C: Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification
- W3C: HTML5 (HTML5)
- W3C: Extensible Markup Language (XML) 1.0 (Fifth Edition)
- W3C: XML Schema Definition Language (XSD) 1.1 Part 1: Structures
- W3C: XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes
- W3C: XSL Transformations (XSLT) Version 3.0
- W3C: Document Object Model (DOM) Level 3 Core Specification
- W3C: XML Path Language (XPath) 3.1
- OMG: Data Distribution Service (DDS)
- OMG: DDS Interoperability Wire Protocol (DDSI-RTPS)
- OMG: ISO/IEC C++ 2003 Language DDS PSM (DDS-PSM-Cxx)
- OMG: Java 5 Language PSM for DDS (DDS-Java)
- OMG: OPC-UA/DDS Gateway (DDS-OPCUA)
- OMG: RPC Over DDS (DDS-RPC)
- OMG: DDS Security (DDS-SECURITY)
- OMG: Web-Enabled DDS (DDS-WEB)
- OMG: DDS Consolidated XML Syntax (DDS-XML)
- OMG: DDS For Extremely Resource Constrained Environments (DDS-XRCE)
- OMG: Extensible and Dynamic Topic Types for DDS (DDS-XTypes)

de facto Standards

- Apache: Log4j
- Apache: Log4cxx
- Apache: log4php
- Apache: log4net
- Apache: log4jscala
- Bitcoin: Developer's Guidance
- Ethereum: cpp Project
- Ethereum: Ethereumh Project
- Ethereum: Ethereumjs-lib Project
- Ethereum: Ethereum_j Project
- Ethereum: Go-ethereum Project
- Ethereum: Parity Project
- Ethereum: Pyethapp Project
- Ethereum: Ruby-ethereum Project
- EIP 20: ERC-20 Token Standard
- Ethereum: Ethereum Virtual Machine (EVM)

- Ethereum: Solidity Language Specification
- Linux Foundation: Hyperledger
- Oracle: The Java® Language Specification SE 8 Edition
- Oracle: The Java® Virtual Machine Specification JVM
- Oracle: Java logger API
- Google: Go (software language)
- InterPlanetary File System (IPFS)

Tools

- None at this time

44)

https://en.wikipedia.org/wiki/Ada_(programming_language

45)

"Understanding the JVM Architecture", Joydip Kanjilal, Developer.com, 16 February 2015, https://www.developer.com/java/data/understanding-the-jvm-architecture.html

46)

"Components of .Net Framework", DeveloperIn.Net, Jayanthan JVP, http://www.developerin.net/a/39-Intro-to-.Net-FrameWork/23-Components-of-.Net-Framework

47)

"Cross platform, open source .Net Framework", The Mono Project, https://www.mono-project.com/

48)

"Language Run-Time Systems: An Overview", Evgenij Belikov, Heriot-Watt University, School of Mathematical and Computer Sciences Riccarton, EH14 4AS, Edinburgh, Scotland, UK https://pdfs.semanticscholar.org/b20c/0295a0661a4c175fcd5acc0ea49e9caf3ca1.pdf

2.2.2.2.3 DIDO Platform

return to Node View

A DIDO platform definition encapsulates the complete environment that supports a running DIDO. The DIDO platform covers the hardware, operating system (OS), DIDO software, and the connection to the network. There are potentially many related DIDO platforms defined for any particular domain. Each DIDO platform can vary the hardware, OS, type of network connection, and potentially the DIDO software if the software can interoperate.

For example, a domain defined for a supply chain might have a set of DIDO platforms defined. A DIDO platform for Windows, Linux, UNIX, Android, Mac OS, and IOS can work on TCP/IP machines or TCP/UDP protocols.

Standards

Technical Standards

- None at this time

de facto Standards

- Bitcoin: Bitcoinj Developer's Documentation
- Bitcoin: Developer's Guidance
- Bitcoin: Bitcoin Improvement Proposals (BIPs)
- Ethereum: cpp Project
- Ethereum: Ethereumh Project
- Ethereum: Ethereumjs-lib Project
- Ethereum: Ethereum_j Project
- Ethereum: Go-ethereum Project
- Ethereum: Parity Project
- Ethereum: Pyethapp Project
- Ethereum: Ruby-ethereum Project
- Google: Protocol Buffers

Tools

- None at this time

2.2.2.2.4 Distributed Applications

return to Node View

A **distributed application** is software that does not reside on a single computer, centralized server, decentralized server, or set of clustered servers. The application is distributed among nodes on the network. Each instance of the application on each node executes the same code and gets the same results based on the current state of the data and the input data sent to it. In other words, the distributed application's nodes are deterministic in nature.

Standards

Technical Standards

- RFC6455 - The WebSocket Protocol
- RFC0793 - Transmission Control Protocol
- RFC2104 - Keyed-Hashing for Message Authentication (HMAC)
- RFC7235 - Hypertext Transfer Protocol (HTTP/1.1): Authentication
- RFC2818 - HTTP Over TLS (HTTPS)
- RFC0791 - Internet Protocol (IPv4)
- RFC2460 - Internet Protocol, Version 6 (IPv6) Specification
- RFC6749 - The OAuth 2.0 Authorization Framework
- RFC6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage
- RFC1112 - Host Extensions for IP Multicasting
- RFC2315 - Cryptographic Message Syntax
- RFC3447 - PKCS #1: RSA Cryptography Specifications
- RFC6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0
- RFC2246 - The TLS Protocol
- RFC0768 - User Datagram Protocol (UDP)
- ISO/IEC 9899:2018 Programming languages -- C
- ISO/IEC 14882:2017 Programming languages -- C++
- ECMA: Standard ECMA-262 - ECMAScript® 2018 Language Specification (Javascript)
- ECMA: Standard ECMA-334 - C# Language Specification
- ECMA: Standard ECMA-335 - Common Language Infrastructure (CLI)
- ECMA: Technical Report TR/84 - Common Language Infrastructure (CLI) - Information Derived from Partition IV XML File
- ECMA: Technical Report TR/89 - Common Language Infrastructure (CLI) - Common Generics
- RFC5424 - The Syslog Protocol (SYSLOG)
- W3C: RDF 1.1 Terse RDF Triple Language (Turtle)
- W3C: OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (second Edition)
- W3C: RDF 1.1 Concepts and Abstract Syntax (RDF)
- W3C: SPARQL 1.1 Overview (SPARQL)
- W3C: Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification

- W3C: HTML5 (HTML5)
- W3C: Extensible Markup Language (XML) 1.0 (Fifth Edition)
- W3C: XML Schema Definition Language (XSD) 1.1 Part 1: Structures
- W3C: XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes
- W3C: XSL Transformations (XSLT) Version 3.0
- W3C: Document Object Model (DOM) Level 3 Core Specification
- W3C: XML Path Language (XPath) 3.1
- OMG: Data Distribution Service (DDS)
- OMG: DDS Interoperability Wire Protocol (DDSI-RTPS)
- OMG: ISO/IEC C++ 2003 Language DDS PSM (DDS-PSM-Cxx)
- OMG: Java 5 Language PSM for DDS (DDS-Java)
- OMG: OPC-UA/DDS Gateway (DDS-OPCUA)
- OMG: RPC Over DDS (DDS-RPC)
- OMG: DDS Security (DDS-SECURITY)
- OMG: Web-Enabled DDS (DDS-WEB)
- OMG: DDS Consolidated XML Syntax (DDS-XML)
- OMG: DDS For Extremely Resource Constrained Environments (DDS-XRCE)
- OMG: Extensible and Dynamic Topic Types for DDS (DDS-XTypes)

de facto Standards

- Apache: Log4j
- Apache: Log4cxx
- Apache: log4php
- Apache: log4net
- Apache: log4jscala
- Bitcoin: Developer's Guidance
- Ethereum: cpp Project
- Ethereum: Ethereumh Project
- Ethereum: Ethereumjs-lib Project
- Ethereum: Ethereum_j Project
- Ethereum: Go-ethereum Project
- Ethereum: Parity Project
- Ethereum: Pyethapp Project
- Ethereum: Ruby-ethereum Project
- EIP 20: ERC-20 Token Standard
- Ethereum: Ethereum Virtual Machine (EVM)
- Ethereum: Solidity Language Specification
- Linux Foundation: Hyperledger
- Oracle: The Java® Language Specification SE 8 Edition
- Oracle: The Java® Virtual Machine Specification JVM
- Oracle: Java logger API
- Google: Go (software language)
- InterPlanetary File System (IPFS)

Tools

- Tools: Network Traffic Analysis

2.2.2.3 Node Architecture

return to Node Network View

The DIDO node architecture establishes a component model as a reference for evaluating the functionality or data available within a DIDO implementation. The reference components are conceptual in nature; thus they may or may not represent actual components within a DIDO implementation. However, the functionality of these reference components should be part of the implementations.

At the most elementary level, reference components provide a common vocabulary for DIDO implementations. For example, Bitcoin does not have a separate standalone component that performs secure messaging; however, the Bitcoin software *does* provide secure messaging functionality using a protocol built upon cryptography. This binds the secure messaging and protocol together. It is possible to write applications that adhere to this protocol using the cryptographic rules and standards defined by Bitcoin and not use the Bitcoin software; however, the usual approach is to build upon the open source Bitcoin software and use it “as-is.” Another example is where Bitcoin provides an actual component referred to as a Wallet, bearing in mind there are numerous other wallets available that can contain and manage Bitcoins.[49)]

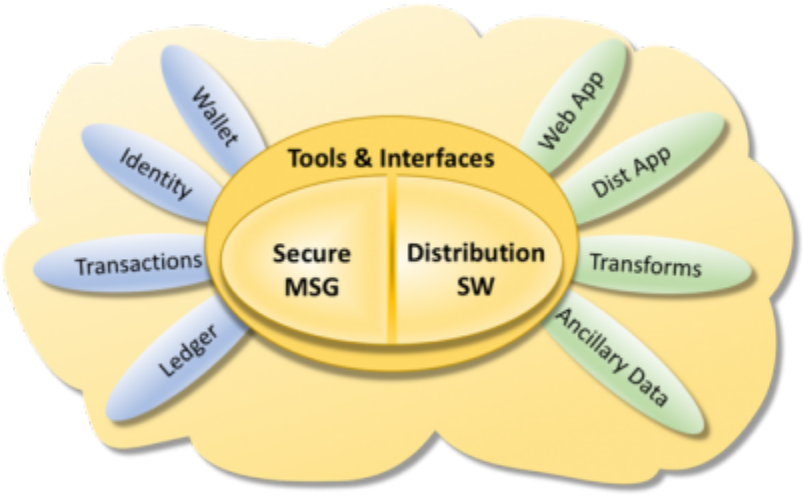


Figure 14: The DIDO Node Component Model

The DIDO node component model presented in this diagram (Figure 14) may look like a traditional stack or layered architecture as described in centralized or decentralized models; however, a very important difference is that the components or a subset of the components of this architecture are usually repeated at every DIDO node that participates in the DIDO network. At a minimum, the secure message and the distribution software components must exist at each node in order for the network to remain distributed and operational. In addition to providing for the core components required to securely communicate, a DIDO node can take on different roles or functions (refer to Figure 15). Some nodes may use all the components within the DIDO node component model while others may only use a subset of these components. For example, a smart contract node may use the identity, transaction, ledger, distributed app, and ancillary data components. In contrast, a Wallet node may just use wallet and identity components.

Figure 15: Examples of kinds of DIDO Nodes

When the entire DIDO network is assembled, it may look something like Figure 16. Each node has the basic common core components and the set of components required for it to fulfill its specific responsibilities.

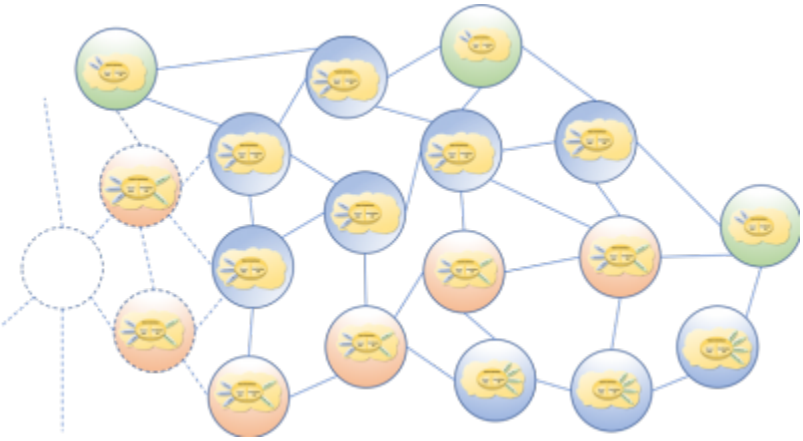


Figure 16: The DIDO Network Model

The DIDO network acts as a single system or entity, in which each node performs the operations required of it as a participant. For example, some DIDO networks may only provide a ledger, account numbers, and support transactions on the ledger. These would require each node to have the secure message, distribution software, ledger, identity and transaction components in common.

All the components within the network must be interoperable at the core functional level. For example, most of the nodes might run the Common Core version 1, whereas a subset might be running version 1.1. As long as the nodes within the network can interoperate and function together, the network is considered viable as a whole.

The interoperability of nodes within the DIDO and overall DIDO network viability become key benefits of DIDO implementations. This means that anything affecting interoperability is critical, and requires that interfaces and interactions (i.e., protocols) between the nodes must be the most conservative of all components. In other words, interfaces and interactions of the nodes are the mission critical aspects of the network and, therefore must be stable from the onset. Interruption or discontinuity in the critical path could result in a fork of the underlying ledger. An example of a fork is the "hard fork" in Ethereum called Byzantium:

> *The Byzantium hard fork is an update to ethereum's blockchain that was implemented in October 2017 at block 4,370,000. It consisted of nine Ethereum Improvement Protocols (EIPs) designed to improve ethereum's privacy, scalability and security attributes.*[50)]

Common Core

The Common Core contains components that are used by both the ledger and ancillary data subset of components, as well as characteristics and attributes that apply to all components within the DIDO network. There are three classes of common components: tools and interfaces, distribution software, and secure messaging. Common components can be used exclusively by one kind of DIDO node as defined in node architecture but can also span across the components within a node. For example, the transaction API is available to both the ledger and the ancillary data node and potentially by some of the tools. However, the transaction API may or may not be used as part of the ancillary data stack.

By definition, a ledger operation node must rely on the transaction API to access the ledger; however, a smart contract node may also require access to the ledger. In that case, its access is made exclusively through the transaction API. It is up to the implementation of a particular DIDO whether to access its ancillary data through a transaction API or a transform API. Conversely, a transaction might require ancillary data (i.e., monetary exchange rate, stock quotes, interest rates, market cap, or Beta, etc.) in order to complete. The Common Core contains an API that defines and allows for this access, sometimes referred to as an oracle.

Note: Transaction API and Transform API are defined in more detail in Section 2.2.2.4 Messaging View.

- 2.2.2.3.1 Immutable Data Objects
- 2.2.2.3.2 Ancillary Data
- 2.2.2.3.3 Semantic Web
- 2.2.2.3.4 Software

49)

A. Hertig, “Does the original Bitcoin Wallet Still Matter?,” 16 September 2016. https://www.coindesk.com/original-bitcoin-wallet-still-matter/.

50)

R. Sharma, “What is the Byzantium Hard Fork in Ethereum?,” 7 March 2018. https://www.investopedia.com/news/what-byzantium-hard-fork-ethereum/

2.2.2.3.1 Immutable Data Objects

return to Node Architecture

An immutable data object is data whose value cannot be changed. Any required update is recorded as a new immutable data object with a link back to its parent(s). An immutable data object represents things either real or virtual. The representation can range from a simple scalar value to a complex data structure of values. The things or items data objects represent are real-world things such as commodities or information. Commodities include things like gold, silver, grains, and so on. Information includes things like certificates of birth, death, marriage, or stock. Data objects can also represent virtual or abstract things such as virtual coins, frequent flier miles, loyalty points, data rights, etc.

The immutable data object component is focused on the care and maintenance of distributed, global data objects usually stored in a ledger, the identifiers (i.e., accounts) associated with the ledger, the transactions used to make updates to the ledger entries, and the wallets that contain identifier (i.e., account) information for a user. When immutable data objects represent a currency such as cryptocurrencies, the identifiers represent accounts; however, when the immutable data objects represent data such as commodities, inventories, and public records, the identifiers may not represent accounts but unique identifiers of the thing.

2.2.2.3.1.1 Ledger

return to Immutable Data Objects

Typically, an immutable data object is stored as a ledger. The term *ledger* is a distributed object that has its root within accounting systems and is comprised of a series of entries that represent an account at different stages in its life. This includes its value, an operation to be performed on the value, and potentially another account that receives or sends an amount. Ledger entries are immutable, i.e., once an entry has been added to the ledger, it can never be modified. New versions of the entry exist, and transactions capture the changes required to move between the data entry values.

For example, if the original value for an entry was AAAA and a transaction wants to modify the value to BBBB, the original value remains in the ledger and a new entry is made with the value BBBB. The new entry points back to the previous value that contained AAAA. Thus, by following the chain of modifications backwards, the provenance and ultimately the pedigree of the value can be determined.

The ledger is not limited to account numbers but can have general identifiers which represent fungible data. For example, a library has books, but the books are referenced by an identifier, not an account. Patrons of the library may have identifiers which might look like accounts, but simply identify the patron.

The ledger exists at all nodes within the network. In the blockchain implementations of DIDO, all the values in the ledger for a particular entity will be identical when all the outstanding transactions contained within a block have been validated and verified and properly distributed throughout the network. There are alternatives to blocks and blockchains. For example, Hashgraphs which have been proven to also solve the Byzantine general problem asynchronously without having to use expensive Bitcoin PoW algorithms. [51)]

51)

G. Kinglsy, “Hashgraph vs. Blockchain Is the end of Bitcoin and Ethereum near?,” coincodex, January 2018. https://coincodex.com/article/1151/hashgraph-vs-blockchain-is-the-end-of-bitcoin-and-ethereum-near/.

2.2.2.3.1.2 Transactions

return to Immutable Data Objects

Transactions contain cryptographically signed data required to describe the creation, transfer, or destruction of fungible data representing an asset stored within the ledger. The transaction captures the change in state of the contents of the ledger. Currently, each implementation of Blockchain defines its own unique concept of a transaction which depends on the kind of fungible data stored in the ledger.

Transactions represent operations that can be performed on fungible data represented in the ledger. In a financial ledger, the money associated with an account is the fungible data and it is represented by the balance associated with an account associated with an entry in the ledger. Money can only be added to or deducted from an account. Note: The actual money is not stored in the ledger, only a balance representing money is stored in the ledger. A slightly higher level concept would be the transfer money from one account to another (i.e., deduct from account nnn1 and add to account nnn2).

2.2.2.3.1.3 Identities

return to Immutable Data Objects

Associated with items in the ledger are identifiers which associate fungible data represented in the ledger with accounts or associated with transactions. In classic financial Ledgers, these identifiers are generally account numbers that contain a balance (a tally) of the fungible data associated with the account and the entry number of the row within the ledger. For example, account nnn1 has a balance of $50. Ledger Entry ''ttt2'' adds $25 to account nnn1.

The evolution of advanced symbologies has helped the securities industry grow, but the limitations and costs imposed by the closed systems have become more apparent as companies and institutions continue to integrate operations on a global scale. Proprietary symbology now stands as one of the most significant barriers to increased efficiency and innovation in an industry that sorely needs it. Moreover, the lack of common identifiers is a key roadblock to achieving the holy grail of Straight-through Processing (StP). [52)]

[52)]

Object Management Group (OMG), “Financial Industry Global Identifier® (FIGI™), v1.0,” December 2015. http://www.omg.org/spec/FIGI/1.0/.

2.2.2.3.1.4 Wallets

return to Immutable Data Objects

Wallets are not quite like physical wallets carried around in pockets or handbags. They are repositories where the cryptographic keys associated with the fungible data managed by the ledger are stored. These keys are required to unlock access to the fungible data in the ledger. For example, within the Bitcoin Blockchain, there is an identifier associated with each Bitcoin. To use the Bitcoin, a public and a private key are required. The private keys are stored in the wallet.

2.2.2.3.2 Ancillary Data

return to Node Architecture

Ancillary ddta is focused on data that is not directly part of the fungible data stored in the ledger but is used in the support, care, and maintenance of the fungible data. Ancillary data is global in nature (all nodes need access to it) and is like the fungible data in the ledger, distributed in nature (all nodes have copies of the data). Some DIDO implementations may provide access to ancillary data through the use of oracles; however, if the ancillary data is not also implemented as a DIDO but as frontend to centralized or even decentralized data, many of the benefits of the distributed data are lost. For example, if the exchange rate between currencies is not also distributed but offered from a server through an oracle, access to the exchange data becomes a vulnerability to the use of the cryptocurrency data when an exchange rate is required.

Access to ancillary data depends on the source of the data. Some implementations will store all the ancillary data on the centralized server, others will provide access to ancillary data stored on a series of decentralized servers, while others will provide access to the data as fully decentralized networks (i.e., other DIDOs). Ancillary data is accessed through an oracle.

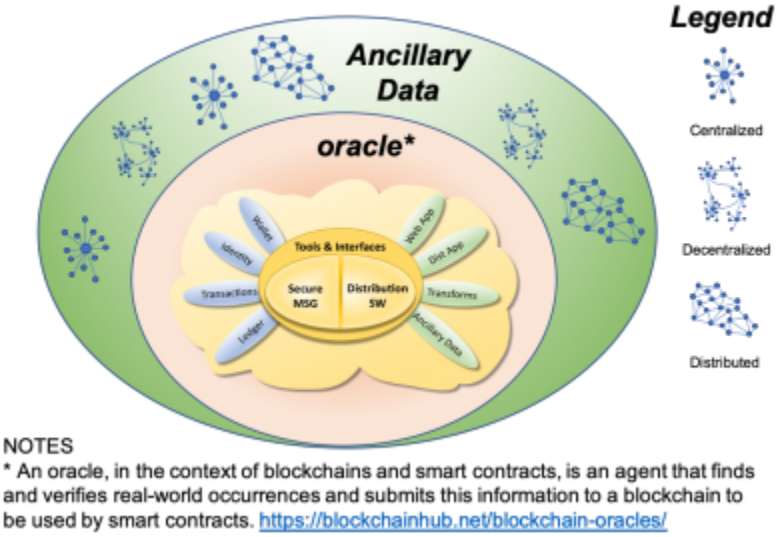


Figure 17: Relationship of Ledger, Ancillary Data and Oracles to other Network Topographies

The ancillary data, when implemented as a DIDO, is hosted in its own network of nodes which may or may not be the same set of nodes as the fiduciary data. It is comprised of the ancillary data itself, a mechanism for transforming the data (i.e., not necessarily a ledger transaction), and software that is distributed on all the nodes in its network, called web applications, which may or not run on a node within the network.

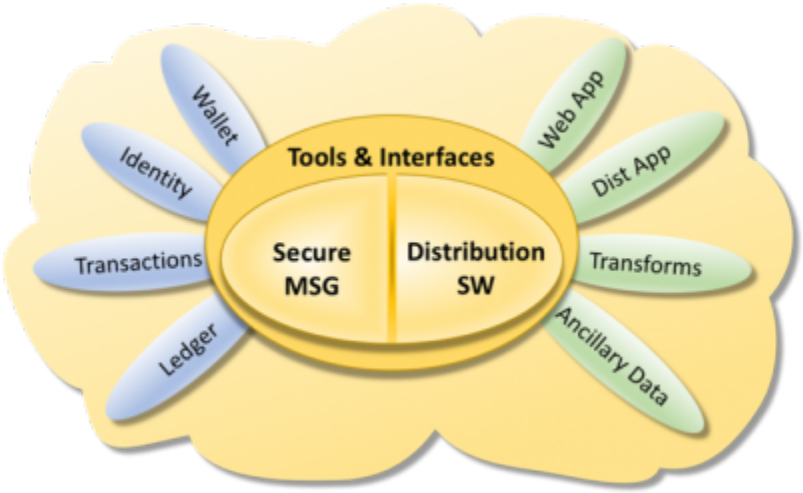


Figure 18: The Components in the Ancillary Data Stack of the DIDO Node

- 2.2.2.3.2.1 Journal
- 2.2.2.3.2.2 Transforms
- 2.2.2.3.2.3 Distributed Applications
- 2.2.2.3.2.4 Web Applications
- 2.2.2.3.2.5 Exchanges

2.2.2.3.2.1 Journal

return to Ancillary Data

The journal is data that is ancillary data to the main fungible data. It may or may not be implemented using a separate ledger and it may exist within another DIDO network, but its care and maintenance does not have to be part of the ledger's transactions. For example, the monetary exchange rate between two currencies is ancillary to a cryptocurrency ledger; however, the almost instantaneous updates to the exchange rate would not be appropriate in the cryptocurrency ledger itself.

2.2.2.3.2.2 Transforms

return to Ancillary Data

There are different ways ancillary data can be transformed. If the ancillary data is implemented as another “blockchain” DIDO, then the ancillary data is stored within another, parallel ledger and then, naturally, the transforms would be accomplished using that ledger stack’s transactions.

However, another mechanism available to transform the ancillary data is operational transformation (OT). These are similar to transactions but there can be multiple changes made to the data at the same time by multiple parties. Ancillary data using operational transforms is not necessarily immutable.

2.2.2.3.2.3 Distributed Applications

return to Ancillary Data

One of the major extensions to the original Satoshi Nakamoto [1] [27] papers on peer-to-peer electronic cash systems is the use of oracles to access external data and Smart Contract to enforce business rules on ledger transactions. For example, an account holder wants to transfer Bitcoins from one account to another, but there is a “reserve” placed on the account to cover potential debts from option trading. This “reserve” information would not be part of the Ledger, but would be ancillary data. The enforcement of the “reserve” is done by a distributed application called a smart contract.

> *A distributed application is software that is executed or runs on multiple computers within a network. These applications interact in order to achieve a specific goal or task. Traditional applications relied on a single system to run them. Even in the client-server model, the application software had to run on either the [dido:public:ra:xapend:xapend.a_glossary:c:client]], or the server that the client was accessing. However, distributed applications run on both simultaneously.*
>
> *With distributed applications, if a node that is running a particular application goes down, another node can resume the task.* [1)]

A distributed application (DApp) is software that is executed or runs on multiple computers within a network simultaneously. The software is deterministic meaning that given the same inputs, they all produce the same outputs. One of the main benefits of DApps is they are extremely durable and hardened against a single point of failure. Therefore, to be distributed, deterministic and durable, any services that the DApp must interact with must also be distributed, deterministic, and durable. This makes Distributed Application (ĐApp or DApp) interaction with traditional client/server cloud services difficult. Naturally, cloud services applications such as Software as a Service (SaaS) and Data as a Service (DaaS) have some redundancy and reliability built into them; however, they cannot achieve the same level of robustness as a DApp. Therefore, SaaS and DaaS need to expose their functionality using a proxy DApp, which by its nature has latency. This latency could adversely affect the DApp’s deterministic and durability nature.

Another important aspect of a DApp is that it should be runtime environment agnostic: allowing as many DIDO nodes into the DIDO network as possible. This can be achieved using Virtual Machine (VM) such as a Java Virtual Machine (JVM) or interpretive engines such as those available with ECMAScript. There are some platform specific virtual machines available such as the Common Language Runtime (CLR) which are not standardized and do not run with the deterministic rigor on non-Windows platforms. Consequently, the list of languages allowable in DApps is limited to those that have a standardized runtime environment (i.e., VM or Engine) that runs on multiple platforms.

1)

Techopedia, “Techopedia Definitions,” 1 December 2017. https://www.techopedia.com/definition/23971/distributed-application.

2.2.2.3.2.4 Web Applications

return to Ancillary Data

Web applications are the main interface between the human end-users of a DIDO and the rest of the DIDO. The web applications do not have the same requirements of determinism as defined in the DIDO and can be subject to far less stringent timing requirements. The equivalent of a web application in the ledger stack is a wallet.

> *A web application or "web app" is a software program that runs on a web server. Unlike traditional desktop applications, which are launched by your operating system, web apps must be accessed through a web browser.* [53]

=-
=-

[53] TechTerms, "TechTerms Definitions," 1 December 2017. https://techterms.com/definition/web_application.

2.2.2.3.2.5 Exchanges

return to Ancillary Data

Investopedia defines an Exchange as:

An exchange is a marketplace in which securities, commodities, derivatives and other financial instruments are traded. The core function of an exchange is to ensure fair and orderly trading and the efficient dissemination of price information for any securities trading on that exchange. Exchanges give companies, governments and other groups a platform from which to sell securities to the investing public. [54)]

Although this definition is meant to be specific to existing physical and electronic exchanges covering "securities, commodities, derivatives and other financial instruments" offered on the New York Stock Exchange (NYSE), Nasdaq, London Stock Exchange (LSE), and Tokyo Stock Exchange (TSE), it is also applicable when immutable data objects themselves become financial instruments and when they are applied to public records, certificates, or supply chains.

An exchange on the surface may seem to be a trivial concept and if this were so, the development of laws, rules, and regulations governing them would also be trivial. However, a quick look at the rules governing the NYSE alone is daunting [55)] . These rules are not to torment or torture traders, or to limit trades; rather they are to "ensure fair and orderly trading".

As the DIDO exchange concepts grow and mature, the need for, and the development of, rules and regulations to "ensure fair and orderly trading" must also adopt the rules and regulations expected from any exchange and evolve to handle the unique benefits, functions, capabilities of DIDOs.

Some of the types of trades that an exchange could handle are:

- **Market Order (MKT)** – A market order is an order to buy or sell a stock at the best available price. Generally, this type of order will be executed immediately. However, the price at which a market order will be executed is not guaranteed. It is important for investors to remember that the last-traded price is not necessarily the price at which a market order will be executed. In fast-moving markets, the price at which a market order will execute often deviates from the last-traded price or "real time" quote. [56)]
- **Limit Orders (LMT)** – A limit order is an order to buy or sell a stock at a specific price or better. A buy limit order can only be executed at the limit price or lower, and a sell limit order can only be executed at the limit price or higher. A limit order is not guaranteed to execute. A limit order can only be filled if the stock's market price reaches the limit price. While limit orders do not guarantee execution, they help ensure that an investor does not pay more than a pre-determined price for a stock. [57)]
- **Stop Order (STP)** – A stop order, also referred to as a stop-loss order, is an order to buy or sell a stock once the price of the stock reaches a specified price, known as the stop price. When the stop price is reached, a stop order becomes a market order. A buy stop order is entered at a stop price above the current market price. Investors generally use a buy stop order to limit a loss or to protect

a profit on a stock that they have sold short. A sell stop order is entered at a stop price below the current market price. Investors generally use a sell stop order to limit a loss or to protect a profit on a stock that they own. [58)]

- **Stop Limit Order (STPLMT)** – A stop-limit order is an order to buy or sell a stock that combines the features of a stop order and a limit order. once the stop price is reached, a stop-limit order becomes a limit order that will be executed at a specified price (or better). The benefit of a stop-limit order is that the investor can control the price at which the order can be executed. [59)]
- Some of the types of trades that need to be considered during an Exchange are **Fill-Or-Kill (FOK)** – An FOK order is an order to buy or sell a stock that must be executed immediately in its entirety; otherwise, the entire order will be cancelled (i.e., no partial execution of the order is allowed). [60)]
- **All-Or-None (AON)** – An Aon order is an order to buy or sell a stock that must be executed in its entirety, or not executed at all. However, unlike the FoK orders, Aon orders that cannot be executed immediately remain active until they are executed or cancelled. [61)]
- **Market If Touched (MIT)** – A market-if-touched, or MIT, order is a conditional order that becomes a market order when a security reaches a specified price. When using a buy market-if-touched order, a broker will wait until the security falls to a certain level before purchasing the asset. A sell market-if-touched order will activate when the price of a security rises to the specified level. [62)]

This document provides a list of standards associated with each of the components within the DIDO Reference Architecture. In this section the descriptive text for each standard or specification is taken directly from the original standard or specification and is properly attributed with a reference to that standard or specification. This has been done in order to aid readers of the DIDO RA to be able to determine applicability and usefulness of the standard or specification to the particular instance of a DIDO they are working on.

Re-writing the descriptive text can result in a misunderstanding of the original intent of those standards and specifications. If you have ever been involved in the writing of a standard or a specification, you'll be familiar with the long discussions and debates on the selection of each word and the punctuation used. Therefore, the text in the standards listed below is duplicated as much as possible "intact".

54)
Investopedia, "Exchange,". https://www.investopedia.com/terms/e/exchange.asp#ixzz5OGeT4QUj.
55)
New York Stock Exchange, "NYSE Tools," http://wallstreet.cch.com/NYSETools/PlatformViewer.asp?selectednode=chp%5F1%5F2%5F1%5F35&manual=%2Fnyse%2Frules%2Fnyse%2Drules%2F.
56)
U.S. Securities and Exchange Commission, "Market Order," https://www.sec.gov/fast-answers/answersmktordhtm.html.
57)
U.S. Securities and Exchange Commission, "Limit Orders," SEC, https://www.sec.gov/fast-answers/answerslimithtm.html.
58)
U.S. Securities and Exchange Commission, "Stop Order," SEC, https://www.sec.gov/fast-answers/answersstopordhtm.html.
59)
U.S. Securities and Exchange Commission, SEC,

https://www.sec.gov/fast-answers/answersstoplimhtm.html.
60)

U.S. Security and Exchange Commission, SEC,
https://www.investor.gov/additional-resources/general-resources/glossary/fill-or-kill-order.
61)

U.S. Security and Exchange Commission, SEC
https://www.investor.gov/additional-resources/general-resources/glossary/all-or-none-order.
62)

Investopedia, Market If Touched - MIT,
https://www.investopedia.com/terms/m/marketiftouched.asp#ixzz5OHOMW2ba]].

2.2.2.3.3 Semantic Web

return to Node Architecture

A major promise of DIDO is the ability to use machines to automatically process data and transactions rather than to rely on humans “in the middle.” A key to automating many of these processes is realized through the adoption of the semantic web. In essence, the semantic web is the institutional memory and experience captured in machine readable form and available at each node. This eliminates the centralized human-centric requirements of the past and supports a distributed, automated solution. The semantic web uses formal semantics that unambiguously capture the vocabulary and ontology of the domain.

2.2.2.3.4 Software

return to Node Architecture

By their very nature, DIDOs are run on distributed, decentralized computers that require software to operate. The DIDO software itself can be classified as ancillary data and therefore can be distributed just like data to each DIDO node. The DIDO software could include Smart Contract, Distributed Application (ĐApp or DApp), scripts, containers, and DIDO command line interface (CLI) commands.

Smart contracts are programs actually stored on the blockchain, and “triggered” to execute when a set of conditions are met. dApps are applications that don't reside on the blockchain but interact with the blockchain. dApps are used to communicate with smart contacts, and consequently with blockchain.

In other words, dApps can be considered as “blockchain-enabled” application, and smart contracts provide easy access the blockchain. This division of responsibilities is similar to the traditional separation used in Web Application (Web App) between the user-front-end (i.e., Client) and the backed (i.e., Server). Contract development is concerned with managing agreements or transactions.

There are many non-functional benefits of dApps such as:

- Maintainability
- Portability
- Scalability
- Interoperability

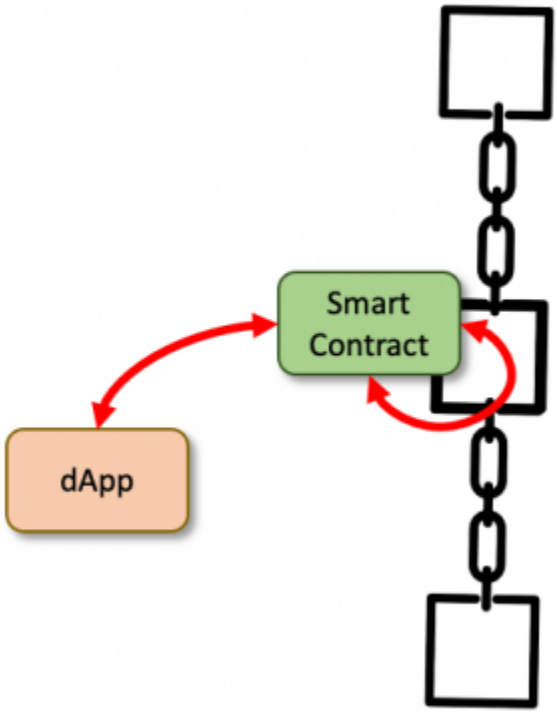


Figure 19: The Relationship Between the dApp, the Smart Contract and the blockchain

- **Note:** Not only is the data deployed on the blockchain, the smart contract itself is deployed on the blockchain. Therefore, before the Smart Contract can be used, it must be deployed to the blockchain. With slight modification, the dApp could interact with any system regardless of it is a distributed system or not as long as the interface remains the same.

2.2.2.4 Messaging View

return to Node Network View

There are three classes of messages that are used within the DIDO node network.

Transactions

Transactions are messages containing instructions on how to change the ledger from one known state to the next state. Transactions are sent to all the nodes within the Node Network providing instructions on how to modify the ledger data to the next state. When all the transactions have been applied to all ledgers on all of the individual nodes within the node network, the ledgers will have the same state and can be treated as a single datastore. These transactions are executed using a transaction Application Programming Interface (API), an API.

Transforms

Like transactions, transforms are instructions that change the state of a “ledger” from one state to another. Transforms are not as order dependent as transactions and are NOT sent to all the nodes. Transforms may or may not be sent to DIDO Nodes. The Transform Network represents Transform Nodes participating in the transformation of some content. Once the content transformation is complete, the changes made from the initial state of the content to the new state of the content are formulated into a Transaction and sent to all the Nodes in the DIDO Network. A Transform API is used to perform this action.

Streams

Streams are a way of subsetting or filtering DIDO Transactions using a publish/subscribe paradigm. This allows any particular Node within the Node Network not to have to listen-to or process-all the transactions presented to it. This is similar to Transforms; however, the participants in the transforms do not have to be part of the Node Network.

2.2.3 Decentralized Finance (DeFi) Layers

return to Technical Views

The Decentralized Finance (DeFi) have defined layers 0 through 3[63)]

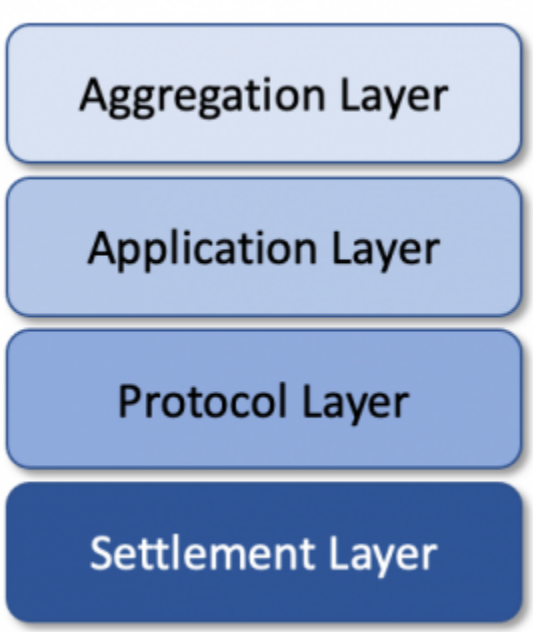


Figure 20: The Layers used in Decentralized Finance (DeFi).

Sharma defines the four layers as:

- **Settlement Layer** is also referred to as Layer 0 because it is the base layer upon which other DeFi transactions are built. It consists of a public blockchain and its native digital currency or cryptocurrency. Transactions occurring on DeFi apps are settled using this currency, which may or may not be traded in public markets. One example of the settlement layer is Ethereum and its native token ether (ETH), which is traded at crypto exchanges. The settlement layer can also have tokenized versions of assets, such as the U.S. dollar, or tokens that are digital representations of real-world assets. For example, a real estate token might represent ownership of a parcel of land.
- **Protocol Layer** are the standards and rules written to govern specific tasks or activities. In parallel with real-world institutions, this would be a set of principles and rules that all participants in a given industry have agreed to follow as a prerequisite to operating in the industry. DeFi protocols are interoperable, meaning they can be used by multiple entities at the same time to build a service or an app. The protocol layer provides liquidity to the DeFi ecosystem. One example of a DeFi protocol is Synthetix, a derivatives trading protocol on Ethereum. It is used to create synthetic versions of real-world assets.
- **Application Layer** as the name indicates, is where consumer-facing applications reside. These applications abstract underlying protocols into simple consumer-focused services. Most common applications in the cryptocurrency ecosystem, such as decentralized cryptocurrency exchanges and lending services, reside on this layer.
- **Aggregation Layer**, is the aggregation layer consists of aggregators who connect various applications from the previous layer to provide a service to investors. For example, they might

enable the seamless transfer of money between different financial instruments to maximize returns. In a physical setup, such trading actions would entail considerable paperwork and coordination. But a technology-based framework should smoothen the investing rails, allowing traders to switch between different services quickly. Lending and borrowing is an example of a service that exists on the aggregation layer. Banking services and crypto wallets are other examples.

63)

Rakesh Sharma, Investopedia, 24 March 2021, Decentralized Finance (DeFi) Definition, Accessed 24 May 2021, https://www.investopedia.com/decentralized-finance-defi-5113835

2.3 Taxonomic Views

return to Architectural Views

A taxonomy is a way of organizing things into useful and convenient classes. Each class can then be treated as an abstraction of the individual elements. For example, **Dog** is a class of **Animal**. Once an individual entity is classified as a **Dog**, generalizations can be made about what to expect from the individual entity. There can be many taxonomies that classify entities. For example, one taxonomic classification places an individual in the animal kingdom; however, another taxonomy classifies an individual according to their state of employment (working, retired, etc.). Each is valid; the only difference is perspective.

The DIDO RA employs the following four taxonomies:

2.3.1 Network Topology Taxonomy

return to Taxonomic Views

Probably one of the most significant differences between DIDO architectures and other architectures is the simple, but powerful, topology of the network of nodes it uses to connect peers. DIDOs rely primarily on a distributed rather than a decentralized or centralized topology. Each network topology defines a community of nodes that act as peers, which collectively provide a solution to a problem that is then distributed. This community of nodes also employs redundant data storage, redundant computing, or both.

Most DIDO implementations rely heavily on data and computational power that exists externally and, therefore, is beyond the primary focus of the DIDO. For example, an account holder in a cryptocurrency DIDO application may require information such as currency exchange rates, the holder's nation of origin, tax IDs, or certificates of trust. In a greenfield development, all this external data would be held within the DIDO. The reality is that it is not possible to build everything from scratch. Therefore, this data might be held in other network topologies (see section 2.3.4.2 Ancillary Data).

These three kinds of network topologies are represented graphically in the following figure.

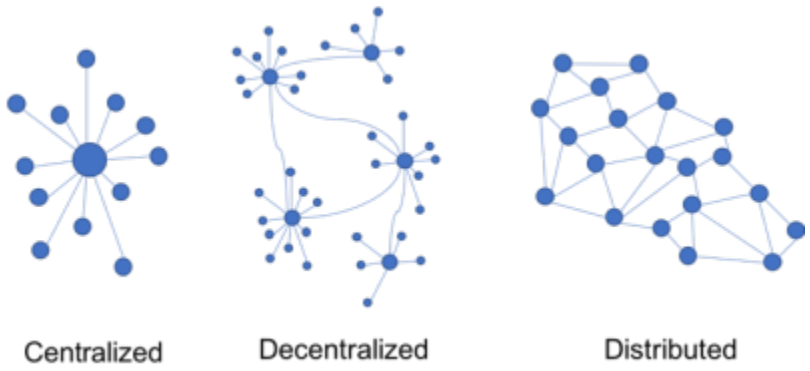


Figure 21: The Difference between Centralized, Decentralized, and Distributed Network Topologies

- 2.3.1.1 Centralized Network Topology
- 2.3.1.2 Decentralized Network Topology
- 2.3.1.3 Distributed Network Topology

2.3.1.1 Centralized Network Topology

return to Network Topology Taxonomy

The centralized network topology (usually associated with mainframes) has a very large centralized server node that all other nodes connect to. It provides most of the services such as data storage, processing, backups, and computational power for the other nodes. It is possible to implement a “ledger” using the centralized model. In many ways, the centralized model that holds a single ledger is easier to implement and maintain since there is only one version of the data in one place and consequently, by definition, the data is the canonical data (i.e., authoritative or standard data).

Figure 22: Centralized Network Topology

Trying to remove redundant, replicated, or duplicate data is a fundamental principle of the centralized model. The multiple “copy of data” problem was first formally addressed by E. F. Codd in his development of A Relational Model of Data for Large Shared Data Banks[64] and has become the cornerstone of Relational Database Management Systems (RDBMS), used extensively today in products such as Oracle Corporation’s Oracle, IBM’s DB2, or Computer Associates INGRES [65] . The following chart shows that the centralized database market is not shrinking, and by 2017 had grown to a net worth of $50 billion with no signs of change in this growth trend. [66] This highlights the magnitude of the effort to convert the world to DIDO technology. Though it may eventually happen, it’s going to take a long time.

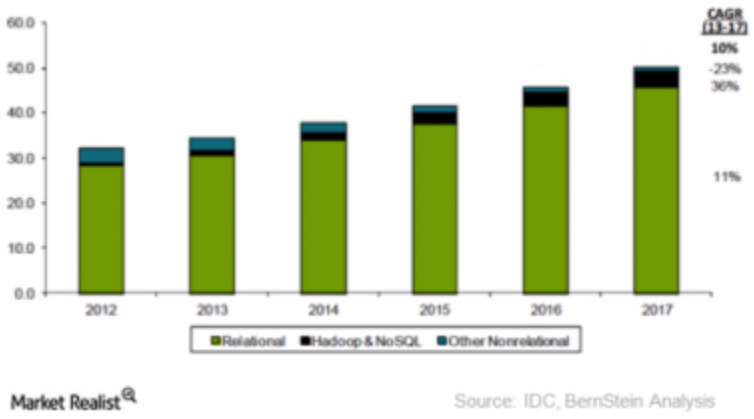


Figure 23: Centralized Global Database Market ($ Billions)

64) E. F. Codd, “A relational model of data for large shared data banks,” Communications of the ACM, vol. 13, n. 6, pp. 377-387, June 1970

65) G2 Crowd, “Best Relational Databases Software in 2018,” 2018. [Online]. Available: https://www.g2crowd.com/categories/relational-databases [Accessed 12 June 2018].

66)

A. Shields, “Is Oracle's Position Secure in the Database Space?,” 18 January 2016. [Online]. Available: http://marketrealist.com/2016/01/oracles-position-secure-database-space/. [Accessed 22 July 2017].

2.3.1.2 Decentralized Network Topology

return to Network Topology Taxonomy

In a decentralized network topology, there are several “central” nodes, each providing redundancy and failover capabilities for the other. Often, each of the centralized nodes are geographically distributed (i.e., North American, Europe, Southeast Asia, etc.).

Figure 24: Decentralized Network Topology

Decentralized systems overcome some of the problems associated with a centralized system: they do not have a single point of failure, resulting in fault tolerance.[67),68)] Thus, each node can be maintained independently, which ultimately results in a more stable system. Moreover, the load on the system can be reduced by increasing the number of centralized nodes thereby distributing the work load.

Nevertheless, the scalability of the system is moderate, since the cost of expansion per node is generally steep. Granted, much of this cost has come down with the availability of Platform as a Service (PaaS) offerings from Amazon, Microsoft, and others. Since the nodes are arranged in clusters around one of the centralized servers, there is only partial fault tolerance with the result that occasionally parts of the system are rendered unavailable. The dependence on the underlying network topology means that, depending on which network links are broken, there is a chance that the data within one of the servers is obsolete.

The decentralized model is used extensively in cloud computing, specifically Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform-as-a-Service (PaaS). Although there are implementations of cloud computing which do not use the decentralized model (e.g. blockchains), the following chart from Gartner summarizes the projected growth of the “as a Service” offerings and indicates that it is on the rise. [69)] It also highlights the magnitude of trying to “convert” the world to one that is completely distributed. Even were it to occur, it’s going to take a long time.

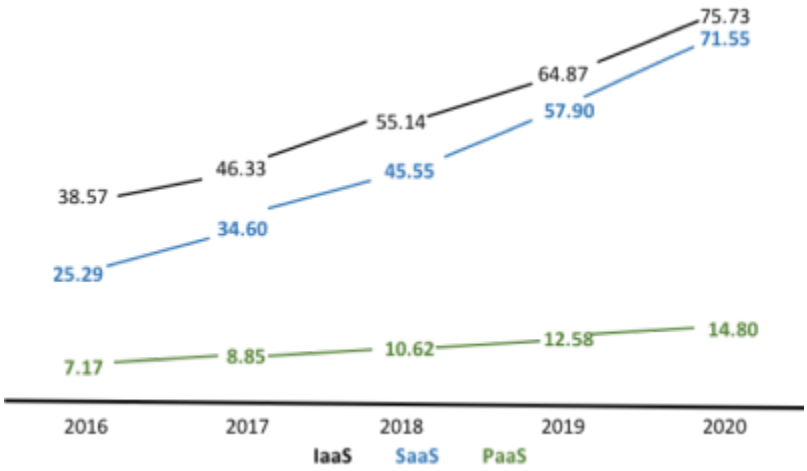


Figure 25: Decentralized Cloud Market Revenue ($ Billions)
67)

Nonetheless, the October 2016 Distributed Denial of Service (DDoS) attack on Domain Name System (DNS) servers on the US East coast and in Europe highlighted a weakness in the decentralized model.
68)

A. Shields, “Is Oracle's Position Secure in the Database Space?,” 18 January 2016. [Online]. Available: http://marketrealist.com/2016/01/oracles-position-secure-database-space/
69)

C. Coles, “Overview of Cloud Market in 2017 and Beyond,” 2016. [Online]. Available: https://www.skyhighnetworks.com/cloud-security-blog/microsoft-azure-closes-iaas-adoption-gap-with-amazon-aws/. [Accessed 24 July 2017].

2.3.1.3 Distributed Network Topology

return to Network Topology Taxonomy

In the distributed network topology, all nodes are equal peers within the system, with each acting as a redundant copy of other nodes. This provides for extremely high fault tolerance, tied to the number of nodes. The more nodes, the greater the tolerance to faults. Since each node is an equal peer, there is no single point of failure and there is no one master copy of the data. As a result, the distributed system becomes almost infinitely scalable.

Figure 26: Distributed Network Topology

However, all nodes are not really considered “equal” in the truest sense. Some nodes are used simply to create a simple transaction, others record all the information within the blockchain, and others go on to validate transactions by mining.

Figure 27: Market Cap of Cryptocurrencies ($ Billions)

2.3.1.4 Relevant Networking Standards

return to Network Topology Taxonomy

The following standards are applicable to all network views.

Technical Standards

- RFC0793 - Transmission Control Protocol
- RFC2104 - Keyed-Hashing for Message Authentication (HMAC)
- RFC7235 - Hypertext Transfer Protocol (HTTP/1.1): Authentication
- RFC2818 - HTTP Over TLS (HTTPS)
- RFC0791 - Internet Protocol (IPv4)
- RFC2460 - Internet Protocol, Version 6 (IPv6) Specification
- RFC6749 - The OAuth 2.0 Authorization Framework
- RFC6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage
- RFC1112 - Host Extensions for IP Multicasting
- RFC2315 - Cryptographic Message Syntax
- RFC3447 - PKCS #1: RSA Cryptography Specifications
- RFC6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0
- RFC2246 - The TLS Protocol
- RFC0768 - User Datagram Protocol (UDP)

de facto Standards

- None identified

2.3.2 Network Access Control Taxonomy

return to Taxonomic Views

Network access control taxonomy classifies the types of access individuals (i.e., nodes) have from outside and from within the node network. The two main classes of access control are permissionless and permissioned.[70)]

Within each of these two classifications it is possible to have public and private access. Public and private access define who is able to write data onto a network or ledger. In contrast, open (i.e., permissionless) and closed (i.e., permissioned) determine who is able to read the data. Networks are classified as[71)]:

- public and open
- public and closed
- private and open
- private and closed

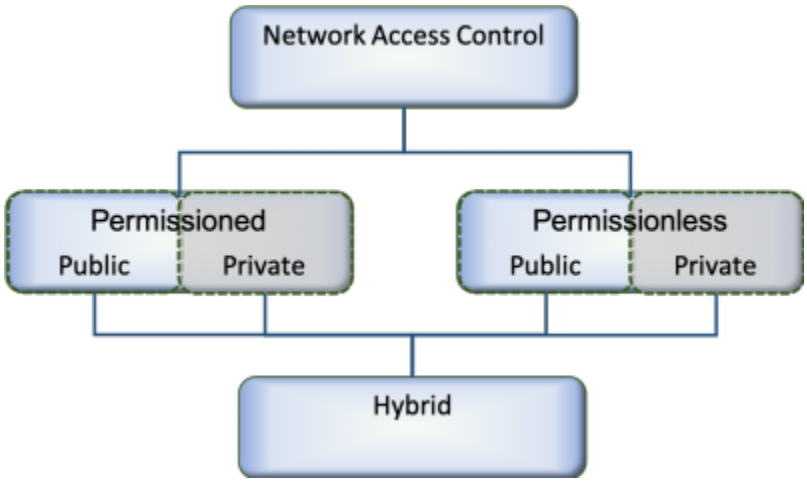


Figure 28: The Node Network Access Taxonomy

Another category of networks is a hybrid network, which makes it possible to restrict the visibility of information on the network using a combination of public, private, permissionless and permissioned networks. Therefore, hybrid networks are appealing to regulated markets because they offer the benefits of public blockchain and private blockchain together.[72)]

Table 1: Taxonomy of DIDO Access Control

Permissionless Networks	Permissioned Networks
Public Networks	Private Networks
Hybrid Networks	

70) , 71)
"Public Vs Private Blockchain In A Nutshell", Demiro Massessi, 12 December 2018, https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f

72)
"Hybrid Blockchain: Decentralized Option for Highly Regulated Markets - Few players in highly regulated markets have adopted blockchain technology. However, hybrid blockchain will change this.", Mina Down,

14 November 2018
https://blog.goodaudience.com/hybrid-blockchain-decentralize-highly-regulated-markets-900f30a37903

2.3.2.1 Permissionless Networks

return to Network Access Control

Permissionless Networks require no permission to use them. In other words, there is no barrier to entry. Anyone can run a node, run mining software/hardware, access a wallet, and write data onto and transact within the blockchain (as long as they follow the rules of the blockchain). There is no way to censor anyone, ever, on a permissionless Bitcoin blockchain.[73)]

Benefits of Permissionless Networks

Decentralized

Permissionless networks are decentralized and distributed. In other words, no one entity can close or terminate the network, modify the content, or censor parts of it. The larger the distributed and decentralized networks and or history are, the harder it is to tamper with.[74)]

Transparency

Users or nodes have complete access to the ledger, transactions, and blocks in the blockchains, which allows for complete auditing of permissionless networks[75)].

Anonymity

In permissionless networks, users or nodes of the network are anonymized. Technically, permissionless networks like Bitcoin are pseudonymous, and not truly anonymous.[76)]

Governance

As a general rule, permissionless networks rely on open source software, which is ruled by open source communities (see Talk Openly Develop Openly (TODO)). The governance of the network is by consensus. Consensus is different for many of the permissionless networksm(i.e., Proof of Work (PoW), Proof of Stake (PoS), Proof of Authority (PoA), etc). [77)]

Tokens

Permissionless blockchains employ fat protocols that compensate network contributors with

Tokens. As the value and utility of the network increases, the value of the underlying tokens increases as well. This is the premise of cryptoeconomics and Initial Coin Offering (ICO) based fundraising. There are two predominant types of tokens today: monetary value tokens and utility tokens. Monetary value tokens are used in myriad ways as instruments for exchanging value. Utility tokens are akin to loyalty points: they have intrinsic value but no monetary value outside of that ecosystem.[78)]

Scalability and Performance

For all the value blockchains bring to modern business processes, their Achilles heel often involves scalability and performance. Both Bitcoin and Ethereum blockchains suffer from poor scores in this area. For example, a recent blockchain game called Crypto kittles clogged the Ethereum network. Having said that, these are just early teething troubles, and startups are experimenting with various strategies to address this issue. Hopefully it is only a matter of time before this issue becomes a non-entity.[79)]

73)

"Permissioned vs. Permissionless blockchains: Who will win and will it matter?", Dustin D, 22 March 2018, Permissionless

74) , 75) , 76) , 77) , 78) , 79)

"Nuances Between Permissionless and Permissioned Blockchains", Anant Kadiyala, 18 February 2018, https://medium.com/@akadiyala/nuances-between-permissionless-and-permissioned-blockchains-f5b566f5d483

2.3.2.2 Permissioned Networks

return to Network Access Control

Permissioned Networks (Glossary Definition)

Permissioned blockchains combine the properties of both the public network and private network. Each permissioned network is unique and represents a careful balance of public and private networks to meet specific business use cases.

The available options include allowing anyone to join the permissioned network after suitable verification of their identity, and the allocation of select and designated permissions to perform only certain activities on the network. [80)]

Benefits of Permissioned Networks

Decentralization

The degree of decentralization for permissioned networks is a business decision. The extent and quality of decentralization depends upon the number of peers (i.e., nodes), the expected number of bad nodes in the network, and the type of consensus mechanism determined by the stakeholder. Permissioned blockchains usually employ an algorithm such as Byzantine Fault Tolerance, which differs from the proof of work (PoW) algorithm[81)].

Transparency

Transparency is not a driving force in permissioned networks and is often a major factor in the business decision to choose permissioned over permissionless networks. Most permissioned blockchains do not use cryptoeconomic coins incentive or tokens. The primary incentive of permissioned blockchain participants is to minimize the transparency, cost, time, and ease of sharing information[82)].

Privacy

Permissioned blockchains offer fine-grained visibility into transaction details, as well as, metadata about those transactions which, in many ways, compromises the privacy of the Network participants[83)].

Governance

There are fundamental differences between permissionless and permissioned network governance. Permissioned governance is decided and agreed upon by members of the business network. Economic incentives, code quality, code changes, and power allocation among peers are based on the business dynamics and the common purpose and goals of the permissioned members. This allows for agile and responsive networks desired by businesses[84)].

Tokens

Permissioned blockchains generally do not employ a cryptoeconomic coins incentive or tokens[85)].

Scalability and Performance

Permissioned blockchains use consensus mechanisms, which are computationally inexpensive (when compared to proof of work (PoW)). Therefore, they enjoy substantially better scalability and performance than their permissionless network cousins[86)].

80)
"Public, Private, Permissioned Blockchains Compared", Shobhit Seth, Investopedia, 10 April 2018, https://www.investopedia.com/news/public-private-permissioned-blockchains-compared/
81) , 82) , 83) , 84) , 85) , 86)
"Nuances Between Permissionless and Permissioned Blockchains", Anant Kadiyala, 18 February 2018, https://medium.com/@akadiyala/nuances-between-permissionless-and-permissioned-blockchains-f5b566f5d483

2.3.2.3 Public Networks

return to Network Access Control

Public networks are distributed and open, allowing participation in core activities of the network from any node sponsored by anyone. Participation includes joining, leaving, reading, writing, and auditing the activities on the network. There are no authorities or discrimination of nodes.

Benefits of Public Networks

Open Read and Write[87)]

Anyone can participate by submitting transactions to the blockchain, such as Ethereum or Bitcoin; transactions can be viewed on the blockchain explorer.

Ledger Is Distributed[88)]

The database is not centralized like in a client-server approach, and all nodes in the blockchain participate in the transaction validation.

Immutable[89)]

When something is written to the blockchain, it can not be changed, in other words it is immutable.

Secure Due to Mining (protection from the Fifty-One Percent (51% Attack)[90)])

For example, with Bitcoin, obtaining a majority of network power could potentially enable massive double spending, and the ability to prevent transaction confirmations, in addition to other potentially malicious acts.

=-
=-

87)
“Public Vs Private Blockchain In A Nutshell”, Demiro Massessi, 12 December 2018, https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f

88), 89), 90)
“Public Vs Private Blockchain In A Nutshell”, Demiro Massessi, 12 December 2018, https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f

2.3.2.4 Private Networks

return to Network Access Control

A **private network** limits access to the network to individuals or nodes granted and verified to have permission to participate in joining, leaving, reading, writing, and auditing activities on the network. A participant joins a private network only through an authentic and verified invitation; validation is performed either by the network operator(s) or using a clearly defined set of protocols implemented by the network.[91)]

Benefits of Private Networks

Enterprise Permissioned[92)]:

The enterprise controls the resources and access to the blockchain, hence private and/or permissioned.

Faster Transactions[93)]:

When you distribute the nodes locally, but also have far fewer nodes that participate in the ledger, performance is faster.

Better Scalability[94)]:

Being able to add nodes and services on demand can provide a great advantage to the enterprise.

Compliance Support[95)]:

As an enterprise, you would likely have compliance requirements to adhere to; having control of your infrastructure enhances ability to satisfy this requirement more seamlessly.

Consensus More Efficient (fewer nodes)[96)]:

Enterprise or private blockchains have fewer nodes and usually a different consensus algorithm, such as BFT vs PoW.

91)

“Public, Private, Permissioned Blockchain Compared”, Shobhit Seth, Investopedia, 10 April 2018, https://www.investopedia.com/news/public-private-permissioned-blockchains-compared/

92) , 93) , 94) , 95) , 96)

“Public Vs Private Blockchain In A Nutshell”, Demiro Massessi, 12 December 2018, https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f

2.3.2.5 Hybrid Networks

return to Network Access Control

A **Hybrid Blockchain** is unique in that it is decentralized while also making it possible to restrict the visibility of information on the network with a combination of public, private, permissionless and permissioned Networks. Thus, a hybrid blockchain is appealing for regulated markets as it offers the benefits of public blockchain and private blockchain together.[97)]

The hybrid blockchain's decentralized, secure, transparent, and immutable nature provides benefits similar to those offered by permissionless and public networks: it allows for restrictions on rights to view, modify, and append/approve transactions to approved participants. In simple words, if a network member does not want their transaction data to be visible or accessible without their permission, they can earmark particular rights to view, modify, or get into consensus with different members.[98)]

Benefits of Hybrid Networks

Private Transactions

Transaction are private but verifiable using the ledger's immutable data objects (i.e., leverage its public state). In its public state, each transaction gets approved by a massive network and is essentially secure and trustworthy. Hence, there is no need for a central governing body or an exhaustive chain of intermediaries to supervise things. So, any change done to a transaction will undergo a “kindred” approval process, making it next to impossible for a single actor to meddle with the transaction or entries[99)].

Equality

Everyone in the network has equal rights to view, modify, and append their consent to a transaction. In addition, the identity of transacting parties is never disclosed to all the visible network participants. [100)].

Non-Repudiation

Anonymity is simply not acceptable to financial institutions and regulated industries with their strict Know Your Customer (KYC) standards. [101)].

Confidentiality

Unrestricted visibility of the public state of the network exposes all the data to a colossal network breach, which is counter to data confidentiality obligations, as well as their business concerns.[102)].

97)

“Hybrid Blockchain: Decentralized Option for Highly Regulated Markets - Few players in highly regulated markets have adopted blockchain technology. However, hybrid blockchain will change this.”, Mina Down, 14 November 2018 https://blog.goodaudience.com/hybrid-blockchain-decentralize-highly-regulated-markets-900f30a37903

98) , 100) , 101) , 102)

“If you Thought Blockchain was Amazing, Wait till You Read about Hybrid Blockchain”, Atul Khekade, 20 January 2018, https://www.entrepreneur.com/article/307794

99)

“If you Thought Blockchain was Amazing, Wait till You Read about Hybrid Blockchain”, Atul Khekade, 20 January 2018, https://www.entrepreneur.com/article/307794. This article uses the term “agnate approval” rather than “kindred approval”; however, agnate limits a kindred relationship to males only. Thus, we prefer the term “kindred” over “agnate”

2.3.3 Node Taxonomy

return to Taxonomic Views

A node[103] is any participant in a DIDO Node Network. However, there are different types of nodes within the node network. The types are classified according to the kind of activities (i.e. roles) they need or want to perform within the node network.

All nodes must provide their own hardware to participate. The hardware requirements can range from fairly simple handheld devices to larger servers capable of storing vast amounts of data and processing many transactions. However, servers are not essential for the operation of the node network since all participants operate as peers within a peer-to-peer (P2P) network of equals (i.e., it is not a client-server model). The correct value of data (i.e. “Truth”) is achieved through consensus and results in each node having a local copy of the “true” values. Thus, there is no need to go to a central server to get the true, accurate, or current values of the data.

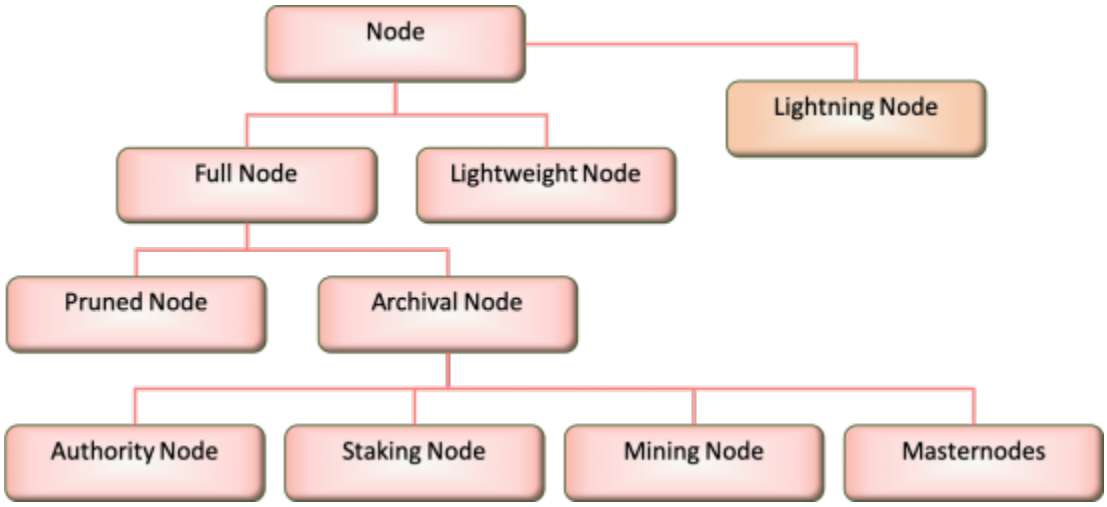


Figure 29: DIDO Node Taxonomy: Node Types

The classifications shown in Figure 1 are generalizations; each domain may define more or fewer types of nodes with each assigned different roles than those described here. For example, Bitcoin really defines only two kinds of nodes: full nodes and lightweight nodes. Full nodes have the entire copy of the ledger and can create transactions on their own. Lightweight nodes must work with a full node in order to synchronize the current “true” value of the data. Bitcoin also has a mining node, which is a full node used to validate that a block of transactions is “true.” Sometimes DIDO nodes can be referred to as clients[104] and are classified by their level of engagement with the DIDO; however, any domain can define or modify the definitions for node types.

Full Node			Lightweight	Lightening
Pruned	Archival			Permanode
	Authority	Staking	Mining	Master

103)

“Blockchain Nodes: An In-Depth Guide”, https://nodes.com/

104)

Clients are nodes able to parse and verify blockchain ledger, transactions, and smart contracts. Clients have access to APIs with which to create transactions and mine blocks.
https://ethereum.stackexchange.com/questions/269/what-exactly-is-an-ethereum-client-and-what-clients-are-there

2.3.3.1 Full Node

return to Node Taxonomy

Full nodes keep a full copy of the blockchain transactions[105]. There can be any number of full nodes within the node network, all acting as redundant data sources. Some of the activities of a full node are maintaining consensus between other nodes, verification of transactions, and storing the ledger.

In many ways the full nodes' functionality is analogous to those of servers in decentralized networks. However, there is no centralized “truth” or final judge. Instead, the “truth” is determined by consensus among the full nodes. However, this consensus based methodology is not without its pitfalls. When more than 51% of the full nodes cannot reach consensus (i.e., agree with a transaction or a proposition), the proposed change is skipped. This can lead to a Hard Fork in the ledger and the opposing groups diverge, creating two or more chains [106]. Sometimes the 51% problem can be part of an orchestrated effort, referred to as a 51% attack [107]. The more nodes in the node network, the harder it is to successfully launch a 51% attack.

A well-known example of this kind of ledger divergence, leading to a hard fork, was the Bitcoin Cash Fork.[108]

Full node contains:

- 2.3.3.1.1 Pruned Node
- 2.3.3.1.2 Archival Node

105)
Osita Chibuike, 21 May 2018, Legobox, https://dev.to/legobox/how-to-setup-an-ethereum-node-41a7
106)
“Blockchain Nodes: An In-Depth Guide”, https://nodes.com/
107)
“51% Attack”, Jake Frankenfield, 6 May, 2019, https://www.investopedia.com/terms/1/51-attack.asp
108)
“Bitcoin Cash’s Scheduled Hard Fork Tripped Up By Software Bug”, Christine Kim, 15 May 2019, https://www.coindesk.com/bitcoin-cash-scheduled-hard-fork-tripped-up-by-software-bug

2.3.3.1.1 Pruned Node

return to Full Node

A **pruned node** is a full node for all intents and purposes with one major difference: in order to save space on the node, the pruned node begins downloading blocks from the beginning of the ledger and when a preset threshold on space is exceeded, the oldest transactions are deleted, retaining only the headers and placement within the blockchain.

For example, a size limit threshold of 550MB allows storage of the latest blocks that fit within the 550MB threshold. However, to retain the integrity of the ledger, all the transactions must be processed and checked for validity in the order of their creation. At the end of the load, the state of the data is correct and only the history is of the transactions is lost.[109)]

Note: Pruned nodes are considered full nodes and thus can also verify transactions and be involved in consensus.

Standards

Technical Standards

- None at this time

de facto Standards

- None at this time

Tools

- None at this time

109)
“Blockchain Nodes: An In-Depth Guide”, https://nodes.com/

2.3.3.1.2 Archival Node

return to Full Node

In more traditional DIDOs, an **archival full node** is primarily responsible for storing the entire ledger and for providing consensus to the entire node network by validating blocks and potentially receiving a reward [110)].

As the size of the ledgers grow, there will be fewer archival nodes and more pruned nodes. IOTA refers to these archival nodes as “permanodes” and believes that business services will evolve around either charging for the storage or charging to retrieve the data that is stored or perhaps both. In some cases, such as in the Industrial Internet of Things (IIoT), the nodes that publish data might be rewarded when or if the the data is retrieved.

> **Note:** The difference between pruned and archival nodes is that pruned nodes only keep the latest validated data and require far less storage on the local node.

Archival nodes are divided into subtypes, three that can add blocks to a blockchain and one that cannot:

- Can add blocks: authority node, staking node, and mining node
- Cannot add blocks: masternode

110)
“Blockchain Nodes: An In-Depth Guide”, https://nodes.com/. Article covers the various kinds of “rewards” a node can receive. Descriptions of the Archival Node subtypes also covers rewards.

2.3.3.1.2.1 Authority Node

return to Archival Node

Traditionally, the participation in DIDO node networks to perform tasks is permissionless, requiring no outside authority from anyone or anything. Having permissionless access to a node network is one of the original guiding principles that is integral to the decentralized nature of DIDOs[111)].

> **Note:** Unfortunately, there are drawbacks to this approach. Solutions involving a level of centralization for granting permission can provide benefits like increased speed because there is no need for the costly, time consuming consensus algorithms such as Delegated Proof of Stake (DPoS), Delegated Byzantine Fault Tolerant (dBFT), Proof of Authority (PoA) and others. However, centralized permissioned systems are vulnerable to malicious attacks such as denial of service, thereby undermining many of the “democratizing” aspects of the DIDO.

Networks making use of PoA networks define a fixed number of **authority nodes**. The number of nodes and associated PoA designation is voted on by the PoA community or defined by the development team. Authority nodes are similar to full nodes and can create and validate blocks. All other nodes in the node network run as lightweight nodes, depending on broadcasted data to participate in the blockchain.[112)]

Iota refers to authority nodes as **permanodes** (definition) because they keep all transaction data even after snapshots are made (concept introduced by Iota [113)])

111)
S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 24 May 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

112)
“Blockchain Nodes: An In-Depth Guide”, https://nodes.com/

113)
“Full node vs permanode”, Helmar, 5 January 2018, https://iota.stackexchange.com/questions/782/full-node-vs-permanode/783

2.3.3.1.2.2 Staking Node

return to Archival Node

A **staking node** is usually a lightweight node (wallet). It allows for the aggregation of individual stakeholders stakes in Proof of Stake (PoS) a node network. The collection of tokens (i.e., coins) has a higher weight [114)] and improves the chances of receiving more rewards for validating a block of transactions. [115)]

Characteristics of Staking Nodes[116)]

- Ease of setup (basically a lightweight node (wallet))
- No initial startup costs
- No payout delay
- No penalty for being offline
- No minimum balance
- No guarantee of returns
- Usually smaller returns than a masternode

114)
See Weight of Network
115)
“Blockchain Nodes: An In-Depth Guide”, https://nodes.com/
116)
“Masternode vs Staking”, Solaris Support Center, 26 June 2019, https://solaris.helpsite.com/articles/24861-masternode-vs-staking

2.3.3.1.2.3 Mining Node

return to Archival Node

Mining nodes [117) 118)] are full nodes or lightweight nodes within the node network that attempt to prove they have worked and completed the required challenge before anyone else to create a block of transactions (see proof of work (PoW)). To complete the task, mining nodes perform either as full archival nodes, or receive data from other full nodes on the node network to obtain the current state of the blockchain and parameters required to describe the next block.

Mining nodes employ hardware components (i.e., CPUs, GPUs or ASICs) to solve a cryptographic problem. The first mining node to complete the task broadcasts the results to the node network for verification by full nodes. Once consensus is achieved on the determination that the solution to the problem is correct and the solution is the first to be posted, the mining node is granted the right to add a block to the existing blockchain.

As a reward for performing the work, mining nodes are given a preset number of tokens (i.e., coins), as well as the transaction fees associated with the block of transactions. The preset reward is often referred to as a *coinbase* or a *coinbase transaction*. The reward (i.e., coinbase) is usually the first transaction in the block and free. [119),120)]

117)
Not all DIDOs use “mining” as originally defined in Bitcoin as Proof of Work (PoW). This means other DIDO implementations (i.e., IOTA) may not require a full blockchain Ledger to verify a transactions validity.
118)
S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 24 May 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf.
119)
“Blockchain Nodes: An In-Depth Guide”, https://nodes.com/
120)
Osita Chibuike, 21 May 2018, Legobox, https://dev.to/legobox/how-to-setup-an-ethereum-node-41a7

2.3.3.1.2.4 Masternode

return to Archival Node

A **masternode**[121] is combination of a staking node using a Proof of Stake (PoS) node network, which relies on the weight of the stake[122], and a server.

The requirements for a successful staking node are a server, a stable internet connection, a minimum number of coins used for staking, and time to mine the server. Unlike the staking node, the masternode does not wait for randomly assigned blocks of transactions to validate but is instead constantly engaged: obtaining rewards and constantly paying out a certain number of tokens (i.e., coins) (thus the minimum stake).[123]

Characteristics of Masternodes[124]

- Higher rewards than a simple staking node
- Ability to participate in votes on proposals
- Hosting does not have to be local
- Higher cost and resource requirements than staking node
- More complexity
- Requires an initial stake

121)
"Blockchain Nodes: An In-Depth Guide", https://nodes.com/
122)
See Weight of Network
123)
"What's the difference between staking and masternode?", darkangel11, 03 February 2018, https://bitcointalk.org/index.php?topic=2874856.0
124)
"Masternode vs Staking", Solaris Support Center, 26 June 2019, https://solaris.helpsite.com/articles/24861-masternode-vs-staking

2.3.3.2 Lightweight Node (Wallet)

return to Node Taxonomy

Lightweight nodes keep a shallow copy of blockchain transactions[125)] and are used in day-to-day crypto operations. Within the cryptocurrency world, lightweight nodes are also referred to as simple payment verification (SPV) nodes or wallet nodes. Lightweight nodes communicate with the ledger through full nodes.[126)]

Standards

Technical Standards

- ISO/IEC 9899:2018 Programming languages -- C
- ISO/IEC 14882:2017 Programming languages -- C++
- ECMA: Standard ECMA-262 - ECMAScript® 2018 Language Specification (Javascript)
- ECMA: Standard ECMA-334 - C# Language Specification
- ECMA: Standard ECMA-335 - Common Language Infrastructure (CLI)
- ECMA: Technical Report TR/84 - Common Language Infrastructure (CLI) - Information Derived from Partition IV XML File
- ECMA: Technical Report TR/89 - Common Language Infrastructure (CLI) - Common Generics
- RFC5424 - The Syslog Protocol (SYSLOG)
- W3C: RDF 1.1 Terse RDF Triple Language (Turtle)
- W3C: OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (second Edition)
- W3C: RDF 1.1 Concepts and Abstract Syntax (RDF)
- W3C: SPARQL 1.1 Overview (SPARQL)
- W3C: Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification
- W3C: HTML5 (HTML5)
- W3C: Extensible Markup Language (XML) 1.0 (Fifth Edition)
- W3C: XML Schema Definition Language (XSD) 1.1 Part 1: Structures
- W3C: XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes
- W3C: XSL Transformations (XSLT) Version 3.0
- W3C: Document Object Model (DOM) Level 3 Core Specification
- W3C: XML Path Language (XPath) 3.1
- OMG: Data Distribution Service (DDS)
- OMG: DDS Interoperability Wire Protocol (DDSI-RTPS)
- OMG: ISO/IEC C++ 2003 Language DDS PSM (DDS-PSM-Cxx)
- OMG: Java 5 Language PSM for DDS (DDS-Java)
- OMG: OPC-UA/DDS Gateway (DDS-OPCUA)
- OMG: RPC Over DDS (DDS-RPC)
- OMG: DDS Security (DDS-SECURITY)
- OMG: Web-Enabled DDS (DDS-WEB)

- OMG: DDS Consolidated XML Syntax (DDS-XML)
- OMG: DDS For Extremely Resource Constrained Environments (DDS-XRCE)
- OMG: Extensible and Dynamic Topic Types for DDS (DDS-XTypes)

de facto Standards

- Apache: Log4j
- Apache: Log4cxx
- Apache: log4php
- Apache: log4net
- Apache: log4jscala
- Bitcoin: Developer's Guidance
- Ethereum: cpp Project
- Ethereum: Ethereumh Project
- Ethereum: Ethereumjs-lib Project
- Ethereum: Ethereum_j Project
- Ethereum: Go-ethereum Project
- Ethereum: Parity Project
- Ethereum: Pyethapp Project
- Ethereum: Ruby-ethereum Project
- EIP 20: ERC-20 Token Standard
- Ethereum: Ethereum Virtual Machine (EVM)
- Ethereum: Solidity Language Specification
- Linux Foundation: Hyperledger
- Oracle: The Java® Language Specification SE 8 Edition
- Oracle: The Java® Virtual Machine Specification JVM
- Oracle: Java logger API
- Google: Go (software language)
- InterPlanetary File System (IPFS)

Tools

- None at this time

125)

Osita Chibuike, 21 May 2018, Legobox, https://dev.to/legobox/how-to-setup-an-ethereum-node-41a7

126)

“Blockchain Nodes: An In-Depth Guide”, https://nodes.com/

2.3.3.3 Lightning Node

return to Node Taxonomy

A **lightning node** is a participant in a lightning network[127] using multi-signature (multisig)[128] on a payment channel[129]. Lightning nodes create a small community (2 to n) of blockchain nodes. The community can trade without having to use the expensive ledger transaction and has been proposed as a way to provide scalability to Bitcoin.

127)
"Blockchain Nodes: An In-Depth Guide", https://nodes.com/

128)
"How to Use Multisig to Keep Your Coins Ultra-Safe", Kai Sedgewick, 2 March 2019, https://news.bitcoin.com/how-to-use-multisig-to-keep-your-coins-ultra-safe/

129)
"Bitcoin's Lightning Network Payment Channel Explained !!", Sudhir Khatwani, 11 March 2019, https://themoneymongers.com/payment-channels/

2.3.3.4 Permanode

return to Node Taxonomy

Permanode is an IOTA-unique term for an authority node, a variant of an archival node that retains transaction data even after a snapshot is taken. It is not shown in the node taxonomy figure in Section 2.3.3 Node Taxonomy.

2.3.4 Data Taxonomy

return to Taxonomic Views

The third word in the acronym DIDO is *Data* and every node therefore, manages and controls data. The data within the node is classified as either: ledger data, ancillary data or external data.

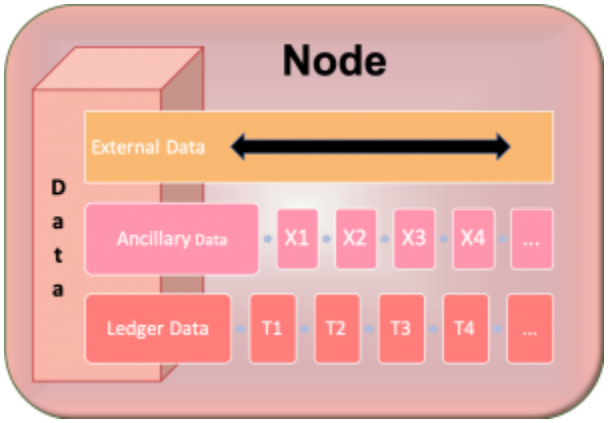


Figure 30: Types of Node Data

- Ledger Data
- Ancillary Data
- External Data

Standards

Technical Standards

- None at this time

de facto Standards

- None at this time

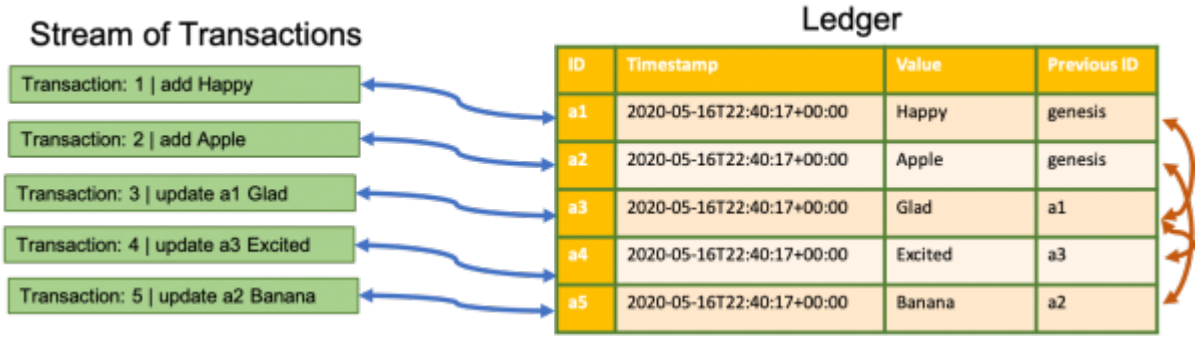
Tools

- None at this time

2.3.4.1 Ledger Data

return to Data Taxonomy

Ledger data is comprised of records containing immutable data. Even though the data captured within the ledger is immutable, this does not mean the ledger itself is immutable. The ledger can be updated by adding newer versions of existing data that then point back to the original data. Updates are made by applying transactions to the ledger in a prescribed order.



Standards

Technical Standards

- None at this time

de facto Standards

- None at this time

Tools

- None at this time

2.3.4.2 Ancillary Data

return to Data Taxonomy

Ancillary data is adjunct information used in the formulation of transactions. It differs from transaction data because it is not used to define the operation described within a transaction. Probably the best way to explain ancillary data is using examples.

Examples of Ancillary Data

Supporting Data

Data needs to be exchanged between two individuals that describe the exchange of one kind of currency with another (e.g., US dollars to EU euros or Bitcoins). The transaction can simply be something like this:

```
TRANSFER 500 USD to 1 Bitcoin in MyAccount
```

The transaction seems legitimate and can probably be verified easily enough, or can it? In order to verify the transaction, the exchange rate used also needs to be provided. It might also require the timestamp of when the exchange rate was calculated and even where the exchange rate was obtained from. This “extra” information used to verify the transaction is called ancillary data. It ultimately needs to be coded into the transaction. A classic example of why ancillary data needs to be captured is to avoid salami slicing.

Business Process Management

Data needs to be exchanged between two different business entities. Each business entity has its own business process describing the steps needed on its side in order to approve a transaction. The data required to approve the transaction is transformational in nature rather than transactional.

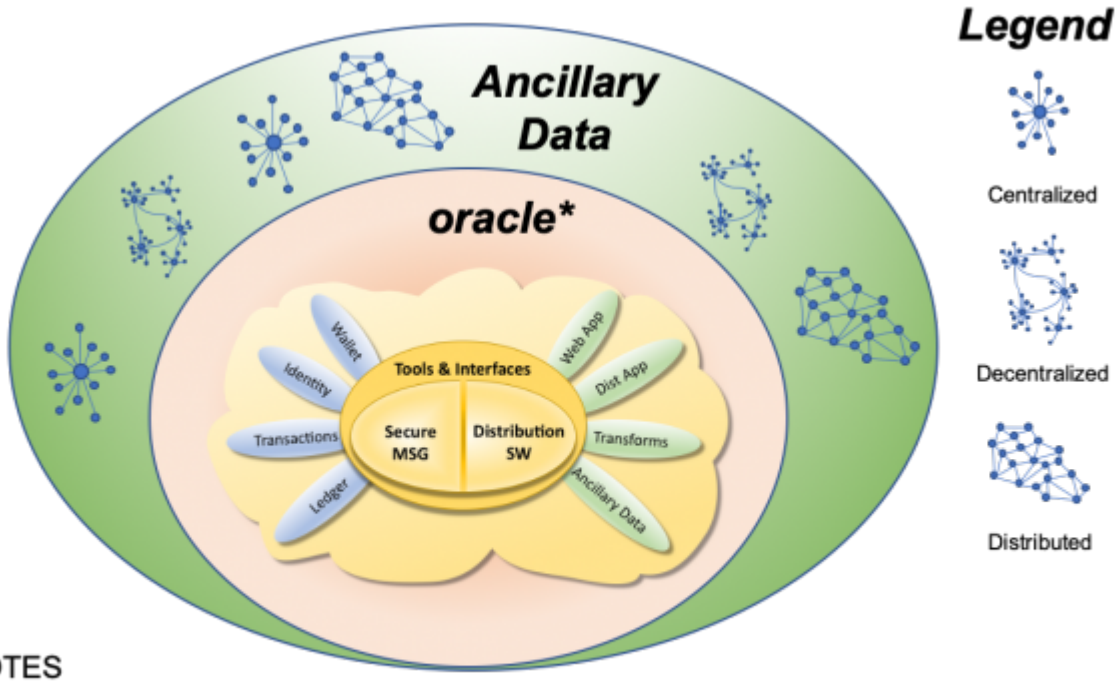
```
TRANSFER 200 AMZN Shares from MyOrg to YourOrg
```

At a transactional level, all that is needed is verification that MyOrg has the shares, and YourOrg exists in order to prevent “double spend” fraud, i.e., spending the same money more than once. However, many organization have business processes in place that require signatures from various people and that these signatures must be in a prescribed sequence (i.e., Director signs, VP signs, Comptroller signs). To verify the transaction, the signatory data needs to be provided and

confirmed that it is valid. This can occur outside the transaction and be transformational in nature. Transformational data can be undone, deleted, or modified until the business process commits to the the transaction.

DIDO Implementation of Ancillary Data

Some DIDO implementations may provide direct support of ancillary data within the DIDO or they may provide access to external ancillary data through oracles. However, if ancillary data is not also implemented within the DIDO, but implemented externally through and accessed through the use of oracles, many of the benefits of the distributed architecture are lost because access to the ancillary data can become a bottleneck.



NOTES
* An oracle, in the context of blockchains and smart contracts, is an agent that finds and verifies real-world occurrences and submits this information to a blockchain to be used by smart contracts. https://blockchainhub.net/blockchain-oracles/

Standards

Technical Standards

- None at this time

de facto Standards

- None at this time

Tools

- None at this time

2.3.4.3 External Data

return to Data Taxonomy

External data is data that is neither ledger data nor ancillary data but is required within the blockchain or smart contracts. The external data is accessed using an oracle. There are numerous formats for external data sources: data files, databases, rich site summary (RSS) feeds, RESTful interfaces, etc.

Standards

Technical Standards

- OMG: Data Distribution Service (DDS)
- RFC8259 - The JavaScript Object Notation (JSON) Data Interchange Format
- W3C: HTML5 (HTML5)
- Linux Foundation: Open Messaging
- Linux Foundation: Open Middleware Agnostic Messaging API (OpenMAMA)

de facto Standards

- None at this time

Tools

- None at this time

3 Governance

return to Reference Architecture

For practical reasons, the acronym Distributed Immutable Data Object (DIDO) represents a set of distributed computing technologies that focus on distributed data (i.e., blockchains, distributed ledgers, distributed file systems or distributed data). A major problem confronting the adoption of DIDO technologies is that it requires shifting away from corporate models of governance to an open community model of development, especially for industry or public ecosystems (e.g., financial, supply chains, public records) and domains (e.g., interest swaps, produce supply chain, cryptocurrencies, carbon credits, air pollution, traffic conditions). In the corporate model of governance, a single entity is responsible for the costs and the entire lifecycle of a product or project. Governance is accomplished through a formal chain of command, usually with a single individual responsible for the success or failure of the product or project.

However, in distributed computing, not all the resources are owned, paid for, or controlled by a single entity. In fact, the more entities involved in the distributed computing solution, the better. These differences in governance models make the adoption of distributed computing difficult by corporate entities since they have to rely on a larger, more inclusive community, which may include competitors, to measure the success of the solution. Ultimately, the success of the project or product comes down to controlling and minimizing risks. Being part of a larger, more diverse community increases some kinds of risks but may decrease others.

For example, the risks associated with specifying, architecting, designing, implementing, testing, maintaining, and sunsetting code are shared over the entire community, thus practically lowering the risks to any individual. The risks associated with setting requirements, priorities, and so on may be increased since these are now set externally to any single entity.

Currently, with the rise of open source solutions versus proprietary solutions, many efforts have moved from the centralized governing corporate model towards the decentralized, distributed community governing model with a great deal of success and broad adoption. Some examples of open source solutions are Operating System (OS), DataBase Management System (DBMS), application servers, web servers, file repositories, bug tracking tools, virtualization tools, and]]dido:public:ra:xapend.glossary:d:dlt]]. Most corporations would now rarely choose to build any of these products on their own, but have readily adapted to the adoption of these open source community products and solutions.

This section defines and recommends community of interest (CoI) governance structures needed for successful, robust, inclusive, and broadly adopted solutions. Some CoIs require a formal legal entity recognized by a government authority such as a state government. Other CoIs can operate within the confines of another legal CoI. For example, World Wide Web Consortium (W3C) and Object Management Group (OMG) are legal entities. Special interest groups (SIGs) or working groups can operate within the W3C or OMG.

Manage an Open Source Program

There have been many books written on this subject:[130)]

- RFC2026 - The Internet Standards Process
- TODO: Using open source code
- TODO: Participating in open source communities
- TODO: Recruiting open source developers
- TODO: Starting an open source project
- TODO: Improve your open source development impact
- TODO: Shutting down an open source project
- TODO: Building leadership in an open source community
- TODO: Setting an Open Source Strategy

130)

TODO Open Source Reading List, https://todogroup.org/guides/open-source-reading-list/

3.1 DIDO Communities

Return to the Governance page

As introduced in Section 2.1, a DIDO community is a community of interest (CoI) responsible for the architecture, design, implementation, maintenance, and eventual sunset (yes, all good things come to an end) of DIDO networks and its DIDO nodes. DIDO communities should have a well-documented, repeatable, traceable, transparent, and unbiased process that is managed and run by a governing board (or hierarchy of governing boards). Often the DIDO community is divided into two organizations: one responsible for the software, usually as an open source software (OSS) project, and the other responsible for managing the fungible data (e.g., currency or commodity) managed by the software. Most DIDO communities choose or select DIDO software designed, implemented, and maintained by another party. For example, Ethereum, Hyperledger, and IOTA all provide their software for others to use as OSS.

Some DIDO communities may support multiple kinds of immutable data objects within a single DIDO network, or they can support multiple networks that each have different kinds of immutable data object types. For example, one DIDO community's OSS might support cryptocurrencies on one DIDO network and support immutable data objects representing supply chain commodities on another.

Unless a DIDO network's focus is very narrow, there is generally a need to have an exchange that allows different immutable data objects to be exchanged (e.g., US dollars to Bitcoins). The exchange might support the exchange of immutable data objects that are all contained within one DIDO network, or it might support the exchange of information managed by one DIDO community but on different DIDO networks. In this case, the exchange might manage converting customer loyalty reward points to a currency (i.e., euros or cryptocurrency) or supply chain commodities to cryptocurrencies and back. Other exchanges might exchange immutable data objects managed by one DIDO community and DIDO network with a cryptocurrency managed by another DIDO community; for example, the exchange of Bitcoins to Ethereum or Iotas.

3.1.1 Stakeholder Communities

Return to DIDO Communities Page

Given that DIDOs are centered around stakeholders, a stakeholder community is key to creating a thriving community of stakeholders. Unless the target of a DIDO is a private, permissioned network, there will be stakeholders that exist outside of any corporation and/or a corporation's customers or clients. In many cases, the stakeholder community will also include competitors or other participants (i.e., supply chain players) that want to integrate services around a common concept or idea. This common concept or idea is generally a distributed object.

In other words, the stakeholder community is generally defined as people, groups, organizations, or businesses that have interest or concern in the distributed object. Stakeholders affect or are affected by the community's actions, objectives and policies.

A stakeholder community can be informal or formal. Informal communities are best described as a confederation of the willing and generally have one or more core participants who make decisions for the group. When the stakeholder community is large, broad and with sometimes competing stakeholders, these communities should be formal, well organized organizations (usually non-profit).

Note: Refer to Section 2.1 Stakeholder Views for how the various stakeholder communities are organized, as well as the definitions and descriptions of ecosphere, ecosystem, and domain. Also, it should be understood that each of these terms refer to a kind of community of interest (CoI). Thus, saying “ecosphere” should be read as “ecosphere CoI”. Likewise for “domain” and “ecosystem”.

Steps for Establishing an Ecosphere

Steps for establishing and building a formal ecosphere CoI[131)]

1. Define a Mission Statement
2. Select a Leadership Team as Governing Board
3. Secure Funding
4. Create Legal Documents:
   a. Charter
   b. Bylaws
   c. Policy and Procedures (P&P)
5. Legally Incorporate
6. If Non-Profit, officially register as Non-Profit (In US, 501 ©(3))
7. Identify Partners (Founding Members)
8. Establish Milestones, Deliverables, and Schedules
9. Create Community Guides consistent with Charter, Bylaws, and Policies and Procedures
10. Recruit more members
11. Establish liaisons with other CoIs

131)

How To Start A Nonprofit Organization, Snowball Fundraising, Accessed 21 May 2020, https://snowballfundraising.com/starting-a-nonprofit-checklist/

3.1.2 Software Communities

Return to DIDO Communities Page

The cornerstone of a DIDO community is the software, which acts as the engine of the distributed network of peers. Software is responsible for maintaining the ledgers on each node: securely and reliably sending transactions to each of the nodes and mitigating any conflicts that may arise while processing these transactions (e.g., the double spend problem). As a general rule, the software is open source software (OSS), governed by OSS rules for its maintenance and the development of new features.

In some communities, the software extends beyond the core software required to maintain the node within a network of nodes. For example, the Ethereum Foundation includes both the software to maintain the network of nodes and smart contracts to monitor and oversee transactions. However, smart contracts themselves may or may not be built, maintained, and supported by the Ethereum Foundation's software team. They may be created, maintained, and supported by external third parties.

3.2 Legal Documents

return to Governance

Given the intentional distributed, decentralized nature of DIDOs, along with the potential for them to run on hardware and/or use software not owned or developed by a single entity, it is important to employ a community of interest (CoI) approach to develop, manage, and govern DIDO solutions. Such an approach allows for the establishment and specification of requirements for a formal organization that balance the various goals of its participants in pursuit of a single shared mission and, at the same time, protects the joint intellectual property (IP) resulting from their collaboration.

- **Note:** Legal Documents serve as the Regulatory Aspect within the Governing Model. See Governing Roles and the Governing Model for more information.

When an organization is a joint venture between different legal entities, it must operate in accordance with legal documents that serve to manage the expectations of, govern the interactions between, and elaborate the collective and individual deliverables expected from the legal entities participating in that organization. In the context of DIDO, *ecosphere* CoI is the term used to refer to such a joint venture. Table 3 lists the legal documents required to formally launch an ecosphere CoI. These documents support the process described in stakeholder communities. Depending on the country of incorporation, all three may be required for an ecosphere COI to be formally incorporated. In general terms (more details provided in Sections 3.2.1, 3.2.2, and 3.2.3), the charter defines how the ecosphere CoI interacts with the outside world; the bylaws define how it operates internally (e.g., fundamental governing rules, internal organization); and the policies & procedures (P&P), a document required for incorporation in the United States, establishes a set of principles to achieve the goals and vision of its charter and the specific methods employed to express these policies in action.

As already discussed in stakeholder views, other types of CoIs or special interest goups (SIGs) can be formed under the umbrella of a parent ecosphere: *ecosystems* and *domains*.[132)] As shown in Table 3, they must each be chartered according to the governing rules established in the bylaws and abide by the P&P of their parent ecosphere CoI. Ecosystem P&Ps may include specific extensions to the parent ecosphere's CoI P&P. In like manner, domain P&Ps may include specific extensions to the their parent ecosystem CoI P&P. All extensions to the parent P&P must be **additive**; they may not replace or override anything in the P&P of their parent ecosphere CoI.

Table 3: Documents required to Create and Govern a DIDO CoI

DIDO CoI	Charter	Bylaws	Policies and Procedures
Ecosphere	Yes(§)	Yes(†)	Yes(‡)
Ecosystem	Subcharter of Ecosphere	covered by Ecosphere	covered by Ecosphere + extensions
Domain	Subcharter of Ecosystem	covered by Ecosphere	covered by Ecosphere + extensions from Ecosystem _ local

- *((§) Initially, a legal statement created by the founders of the organization that lays out the goals, missions and officers for the organization*
- *(†) a legal document reviewed by lawyers from all the participating parties*
- *(‡) Some Policies and Procedures may be mandated by law (i.e., discrimination, ADA, Safety, etc.*

while others may be added by local governing boards and should be drafted/reviewed by lawyers of all participating parties

NOTE: There are no standards for these documents. The initial versions of these documents, particularly the Bylaws, should be drafted by attorneys. In the case of the P&P, the initial version should be drafted with input from attorneys to address legally mandated items.

- 3.2.1 Charter
- 3.2.2 Bylaws
- 3.2.3 Policies and Procedures (P&P)

132)

Described in more detail, along with illustrations, in Ecosystem View and Domain View.

3.2.1 Charter

return to Legal Documents

A charter is a legal document providing basic information about a community of interest (CoI): its location, purpose[133)], profit status (usually non-profit), governing board (e.g., board of directors) composition, and stakeholder (or ownership) structure. Sometimes the charter is referred to as the articles of incorporation or a certificate of incorporation. In the case of a for-profit corporation, the articles of incorporation must include the number, classes, and par values of authorized shares. In the United States, most states require the name and address of the company's registered agent as well. Source: Charters: Creating the Organization

The CoI charter is important because when it is filed and approved by a legal entity, such as a secretary of state, it "gives birth" to a new CoI as a formal corporation. A CoI charter is not the same as the CoI bylaws, which build off the charter and add things like fundamental governing rules, board meeting schedules, conditions of membership, etc.

Note:

1. Charter, as described in this section, only applies to ecosphere CoIs that file for incorporation.
2. Ecosystem CoIs and domain CoIs are also created using a charter, but these charters are not legal or legally binding documents. Instead they are "subcharters" of, and must be approved by, their parent ecosphere using a process defined in the parent ecosphere's P&P.
3. The Charter serves as part of the Regulatory Aspect within the Governing Model. See Governing Roles and the Governing Model for more information.

Essential Elements of a Charter

- **Name**
- **Address**
- **Statement of Purpose**
- **Profit Status (Usually non-Profit)**
- **Registered Agent**
- **Agent's Address**
- **Number of Shares Authorized**
- **The Classes and par value of the shares**
- **Roles** and **Responsibilities**
- **Directors**

Standards

- The charter ID made specifically for the CoI (i.e., ecosphere).

Tools

- Smartsheet Project Charter Template, Smartsheet, 21 May 2020, https://www.smartsheet.com/blog/project-charter-templates-and-guidelines-every-business-need
- UpBoard, Project Charter Online Tools & Templates – Best Practices, accessed 21 May 2020, https://upboard.io/project-charter-online-software-tools-templates/

References

- The Essential Guide to Creating an Effective Team Charter
- Investing Answers, Corporate Charter, Accessed 21 March 2020, https://investinganswers.com/dictionary/c/corporate-charter
- Investopedia, Understanding a Corporate Charter, Accessed 30 May 2020 https://www.investopedia.com/terms/c/corporatecharter.asp

133)

in the context of a DIDO CoI, think of this as another word for “Goal”, “Mission”, or “Vision”

3.2.2 Bylaws

return to Legal Documents

The community of interest (CoI) bylaws (or articles of organization) are the primary legal documents providing governance for the group. The bylaws are originally created during the initial steps of establishing the CoI (i.e., ecosphere). [134)]

Bylaws are supplemental to the rules already defined by the granting government entity's (e.g., states, counties, or cities in the US) code and rules for how the CoI must run. Bylaws specify the rules and regulations governing actions and decisions made by the board. The bylaws should help prevent or resolve conflicts and disagreements, and protect the CoI from potential problems by clearly outlining rules concerning authority levels, rights, and expectations.

> **Note**:
>
> 1. If the board of directors fails to follow the adopted CoI bylaws, it can be held liable for breaching their duty to the CoI.
> 2. The Bylaws serves as part of the Regulatory Aspect within the Governing Model. See Governing Roles and the Governing Model for more information.

Common Provisions in Bylaws

1. Name and Purpose
2. Parliamentary Authority for Election, roles, and terms of officers and board members
3. Membership Issues (categories, responsibilities)
4. Meeting Guidelines (Frequency and Quorum)
5. Board Structure and Size
6. Compensation and indemnification of board members
7. Role of Chief Executive
8. Conflict of Interest Policy
9. Amendment of Bylaws Policy
10. Dissolution of the organization

Standards

- The bylaws are made specifically for the CoI (i.e., ecosphere).

de facto Standards

- Robert, Henry M.; et al. (2011). Robert's Rules of Order Newly Revised (11th ed.). Philadelphia, PA:

Da Capo Press. ISBN 978-0-306-82021-2 (hardcover).
- Robert III, Henry M. (2011). “The Official Robert's Rules of Order Web Site (Home)”. The Official Robert's Rules of Order Web Site. The Robert's Rules Association.
- The Official Robert's Rules Of Order Web Site (robertsrules.com) Site maintained by the Robert's Rules Association

Tools

- Section 7. Writing Bylaws, Community Tool Box, Accessed 21 May 2020, https://ctb.ku.edu/en/table-of-contents/structure/organizational-structure/write-bylaws/main
- Form of Bylaws - California Nonprofit Public Benefit Corporation, Public Coucil Org, Accessed 21 May 2020, http://www.publiccounsel.org/tools/publications/files/Annotated_Bylaws.pdf
- Sample Bylaws, U.S. Chamber of Commerce, Accessed 21 May 2020, https://www.uschamber.com/your-chamber-commerce-guide-starting-and-growing-chamber-commerce/sample-bylaws

References

- Bylaws for an Unincorporated Association, Drew Nelson, September 26, 2017, Accessed 21 May 2020, https://bizfluent.com/list-6981569-bylaws-unincorporated-association.html
- What is the difference between Charter and Bylaws?, William Adkins, bizfluet, 26 September 2017, Accessed 21 May 2020, https://bizfluent.com/facts-5894520-difference-between-charter-bylaws-.html
- The Difference Between Bylaws and Policy, Victoria Duff, 26 September 2017, Accessed 21 May 2020, https://bizfluent.com/facts-5921586-difference-between-bylaws-policy.html
- Bylaws, Policies and Procedures: What's the Difference?, Karen Blewett, presented to Board Leadership Southeast Alberta, 4 March 2017, Accessed 30 May 2020 https://boardleadershipsouth.com/files/march_4_2017/Bylaws,%20Policy%20and%20Procedures%202017.pdf

[134)]

Nonprofit Bylaws Made Easy: Tips and Best Practices, donorbox.org, Acccessed 21 May 2020, https://donorbox.org/nonprofit-blog/nonprofit-bylaws-made-easy/

3.2.3 Policies and Procedures (P&P)

return to Legal Documents

Policies and Procedures (P&P)

One of the legal documents required for the formal incorporation of an ecosphere community of interest (CoI) is the Policies and Procedures (P&P). The P&P establishes a set of principles and policies that serve these principles, in order to achieve the CoI's long-term goals (i.e, the purpose as stated in its charter). The P&P defines specific methods/procedures employed to express these policies in action: detailed rules and guidelines to control the activities of the organization. In the US, some policies are concerned with human resources (HR) issues (e.g., Health Insurance Portability and Accountability Act (HIPAA), or data protection[135] and security), a direct response to the General Data Protection Regulation (GDPR), Data Protection Act 2018, or the California Consumer Privacy Act (CCPA).

The intent of the P&P is to influence and help formulate all the major decisions and actions of the organization. All activities within the organization are to take place within the boundaries set by the P&P.[136] Procedures are specific methods outlining the steps required to fulfill the policies on a day-to-day basis within the organization. Together, policies and procedures help the governing body of an organization meet its mission and goals.[137]

One of the larger policies that is often required by the governing body granting incorporation are the human resources policies, especially those concerning health and human safety.

- **Note** The Policies and Procedures (P&P) serve as part of the Regulatory Aspect within the Governing Model. See Governing Roles and the Governing Model for more information.

Standards

- The charter ID made specifically for the CoI (i.e., ecosphere).

Laws

- Occupational Safety and Health Act of 1970, Public Law 91-596, 84 STAT. 1590, 91st Congress, S.2193, December 29, 1970, as amended through January 1, 2004. https://www.osha.gov/laws-regs/oshact/completeoshact
- Regulation (EU) 2016/679 OF THE European Parliament and of the Council of 27 April 2016 https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

Tools

- Rocket Lawyer, Health and Safety Policy Document Maker, Accessed 21 may 2020,

https://www.rocketlawyer.com/gb/en/documents/health-and-safety-policy
- Rocket Lawyer, Data Protection and Data Security Document Maker, Accessed 21 May 2020, https://www.rocketlawyer.com/gb/en/documents/data-protection-and-data-security-policy

References

- Health and Safety, Rocket Lawyer, accessed 21 May 2020. https://www.rocketlawyer.com/gb/en/quick-guides/health-and-safety
- Overview of the Data protection and data security policy, Rocket Lawyer, accessed 21 May 2020. https://www.rocketlawyer.com/gb/en/documents/data-protection-and-data-security-policy

135)

NOTE: From https://iclg.com/practice-areas/data-protection-laws-and-regulations/usa,

- *There is no single principal **Data Protection** legislation in the United States. Rather, a jumble of hundreds of laws enacted on both the federal and state levels serve to protect the personal data of U.S. residents. At the federal level, the **Federal Trade Commission Act (15 U.S. Code § 41 et seq.)** broadly empowers the U.S. Federal Trade Commission (FTC) to bring enforcement actions to protect consumers against unfair or deceptive practices and to enforce federal privacy and data protection regulations. The FTC has taken the position that "deceptive practices" include a company's failure to comply with its published privacy promises and its failure to provide adequate security of personal information, in addition to its use of deceptive advertising or marketing methods. As described more fully below, other federal statutes primarily address specific sectors, such as financial services or health care.In parallel to the federal regime, state-level statutes protect a wide range of privacy rights of individual residents. The protections afforded by state statutes often differ considerably from one state to another, and cover areas as diverse as protecting library records to keeping homeowners free from drone surveillance.*
- *Although there is no general federal legislation impacting data protection, there are a number of federal data protection laws that are sector-specific, or focus on particular types of data. By way of example,*
  o Driver's Privacy Protection Act of 1994 (DPPA)
  o Children's Online Privacy Protection Act (COPPA)
  o Video Privacy Protection Act (VPPA)
  o Cable Subscriber Protection

136)

Difference Between Policies and Procedures, Key Differences, accessed 21 May 2020,https://keydifferences.com/difference-between-policies-and-procedures.html

137)

Policies and Procedures, Business Dictionary, http://www.businessdictionary.com/definition/policies-and-procedures.html

3.3 Guides

return to Governance

In addition to the legal documents governing the activities of a community of interest (CoI), many CoIs create a simple, easily understood guide to help its members maneuver through and understand all the governing statements contained in the statutes, charter, bylaws, parliamentary authority, special rules, custom practices, and standing rules associated with that particular CoI. For example, the OMG publishes a "Hitchhiker's Guide to the OMG Process". Even though it is not a “normative” document (has no legal standing whatsoever, cannot be cited as a defense for not following process), it is much easier to read and understand than the OMG Policies and Procedures. It provides additional details and insights that, even though they do not belong in a P&P, are essential to know. Technical standards bodies such as the IETF publish a similar guide, which is even more informal and flippant. In any case, the goal of such a guide is to make it easier for the members of a CoI to follow their CoI's process rules.

4 Requirements

return to Reference Architecture

The term requirement first appeared in software engineering in the 1960s.[138)]

The Business Analysis Body of Knowledge® version 2 from IIBA (BABOK),[139)] defines a requirement as having one or more of the following characteristics:

1. *A condition or capability needed by a stakeholder to solve a problem or achieve an objective.*
2. *A condition or capability that must be met or possessed by a solution or solution component to satisfy a contract, standard, specification, or other formally imposed documents.*
3. *A documented representation of a condition or capability as in (1) or (2).*[140)]

This definition is based on IEEE Standardized vocabulary.[141)]

138) Boehm, Barry (2006). A view of 20th and 21st century software engineering. ICSE '06 Proceedings of the 28th international conference on Software engineering. University of Southern California, University Park Campus, Los Angeles, CA: Association for Computing Machinery, ACM New York, NY, USA. pp. 12–29. ISBN 1-59593-375-1. Retrieved January 2, 2013. http://dl.acm.org/citation.cfm?id=1134288

139) 1.3 Key Concepts - IIBA | International Institute of Business Analysis“. www.iiba.org. Retrieved 2016-09-25. http://www.iiba.org/babok-guide/babok-guide-v2/babok-guide-online/chapter-one-introduction/1-3-key-concepts.aspx

140) Wikipedia Requirement

141) IEEE/ISO/IEC 24765-2010 - ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary , 2010-02-02, published 15 December 2015, IEEE, [https://standards.ieee.org/standard/24765-2010.html]]

4.1 About Requirements

Return to Requirements

Although Requirements at first glance appear to be simple, they quickly become very complex as the number of requirements increase, and consequently they need to be managed and governed. Add this complexity to the complexity associated with any distribute system, and it can quickly seem daunting and overwhelming. Like any other large problem that confronts us, it is too hard to handle it as one big problem. In engineering, a common approach is to not address the one big problem but to refine the problem into a series of little problems which can be solved. Consequently, that is why the bigger solution requires governance.

The following chapters help refine this big problem into smaller ones.

4.1.1 Governance Requirements Model

Return to About Requirements

In addition to this definition, requirements can be applied to various aspects of a project from a governance perspective. There are three different aspects of governance: Regulation, Execution and Compliance (See Appendix J: Governance Model. [142)]

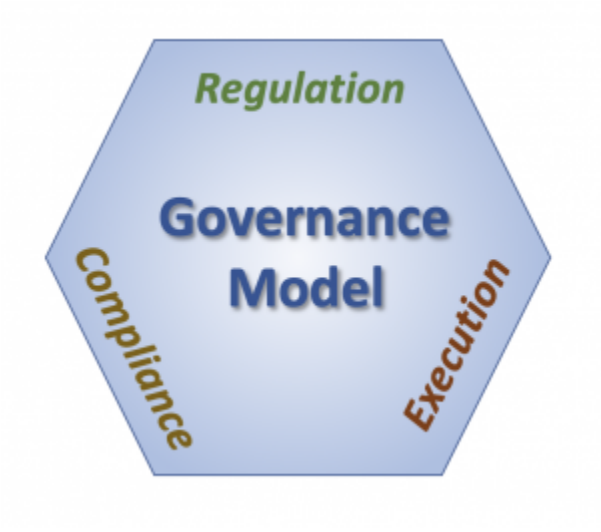


Figure 31: General Governance Model

- **Regulation** covers the specification of the requirements of the product or service. This can be either formal government regulation such as U.S. ADA compliance[143)144)], or it can be project specific such as the contract or the technical specification of a contract or can be as detailed as the steps defined for a particular test plan.
- **Execution** covers the lifecycle of a product or service being governed (i.e., design, building, maintenance, etc). This covers the functional and non-functional requirements of the product or service and can be in terms of static or dynamic compliance points
- **Compliance** covers the oversight of the product or service being governed. This covers the not only the product or service itself but also the process of Regulation and Execution. For example, is the contract or technical specification well written and maintained throughout the lifecycle of the product or service.

Understanding requirements is an important part of specifying, building, using and maintaining any product. And when the System is Distributed on multiple machines, supporting multiple stakeholders the need for the proper governance is essential. The requirements must cover all three aspects of governance. For example, there can be a requirement that is expressed in the regulatory aspect which requires reporting of information. The Compliance aspect will have a corresponding requirement that validate and verifies the reporting of the information. These are then also reflected in the Execution Aspect that has to have processes and systems designed to collect and report the proper data.

142)

Stavros, Robert W. and Albrant, Jeremiah; Engineering Governance, SPAWAR, October 9, 2007,

143)

ADA Website Accessibility Under Title II of the Americans with Disability Act (ADA) - ADA Best Practices

Tool Kit for State and Local Governments, Website Accessibility Under Title II of the ADA, Chapter 5, https://www.ada.gov/pcatoolkit/chap5toolkit.htm
144)

Accessibility of State and Local Government Websites to People with Disabilities, U.S. Department of Justice, Civil Rights Division, Disability Rights Section https://www.ada.gov/websites2.htm

4.1.2 Cognitive Requirements Model

Return to About Requirements

In addition to the definition of a requirement and the specification of the requirements at each of the Governance Aspects, there is a need to further refine the requirements according to the level of specificity of the requirement. There are five different layers to specificity which are based upon the layers of the Cognitive Model. (See Appendix I: Cognitive Model).

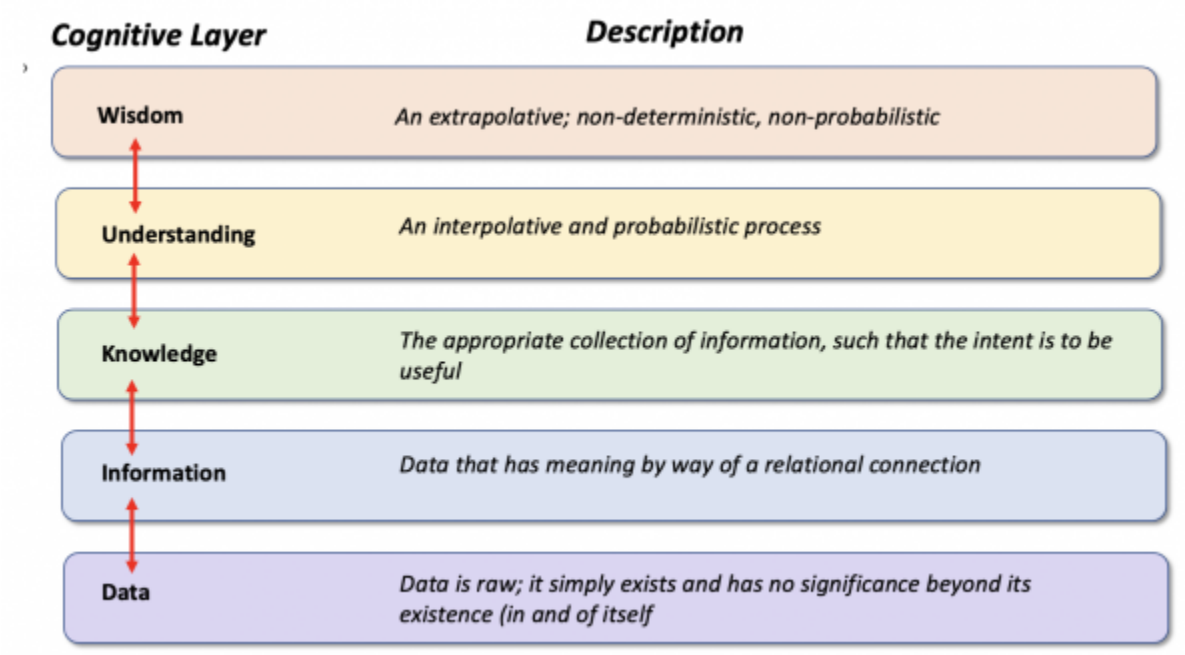


Figure 32: Cognitive Model

- **Wisdom** - An extrapolative and nondeterministic, non-probabilistic concept which is built upon **Understanding** concepts.
- **Understanding** - An interpolative and probabilistic concept that reflects **Wisdom** concepts
- **Knowledge** - A concept relating appropriate collection of **Information** concepts together, such that it's intent is useful
- **Information** - A concept aggregating data concepts together, in other words, **Data** that has been given meaning by way of relational connection
- **Data** - Data that is a raw concept; it simply exists and has no significance beyond its existence (in and of itself). It is the basis upon which **Information** concepts are built

The flow between the cognitive layers is bidirectional and can have many-to-many relationships. For example, any particular **Wisdom** concept can be associated with any number of **Understanding** concepts. The inverse is also true; **Understanding** concepts can also be used to help support any number of **Wisdom** Concepts.

4.1.3 Governing Roles - Combined Requirements Model

Return to About Requirements

In order to be effective, it is best to combine the Governance and the cognitive models together. The results look something like Figure 33. Each cell in the overlaid models represents a single Role or area of consistent governance providing some context for the requirements. For example, at the **Data** x **Regulation** cell, there is specific data that is required to be collected according to the regulations. There is a regulation that requires a bank to collect taxpayer IDs for each account. During the **Execution** aspect (i.e., the Bank's Policies and Procedures(P&P)) the taxpayer ID is collected, the specific bank actually collects and records the taxpayer ID. During the **Compliance** aspect, there is a requirement to verify that each Bank actually has a taxpayer id with each account. This consistency in governance can be repeated for each row (i.e., Wisdom, Understanding, Knowledge, Information and Data) and for each column (i.e., Regulation, Execution and Compliance).

If any of the Roles (i.e., cells have no requirements, the governance is incongruent and can lead to a potential flaw or hole in the governance which is vulnerable to exploitation.

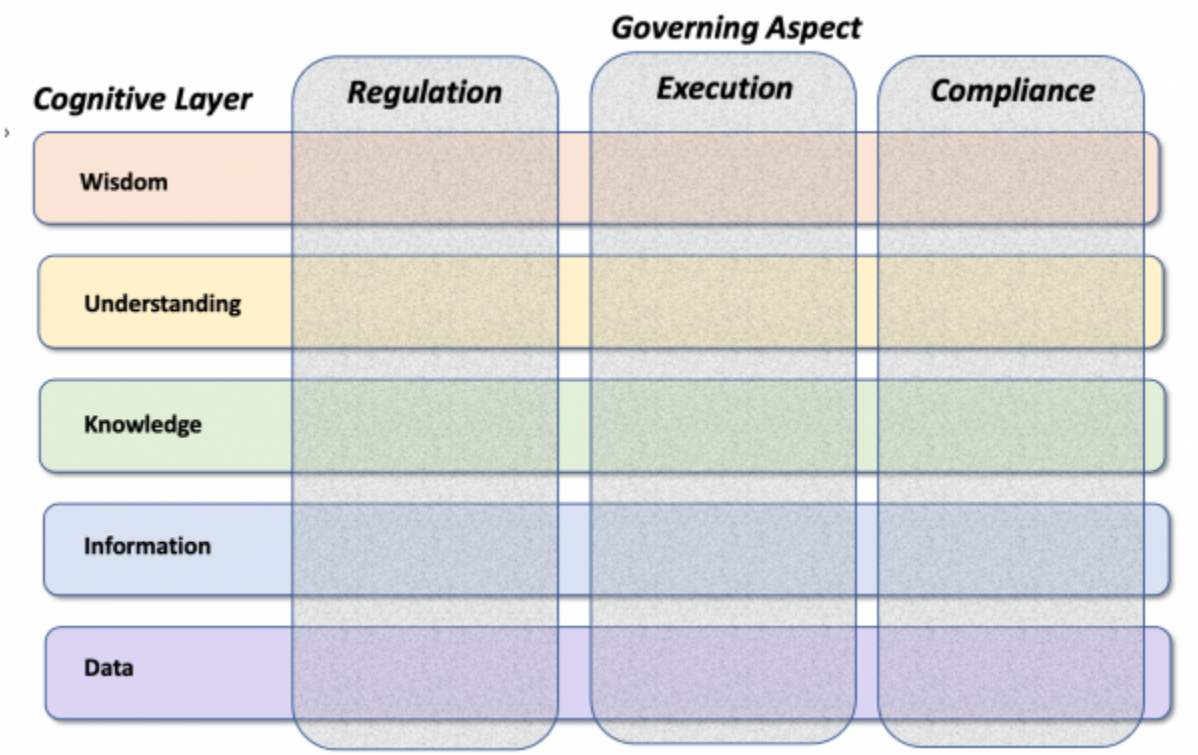


Figure 33: Combined Governing and Cognitive Models - Each Cell represents a Governing Role

4.1.4 Example of a Using the Combined Requirements Model

Return to About Requirements

The following is a Normative[145)] example of using the Combined Requirements Model to review requirements for any Federal Deposit Insurance Corporation (FDIC) regulation. The actual data may be invalid or based on some convenient assumptions but is included as an example of how the combined model can be used.]

In the example, the regulation starts with **Wisdom**. Within Regulation, the **Wisdom** is the U.S. Code of Federal Regulations. Although we can make light of Federal Regulations as being **Wisdom** try and think about it this way: *"when working within the confines of a government, it is not wise to ignore or marginalize regulation"*. When working within the banking industry, **Understanding** is knowing that there are U.S. Federal Regulations that pertain to banking in general. **Knowledge** is to know the specific laws that pertain to the Area-of-Interest (AoI), in this case banking and Federal Insurance covering banking(i.e, FDIC). There is knowledge that is provided within the FDIC legislation the that specifies which laws must be followed by an institution wanting to be insured (i.e., *Section 18 of the FDIC regulations*). Finally, at the **Data** layer, there are specific rules about what needs to be reported and how it is to be reported.

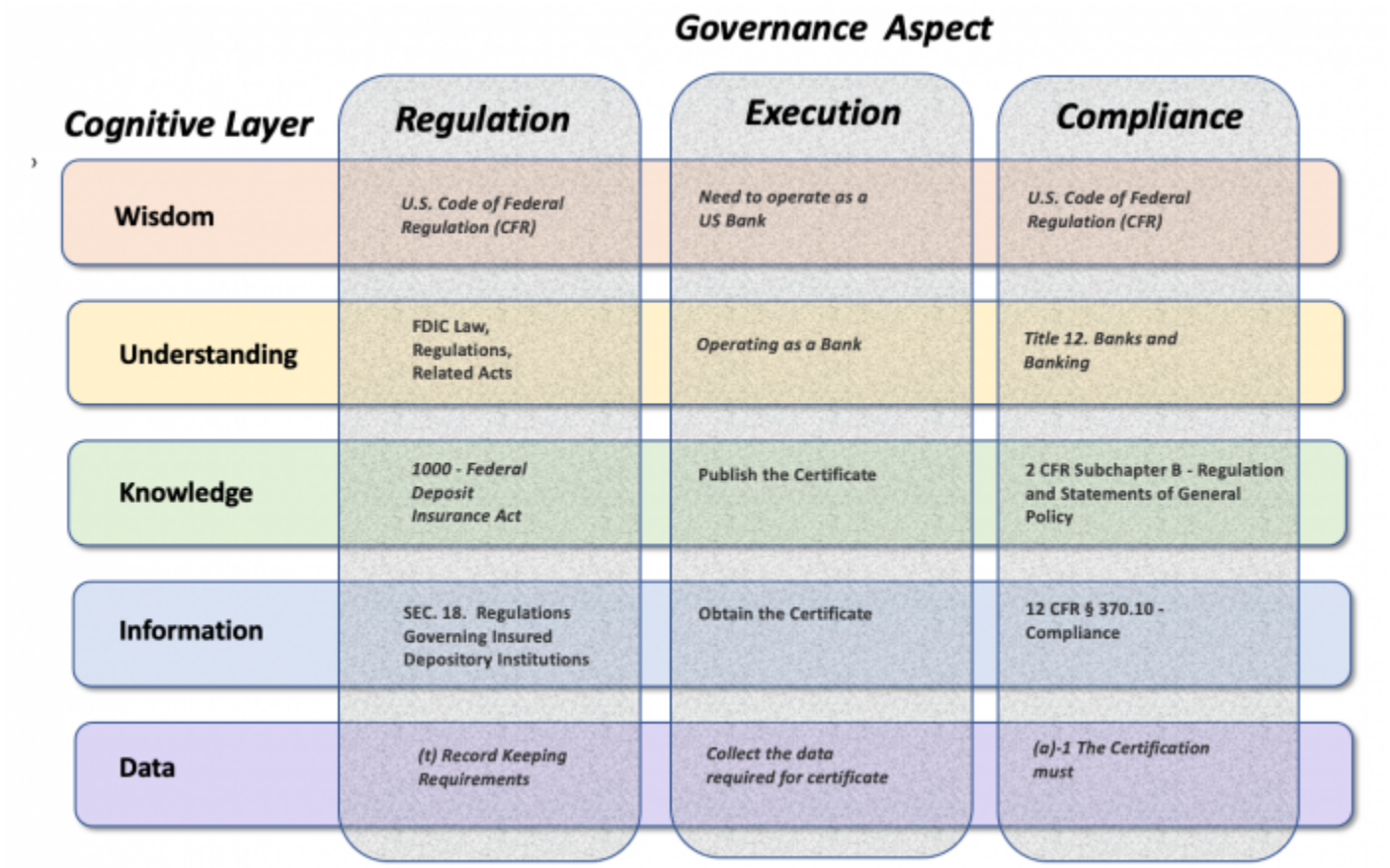


Figure 34: Normative Example of the Combined Governing and Cognitive Models

Figure 34 roughly lays out a “straw man” of what the overall combined governance could be like. Within each governing Aspect and within each Cognitive layer there are specific requirements that covers that Role (i.e., cell).

145)

Normative - are statements based on opinions about what should happen. They are subjective rather than objective because they involve value judgment about what is right and what is wrong.

4.1.5 The Current State of DIDO Requirements

Return to About Requirements

The current state of DIDO governance is basically “Execution Centric”. This basically means that the governance is being developed by those that are actually creating the DIDO platforms (i.e., products). Although this approach may seem efficient as a way to overcome bureaucratic obstacles, it ultimately leads to failures in the end results. In Figure 2, the **Execution** is represented as a pyramid. This represents the bottom up approach used to developed most DIDO platforms today and is a reflection of the Agile Model Agile methodology popular on non-Mission Critical Systems.

As cryptocurrencies attempt to become financial instruments and represent actual national currencies, they are essentially positioning themselves as Mission Critical for the nations, their economies and their citizens. Therefore, to be successful, they must be governed properly which implies that each Role (i.e., cell) within the combined Governing and Cognitive models (See Figure 2) need to have an Actor (i.e. be completed).

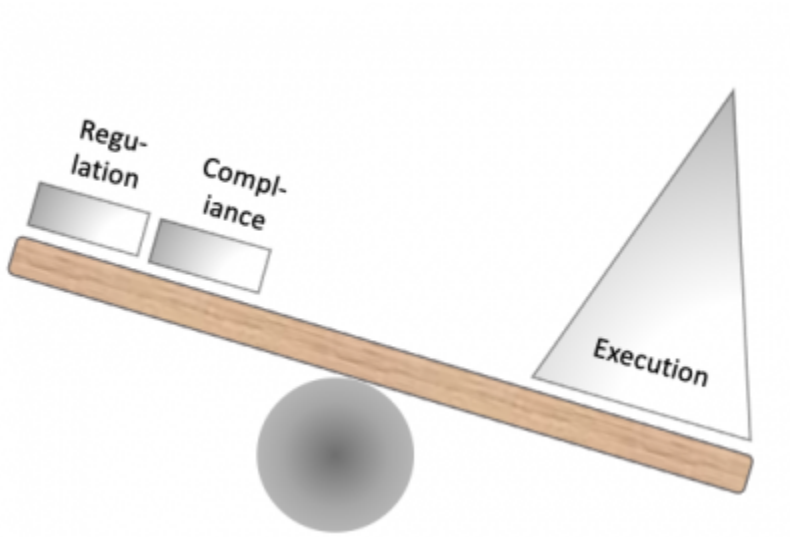


Figure 2: The current state of DIDO Governance

This is not unusual when a new disruptive product or service comes along, it is to be expected. However, in order to mature to the next level and gain wider acceptance, the product or service needs to mature its governance. The Combined Governing and Cognitive Model described and depicted in Section 4.1.3 offers a good framework for a checklist of what is done and what needs to be done.

The relationship between the Roles (i.e., cells) in a many-to-many relationship that is not only bidirectional in up and down the cognitive layers but also between the governance aspects.

4.1.6 One Degree of Freedom Rule

Return to About Requirements

Probably one of the most important rules is to not skip roles. Each and every role is important and all too often, in the name of expedience, attempts are made to short circuit the model and skip roles. For example, trying to specify in “regulatory wisdom” how to inspect products built during execution. This does not however mean that the roles are completely isolated. Within each of the Governance Aspects, the Cognitive Model’s hierarchy still applies. This can be summarized as the “one degree of freedom” rule.

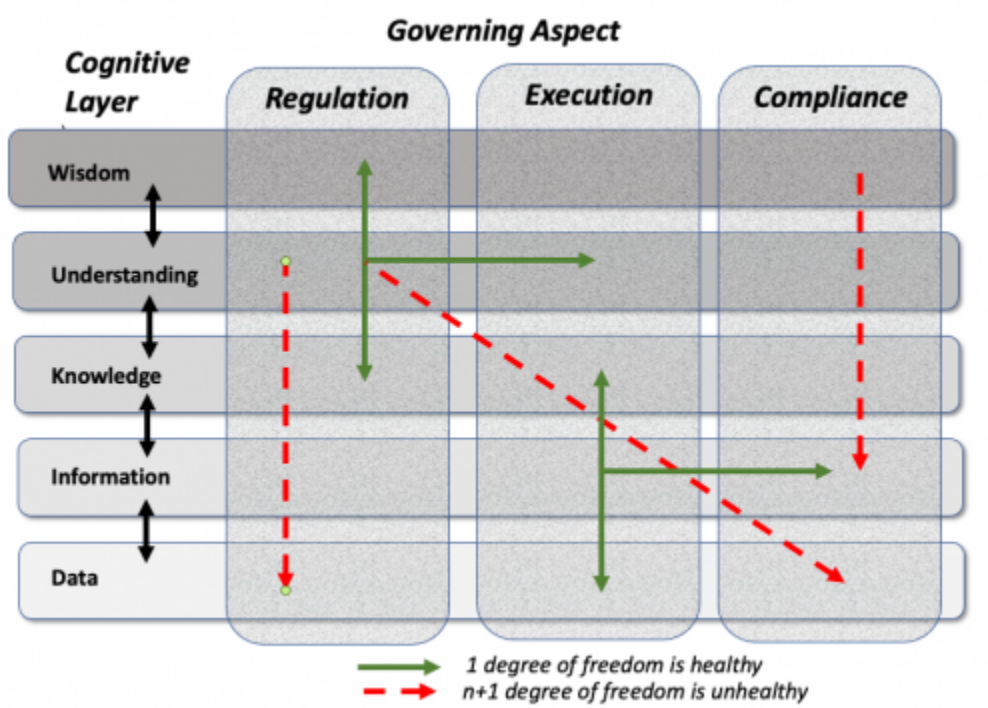


Figure 36: The One Degree of Freedom Rule

4.1.7 Specifying Requirements

Return to About Requirements

Requirements are captured in many ways. In the government realm, this is usually done through codification into laws, regulations and contracts including Performance and Conformance Specification. In the corporate realm, it often comes in the form of Charters, Bylaws and Policies and Procedures (P&P) and contracts with other entities (see above).

Regardless of where the requirements are captured or by what organizations, they can in general be considered governing statements. The following are some guidance on how to write healthy governing statements and consequently also requirements.

A Governance Statement based on an Engineering Governance Model developed at US Navy SPAWAR[146)] is defined as atomic, succinct, absolute and definitive in nature. It contains specific instructions which can be validated through observation, measurement or testing.

- **Atomic** - A Governance Statement only addresses a single topic. Indicators of non-atomic guidance are use of highly complex sentences, multiple sentences or conjunctions such as and, or, etc.
- **Succinct** - A Governance Statement are short and to the point. The definition of terms or caveats that explain when a statement is applicable are not acceptable as part of the Governance Statement. Indicators of non-succinct statements are the use of words or expressions such as: consider, when possible, if, etc.
- **Absolute** - A Governance Statement is evaluatable with one or more non-subjective questions. Indicators of non-absolute statements are those which are subject to the interpretation of the evaluator. For example, “All menus must be user-friendly“. No one produces menu's that they feel are user hostile.
- **Definitive** - A Governance Statement is precisely worded and explicit in nature. Their words, terms and expressions need to be defined and not subject to interpretation. Indicators of non-definitive guidance are words that are not intuitively obvious to an outside reader. Some words that are examples of non-explicit words are: object, service and function.

Another issue or controversy with specifying requirements is how the statements use imperatives, words like:

- Shall (Requirement)
- Must (Requirement)
- Will (Requirement)
- Should (Requirement)

A major sticking point is the use of the word **Shall**[147)] which basically claims:

- lawyers regularly misuse it to mean something other than “has a duty to.” It has become so corrupted by misuse that it has no firm meaning.
- it breeds litigation. There are 76 pages in “Words and Phrases” (a legal reference) that summarize hundreds of cases interpreting “shall.”
- nobody uses “shall” in common speech. It’s one more example of unnecessary lawyer talk. Nobody

says, “You shall finish the project in a week.”

However, the counterargument is that ***Must should not be use because no one has defined how must is different from shall. Also, shall has held up in court, must has not.*** [148]. So, whether you should use **Shall** or **Must** is a matter for the Community of Interest (CoI) to determine, and it should be consistent.

There is an excellent reference in How to Write and Exceptionally Clear Requirements Document[149].

146)

Stavros, Robert W. and Albrant, Jeremiah; Engineering Governance, SPAWAR, October 9, 2007,

147)

PlainLanguage.gov, Shall and Must, Accessed 5 March 2021, https://www.plainlanguage.gov/guidelines/conversational/shall-and-must/

148)

Wheatcraft, Lou, Requirement Experts, 9 October 2021, Accessed 5 March 2021, https://reqexperts.com/2012/10/09/using-the-correct-terms-shall-will-should/

149)

QRA, How to Write and Exceptionally Clear Requirements Document, Accessed 5 March 2021, https://qracorp.com/write-clear-requirements-document/#elementor-toc__heading-anchor-1

4.2 Functional Requirements

Return to Requirements

Functional Requirements define the basic system behavior. Essentially, they are requirements stating what the system does or must not do, and can be thought of in terms of how the system responds to inputs. Functional requirements usually define if/then behaviors and include calculations, data input, and business processes.

Functional Requirements are features that allow the system to function as it was intended. Put another way, if the functional requirements are not met, the system will not work. Functional requirements are product features and focus on user requirements. Functional Requirements can be used during all phases of a project Lifecycle independent of the development model (i.e., Waterfall Model or Agile Model). In the Waterfall method, these requirements are generally specified early on in the process. In the Agile method, they can be applied throughout each Sprint or applied during specific Sprints.

The DIDO RA views functional requirements from the following perspectives:

4.2.1 Platforms

Return to Functional Requirements

A **Platform** is an overloaded term and depends on the context it is used in. Sometimes, Platform refers to just the hardware (i.e., x86, 68000, CISC, RISC, ARM, etc.), other times it can refer to the Operating system (i.e., Windows, Linux, MacOS, Android, iOS), sometimes it can refer to the run-time environment provided by the programming languages used (i.e., C, C++, C#, Java or .NET), while othertimes it can refer to the networking used to connect computers together (i.e., Transmission Control Protocol (TCP)/Internet Protocol (IP)/User Datagram Protocol (UDP), Bluetooth, ZigBee).

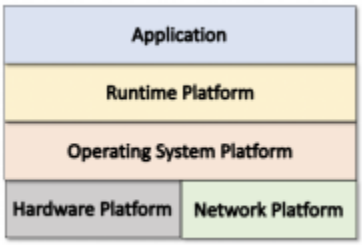


Figure 37: The Kinds of Platforms

As a consequence, in order to identify a specific platform, all the individual platforms and the specific versions need to be identified. This could result in hundreds if not thousands of combinations of platforms. This can quickly become a maintenance nightmare with versions that can almost change daily to apply patches.

```
HW-vvv:OS-vvv:RT-vvv
```

Where:

- HW - represents the specific hardware such as x86, 6800, ARM, etc
- OS - represents the specific Operating System such as Windows, MacOS, iOS, Android, etc
- RT - represents the specific runtime environment such as C, Java, Solidity, C#, eyc
- vvv - represents the specific version of the platform such as 10.15.7

Another way to represent a platform is to use an Application Container that encapsulates the platforms into the container. This simplifies the deployment and the number of user platforms that have to be supported. For example, any user platform that can support a container, will be able to deploy and use the Application.

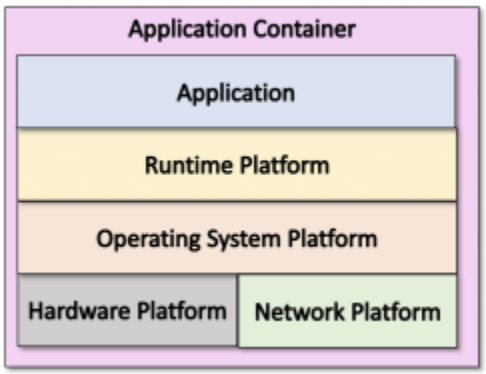


Figure 38: The composition of a Application Container.

- 4.2.1.5 Virtualized Nodes

4.2.1.1 Hardware Platform

Return to Platforms

About

In a Distributed System, each Network Node in the Node Network is associated with a Computer Architecture. As a consequence, a requirement of the distributed system is to identify specifically the computer architectures that are permitted.

It is not a simple task. For example, a coin DIDO might not initially consider a need to support embedded systems, however, the system might need to support Point-of-Sale operations, and as a general rule, these are implemented as embedded systems.

Another example, might be a supply chain DIDO. It might be obvious that this implementation requires embedded systems to read bar codes, RFIDs, and to monitor sensors used in production. As initially conceived, the system may not need Handheld Computers since the results of the embedded systems will be processed by Servers. However, often inventory is now being made available using general purpose smartphones that can even provide maps of where to find the item within the warehouse. Is it far fetched to believe that Amazon may need to employ a supercomputer to help analyze the 1.3B transactions a day? Imagine trying to play “what-if” games with that kind of data and getting responses back in a useful time frame. [150)]

- Embedded Systems
- Servers
- Desktops
- Handheld Computers
- Supercomputers
- Network Computers

150)
Conga; What’s Under the Hood Supporting 1.3B Transactions a Day? Salesforce by the Numbers, 31 October 2020, Accessed 8 December 2020, https://apttus.com/blog/salesforce-by-the-numbers/

4.2.1.2 Operating System Platform

Return to Platforms

On every Network Node in the Node Network is built using an Operating System (OS) Platform, therefore, and each instance of the Distributed Application (ĐApp or DApp) distributed throughout the Node Network must be created specifically for the operating system running on that node. Each build of the Dapp requires work, not just in the process of building the software, but also in terms of maintenance and support. Diversity in the number of OSs supported can drastically increase the cost of of the Dapp throughout the System Lifecycle of the Dapp. It also brings more problems when an OS is considered deprecated and is at End-of-life (EoL).

As a consequence, each Dapp must determine the Operating systems it will support: too few and it may adversely effect adoption, too many and the cost of maintenance may make the Dapp too costly to maintain.

One way to limit the number of operating systems is to select operating systems that are tailored to the specific environment. For example, what are the target environments for the Dapp:

- Embedded Processors?
- Mobile devices such as tablets and phones?
- Network devices such as Network Storage Devices (NDS) or Storage Area Networks?
- Enterprise Servers?
- Desktops?
- Workstations?

The following table lists most of the common OSs and the environments they support. It can be used to help provide a functional list of OSs required for the project.

Operating System	Embedded Systems	Handheld Devices	Desktops	Workstations	Enterprise	Network Devices
Android	X	X	X			
Apstra					X	X
Azure Real Time Operating System (or Azure RTOS)	X					
balenaOS	X					
Blackberry QNX	X	X				
CentOS			X	X	X	
Chromium OS			X			
Cisco Digital Network Architecture (Cisco DNA)						X
Cisco Internetwork Operating System (IOS)						X
Cisco IOS XR						X
Cisco NX-OS						X

Operating System	Embedded Systems	Handheld Devices	Desktops	Workstations	Enterprise	Network Devices
ClearOS					X	X
CloudReady						
ExtremeXOS						X
FreeBSD	X	X	X	X	X	X
FreeRTOS	X					
IBM i					X	
iOS		X				
Junos operating system (Junos OS)						X
LynxOS RTOS	X					
Nokia X Software Platform	X	X				X
Open Network Linux						X
OpenServer						X
Oracle Linux (OL)					X	
Oracle Solaris				X	X	
Red Hat Enterprise Linux (RHEL)					X	
SANtricity Software Operating System (OS)					X	X
SCO UnixWare					X	
SUSE Linux Enterprise Server (SLES)	X		X	X	X	
TrueNAS			X	X	X	X
Ubuntu Linux		X	X			
Windows Server						

4.2.1.3 Runtime Platforms

Return to Platforms

About

A **Runtime Platform** is the software components that are required to run application software that are not part of the Application itself, Hardware Platform or Operating Systems Platform. Some examples of Runtime Platform components are language specific libraries, or language specific virtual machines.

In general, an application is a sequence of steps or events directed and coordinated by the application. Some of the steps are actually contained within the application while others are executed by the Runtime Platform, Operating Systems Platform or the Hardware Platform on behalf of the application.[151)].

When an application is translated into a target hardware instruction set by a compiler, there would be an extreme enlargement of the executable code if all the reused code from the software and hardware platforms were added to the executable. Alternatively, compilers often use compiler-specific external functions in pre-compiled code and assembled into runtime libraries that are linked during execution. These libraries are generally optimized for efficiency and for segregation of functionality for improved accuracy, prevention of common runtime errors, or security concerns.

Runtime libraries provide functionality by accessing the underlying operating or hardware platforms. For example, the use of floating point arithmetic or array/vector processing that use array processors or multiple cores. These considerations can often blur the boundary between standard application runtime libraries and language specific runtime libraries. Often compilers provide copies of runtime libraries optimized for the features of the OS and the hardware.

The concept of a runtime library should not be confused with an ordinary program library like those created by application programmers or delivered as third party such as Dynamic Link Library (.dll) or Shared Object (.so), meaning a program library linked at run time. For example, the programming language C requires only a minimal runtime library (commonly called `crt0`) but defines a large standard library (called `C standard library`) that each implementation delivers.

Consequently, the successful execution of an application depends on the proper versions and revisions of all other the Relocatable Objects. It also means that some kinds of errors can only be caught at execution or run time. These kinds of errors are best caught using strategic testing methodologies and strategies such as regression or unit testing (see: testability for more on testing).

151)

Mansourov, Nikolai and Campara, Djenana; 2011, Accessed: 10 December 2020, https://www.sciencedirect.com/topics/computer-science/runtime-platform

4.2.1.4 Network Platforms

Return to Platforms

A DIDO, by definition, is a collection of networked nodes. Traditionally, the network is assumed to be Ethernet with nodes connected over Local Area Network (LAN) and/or a Wide Area Network (WAN) using a Network Device.

The connections can either be:

- Wired Network using
  - Network Cabling
  - Universal Serial Bus (USB)
- Wireless Network using Wireless Fidelity (Wi-Fi)

Note: for Private, permissioned DIDOs, there may be explicit bans on wireless or even USB connections.

However, there is a lot of growth and development using other wireless connections other than Ethernet or USB. For example:

- Bluetooth
- ZigBee
- Near-Field-Communication (NFC)

Consequently, it is important to identify the kinds of connections that are required to support the DIDO. Some example are:

- Many contactless payments systems use Radio Frequency Identification (RFID) and NFC
- Many supply chains use RFID
- Many smart home efforts use WiFi and zigbee
- Many automobiles use Bluetooth to connect phones, make queries, play music, etc.

4.2.1.5 Virtualized Nodes

Return to Platforms

About

A Virtual Node can reside on any machine that supports either Virtual Machines (VMs) or Application Containers (or refered to as Containers). These can run many different Hardware Platforms and or Operating Systems (OS) Platforms. VMs are limited more by the hardware platforms they run on because of the footprint size of hypervisor, whereas containers are smaller and can consequently run on more machines.

Superficially, the two can appear almost identical. However, VMs have a full Host OS as well as a Guest OS while the Application Containers are lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. They are a bit less secure than VMs since Containers manage memory versus giving each instance its own “machine” complete with memory.

Containers and VMs have similar goals: to isolate an application and its dependencies into a self-contained unit that can run anywhere (i.e., Virtual Nodes)

Moreover, containers and VMs remove the need for physical hardware, allowing for more efficient use of computing resources through sharing, both in terms of energy consumption and cost effectiveness.

The main difference between containers and VMs is in their architectural approach.

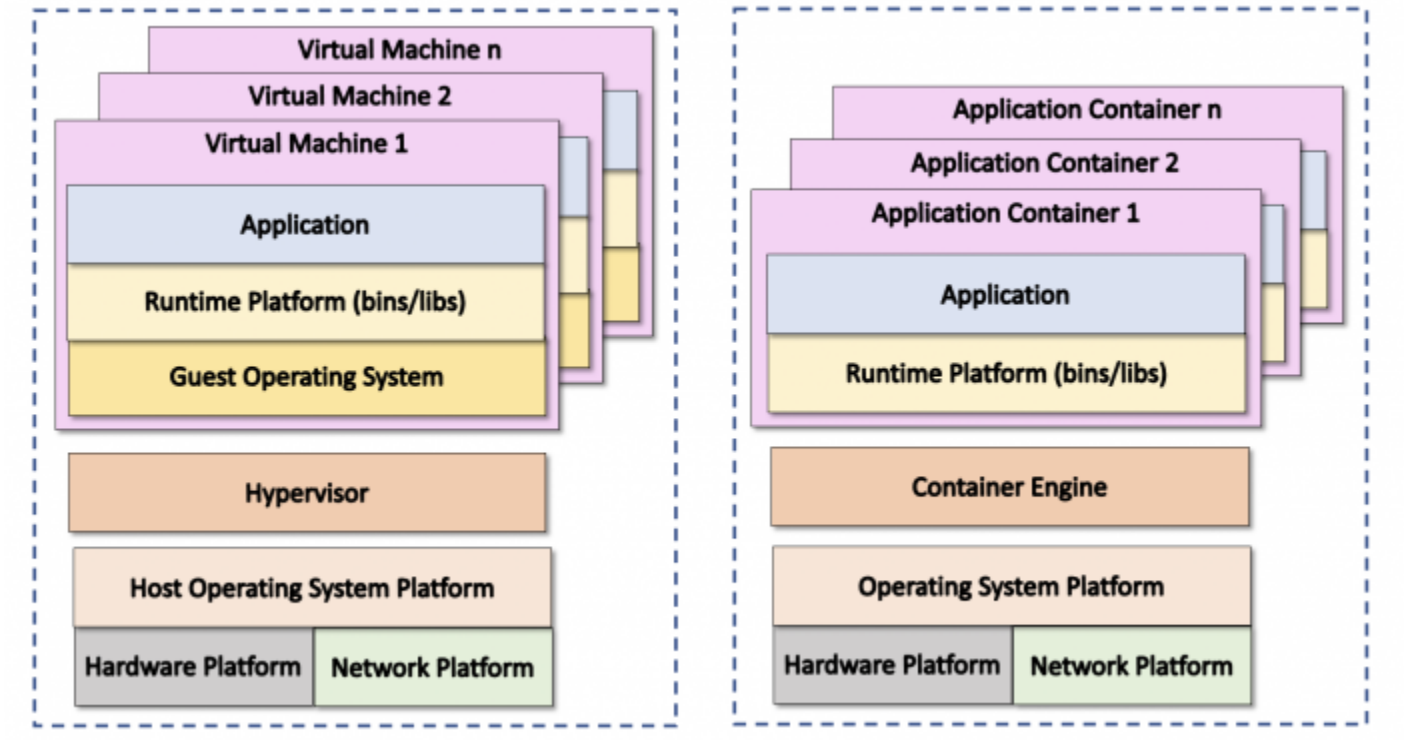


Figure 39: TVirtual Machines and Application Containers.

Virtual Machines

Return to Top

A Virtual Machine (VM) is a software program that behaves as if it were a complete computer. Often VMs are software versions of an existing Hardware Platform. For example, the Hardware Server running an Reduced Instruction Set Computer (RISC) processor, might host a VM that behave as if it were an x86, 68000, CISC, RISC, ARM, etc. The VM will have a virtual processor that executes the appropriate instruction set such as Complex Instruction Set Computer (CISC). Generally, the Hypervisor also known as a Virtual Machine Monitor (VMM) creates, runs and monitors the execution of the VMs. Each VM is isolated from each other and from the host machine which is a good for security reasons. For example, the memory (also virtual) is allocated for only for one VM. If another VM needs to use the same physical memory, the memory is “zeroed” before it can be reused.

As a general rule, Hypervisor's are heavy when compared to Container Engines and Virtual Machines are heavier than Containers. The “heaviness” is in reference to the resources required. For example, a Hypervisor takes more memory and Central Processing Unit (CPU) than a Container Engine because of the amount of work that is required to create and manage the VMs. This means that the smaller the host machine, the more likely that Containers are a better solution. Another reason that VMs are heavier is that each VM has its own copy of a a Guest Operating system while Containers are designed to share key parts of the host operating system.

Application Containers

Return to Top

An Application Container (also know as a Container) is a stand-alone, all-in-one package for a software application that run on an Container Engine. Containers include the the binaries and libraries used to run the application as well the support for the hardware needed to execute the application. Another way to think about a Container is that it is that everything need to execute the application is bundled into a single package (general one file).

Core Operating System (Core OS) is a system for container-based virtualization. Core OS deploys applications in virtual containers as a way to provide effective hardware virtualization for businesses.

- Container Engine
- Disk Image
- Virtual Disk Image (VDI)
- Virtual Machine Images

4.2.2 Access Control

Return to Functional Requirements

A major functional requirement is to provide a classifications of allowable nodes within the DIDO network. A detailed explanation of DIDO networks is provided by in Network Access Control.

Within each of these two classifications it is possible to have public and private access. Public and private access define who is able to write data onto a network or ledger. In contrast, open (i.e., permissionless) and closed (i.e., permissioned) determine who is able to read the data. Networks are classified as[152]:

- Permissionless Networks and Public Network - public and open
- Permissionless Networks and Private Network - public and closed
- Permissioned Networks and Public Network - private and open
- Permissioned Networks and Private Network - private and closed

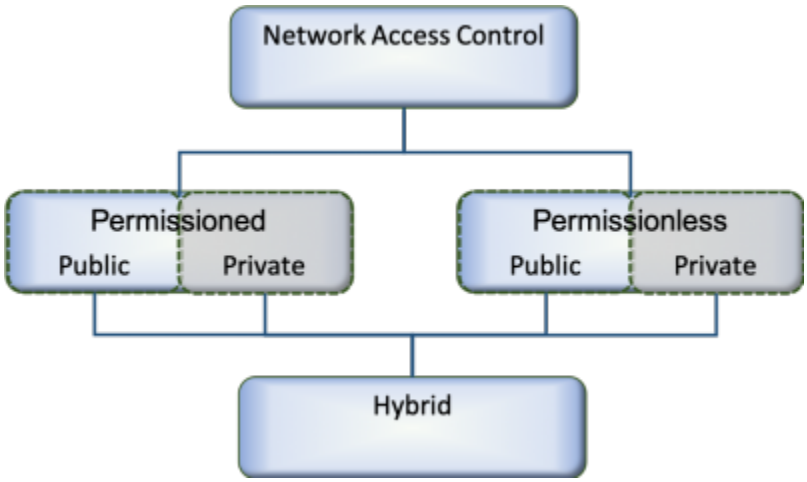


Figure 40: The Node Network Access Taxonomy

Determine the Access Control required for this DIDO by completing the worksheet presented in Table 4. Determine the characteristic required for the particular project of intrest. For example, for the Decentralization the answer should be either Permissioned or Permisionless. When the worksheet is done, use the answers to make the appropriate requirement of Permissionless versus Permissioned, public versus private, or if the requirements are a hybrid. Defining these requirements early can help avoid costly and time consuming changes later.

Table 4: Network Access Worksheet

Characteristic	Characteristic	Description
Decentralization		• **Permissionless** - Permissionless networks are decentralized and distributed. In other words, no one entity can close or terminate the network, modify the content, or censor parts of it. The larger the distributed and decentralized networks and or history are, the harder it is to tamper with.[153)] • **Permissioned** - The degree of decentralization for permissioned networks is a business decision. The extent and quality of decentralization depends upon the number of peers (i.e., nodes), the expected number of bad nodes in the network, and the type of consensus mechanism determined by the stakeholder. Permissioned blockchains usually employ an algorithm such as Byzantine Fault Tolerance, which differs from the proof of work (PoW) algorithm[154)].
Transparency		• **Permisionless** - Users or nodes have complete access to the ledger, transactions, and blocks in the blockchains, which allows for complete auditing of permissionless networks[155)]. • **Permissioned** - Transparency is not a driving force in permissioned networks and is often a major factor in the business decision to choose permissioned over permissionless networks. Most permissioned blockchains do not use cryptoeconomic coins incentive or tokens. The primary incentive of permissioned blockchain participants is to minimize the transparency, cost, time, and ease of sharing information3).
Privacy/Anonymity		• **Permisionless** - Privacy - In permissionless networks, users or nodes of the network are anonymized. Technically, permissionless networks like Bitcoin are pseudonymous, and not truly anonymous.[156)] • **Permissioned** - Anonymity - Permissioned blockchains offer fine-grained visibility into transaction details, as well as, metadata about those transactions which, in many ways, compromises the privacy of the Network participants[157)].

Characteristic	Characteristic	Description
Governance		• **Permisionless** - As a general rule, permissionless networks rely on open source software, which is ruled by open source communities (see Talk Openly Develop Openly (TODO)). The governance of the network is by consensus. Consensus is different for many of the permissionless networksm(i.e., Proof of Work (PoW), Proof of Stake (PoS), Proof of Authority (PoA), etc). [158)] • **Permissioned** - There are fundamental differences between permissionless and permissioned network governance. Permissioned governance is decided and agreed upon by members of the business network. Economic incentives, code quality, code changes, and power allocation among peers are based on the business dynamics and the common purpose and goals of the permissioned members. This allows for agile and responsive networks desired by businesses[159)].
Tokens		• **Permisionless** - Permissionless blockchains employ fat protocols that compensate network contributors with Tokens. As the value and utility of the network increases, the value of the underlying tokens increases as well. This is the premise of cryptoeconomics and Initial Coin Offering (ICO) based fundraising. There are two predominant types of tokens today: monetary value tokens and utility tokens. Monetary value tokens are used in myriad ways as instruments for exchanging value. Utility tokens are akin to loyalty points: they have intrinsic value but no monetary value outside of that ecosystem.[160)] • **Permissioned** - Permissioned blockchains generally do not employ a cryptoeconomic coins incentive or tokens[161)].
Scalability & Performance		• **Permisionless** - For all the value blockchains bring to modern business processes, their Achilles heel often involves scalability and performance. Both Bitcoin and Ethereum blockchains suffer from poor scores in this area. For example, a recent blockchain game called Crypto kittles clogged the Ethereum network. Having said that, these are just early teething troubles, and startups are experimenting with various strategies to address this issue. Hopefully it is only a matter of time before this issue becomes a non-entity.[162)] • **Permissioned** - Permissioned blockchains use consensus mechanisms, which are computationally inexpensive (when compared to proof of work (PoW)). Therefore, they enjoy substantially better scalability and performance than their permissionless network cousins[163)].

Characteristic	Characteristic	Description
Open Read and Write		• **Public** - Anyone can participate by submitting transactions to the blockchain, such as Ethereum or Bitcoin; transactions can be viewed on the blockchain explorer.[164)]
Ledger Is Distributed		• **Public** - The database is not centralized like in a client-server approach, and all nodes in the blockchain participate in the transaction validation.[165)]
Immutable		• **Public** - When something is written to the blockchain, it can not be changed, in other words it is immutable.[166)]
Secure Due to Mining		• **Public** - For example, with Bitcoin, obtaining a majority of network power could potentially enable massive double spending, and the ability to prevent transaction confirmations, in addition to other potentially malicious acts.[167)]
Enterprise Permissioned		• **Private** - The enterprise controls the resources and access to the blockchain, hence private and/or permissioned.[168)]
Faster Transactions		• **Private** - When you distribute the nodes locally, but also have far fewer nodes that participate in the ledger, performance is faster.[169)]
Better Scalability		• **Private** - Being able to add nodes and services on demand can provide a great advantage to the enterprise.[170)]
Compliance Support		• **Private** - As an enterprise, you would likely have compliance requirements to adhere to; having control of your infrastructure enhances ability to satisfy this requirement more seamlessly.[171)]
Consensus More Efficient		• **Private** - Enterprise or private blockchains have fewer nodes and usually a different consensus algorithm, such as BFT vs PoW.[172)]
Private Transactions		• **Hybrid** - Transaction are private but verifiable using the ledger's immutable data objects (i.e., leverage its public state). In its public state, each transaction gets approved by a massive network and is essentially secure and trustworthy. Hence, there is no need for a central governing body or an exhaustive chain of intermediaries to supervise things. So, any change done to a transaction will undergo a “kindred” approval process, making it next to impossible for a single actor to meddle with the transaction or entries[173)].

Characteristic	Characteristic	Description
Equality		• **Hybrid** - Everyone in the network has equal rights to view, modify, and append their consent to a transaction. In addition, the identity of transacting parties is never disclosed to all the visible network participants. [174].
Non-Repuddiation		• **Hybrid** - Anonymity is simply not acceptable to financial institutions and regulated industries with their strict Know Your Customer (KYC) standards. [175].
Confidentiality		• **Hybrid** - Unrestricted visibility of the public state of the network exposes all the data to a colossal network breach, which is counter to data confidentiality obligations, as well as their business concerns.[176].

- **Note:** Another category of networks is a hybrid network, which makes it possible to restrict the visibility of information on the network using a combination of

public, private, permissionless and permissioned networks. Therefore, hybrid networks are appealing to regulated markets because they offer the benefits of public blockchain and private blockchain together.[177]

152) , 165) , 166) , 167) , 168) , 169) , 170) , 171) , 172)
"Public Vs Private Blockchain In A Nutshell", Demiro Massessi, 12 December 2018, https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f

153) , 154) , 155) , 156) , 157) , 158) , 159) , 160) , 161) , 162) , 163)
"Nuances Between Permissionless and Permissioned Blockchains", Anant Kadiyala, 18 February 2018, https://medium.com/@akadiyala/nuances-between-permissionless-and-permissioned-blockchains-f5b566f5d483

164)
"Public Vs Private Blockchain In A Nutshell", Demiro Massessi, 12 December 2018, https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f

173)
"If you Thought Blockchain was Amazing, Wait till You Read about Hybrid Blockchain", Atul Khekade, 20 January 2018, https://www.entrepreneur.com/article/307794. This article uses the term "agnate approval" rather than "kindred approval"; however, agnate limits a kindred relationship to males only. Thus, we prefer the term "kindred" over "agnate"

174) , 175) , 176)
"If you Thought Blockchain was Amazing, Wait till You Read about Hybrid Blockchain", Atul Khekade, 20 January 2018, https://www.entrepreneur.com/article/307794

177)
"Hybrid Blockchain: Decentralized Option for Highly Regulated Markets - Few players in highly regulated markets have adopted blockchain technology. However, hybrid blockchain will change this.", Mina Down, 14 November 2018 https://blog.goodaudience.com/hybrid-blockchain-decentralize-highly-regulated-markets-900f30a37903

4.3 Non-Functional Requirements

Return to Requirements

About

Non-functional requirements are often incorrectly assumed rather than being explicitly defined by users. This can lead to problems towards the end of a project as the user expectations for non-functional requirements are not met. Many times, the developers dismiss non-functional requirements as non-testable and therefore not enforceable.

This lack of specificity in non-functional requirements sets the stage for conflicts between the users, system architects, systems engineers, and developers. For example, users expect software to start and run every time it is used however, the non-functional requirement of reliability may never have been explicitly specified.

Users expect new features to be added to a system and tested before they use them. Users assume the software is maintainable without an explicit declaration for "maintainability". In many ways, they expect it to be an unwritten requirement and or goal. In other words, users expect the system to be analyzable, changeable, stable and testable[4)]. For example, smartphone users will switch apps to other apps if the energy consumed by the app is not efficient. Efficiency is therefore a non-functional requirement. Energy consumption may also be a functional requirements (i.e., An application can not use more than 1040 mW (milli-Watt) per SMS message. [5)].

The DIDO RA assumes the following non-functional requirements:

- 4.3.1 Portability
- 4.3.2 Reliability
- 4.3.3 Maintainability
- 4.3.4 Securability
- 4.3.5 Manageability
- 4.3.6 Usability
- 4.3.7 Performance
- 4.3.8 Interoperability
- 4.3.9 Elasticity
- 4.3.10 Scalability

Creating a Trade Study

Return to Top

A trade study or trade-off study helps consumers compare products on an equal footing, in other words, to help a consumer compare apples-to-apples so to speak. For example, when comparing cameras it is

important to know the resolution metric of the camera. Simplistically, the higher the resolution, the better the camera. Speakers on the other hand might use the highest or lowest speaker frequency responses as a metric. Each metric is unique to the product being evaluated. However, if two cameras each have the same camera resolution, then the additional metric of frequency response might be used as a differentiator.

In the examples above, the camera resolution and frequency responses are specific to the product being evaluated and are referred to as functional requirements. However, there are also a set of requirements with corresponding metrics that capture products non-functional requirements (i.e., sometimes referred to as the "**ilities**" and include things like maintainab**ility**, portab**ility**, reliab**ility**, etc.)

Often a trade study is based on a collection of Figure of Merits with each FoM representing a single evaluation score for a single function. In the examples above, the camera resolution, or the high frequency response. An entire camera may have a FoM, but it represents the cumulative FoM of each function. A Camera may have other FoMs such as weight, battery life, size, or exchangeable lenses.

- How to Use the Boiler Plate

4)
Prolifics Testiing, Achieving Requirements Testability, 10 October 2018, Accessed 10 November 2020, https://www.prolifics-testing.com/news/achieving-requirements-testability
5)
Sai Suren Kumar Kasireddy and Vishnuvardhan Reddy Bojja, Measurements of EnergyConsumption in MobileApplications with respect toQuality of Experience, School of Computing, Blekinge Institute of Technology, 37179 Karlskrona, Sweden, March 2012, Acessed on 10 November 2020, https://www.diva-portal.org/smash/get/diva2:829733/FULLTEXT01.pdf

4.3.1 Portability

return to Non-Functional Requirements

About

Portability is the *degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another. This characteristic is composed of the following sub-characteristics:*[180)]

- ***Adaptability*** *- Degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments.*
- ***Installability*** *- Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment.*
- ***Replaceability*** *- Degree to which a product can replace another specified software product for the same purpose in the same environment.*

The phrase “to port” means to modify software and make it adaptable to work on a different computer system. For example, to port an application to Linux means to modify the program so that it can be run in a Linux environment.

Portability also refers to the ability of an application to move across environments, not just across platforms. To clarify, a computer platform generally refers only to the operating system and computer hardware. A computer environment is much broader and may include the hardware, the operating system and the interfaces with other software, users and programmers.

180)

ISO/IEC 25010, Portability, Accessed 27 July 2020, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/64-portability

4.3.1.1 Adaptability

Return to Portability

About

Return to the Top

Adaptability is the degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments. In other words, it is the extent to which software systems adapts to changes in its environment such as operating system, databases, runtime environments, etc. An adaptable software system tolerates environmental changes without the need for external intervention. For example, a dual-mode cell phone can find out by itself if any one of the two wireless standards it supports is available at its current location and, if so, starts using that standard.[181)].

Nary Subramanian & Lawrence Chung[1] describe software architectures as composed of two elements: components and connectors. Adaptability in software can occur when either the components of the software change or the connectors between the software change.

Software Adaptability is when a software component with a well-defined, stable Application Programming Interface (API) can be exchanged using another component with minimal effort, as long as that component adheres to the API. For example, SQL describes an API for a database component. As long as the software adheres to the standard SQL API, the DataBase Management System (DBMS) can be exchanged between, for example, Oracle and PostgreSQL, with no to minimal impact.

Architecture Adaptability is when the connectors between software components change without having to change the components. This again comes down to having well-defined, stable APIs for the connectors. For example, the Unix File System (UnixFS) is a connector between software components and the physical filesystem. The associated UnixFS library can be exchanged for the InterPlanetary File System (IPFS) UnixFS connector and the software component should have no to minimal impact.

Nary Subramanian & Lawrence Chung[1] further define the following adaptability indices.

- **EAI** - **Element Adaptability Index** is a general purpose index, which represents a weighted value of adaptability where **1** represents complete adaptability and **0** represents no adaptability
- **AAI** - **Architecture Adaptability Index** is an EAI based on the number of elements in the architecture

$$AAI = \sum_{n=1}^{+\infty} EIA_{arc\ elements}$$

- **SAI** - **Software adaptability index** is an EAI based on the number of architectures for the software

$$SAI = \sum_{n=1}^{+\infty} EIA_{sw\,elements}$$

DIDO Specifics

Return to the Top

To be added/expanded in future revisions of the DIDO RA

=-

181) Nary Subramanian and Lawrence Chung, Metrics for Software Adaptability, University of Texas - Dallas, Accessed 29 July 2020, https://personal.utdallas.edu/~chung/ftp/sqm.pdf

4.3.1.2 Installability

Return to Portability

About

Installability and its mirror requirement of un-installability allows for an individual product or an entire system to be installed and uninstalled on a system easily, efficiently and with a minimum number of side effects. A side effect occurs when removing a product or system has adverse effects on other products or systems running on the same computer or network. For example, when a product was installed, it installed a library to access a particular DataBase Management System (DBMS) but, in so doing, removed a library required to access a different DBMS. These side effects can also occur when a product or system is un-installed. For example, when a product is un-installed, it removes a library used to access a particular DBMS; however, this library is required by other products or systems on the computer or network.

In addition, the complexity of installing or un-installing a product or system may be extremely tedious, detailed, and laborious with many steps and decision points. Each step in this process introduces risk. It is important to remember that risk is multiplicative, thus as the number of steps involved increases, even if the individual risk of each step is low, the overall risk to the success of the installation or un-installation increases.

Many of these issues were the bane of new systems and products in the past. Fortunately many of these issues have been addressed through the use of Wizards, Package Manager tools, containerization processes and orchestration tools. Installation usually entails the following checks and functions:

- The target system has the correct system resources available
  - Hardware architecture (32-Bit or 64-Bit Central Processing Unit (CPU) etc)
  - Operating System (OS) such as Android, IOS, Linux, MacOS, Windows, Unix, etc.)
  - Network connectivity
  - Hardware resources such as memory, disk space, etc.
- The target system does not already have the software installed (i.e., previous or current version of the software). If so, perform an upgrade.
- The target system has the proper directories, files and operating system privileges
- Add configuration data (i.e., configuration files, Environment Variables, or Windows Registry entries) to the target system
- Make software accessible on the target system (i.e., setting privileges, creating links, and shortcuts)
- Configure components required to run the target system (i.e., Daemon, services, etc)
- Activate software on the target system (i.e., license agreements, license activation, system registration, etc.)
- Update other third party components to acceptable levels on the target system

Installers can be classified according to the amount of interaction with the user:

Table 1: Installer Classifications

Kind of Installer	Description
Attended installation	This generally requires a user to attend the installation process to answer questions about where to do the install on the target system, provide information about the user, accept any terms and conditions, etc. [1]
Silent installation	This allows installation of a system or program on a target system without any notification to the user. This is often the backdoor used by malware. It differs from **Attended** and **Unattended** installations only in that the user may be completely unaware of the installation.
Unattended installation	This installation is similar to the **Attended Installation** but does not require any human interaction, thus allowing for systems or programs to be installed with the user just monitoring the installation rather than interacting with it.
Headless installation	This refers to the graphics head usually used to drive a monitor. If there is no graphics head, there are only command line interfaces and logs for detailing the installation process. Often these installations use Telnet where the installation on one computer (or machine) is being done from another computer (or machine). **Note:** could be a virtual machine.
Scheduled or automated installation	This term is usually used with **Attended** or **Unattended** installation. The installation process is scheduled for a later time, or on a schedule (i.e., every first Tuesday of the month).
Clean installation	This installation can be considered “hostile” in that no regard is made for previous installations. It can also mean that it does not care about other systems or programs that are installed. It `makes clean` and then starts the installation process anew.
Network installation	This kind of installation is made using a shared network resource. It often requires the installation of a minimal or skeleton operating system before the rest of the installation can occur. This kind of installation is used often for large corporate, government or other institutional organization.

[1] **Note:** It is possible to have hybrids of these kinds of installers. For example, an installer is **Attended** for the first part of an installation and then unattended for the remainder of the installation

In addition to the kinds of installations. Installers programs can either be **self contained**[1] and specific[2] in what they can install (i.e., **Wizard**) or they can be generalized to handle any kind of installation (i.e., **Package Manager**). Sometimes, installer programs themselves need to be installed or updated, a pattern referred to as "bootstrapping", which entails the use of a lightweight, simple and small executable file that is initially downloaded and started on the target system. This executable updates the installer software and then launches into the installation of the desired software. Often in complex systems or programs, the initial bootstrapping process updates other components that the desired software depends upon. For example, an operating system or DBMS update.

[1] **Note:** Contain all the files they need to perform the installation.
[2] **Note:** product based such as Microsoft Word, Parallels

DIDO Specifics

Return to the Top

To be added/expanded in future revisions of the DIDO RA

4.3.1.3 Replaceability

Return to Portability

About

Replaceability occurs when software components are developed using open, well written, standard specifications, usually captured as an Application Programming Interface (API). The standards can be either technical (i.e., developed by a Standards Organization) or de facto (i.e., developed by a for-profit or not-for-profit corporation). Replaceability is also a key factor in preventing Vendor Lock-In.

Replaceability is not just about the ability to switch suppliers and avoid Vendor Lock-In[182)] of the components, it's also about managing risk to the target system. This is especially true because each component can have its own Lifecycle with its own End-of-life (EoL) timelines, independent of the target system. In addition to the components' lifecycle, many components are now Open Source Software (OSS), which can often have forks spawning newer and competing products with similar, but not identical APIs. A recent article describes the Best Message Queue (MQ) Software of 2020. It describes 30 of the "top" Message Queue(MQ) Message-Oriented Middleware (MOM) software products.[183)]:

1. MuleSoft Anypoint Platform
2. IBM MQ
3. Azure Scheduler
4. Apache Kafka
5. Google Cloud Pub/Sub
6. Amazon MQ
7. RabbitMQ
8. Apache ActiveMQ
9. KubeMQ
10. IBM MQ on Cloud
11. Azure Queue Storage
12. Alibaba Message Queue
13. Alibaba Message Service
14. CloudAMQP
15. Intel MPI Library
16. ZeroMQ
17. Apache Qpid
18. Apache RocketMQ
19. IBM Compose for RabbitMQ
20. IronMQ
21. PubSub+
22. Red Hat AMQ
23. TIBCO Enterprise Message Service
24. Bottomline GTBridge
25. CloudMQTT

26. Compose Hosted RabbitMQ
27. deepestream.io
28. Enduro/Z
29. EnMasse
30. IBM Cloud Pak for Integration

Obviously, not all these products have the same API. They definitely do not have the same Wire Protocol, so it is very difficult to “mix and match” Publishers and Subscribers. In a distributed system, this would require all the components within the system to upgrade at the same time, sometimes referred to as a “Reboot the World Problem”. And when the upgrades occur, each part of the system requires complete Regression Testing to make sure that all the parts of the system continue to operate and function according to the specifications.

Replaceability is closely related to Adaptability and the kinds of Installers Architecture and Software Adaptability. Whenever there is an architecture or software adaptability issue, there is probably a Replaceability issue as well. Additionally, the concepts of Installability and Replaceability overlap. The harder a system or a product is to install, the greater the probability it will also be hard to replace.

Competitive products within the same domain are ideal candidates for Replaceability. However, replacing products should not just be driven by the cost of acquisition (i.e., purchase price) but also on the Total Cost of Ownership (TCO), which considers the cost throughout the lifecycle of the target system or component including maintenance (see 4.3.3 Maintainability). This cost can be tied to other tangential products such as debuggers, performance monitors, loggers, or any of the Non-Functional Requirements.[184)]

DIDO Specifics

Return to the Top

To be added/expanded in future revisions of the DIDO RA

182)

Note: Vendors are not just proprietary corporations; Open Source projects produce and sell products also. The software might be “free”, but the producers are competitors that have the same drive to lock-in customers as the corporations

183)

Best Message Queue (MQ) Software, https://www.g2.com/,

184)

Portability Testing Guide with Practical Examples, Software Testing Help, 30 June 2020, Accessed 31 July 2020, https://www.softwaretestinghelp.com/what-is-portability-testing/

4.3.2 Reliability

return to Non-Functional Requirements

About

https://www.sebokwiki.org/wiki/Reliability,_Availability,_and_Maintainability#RAM_Considerations_during_Systems_Development

Reliability *is the degree to which a system, product or component performs specified functions under specified conditions for a specified period of time. This characteristic is composed of the following sub-characteristics:*[185)]

- ***Maturity*** *- Degree to which a system, product or component meets needs for reliability under normal operation.*
- ***Availability*** *- Degree to which a system, product or component is operational and accessible when required for use.*
- ***Fault tolerance*** *- Degree to which a system, product or component operates as intended despite the presence of hardware or software faults.*
- ***Recoverability*** *- Degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system.*

Reliability, maintainability, and availability (RAM) are three system attributes that are of great interest to systems engineers, logisticians, and users. Collectively, they affect both the utility and the life-cycle costs of a product or system. The origins of contemporary reliability engineering can be traced to World War II. The discipline's first concerns were electronic and mechanical components (Ebeling, 2010). However, current trends point to a dramatic rise in the number of industrial, military, and consumer products with integrated computing functions. Given the rapidly increasing integration of computers into products and systems used by consumers, industry, governments, and the military, reliability must consider both hardware and software.

Maintainability models present some interesting challenges. The time to repair an item is the sum of the time required for evacuation, diagnosis, assembly of resources (parts, bays, tool, and mechanics), repair, inspection, and return. Administrative delay (such as holidays) can also affect repair times. Often these sub-processes have a minimum time to complete that is not zero, resulting in the distribution used to model maintainability having a threshold parameter.

A threshold parameter is defined as the minimum probable time to repair. Estimation of maintainability can be further complicated by queuing effects, resulting in times to repair that are not independent. This dependency frequently makes the analytical solution of problems involving maintainability intractable and promotes the use of simulation to support analysis.

DDS Specifics

return to Top

To be added/expanded in future revisions of the DIDO RA

185)

ISO/IEC 25010, Reliability, Accessed 27 July 2020, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/62-reliability

4.3.2.1 Maturity

Return to Glossary

About

Return to Top

There are two ways to think about maturity: the maturity of the products or systems and the maturity of the communities which develop the systems or products. Usually, the two kinds of maturity go hand in hand. A mature product or system is the result of a mature community process and, visa versa, a mature community process produces mature products.

- **Note:** Community, in this case, can refer to a Community of Interest (CoI) or a corporation.

Products or Systems

Return to Top

Product or System maturity is an assessment (sometimes quantifiable) of how well a product or system meets its requirements for reliability under normal operations.

Maturity of the components selected for inclusion in a system can play a significant role in the overall success of a system. Components that are mature are more likely to be stable and reliable; qualities that directly translate to stable and reliable integrations, which are thereby robust and resilient when inevitable changes to the system are made. This holds true as long as the components are not coming close to End-of-life (EoL). See Manageability Costs .

Rafa E. Al-Qutaish and Alain Abran have proposed a maturity model based on Six Sigma (6Sigma).[186)]

> *The quality of a product can be assessed either directly by looking into the product itself, or indirectly through assessing the process used to develop that product. In the software engineering field, there are currently numerous capability and maturity models for assessing a set of specific software processes, but very few product maturity models for those interested in assessing the quality of software products. This paper presents a maturity model designed to directly assess the quality of a software product, i.e. the Software Product Quality Maturity Model (SPQMM). This model is based on the six-sigma view of product quality and handles – in submodels – the three views of quality specified in ISO 9126, that is, the Software Product Internal Quality Maturity Model (SPIQMM), the Software Product External Quality Maturity Model (SPEQMM), and the Software Product Quality-in-Use Maturity Model (SPQiUMM).*

$$WQL = \frac{(IQL + EQL + iUQL)}{3}$$

Where:

- ***WQL*** *is the quality level of the whole software product, including the quality levels of all three stages of the software product*
- ***IQL*** *is the internal quality level*
- ***EQL*** *is the external quality level*
- ***iUQL*** *is the in-use quality level of the software product*

Table 6: Sigma values based on the quality level and the software integrity level

5	4	3	2	1	0	
Very High	**High**	**Intermediate**	**Low**	**Trivial**	**None**	
Zero Sigma Shift	1.5 Sigma Shift	2.0 Sigma Shift	2.5 Sigma Shift	3.0 Sigma Shift	3.5 Sigma Shift	Assigned Sigma Ranges
QL ≥ 99.99997%	QL ≥ 99.976%	QL ≥ 99.865%	QL ≥ 99.379%	QL ≥ 97.724%	QL ≥ 03.319%	σ ≥ 5
QL < 99.99997%	QL < 99.976%	QL < 99.865%	QL < 99.379%	QL < 97.724%	QL < 93.319%	
and	and	and	and	and	and	5 > σ ≥ 4
QL ≥ 99.996%	QL ≥ 99.379%	QL ≥ 99.724%	QL ≥ 99.319%	QL ≥ 97.134%	QL ≥ 69.146%	
QL < 99.996%	QL < 99.379%	QL < 97.724%	QL < 93.319%	QL < 84.134%	QL < 69.146%	
and	and	and	and	and	and	4 > σ ≥ 3
QL ≥ 99.865%	QL ≥ 93.319%	QL ≥ 84.134%	QL ≥ 69.146%	QL ≥ 50%	QL ≥ 30.853%	
QL < 99.965%	QL < 93.319%	QL < 84.134%	QL < 69.146%	QL < 50%	QL < 30.853%	
and	and	and	and	and	and	3> σ ≥ 2
QL ≥ 97.724%	QL ≥ 69.146%	QL ≥ 50%	QL ≥ 30.853%	QL ≥ 15.865%	QL ≥ 6.680%	
QL < 97.724%	QL < 69.146%	QL < 50%	QL < 30.853%	QL < 15.865%	QL < 6.680%	σ < 2

Communities

Return to Top

There are several ways to establish or assess Community[187)] maturity:

- ISO 9001
- ISO 15288
- ISO 90003-2018
- ISO 10001:2018 Quality management — Customer satisfaction — Guidelines for codes of conduct for organizations

- ISO 10002:2018 Quality management — Customer satisfaction — Guidelines for complaints handling in organizations
- ISO 10003:2018 Quality management — Customer satisfaction — Guidelines for dispute resolution external to organizations
- ISO 10004:2018 Quality management — Customer satisfaction — Guidelines for monitoring and measuring
- OMG: Case Management Model and Notation (CMMN)
- Capability Maturity Model Integration (CMMI)

See also: Talk Openly Develop Openly (TODO).

DIDO Specifics

Return to Top

186)

Rafa E. Al-Qutaish and Alain Abran, A Maturity Model of Software Product Quality, Journal of Research and Practice in Information Technology, 43(4):307-327, November 2011, Accessed 27 July 2020, https://www.researchgate.net/publication/260835325_A_Maturity_Model_of_Software_Product_Quality

187)

Note: Community in this case can refer to a Community of Interest (CoI) or a corporation.

4.3.2.2 Availability

Return to Reliability

About

return to the top

Availability in this context is System Availability. System Availability is the probability the system will function as designed for a particular duration. The duration could be a fixed time period (i.e., 24 hours a day, 7 days a week, or 364.9 days a year) or it could be over a particular mission (i.e., a flying mission, a patient stay, or a growing season). The ability of an item to be in a state to perform a required function under given conditions at a given instant of time or over a time interval, assuming the required external resources are provided. [188)]

It is important to remember that Availability is expressed as a probability expressed in terms of Five Nines, and therefore, the Multiplication Rule of Probability needs to be considered when thinking about a system comprised of parts. Each part has its own probability of success (or failure). The **Multiplication Rule of Probability** means that to find the probability of the intersection of two events, multiply the two probabilities. The intersection of the events occurs when the probability of two events occurring is known. The Multiplication Rule of Probability determines the intersection of two different sets of events, called independent and dependent events.

- An Independent Event is when the probability of an event is not affected by a previous event.
- A Dependent Event is when one event influences the outcome of another event in a probability scenario. To find the intersection of two events, whether they are independent or dependent, multiply the two probabilities together. [189)]

$$A_o = \frac{T_m}{T_m + T_d} \begin{cases} A_o = Operational\ Availability \\ T_m = Mission\ Duration \\ T_d = Observed\ Down\ Time \end{cases}$$

Mission Duration (T_m) is the time the system needs to be operational. T_m can be expressed as a fixed time period (i.e., 24 hours a day, 7 days a week, or 365 days a year) or it could be over a particular mission (i.e., a flying mission, a patient stay, peak energy demand, or a growing season)

Mean Time Between Failure (MTBF) is a calculation of the arithmetic mean (average) time between failures of a system.

- **Note:** If a system is designed with both redundancy and automatic fault bypass, then MTBF is the anticipated lifespan of the system if these features cover all possible failure modes (infinity for all practical purposes). Such systems will continue without noticeable interruption when these conditions are satisfied unless there are secondary failures. This is called active redundancy and

requires no maintenance to prevent mission failure. Active redundancy is required for systems that cannot be maintained, such as satellites. See: Wikipedia, Mean Time Between Failure (MTBF), Accessed 3 July 2020, https://en.wikipedia.org/wiki/Availability_(system)
- **Note:** The term is used for repairable systems[190)]

Downtime (T_d) is the Mission Duration times the sum of all of the different kinds of time required to transition from being down to the time to be fully operational, divided by the Mean Time Between Failure.

- Mean Time To Repair (MTTR) is the time required to restore operations the level defined in the system specification
- Mean Logistics Delay Time (MLDT) is the time required to obtain parts from the part depot or from the manufacturer including transportation to the site
- Mean Active Maintenance Down Time (MAMDT) is the average time required to perform diagnostics and replace parts

DIDO Specifics

return to the top

188)

Note: Availability is part of Reliability, Maintainability, and Availability (RAM)

189)

The Multiplication Rule of Probability: Definition & Examples, Chapter 4, lesson 11, Accessed 3 July 2020, https://study.com/academy/lesson/the-multiplication-rule-of-probability-definition-examples-quiz.html

190)

https://en.wikipedia.org/wiki/Mean_time_between_failures | Mean Time Between Failure (MTBF)]]

4.3.2.3 Fault Tolerance

Return to Reliability

About

Return to the Top

Fault Tolerance is the ability of a system (computer, network, cloud cluster, component, etc.) to continue functioning correctly without interruption during failures. Fault Tolerant systems (or components) prevent disruptions to a system that is considered Safety-Critical System (SCS), Life-Critical System or Mission Critical System. Usually, this requires an understanding of the single points of failure through the multiple critical execution paths in a running system.

The system characteristics of Fault Tolerance High Availability are related in that to achieve high availability, a system must address Fault Tolerance of components on the systems critical paths.

Fault Tolerant systems use redundant (i.e;, spare, backup) components to automatically become available in the event of a component failure to ensure there is no loss of service or data. The ability to use Failover mechanisms to quickly, smoothly and transparently transition to the redundant or backup systems requires a well designed system, with contingency plans and special management processes, hardware or software to ensure the transition. There are some Failover components which are acquired. For example:

- **Power sources** are ruggedized as fault tolerant by incorporating alternative sources and backups like Uninterruptible Power Supply (UPS) and backup generators. A good description of this is provided in the Tactical Microgrid Standard (TMS) use case,
- **Hardware systems** are made Fault Tolerant by deploying identical or equivalent systems that can either be used instead of the original system or use in conjunctions with the original system used as an alternative. For example, a server can be made fault tolerant by using an identical server running in parallel, with all operations mirrored to the backup server.
- **Networks** designed as Fault Tolerant by supporting multiple networks paths between any two endpoints within the Local Area Network (LAN) or Wide Area Network (WAN) are possible but the actual endpoint also needs to be duplicated (i.e., two Network Interface Cards (NICs)). It is also possible to use two different networks such as a wired, Wireless Fidelity (Wi-Fi), Bluetooth, or ZigBee.
- **Software** systems or components become fault tolerant when multiple instances of the software are running in parallel using either operating system threads or even more modern containers such as Docker or orchestration software such as Kubernetes. For example, a database can be continuously replicated to other machines. If the primary database goes down, operations can be automatically redirected to the second database. Another example, would be use of orchestration software such as Kubernetes to automatically use an alternate application container on the same or different machine.

Fault Tolerance needs to be considered in all disaster recovery plans or strategies. For example, Fault

Tolerant systems can use the cloud for backups allowing critical systems to quickly be restored. Although these backups are not true immediate failovers, they can offer a longer timeline for fault tolerance recovery. **Note:** often these backup plans are not geographically local which is particularly important during natural or even human disasters. [191)]

DIDO Specifics

Return to the Top

191)

Mariah Timms, AT&T outage: Internet, 911 disrupted, planes grounded after Nashville explosion. Get the latest updates, Nashville Tennessean, 5 January 2012, Accessed: 8 January 2021, https://www.tennessean.com/story/news/local/2020/12/25/att-outage-internet-down-hours-after-nashville-explosion/4045278001/

4.3.2.4 Recoverability

Return to Reliability

About

Return to the Top

Recoverability is the ability of a system to be rebuilt in the event of a system failure do to human or natural disasters or catastrophic failures in hardware or software. After the system is recovered it is able to resume with full functionality with minimum interruption. For example in DataBase Management System's (DBMS), a Checkpoint is a place in time where the database transactions, operations, and logging are paused long enough to be completed and recorded into the database files. These files are then archived (i.e., copied, backed up) away from the current database files. After the Checkpoints operation is complete, the DBMS can resume with new transactions, operations and logging. Although the concepts of Checkpoints are usually thought of in conjunction with DBMSs, it is also possible to have Checkpoints applied to Operating Systems (OSs) as well. Virtual Machines (VMs) and containers (i.e., Docker) can be thought of as a checkpoint made against the OS files, logs, etc. at a particular point in time. Every time the VM or Container are reloaded, they start from a set known point (a Checkpoint).

DIDO Specifics

Return to the Top

4.3.3 Maintainability

return to Non-Functional Requirements

About

Maintainability is the *characteristic that represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in its environment and requirements. This characteristic is composed of the following sub-characteristics:*[192)]

- ***Modularity*** *- Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.*
- ***Reusability*** *- Degree to which an asset can be used in more than one system, or in building other assets.*
- ***Analysability*** *- Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.*
- ***Modifiability*** *- Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.*
- ***Testability*** *- Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.*

Maintainability is a characteristic of a system that is able to resume full operation after a failure in a component of the system. Components that are mission critical for the system can include equipment, machine, power, air conditioning, software, etc. Maintainability is expressed as the probability of recovery based on a specified time frame, usually done in terms of Five Nines (i.e., 99%, 99.9%, 99.99%, and 99.999%). For example, a 99.999% maintainability would be for 5 minutes, 15 seconds or less of downtime in a year. The downtime must include all the steps required to recover with full operational capabilities, so the time must include removal, diagnostics, assembly of resources required to perform the maintenance (i.e., parts, bays, tools, personnel, etc.) and the re-installation of the failed component.[193) 194)]

Two main aspects of Maintainability need to be addressed for all systems or components:

- Servicability - the ease of conducting scheduled inspections and servicing
- Repairability - the ease of restoring service after a failure

Maintainability is a projection of the downtime of a system. Given that it is, by its very nature, a projection, it should not viewed as a guarantee that a system will only be down for the projected amount of time. Maintainability must therefore rely on models to calculate the probability of failures for components based on actual failure rates for components in the past or test results of the components.

> *There are a wide range of models that estimate and predict reliability (Meeker and Escobar 1998).*

Simple models, such as exponential distribution, can be useful for "back of the envelope" calculations.

System models are used to:

(1) combine probabilities or their surrogates, failure rates and restoration times, at the component level to find a system level probability or
(2) to evaluate a system for maintainability, single points of failure, and failure propagation. The three most common are reliability block diagrams, fault trees, and failure modes and effects analyses.

There are more sophisticated probability models used for life data analysis. These are best characterized by their failure rate behavior, which is defined as the probability that a unit fails in the next small interval of time, given it has lived until the beginning of the interval, and divided by the length of the interval. See: Upkeep's discussion of models in Maintainability Definition & Calculation [195)]

All these models are abstractions of reality; therefore, at best they are only approximations of reality. To the extent they provide useful insights, they are still very valuable. The more complicated the model, the more data necessary to develop precise estimations. The greater the extrapolation required for a prediction, the greater the imprecision. Also, obtaining all the data required as input to the models is difficult, time consuming, and may not even be very accurate.

Measuring the Mean Time To Repair (MTTR), is also used as part of Availability (see 4.3.2.2 Availability).

The MTTR, identifies the average time to restore a system or component after experiencing a failure or breakdown in the expected (i.e., specified) operating conditions. The formula for MTTR is:

$$MTTR = \frac{Total\ downtime\ \left(hours\right)}{Number\ of\ failure\ events}$$

A lower MTTR value corresponds to a higher level of maintainability and which means that maintainable systems take less time to repair.

DIDO Specifics

Return to Top

All systems, regardless of their level of complexity, require maintenance. To reduce the impact of performing maintenance, using physical (hardware) modules (components) that require the fewest number of repairs in a given time frame and choosing hardware and designs that require the least amount of downtime is one way to reduce the impact of maintenance. Another way to reduce the impact of maintenance is to design a system that anticipates maintenance and provides redundancy to handle

the downtime required for maintenance. Nevertheless, managing redundancy is not a trivial task and adds complexity to the overall system. The more components that require redundancy, the more complex the system becomes unless the Middleware can manage the transition seamlessly, easily and transparently.

There are a few major forms of redundancy DDS can help with:

Hardware redundancy	Means that there are multiple modules (components) that provide the same functionality available in the system at the same time. For many systems, a simple dual modular redundancy is sufficient to accomplish the job (i.e., two components). However, for life critical systems, some systems rely on a triple modular redundancy. These systems should have zero to minimum loss from downtime.
Information redundancy	Occurs when there are multiple information sources available, so that modules can use whichever source is active if there is an interruption in information. In a system that has only one network available to it, the information redundancy is of little use if the network needs maintenance.

192)

ISO/IEC 25010, Maintainability, Accessed 27 July 2020, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/64-maintainability

193)

Maintainability Definition & Calculations, Upkeep, answered 4 June 2019, Accessed 13 July 2020, https://www.onupkeep.com/answers/preventive-maintenance/maintainability-definitions-calculations/

194)

Note: Maintainability is part of Reliability, Maintainability, and Availability (RAM)

195)

Maintainability Definition & Calculations, Upkeep, answered 4 June 2019, Accessed 13 July 2020, https://www.sebokwiki.org/wiki/Reliability,_Availability,_and_Maintainability#Models

4.3.3.1 Modularity

Return to Maintainability

About

Modularity describes a characteristic of a system or program to be organized into smaller, reusable components. Each component is self-contained and provides an interface that describes the functionality it offers to other components of the system. It optionally provides a set of interfaces required to fulfill its functionality. In Object-Oriented Programming (OOP), this functionality is encapsulated as set of data attributes. The Module exposes access to these data attributes by defining public interfaces other components can call to manage and manipulate the data attributes of the object (i.e., Module). If the component relies on other components, then it can specify the required components (i.e., Modules).

Many Modules are described by files that only describe the interface to the Module and contain no actual functionality. For example, in C/C++, these Module interfaces are described using header files (i.e., `.h`, `.hpp`) files; in Java these files are regular `.java` files but contain the key word `interface` in their class descriptor; in Common Object Request Broker Architecture (CORBA), interfaces are described as stubs. ECMAScript (i.e., JavaScript), PHP, etc. use the concept of Duck Typing at runtime to accomplish the equivalent of interfaces (i.e., it is based on the methods defined within a Module at runtime rather than on a static, abstract fixed interface).

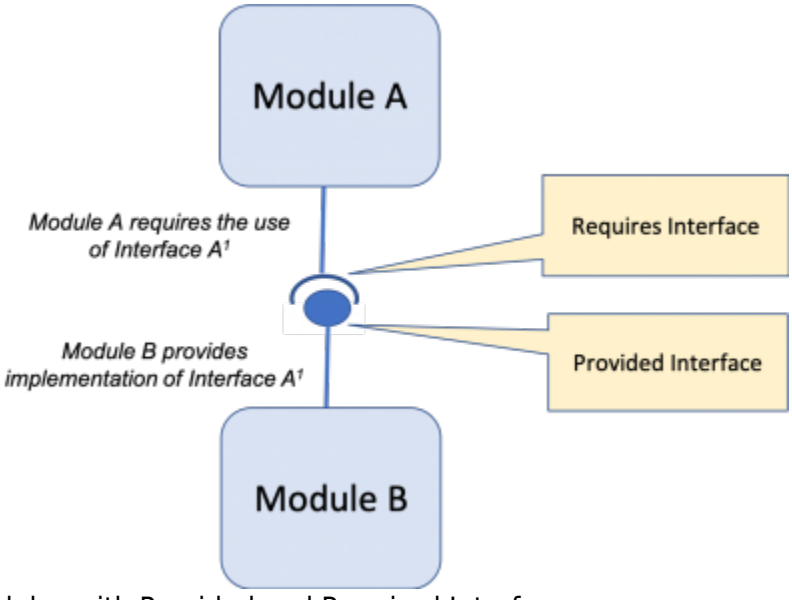


Figure 41: Modeling Modules with Provided and Required Interfaces.

Regardless of the kind of implementation (i.e., interface, stubs, DuckTyping, etc.), APIs need to be public, formalized and freely available to the general public at no cost. The benefit of using interfaces, stubs, or headers is that un-implemented or mismatched functionality can be checked at compile or link time. Dynamic Plug In interfaces (i.e, DuckTyping) can only be checked at runtime. However, dynamic type is very definitely modular. One way around run-time errors is to always provide a default implementation.

The use of Modules allows the architecture and design of a system or program to be re-factored by extracting functionality into new Modules, combining Modules into a new Module, or composing Modules from other Modules. Often the rules for refactoring a system of a program follow many of the same rules as data Normalization.

Yiming et al.[196] introduced the use of complex network theory into software engineering with which to develop metrics for measuring Modularity. They analyzed existing software by representing it as a network using a feature coupling network (FCN). Each method and attribute is considered a node in the node network. Couplings between methods and attributes are considered edges. Edge Weight represents a weighted value based on the number of invocation paths for the node (i.e., method 'A' is called 2 times, has a weight of '2', each attribute is used once, gets a weight of '1').

FCN = (N, E, Ψ)
Where:

- FCN : Feature Coupling Network
- N : the Node set (number of attributes or methods)
- E : the Edge set (couplings between Edge Nodes) of
- Ψ : the Edge Weight (number of invocation paths)

Yiming et al., apply Weyuker's criteria, which are widely used in the field of software metrics, to validate modularity as a software metric theoretically, and also to perform an empirical evaluation using open-source Java software systems to show its effectiveness as a software metric to measure software modularity.

DIDO Specifics

Return to Top

Although Modularity was primarily developed to look at software, it can also be applied to distributed systems by changing the definition of N from the number of attributes or methods to the number of Publishers and Subscribers, and E to the number of couplings between the publishers and subscribers. Ψ would then be a weighting for the volume of couplings between publishers and subscribers.

In practice this means that if functionality is centralized into a normalized set of Endpoints such that couplings become the connections between endpoints and the volume of traffic represents a weighting, the end result is that we are reduced to three variables: N, E and Ψ. Thus, we must address the age old question: is it better to have one command with 100 options, 100 commands with no options, or a set of commands that can share a restricted set of options? It's obvious that the first two choices are unacceptable so we are left with with the last choice where the options are all related based on the functionality of the command.

196)

Yiming Xiang, Weifeng Pan, Haibo Jiang, Yunfang Zhu, Hao Li, Measuring Software Modularity Based on Software Networks, 14 February 2019, Entropy, Accessed 3 Aug 2020, https://www.mdpi.com/1099-4300/21/4/344/htm

4.3.3.2 Reusability

Return to Maintainability

About

Reusability is a way to save time, resources and effort by reusing existing system or software assets that were created previously. Sometimes these assets maintain all the functionality they were originally constructed to provide (e.g., widgets in a Graphical User Interface (GUI)). Sometimes, well written modularized software is abstracted into generalized Modules that can then be used to serve one or more design patterns, e.g., a linked list, hash algorithm, etc.) When this is done, the software is considered to be reusable.

Reusability is defined as the *ability* of something to be used more than once. In contrast, *reuse* is defined as the *action* of using something more than once. Reusability promotes and enables reuse but does not ensure complete reusability.[197)]. Therefore, merely creating a system or program for Reusability is not important unless there is also a “marketing effort” to promote the reuse of the system or product. Marketing in this context does not mean commercial marketing. Often marketing can be achieved just through making the code available as Open Source.

The meaning and application of 'reuse' also depends on who you ask. To a Software Engineer, they naturally think about reuse of code, executables or even software patterns, etc. To a Systems Engineer, reuse makes them think about use cases, state transition diagrams, etc. To an Ontologist, reuse means glossaries, vocabularies and ontologies. To a contracting officer, they makes them think of contracts and performance specifications. Thus, it is obvious that even within a technology area, there are lots of different ideas about what, where, and how to re-use artifacts.

Jones[198)] identifies 10 different areas within a software project where reuse can be applied:

1. Architectures
2. Source Code
3. Data
4. Designs
5. Documentation
6. Estimates
7. Human Interfaces
8. Plans
9. Requirements
10. Test Cases

Types of Reuse

Return to Top

Reuse is a multi-faceted idea, which requires an overall classification for the various types of reuse. Frakes and Terry [199)] have developed a taxonomy for reuse based on an extensive review of the literature on Reuse and Reusability. Table 7 depicts the two major elements of the reuse taxonomy: **Facets**, which cover all the types of reuse, and **Terms**, which are used to describe each **Facet**. For example, the **Development Scope Facet** has two possible **Terms** within it: **Internal** and **External**. When describing the **Development Scoping** used in a company or project, this can be **Internal**, **External** or both. The **Terms** associated with each **Facet** are defined in Table 8.

Table 7: Types of Software Reuse [3)]

Facet					
Development Scope[1)]	**Modification**[2)]	**Approach**[3)]	**Domain Scope**[4)]	**Management**[5)]	**Reused Entity**[6)]
Internal (Private)	White Box	Generative	Vertical	Systematic (Planned)	Code
External (Public)	Black Box (vrbatum)	Compositional	Horizontal	Ad Hoc	Abstract Level
	Adaptive (porting)	In-the-Small			Instance Level
		In-The-Large			Customization Reuse
		Indirect			Generic
		Direct			Source Code
		Carried Over			
		Leveraged			

[1)] **Development Scope** refers to whether the reusable components are from a source external or internal to a project.
[2)] **Modification** refers to how much a reusable asset is changed.
[3)] **Approach** refers to different technical methods for implementing reuse.
[4)] **Domain Scope** refers to whether reuse occurs within a family of systems or between families of systems.
[5)] **Management** refers to the degree to which reuse is done systematically.
[6)] **Reused Entity** refers to the type of the reused object.

Clear definitions of types of reuse are necessary prerequisites to measurement. Table 2 provides a faceted classification of reuse definitions gathered from the literature Each column specifies a facet, with the facet name inbold.

Table 8: Definitions of Types of Reuse [3)]

Type of Reuse	**Description**
abstract-level	Abstract-level reuse is the use of high-level abstractions within an object-oriented inheritance structure as the foundation for new ideas or additional classification schemes.
ad-hoc	Ad-hoc reuse refers to the selection of components that are not designed for reuse from general libraries or where reuse is conducted by an individual in an informal manner

Type of Reuse	Description
adaptive	Adaptive reuse is a reuse strategy that uses large software structures as invariants and restricts variability to low-level, isolated locations. An example is changing arguments to parameterized modules.
black-box	Black-box reuse is the reuse of software components without any modification. See **verbatim**.
Carry-Over	Carry-Over Reuse is when software used with one version of a software component is carried over and used as is in a subsequent version of the same system.
compositional	Compositional reuse is a reuse strategy that uses small parts as invariants and then uses variant functionality to link these parts together. Programming in a high level language is an example. Compositional reuse is the use of existing components as building blocks for new systems. The Unix shell is an example
customization	Customization reuse is the use of object-oriented inheritance to support incremental development. A new application may inherit information from an existing class, overriding some methods in that class and adding new behaviors.
direct	Direct reuse is reuse without going through an intermediate entity.
external	External reuse level is the number of lower level items from an external repository in a higher level item divided by the total number of lower level items in the higher level item. See **Public**.
generative	Generative reuse is reuse at the specification level using application or code generators. Generative reuse offers the “highest potential payoff.” The Refine and MetaTool systems are state of the art examples.
generic	Generic reuse is reuse of generic packages, such as templates for packages or subprograms.
horizontal	Horizontal scope reuse is reuse of generic parts in different applications. Booch Ada Parts and other subroutine libraries are examples.
In-the-large	Reuse-in-the-large is the use of large, self-contained packages such as spreadsheets and operating systems.
In-the-small	Reuse-in-the-small is the reuse of components that are dependent on the environment of the application to achieve full functionality. Favaro asserts that component-oriented reuse is reuse-in-the-small.
indirect	Indirect reuse is reuse through an intermediate entity. The level of indirection is the number of intermediate entities between the reusing item and the item being reused
instance-level	Instance-level reuse is the most common form of reuse in an object-oriented environment. It is defined as simply creating an instance of an existing class.
internal	Internal reuse level is the number of lower level items not derived from an external repository and used more than once divided by the total number of lower level items not derived from an external repository. See **Private**.
leveraged	Leveraged reuse is reuse with modifications.
private	Private reuse is“the extent to which modules within a product are reused within the same product.” See **Internal** .
public	Public reuse is “the proportion of a product which was constructed externally.” See **External**.
source-code	Source code reuse is the low-level modification of an existing object-oriented class in order to change its performance characteristics.

Type of Reuse	Description
systematic (planned mode)	Systematic/planned mode reuse is the systematic and formal practice of reuse as found in software factories.
verbatim	Verbatim reuse is reuse of some item without modifications. See **Black-Box**.
vertical scope	Vertical scope reuse is reuse within the same application or domain. An example is domain analysis or domain modeling.
white-box	White-box reuse is the reuse of components by modification and adaptation.

Types of Metrics and Models

Return to Top

Frakes and Terry categorize the kinds of metrics and models that can be used for evaluating reuse.

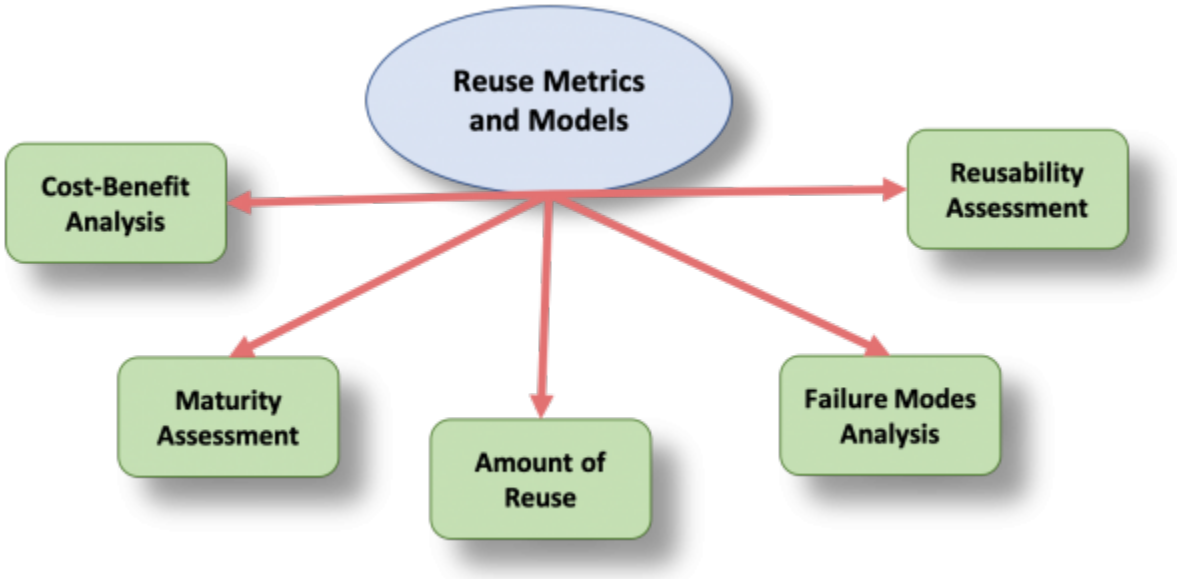


Figure 42: Categorization of Reuse Metrics and Models [3)]

Table 9: Descriptions of Reuse Metrics and Models [3)]

Metric or Model	Description
Cost-Benefit Analysis	Uses **models** to include economic cost/benefit analysis, as well as, quality and productivity payoffs. Maturity assessment models categorize reuse programs by how advanced they are in implementing systematic reuse.
Maturity Assessment	Uses **models** to categorize reuse programs by how advanced they are in implementing systematic reuse.
Amount of Reuse	Uses **metrics** to assess and monitor a reuse improvement effort by tracking the percentages of reuse for life cycle objects.
Failure Modes Analysis	Identifies and orders the impediments to reuse in a given organization.
Reusability Metrics	Indicates the likelihood that an artifact is reusable.
Reuse Library Metrics	Manages and Tracks usage of a reuse repository.

* **Note:** Organizations often encounter the need for these metrics and models in the order presented.

DIDO Specifics

Return to Top

Startup initialization is minimal because of DDS Discovery.

To be added/expanded in future revisions of the DIDO RA

197)
Luis Zafra, Know the Difference: Reusability vs. Reuse, Global Logic, 22 September 2015, Accessed 3 August 2020, https://www.globallogic.com/latam/blog/know-the-difference-reusability-vs-reuse/
198)
Jones, C. Software return on investment preliminary analysis, 1993, Software Productivity Research, Inc.,
199)
William Frakes and Carol Terry, Software Reuse and Reusability Metrics and Models, Virginia Tech, Accessed on 4 August 2020, https://pdfs.semanticscholar.org/53d3/37f49c7d1ef98968ae5ed7e699096974db10.pdf

4.3.3.3 Analysability

Return to Maintainability

About

Analysability is defined by the IEEE glossary of Software Engineering as “the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to changes to the environment”.

One way to understand the Analysability of a system or a program is to understand the size of the system or program. As a rule of thumb, as the size of system or program increases, it becomes increasingly harder to successfully modify the software to correct faults, improve performance, or to adapt to changes in the operating environment. This Analysability can be performed any time while using either the Waterfall Model or the Agile Model during the early stages of analysis and design. As projects mature and use a larger code base, the models can be based directly on either the source code or Unified Modeling Language (UML) models created using reverse engineering. The metrics can also be used during both Greenfield or Brownfield deployments. Another way to examine Analysability could be to collect and use similar metrics on Distributed Computing systems where, instead of using classes (i.e., as in Object-Oriented Programming (OOP)), the number of Nodes, Endpoints and message types could be used. To address **Structural Complexity**, the connections between processes can be used.

Table 10: Metrics for Class complexity[200)]

Type of Metrics	Metric definition
Size Metrics	
Number of Classes (NC)	The total number of classes
Number of Attributes (NA)	The total number of attributes
Number of Methods (NM)	The total number of methods
Structural complexity Metrics	
Number of Associations (NAssoc)	The total number of associations
Number of Aggregations (NAgg)	The total number of aggregation relationships within a class diagram (each whole-part pair in an aggregation relationship)
Number of Dependencies (NDep)	The total number of dependency relationships
Number of Generalisations (NGen)	The total number of generalization relationships within a class diagram (each parent-child pair in a generalization relationship)
Number of Generalization hierarchies (NGenH)	The total number of generalization hierarchies in a class diagram
Maximum DIT (MaxDIT)	It is the maximum DIT value obtained for each class of the class diagram. The DIT value for a class within a generalization hierarchy is the longest path from the class to the root of the hierarchy

Type of Metrics	Metric definition
Size Metrics	
Maximum HAgg (MaxHAgg)	It is the maximum HAgg value obtained for each class of the class diagram. The HAgg value for a class within an aggregation hierarchy is the longest path from the class to the leaves

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

200)

Marcela Genero, Mario Piattini and Ronda de Calatrava, Empirical validation of measures for class diagram structural complexity through controlled experiments, Accessed 4 August 2020, https://pdfs.semanticscholar.org/dd52/5d80c1f370258e56cd956bcb903706c216dc.pdf

4.3.3.4 Modifiability

Return to Maintainability

About

Return to Top

Modifiability is characteristic of a system to successfully support:

- Extensibility
- Updateability
- Deletability

In other words, it is a system or products ability and receptiveness to adapt to future and often unexpected changes. Planning for Modifiability is a bit like looking into a crystal ball about the product, its place within the ecosystem and its ability to adapt to changes in the environment. An easy way to consider Modifiability of a system or product is to ask the following series of questions suggested by McGovern et al. [201]:

Table 11: Questions to consider when assessing Modifiability.

Questions proposed by McGovern et al.	Considerations
How often is it expected that a system change will be required?	This question is trying to understand the maturity (see 4.3.2.1 Maturity) associated with the system or product. It is not just about the product, but the maturity of the domain knowledge surrounding the system or product. For example, the accounting domain has been around for thousands of years and is well documented and understood while the use of applications to address Blockchains is less than a decade old.
What is the usual extent of the change?	This question also relates to maturity (see 4.3.2.1 Maturity) of the domain associated with the system or product. It also has to do with how conservative the attitude is towards changes. For example, changes made to an end-user entertainment application such as TikTok are easily tolerated while changes made to accounting records, which can adversely effect an individual's wealth, are frowned upon.

Questions proposed by McGovern et al.	Considerations
Who is expected to make the changes?	If changes are made by a single individual with minimal review and testing, the ability to modify the system is high, but the risk of failure is increased. Modifiability must consider these risks in the context of the domain. In essence, the better the governance over changes, the higher the probability of success (see 3 Governance). An individual can be extremely disciplined and positive when it comes to adapting to changes, while organizations can be sloppy. So, the maturity of the domain is important and the organization (even if it's a single person) is important. See ISO 90003-2018 and Capability Maturity Model Integration (CMMI).
Is it necessary for the system to use current platform versions?	This addresses the End-of-life (EoL) issues associated with any product (see 4.3.5 Manageability and 4.3.1.3 Replaceability). As the system ages, more and more EoL problems arise. As more Commercial Off-The-Shelf (COTS), Government Off-The-Shelf (GOTS), Modified Off-The-Shelf (MOTS), and NATO Off-The-Shelf (NOTS) products are used by the system and the longer the system exists the risk to the system increases because each subsystem, component or modular needs to be managed. As a case in point, in mid-2020, roughly 200 million PCs worldwide will still be running older Windows versions, mostly Windows 7[https://www.zdnet.com/article/how-many-pcs-are-still-running-windows-7-today/]]. Many of these are probably not modifiable any more.

Zarnekow et al. [202)] did a detailed study of 30 applications in 2015 and found the following time and cost characteristics and that over half (55%) of the cost of the projects can be attributed to maintenance and support. These findings underline the importance of Modifiability. All too often when a project gets started, too many problems are “kicked down the road” with the idea that “we'll cross the bridge when we get there”.

Table 12: Summary of time and cost characteristics found in 30 projects (Zarnekow et al.)

			Time			Actual Cost (in Mill of Euro)					
	# of Users	# of Transactions/yr	Total	Init Dev	Prod	Total Cost	Planning	Init Dev	More Dev	Prod	Shutdown
Minimum	160	23,900	2.0	0.3	0.4	0.50	0.01	0.13	0.00	0.14	0.00
Maximum	135,500	91,250,000,000	16.4	3.0	12.4	137.33	50.00	85.00	38.00	117.8	0.13
Average			5.6	1.7	3.1	35.74	2.97	11.57	5.52	15.6	0.08
% of total			100.0%	30.3%	55.3%	100.00%	8.31%	32.37%	15.44%	43.65%	0.2%

Another paper published by Björklund [203)] reported the cost of software maintenance as 67%.

Table 13: The Cost of Software Maintenance for one project

Lifecycle Phase	Percent of Cos
Requirements Definition	3%
Preliminary Design	3%
Detailed Design	5%
Implementation	7%
Testing	15%

Lifecycle Phase	Percent of Cos
Maintenance	67%

* **Note:**

- Another study found at least 50% of the effort spent on maintenance
- Another study found between 65% and 75% on maintenance
- In embedded real-time systems, maintenance costs may be up to 4 times development costs

Regardless of the actual numbers for a system or a program, all the numbers point to one conclusion: the cost of maintenance is a major driver in the total cost of ownership for systems or projects. Much of the maintenance cost for many projects can be traced back to not planning or considering Modifiability throughout the project lifecycle, especially early on. Modifiability is closely correlated to creating layered, modular and loosely coupled systems or programs. Fortunately, there are tools which can analyze a system or a program during all its phases (see 4.3.3.1 Modularity and 4.3.5 Manageability).

Layering involves separating the system or programs based on technical responsibilities, usually using an N-Tier Architecture. Generally, these tiers are referred to as the *presentation tier*, *processing tier* and *data management tier*. Often the tiers are both logically and physically separated, with each tier running on its own dedicated platforms. In a distributed system, the tiers do not follow the client-server architecture but use a Peer to Peer (P2P) architecture. However, the peers can be categorized as fulfilling presentation, processing, and data management functionality.

In addition, systems or programs that are declarative and configurable are more modifiable, especially if the configuration describes the details of connectivity between the modules (or peers). In other words, these descriptions provide the context and should address the 5-Ws of **who**, **what**, **when**, **where** and **why**; as well as, the **how**. For example:

Table 14: How the 5-Ws and H can be used to help ensure Modifiability

W	context
who	Who can access the module (peer) including privileges: developer, a business user, an analyst, or some combination of these is responsible
what	What is the module (peer) name, version, download URI
when	When can the module (peer) be accessed: event, calendar, time, etc.
where	Where can the module (peer) be found: paths, endpoints, etc.
why	Why is the module (peer) defined: documentation, rules, filters, etc.
how	How is the module (peer) accessed: Library, RESTFul, Remote Procedure Call (RPC), DDS, Message queue, etc.

Expectations of frequent changes driven by business-related changes can be more modifiable if the rules are not codified into software but stored as machine readable rules that can be interpreted at run time using, for example, rule engines. However, the downside of a data-rule driven system is that changes in data or rules can lead to crashes and adverse impacts to stability (see 4.3.5 Manageability).

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

201)

James McGovern, Sameer Tyagi, Michael E. Stevens and Sunil Mathew, Java Web Services Architecture, 2003, ISBN 978-1-55860-900-6, Accessed 5 August 2020
https://www.sciencedirect.com/book/9781558609006/java-web-services-architecture

202)

Ruediger Zarnekow and Walter Brenner, Distribution of Cost Over the Application Lifecycle - A Multi-Case Study, University of St. Gallen, 22 July 2015, Accessed 5 August 2020, Researchgate

203)

Carl Björklund, App Maintenance Cost Can Be Three Times Higher than Development Cost, 15 April 2019, Accessed 5 August 2020,
https://www.econnectivity.se/app-maintenance-cost-can-be-three-times-higher-than-development-cost/

4.3.3.5 Testability

Return to Maintainability

About

Return to Top

Testability, *Testable*, *Testing* and *Test* are not synonyms for each other. Just because a system or program is undergoing *testing* using various *tests* does not necessarily mean that the system or program is actually *Testable*. The following table provides definitions for each of these four terms, associates each with the appropriate Structured Assurance Case, as well as, level in the Cognitive Model's Science and Knowledge Management DIKW (Data, Information, Knowledge and Wisdom) pyramid.

DIKW pyramid level	Structured Assurance Case	Term	Description

DIKW pyramid level	Structured Assurance Case	Term	Description
Understanding	Software Assurance (SwA)	Testability	**Testability** is about documenting the functionality and requirements for a system or program and verifying that these requirements will be or have been met. Functional requirements are generally not a problem since most of these are directly measurable or observable. Functional requirements are often directly measured or observed: a data field needing to be provided, a relationship existing between two pieces of data, or a relationship that is one-to-many or a many-to-many. For example, every person must have a unique company ID number but they may have multiple phone numbers and also belong to multiple organizations. Other functional requirements are not so definite, but expressed in terms of a range of acceptable values. For example, a Graphical User Interface (GUI) will respond in less than 5 seconds or the heart pulse rate is between 35 to 200 beats per minute. In contrast, non-functional requirements are generally more abstract: they relate to the quality of the system or program being delivered (i.e., portable, reliable, maintainable, securable, scalable, etc.) and are usually not directly measurable or observable but must be inferred from characteristics found in the delivered system's or product's architecture, design and implementation. These kinds of requirements require ways to characterize assurance and are specified in terms of claims (i.e., the system has a High Availability), sub-claims, and arguments (i.e., Availability can be predicted using a Mean Time To Repair (MTTR) of 5 minutes, 15 seconds or less of downtime in a year for all components). These kinds of requirements are generally specified in Performance or Functional Specifications. These specifications tend to focus on hardware specifications; however, performance specifications can also capture non-functional metrics. • **Note** Testability metrics are not limited to operational systems or programs but can also take advantage of system or program level artifacts that describe architecture, design, discussion papers, outside references, software and executables. Here is a list of some common “mistakes” found in requirement documents[204] that can make it difficult to determine if requirements are actually “testable”: • **Noise:** Text containing no information relevant to any aspect of the problem. For example, a requirement on a standalone application that does not need access to the Ethernet ◦ The system shall conform to IPV6 ... • **Silence:** A feature not covered by any text within the Requirements documents or specifications • **Over-specification:** Description of the solution rather than the problem. For example, ◦ The distributed system must use blockchain. (blockchain is one of many distributed technologies used by Cryptocurrencies) ◦ The system must use a checkbox to select the appropriate option • **Contradictory:** Mutually incompatible descriptions of the same feature. For example, ◦ The system shall not record any personal information ◦ The system shall record all transactions and parties participating in the transaction • **Ambiguity:** Text that can be interpreted more than one way ◦ The system shall support real-time operations (what is real-time?) • **Forward reference:** Referring to a feature not yet described ◦ The system shall publish all information on a topic (but topic has not been officially defined yet) • **Wishful thinking:** Defining a feature that can’t be validated ◦ The system shall initialize all values with intelligent default choices. (what's the metric for “intelligent”?) • **Weak phrases:** Causing uncertainty (“adequate”, “usually”, “etc.”) For example, ◦ *When possible*, the systems shall ... ◦ The system shall collect their data (whose data?) • **Jigsaw puzzles:** Requirements distributed across a document and then cross-referenced • **Duckspeak:** Requirements included merely to conform to standards that have no or little relationship to the problem at hand. Perhaps required as part of a boilerplate. • **Terminology invention:** “user input/presentation function”; “airplane reservation data validation function”. For example, ◦ The system shall use a double blind logged journal entry (huh, what is that?) • **Putting the onus on developers and testers:** to guess what the requirements really are. ◦ The system shall use a right-handed approach when presenting data

DIKW pyramid level	Structured Assurance Case	Term	Description
Knowledge	Claim	Testable	A **Testable** attribute of a system or program is a functional or nonfunctional requirement that may be testable or not. Some requirements can be directly tested for by running specific tests (i.e., Unit Testing, integration testing, etc.) using test plans that exercise a portion of the system or program software responsible for providing specific functionality. For example, the system is supposed to offer the choice of none, one, and many. Another example might be that when an option is selected, a message is sent out over the network. By design, some requirements are not directly testable, i.e, are untestable. Often, these requirements are met through the use of mathematical proofs or demonstrations. For example, the generation of a Universally Unique IDentifier (UUID) can not be tested directly; instead, the algorithm used to generate them must provide an explanation and proof that no two sets of conditions will produce the same UUID. Often there is a risk of generating the same UUID, but the chances of the same UUID being used in identical domains or environments is even smaller. Another example would be the reCAPTCHA, which shows a series of photos and asks the user to identify the ones with green peas in them. The order of the photos and the thing it is asking you to identify are randomly assigned.
Information	Argument	Testing	Testing is a process that generally involves the execution of the system or program under scripted, controlled situations. The scripts can be human instructions in documents or they can be captured in text files that a testing engine uses to drive the software. Sometimes, a Unit Test is used to test individual modules before they are integrated into the system or program. Below are the various requirement conformity checks that can be performed to verify functional requirements: 1. Unit Testing 2. Integration Testing 3. End-to-End Testing (E2E Testing) 4. Smoke Testing 5. Sanity Testing 6. Regression Testing 7. Acceptance Testing 8. White Box Testing 9. Black Box Testing 10. Interface Testing 11. Interoperability Testing
Data	Evidence	Test	*Test* refers to the act of collecting the evidence used to support arguments, sub-claims and claims made about the system or program. There is not a one-to-one relationship between a Test, an Argument, a Sub-Claim or a Claim. Instead, one piece of data can support multiple Arguments and an Argument can support multiple Sub-Claims or Claims. That is why it is so important to have a Structured Assurance Case Model.

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

204)

Achieving Requirements Testability, ProlificsTesting, 10 October 2018 Accessed on 9 August 2020 https://www.prolifics-testing.com/news/achieving-requirements-testability

4.3.4 Securability

Return to Non-Functional Requirements

About

Security is not a single “thing” that can be added to a system. To be truly secure, the entire End-to-End Solution (E2ES) needs to be secure and needs to be considered during the entire System Lifecycle. As shown in Figure 43, a layered approach is used to help isolate the security needs. Each layer represents a portion of the Information Technology (IT) stack, including the people who use and have access to the IT stack.

Figure 43: The layers of security.

Table 15: Definitions for layers of security

Layer	Description
Physical Security	The physical security is concerned with preventing physical harm to the Computing Platform (e.g., theft, fire, flooding, etc.), as well as, preventing access to the physical platform via “back doors” thereby allowing breaches by potentially malicious actors (e.g., using pluggable USB drives, adding wire sniffers to the network, or the internal threat posed by employees with access).
Data Security	Data security ensures that Data at Rest, Data in Motion, or Data in Use remains intact (i.e., completeness, accuracy and consistency). For example, allowing incomplete data to be stored (i.e., date of a transaction, or *authorized by* fields). Roundoff Error can also affect the accuracy of the data. Modifying a bank account balance introduces inconsistencies that can be detected.

Layer	Description
Network Security	Network security issues are generally the result of unaddressed network vulnerabilities. There are three main network categories for vulnerabilities: software, hardware, or organizational processes. Software and hardware that are not kept current are subject to malicious attacks by merely exploiting known vulnerabilities. Another issue for networks are social engineering attacks where people violate hardware and software protection policy and procedures to compromise the data. The first line of defense for networks is the use of a Hardware Firewall, which predominately protects the Network Nodes inside a Local Area Network (LAN) from external Nodes on the Internet. Another common tool is the use of Networking Access Control Lists (ACLs).
Platform Security	Platform security involves an attack on a Computing Platform by the introduction of malicious software, the modification of access controls or configuration data, or having incorrectly configured settings (i.e., Software Firewall, Access Control List (ACL), Full-Disk Encryption (FDE)). Many platforms can require authorization of peripheral functions such as geo-location services, Bluetooth, TCP/IP networks, radio networks, and segmented portions of the disk storage such as photos.
Application Security	Application Security features include authentication of users using multi-factor authentication such as user id and passwords, asking additional questions only the user knows the answer to, use of a One-Time PIN (OTP) to a known device, a fingerprint or face recognition. Application Security also includes authorization that maps the user's identity with a list of applications the user can access and even the privileges the user has within the application (read/write/delete, etc). Sometimes an application can encrypt information as it moves between the components of the system (CPU, Memory, Disk, network, etc.). Another key aspect is the secure logging of activities occurring within an application (e.g., user granted access, user deletes information, user updates information, etc.)
Culture Security	One of the biggest threats to any system or program is the internal threat caused by inadvertent or overt actions taken by the people within the organization. To overcome these threats, there needs to be Cultural Security (also known as cybersecurity) that seeks to change the the mindset of the authorized users. Some basic examples are not using `password` as a password, not writing the passwords on paper, having a separate password for every person, changing the password every so many days, keeping ACLs up to date and reflecting the current roles and responsibilities of the users. Sometimes, it comes down to just observing the behavior of others[205)]

ISO/IEC 25010 defines Security as the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization. This characteristic is composed of the following sub-characteristics[206)]:

- 4.3.4.1 Confidentiality
- 4.3.4.2 Data Integrity
- 4.3.4.3 Non-Repudiation
- 4.3.4.4 Authenticity
- 4.3.4.5 Accountability

See DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

205)

Cyber Security Culture in organizations, European Union Agency for Network and Information Security (ENISA), November 2017, Accessed on 13 August 2020, https://www.enisa.europa.eu/publications/cyber-security-culture-in-organisations

206)

ISO25000 Software and Data Quality ISO/IEC 25010, 2011, Accessed 13 August 2020, https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

4.3.4.1 Confidentiality

Return to Securability

About

Confidentiality is usually covered by the use of a Confidentiality Agreement or Non-Disclosure Agreement (NDA), which defines a set of rules or a promise limiting access or places restrictions on certain types of information. Areas that have legal agreements covering confidentiality are:

- Legal Confidentiality
- Medical Confidentiality
- Clinical and Counseling Psychology
- Commercial Confidentiality
- Banking Confidentiality
- Public Policy Concerns
- Religious Confidentiality

As a rule of thumb, it is best to treat all Personal Identifiable Information (PII) as confidential and to secure it (i.e., require authentication both to access the data and log access to the data).

The US National Institute of Standards and Technology (NIST) describe the kinds of data that should be treated as PII[207] as:

- Name, such as full name, maiden name, mother's maiden name, or alias
- Personal identification number, such as:
  - Social security number (SSN),
  - Passport number,
  - Driver's license number,
  - Taxpayer identification number,
  - Patient identification number,
  - Financial account number, and
  - Credit card number

NIST also identifies information which potentially can be used to identify people:

- Address information, such as street address or email address
- Asset information, such as Internet Protocol (IP) or Media Access Control (MAC) address or other host-specific persistent static identifier that consistently links to a particular person or small, well-defined group of people
- Telephone numbers, including mobile, business, and personal numbers
- Personal characteristics, including photographic image (especially of face or other distinguishing characteristic), x-rays, fingerprints, or other Biometric image or template data (e.g., retina scan, voice signature, facial geometry)
- Information identifying personally owned property, such as vehicle registration number or title

number and related information
- Information about an individual that is linked or linkable to one of the above (e.g., date of birth, place of birth, race, religion, weight, activities, geographical indicators, employment information, medical information, education information, financial information).

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

207)

Erika McCallister Tim Grance and Karen Scarfone, Guide to Protecting the Confidentiality of Personally Identifiable Information (PII), Special Publication 800-122, April 2010, Accessed on 13 August 2020, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-122.pdf

4.3.4.2 Data Integrity

Return to Securability

About

Return to Top

Data Integrity is the completeness, accuracy and consistency of data throughout the entire data lifecycle of the data as well as when the Data is at Rest, Data in Motion and Data in Use.[208)]

Figure 44 shows the five levels Automation Pyramid and the functionality usually associated with each one. There is Data at Rest at each level of the pyramid. As the data transitions up and down from level to level within the pyramid, the Data is in Motion. Within each level, the data will most likely be accessed therefore, the Data is in Use.

Table 16: The five levels of the Automation Pyramid.

Automation Level	Description
Field Level	The Field Level where products are produced. In other words, this is where the physical work plus monitoring occur. Electric motors, hydraulic and pneumatic actuators to move machinery, proximity switches used to detect that movement or certain materials, photoelectric switches that detect similar things will all play a part in the field level.
Control Level	The Control Level uses the control devices to “run” the devices in the Field Level. The Control Devices make decisions based on information provided by sensors, switches, and other input devices to complete the programmed task.
Supervisory Level	The Supervisory Control and Data Acquisition (SCADA) is combines the Field and Control Levels to provide oversight from a single location. This is usually accomplished using Graphical User Interface (GUI), or Human-machine interface (HMI), to remotely control operations. For example, water plants often employ this technology to control remote water pumps.
Planning Level	The Planning Level uses Manufacturing Execution System (MES) to monitor the entire manufacturing process. For example, in a factory to plan for everything from raw materials to the finished products. This allows management to visualize the current state of operations and aids them in making decisions and adjust raw material orders or shipment plans based on real data received from Supervisory, Control and Field Levels.
Management Level	The management level uses the companies integrated management system such as as Enterprise Resource Planning (ERP). Corporate management visualize and control operations. This level allows the businesses monitor all levels (i.e., manufacturing, to sales, to purchasing, to finance and payroll). The integration of an ERP promotes efficiency and transparency within a company by helping to communicate the levels.

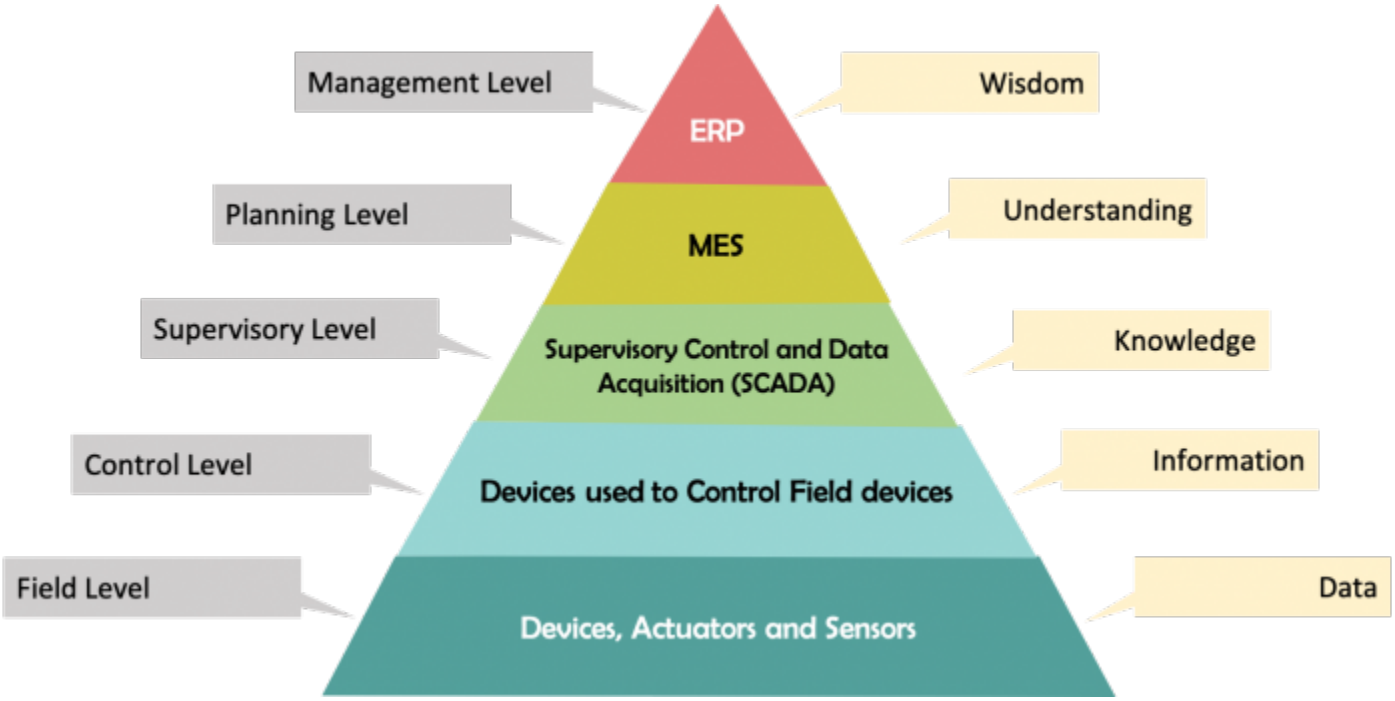


Figure 44: Automation Pyramid

At each level, the Data at Rest can be categorized as one of two kinds of data integrity both of which are a collection of processes and methods intended to enforce Data Integrity.

- Physical Integrity protects data's wholeness and accuracy as it's being used. When expected or unexpected down times occur (i.e., natural disasters strike, power goes out, or hackers disrupt database functions) physical integrity is compromised. Some other issues which can compromise the integrity of the data are Human error, storage erosion, or a host of other issues making it impossible for data processing managers, system programmers, applications programmers, and internal auditors to obtain accurate data.
- Logical Integrity keeps data unchanged as it is accessed. Logical integrity protects data from some of the same issues as Physical Integrity (i.e., human error and hackers as well) but in different ways. There are four types of logical integrity.
  1 - Entity Integrity - supports unique values that identify any particular data entry and that the key is not null.
  2 - Referential Integrity - ensures that references to other data entries exist.
  3 - Data Integrity - ensures that domain rules (i.e., data restrictions) are enforced for the data within the Data Structure. For example, minimum, maximum, number of decimals, nullable, etc. are enforced.
  4 - User Defined Integrity - ensures that business rules are enforced. For example, if a value is set, another value must also be set (unset); if a value exceeds a threshold, a notice must be sent.

Note: Data integrity is not Data Security and not Data Quality.

- Data Security defines the steps taken to prevent corruption from within and from outside attacks by people or processes.
- Data Integrity defines the steps taken to keep the data intact and accurate from internal people and processes and for the entirety of the data's existence.

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA (look at spec if DDSF)

208)

What is Data Integrity, Accessed 8 July 2020, https://www.talend.com/resources/what-is-data-integrity/

4.3.4.3 Non-Repudiation

Return to Securability

About

Non-Repudiation [209] means that it is not possible to repudiate (i.e., deny) that an action has been taken. For example, the signed contract witnessed by two people could not be repudiated. In other words, the contract now has Non-Repudiation.

Non-Repudiation is about providing assurance using evidence that an action has been done. For example, a data sender is provided evidence (i.e., proof) of delivery while the receiver is provided evidence (i.e., proof) of the sender's identity. As a consequence, neither the sender or the receiver can deny having processed the data.

Non-Repudiation applies to more than just sending data between two parties. It can be applied to any action or activity. For example, by digitally signing an email, the receiver has evidence (i.e., proof) that the email is from the entity that signed the email. In other words, it is not possible to repudiate that the email came from the entity that digitally signed the email. Another example is the use of identities in configuration management systems. The change (i.e., transformation) was recorded in a log along with the identity of the individual that made the change. In this way, all changes made to the configuration have Non-Repudiation.[210]

There is a lot of overlap in Non-Repudiation and Access Control. During access to a controlled resource, the identity of the entity trying to access the resource is verified against an Access Control List (ACL). When access is allowed or denied, an entry is made into a log. Once the entry is made, the access has Non-Repudiation. In other words, once an Access Control Function is executed, there is generally sufficient evidence to for Non-Repudiation of access to the controlled resource.

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

209)
Non-Repudiation, Computer Security Resource Center (CSRC) Accessed 14 August 2020, https://csrc.nist.gov/glossary/term/non_repudiation

210)
Evan Wheeler, Security Risk Management, 2011, Accessed 14 August 2020, https://www.sciencedirect.com/science/article/pii/B9781597496155000074

4.3.4.4 Authenticity

Return to Securability

About

Authenticity is a property indicating the source and origin of the information[211)]. The process of authenticating a source starts when an entity (i.e., user, remote process, intelligent agent, etc.) attempts to access resources on a Computing Platform. The entity proves their identity in order to gain access rights. For example, traditionally when logging into a computer, users use a Single-Factor Authentication (SFA) by providing a `usernames` and `passwords` to confirm their identity to allow future authentication for access to resources. However, this `usernames` and `passwords` login combination is no longer considered secure enough, especially if there are poor CyberSecurity Culture (CSC). As a consequence, many systems have added Two-Factor Authentication (2FA) that require Biometrics (i.e., facial recognition, fingerprints, etc) or One-Time PIN (OTP). These 2FA methods generally require the user to be physically present to successfully login.

Public Key Infrastructure (PKI) is generally used to connect servers and clients or even nodes that have no user present to perform the SFA or the 2FA methods of authentication. It is often incorrectly used as a synonym for Encryption. Encryption is an algorithm used to encrypt and decrypt data. PKI is an infrastructure built around asymmetric encryption with two keys: public and private. PKI is used extensively to securely transfer data between Network Nodes. In the PKI infrastructure, entities (i.e., AAA and BBB) exchange public keys. To exchange information, one entity (i.e., AAA) encrypt a document using the other entities (i.e., BBB) public key. Anyone can receive the document encrypted by AAA using BBB's public key, but it remains encrypted until BBB uses the private key in the PKI to decrypt the document.

PKI is the backbone of most of the major secure document exchange sites. Some examples are[212)]:

- Securing emails - Email Security (S/MIME Protocol)
- Securing web communications - Website Security
  - Hypertext Transport Protocol Secure (HTTPS)
  - Secure Sockets Layer (SSL)
  - Transport layer security (TLS)
- Secure Shell Protocol (SSH)
- Digitally signing software, applications or data
- Encrypting and decrypting data
- Smart Card authentication
- Subscriber Identity Module (SIM)

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

=-

211) Authenticity, Computer Security Resource Center (CSRC) Accessed 14 August 2020, https://csrc.nist.gov/glossary/term/authenticity

212) How Does PKI Work ?, Venafi, Accessed 14 August 2020, https://www.venafi.com/education-center/pki/how-does-pki-work

4.3.4.5 Accountability

Return to Securability

About

Accountability is the principle holding an individual entrusted to safeguard and control key components of a system or program (i.e., equipment, keying material, and information) answerable to proper authority for the loss or misuse of that component.[213)]

Accountability is a security goal outlined in ISO/IEC 24010[214)] requiring the actions of an entity to be traced uniquely to that entity. Accountability directly supports Non-Repudiation. It also provides a deterrence, helps with fault isolation, and is useful in intrusion detection and prevention. In many cases, it is a key source of evidence use in and After Action Review (AAR) and can ultimately, if needed, support legal actions.

Accountability is part of an information security plan. The plan should enumerate every individual working with an information system and define specific responsibilities (i.e., tasks) regarding information assurance. Each task needs to me measurable and be subject to oversight by individuals higher up in the command chain.

One example, might be an information security requirement that holds all employees responsible for not installing software from any source other than a company-owned repository (i.e., a task). The individual responsible for upholding the requirement might perform a periodic check of corporate assets to determine that the policy is being followed. Any violations in the requirement would hold the individual responsible for performing the task. The plan makes the individuals aware of the tasks expected of them, and guide continual improvement in compliance with the requirement.[215)]

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

213)
Accountability, Computer Security Resource Center (CSRC) Accessed 14 August 2020, https://csrc.nist.gov/glossary/term/accountability

214)
Accessed 15 August 2020, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&start=6

215)
Computer-Security-Glossary.org, Accessed 15 August 2020,

https://www.computer-security-glossary.org/accountability.html

4.3.5 Manageability

Return to Non-Functional Requirements

Manageability is most important during the second half of System Lifecycle phases (i.e. operation, maintenance, support). Manageability can greatly influence the recurring costs and can increase the chances of a failure. Often a system that is hard to manage is described as fragile since the smallest change can have dire consequences on the system's functionality.

> *Manageability directly influences a system's reliability, availability, security, and safety; therefore, it is a key ingredient of system dependability.*
>
> *Just like security and safety, manageability is generally hard to retrofit in complex systems—it is always easier to build it in from day one. However, in the absence of means to measure manageability and quantify the various tradeoffs, it is difficult to get the design right. We proposed a manageability metric that combines management workloads and weightings based on real world studies with direct measurement of the number of steps involved in management tasks and their duration.* [216)]

DIDO Specifics

Return to the Top

> To be added/expanded in future revisions of the DIDO RA

[216)]
Toward Quantifying System Manageability, George Cadea, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, Accessed 20 July 2020, https://www.usenix.org/legacy/event/hotdep08/tech/full_papers/candea/candea_html/index.html

4.3.5.1 Types of Manageability Functions

Return to the Manageability

About

Return to Top

NI (formerly National Instruments) defines four basic manageability functions [217)] described in the following Table:

Table 17: The kinds of Management Functions needed for Manageability

Kinds of Management	Description
Health Monitoring, Logging, and Alerting	During operations a key part of managing the system is the collection of metrics about the operations that indicate the health of the system. • Monitoring can include usage of Central Processing Unit (CPU), memory, energy, cache, heat, on-line and off-line storage, network connectivity, network loads of particular computers. In a distributed system, this also includes the metrics associated with the connectivity of the nodes that participate in the distributed system (i.e., the latency, lag, error rates, upload and download speeds, etc). • Information about the overall operations need to be handled. For example, data used to Trace, and Debug; provide Informational context; and alert to warnings, errors and fatal situations.
Configuration and Control	During the startup and sometimes during operations, the system needs to obtain configuration details and provide control over the systems as a whole or to the individual components. This becomes even more important in a system that is distributed on a system of nodes. [1)] • Typical configuration management data are Internet Protocol address (IP address), network ports, environment paths to files, and maximum usage limits for things like CPU and disk space. • Typical commands are start, pause, resume and stop. Sometimes abort is also used to capture detailed dump files that can be used for analysis.

Kinds of Management	Description
Deployment and Updates	During operations, • There is sometimes a need for the efficient and sometimes automatic deployment of system resources such as web servers, application servers, database management systems, backups etc. This is useful when systems are made as a collection of other systems but is critical when managing distributed systems. • In addition, this kind of management needed during first-time installation of systems (i.e., wizards) but also in deployed systems for applying patches, performance, and feature updates.
Asset Discovery and Inventory	In remote systems or especially in distributed systems, an accurate and timely discovery of the system or component needs to be made. • In truly distributed systems, this discovery needs to be dynamic and should easily adapt to new locations. In a distributed system this must also support the dynamic creation and destruction of nodes in the system network. • In complex systems deployed on a single computer there is a dependency on other components within the node. An inventory needs to be made of the components required as included in a package management description (i.e., manifest). • In a distributed system running on distributed nodes there is a need to understand the resources dedicated to the system. Having an automated inventory helps with asset management, containing cost controls, and scheduling maintenance or replacement of systems.

[1)] **Note:** The following example of how configuarions can become a manageability issue is from Candea [1)] *- The greatest manageability challenge is posed by stateful systems (e.g., databases, filesystems). By contrast, stateless applications (e.g., Web servers) require little configuration, can be scaled through mere replication, and are reboot-friendly. While one administrator can manage 100s to 1000s of Web servers, it takes approximately one administrator for each TB of data in a database. The number of knobs on stateful systems is overwhelming: the Oracle DBMS has 220 initialization parameters and 1,477 tables of system parameters, while its "Administrator's Guide" is 875 pages long.*

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

[217)]

What is Manageability?, 5 May 2019, National Instruments (NI), Accessed 25 July 2020, https://www.ni.com/en-us/innovations/white-papers/13/what-is-manageability-.html

4.3.5.2 Manageability Costs

Return to the Manageability

About

> *It is no longer necessary to argue that managing enterprise computing systems is complex and time-consuming, or that the cost of managing Information Technology (IT) infrastructures far exceeds the hardware and software costs—numbers speak for themselves: IT operations account for 50%-80% of today's IT budgets, amounting to tens of billions of dollars yearly. Besides the bottom line, poor manageability also impacts reliability, availability, and security in harder-to-quantify ways. As human error becomes the dominant cause of unscheduled downtime, we desire systems that are easier, cheaper, and quicker to manage.* [1)]

The cost of managing systems is expensive and is getting more expensive. This is particularly true when using servers that are aging. The cost of support and administration of a new serer is about `$4,200`/year; by year seven, the cost will rise to about `$17,000`/year. It is logical to try and decrease these costs through refreshing the hardware [218)].

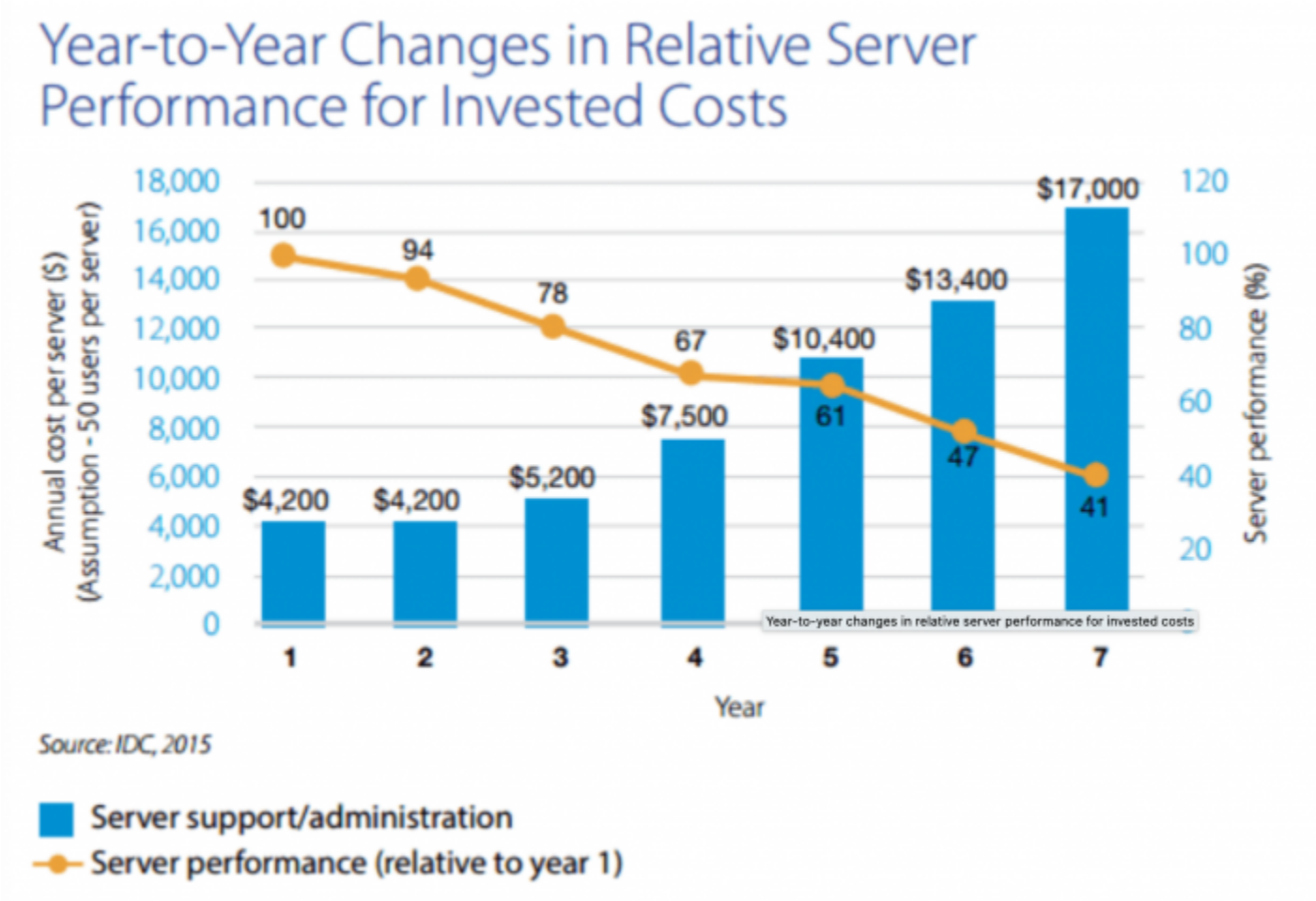


Figure 45: Year-to-Tear Changes in Relative Server Costs

Although these are the costs of owning and operating physical servers, the trend is relevant to almost all IT systems including software. The costs of managing the software is analogous to maintenance of the hardware but is probably worse since application software is dependent on many more layers of hardware and software. For example, a software system sitting on multiple Operating Systems (i.e, Linux, Unix, Windows, MacOS, IOS and Android), using a DataBase Management System (DBMS), a Web Server and using Java, Python, JavaScript, HTML, CSS, and supporting multiple browser (i.e., Firefox, Chrome and Safari) soon is overwhelmed by just keeping each system up-to-date and working. Distributed systems can compound that problem because each node in the System Network needs to be managed as well.

Granted, there are products and tools that can reduce the complexity of managing the various platforms easier and this is a big step towards helping manageability. But the overall goal is always to have systems that are easier, cheaper, and quicker to manage. There is another way to think about manageability called the Bathtub Curve. Although Figure 46 is targeted at hardware, there are similar things that effect software. Probably one of the biggest problems is that as the number of components in the system increase, the number of End-of-life (EoL) or End of Sales(EoS) issues need to be addressed. For example, a system that runs on an Operating System, has to worry about the EoL of the operating system. If it uses a database, it has a similar problem. These are generally not a problem in the early stags of the a system's lifcycle and hopefully of minimum problem during its “useful Life” phase. But as the system ages, more and more EoL problems arise. The more Commercial Off-The-Shelf (COTS), Government Off-The-Shelf (GOTS), Modified Off-The-Shelf (MOTS), and NATO Off-The-Shelf (NOTS) products used by the system, the longer the system exists there is an increase in risk to the system because each subsystem, component or modular need to be managed.

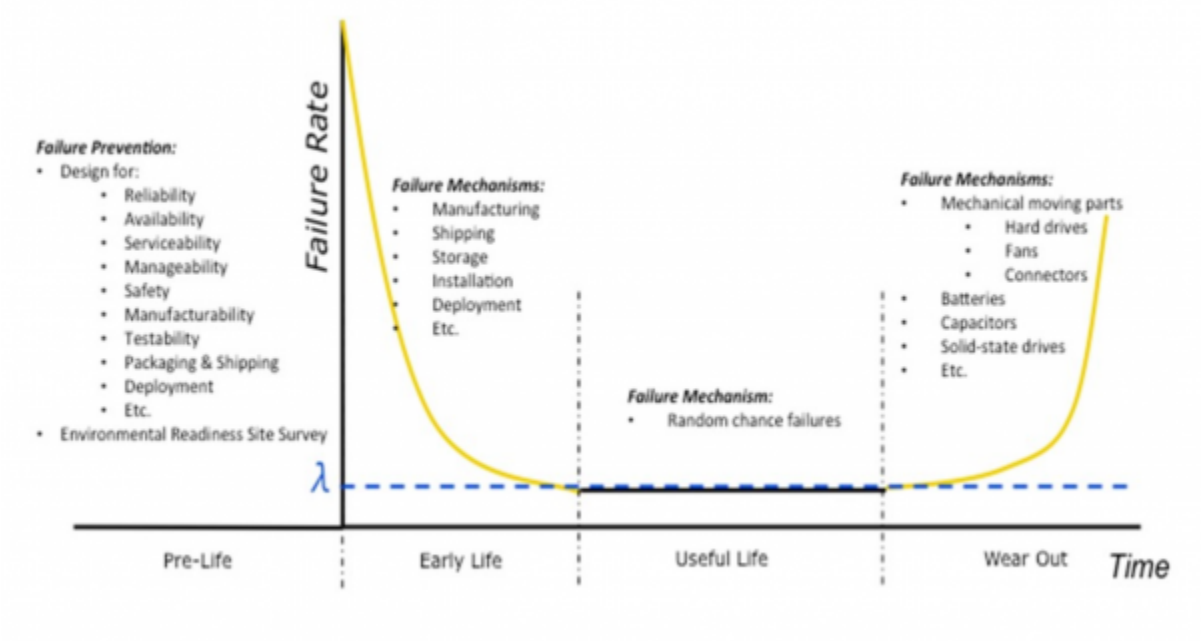


Figure 46: The Bathtub Curve [1)]

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

218)

The Hidden Costs of your aging IT Infrastructure, Barry Angell, 9 January 2017, accessed 15 July 2020, https://blog.juriba.com/the-hidden-costs-of-an-aging-it-infrastructure

4.3.5.3 System Manageability Issues

Return to the Manageability

About

Subsystem, Component and Module Lifecycle Issues

"Studies have shown the average software program lifespan over the last 20 years to be around 6-8 years. Longevity increases somewhat for larger programs, so for extremely large complex programs (i.e., over a million Lines of Code – LOC) the average climbs as high as 12-14 years."[219)] Obviously, there is not just the lifespan of the target system, but there are independent lifespans for each version for each subsystem, module or component that is developed externally. For example, the Windows Operating System (OS) first appeared in the mid 1980s with version 1.0.[220)] and the current version is 10. About every 10 years, Microsoft release another major release of Windows. [221)]. IPV4 was originally available in 1883. Windows 7 was release in October 2009 and IPV4 was the dominate Internet Protocol (IP). By 2012, IPV6 gained dominance[222)]. Therefore, if your system was released in 20010 using IPV4 and Windows 7, by 2012 the network protocol needed to be upgraded which can have cascading maintenance effects throughout your system. By 2015, Windows 10 was released again having a cascading effect on upgrades.

Figure 47 developed by the Industrial Internet Consortium[223)] illustrates some of the architectural components required in a generic Industrial Internet of Things (IIoT) system. When a system is deployed, each of these components needs to be managed. Each of these components has its own unique System Lifecycle which evolves independently of the target system.

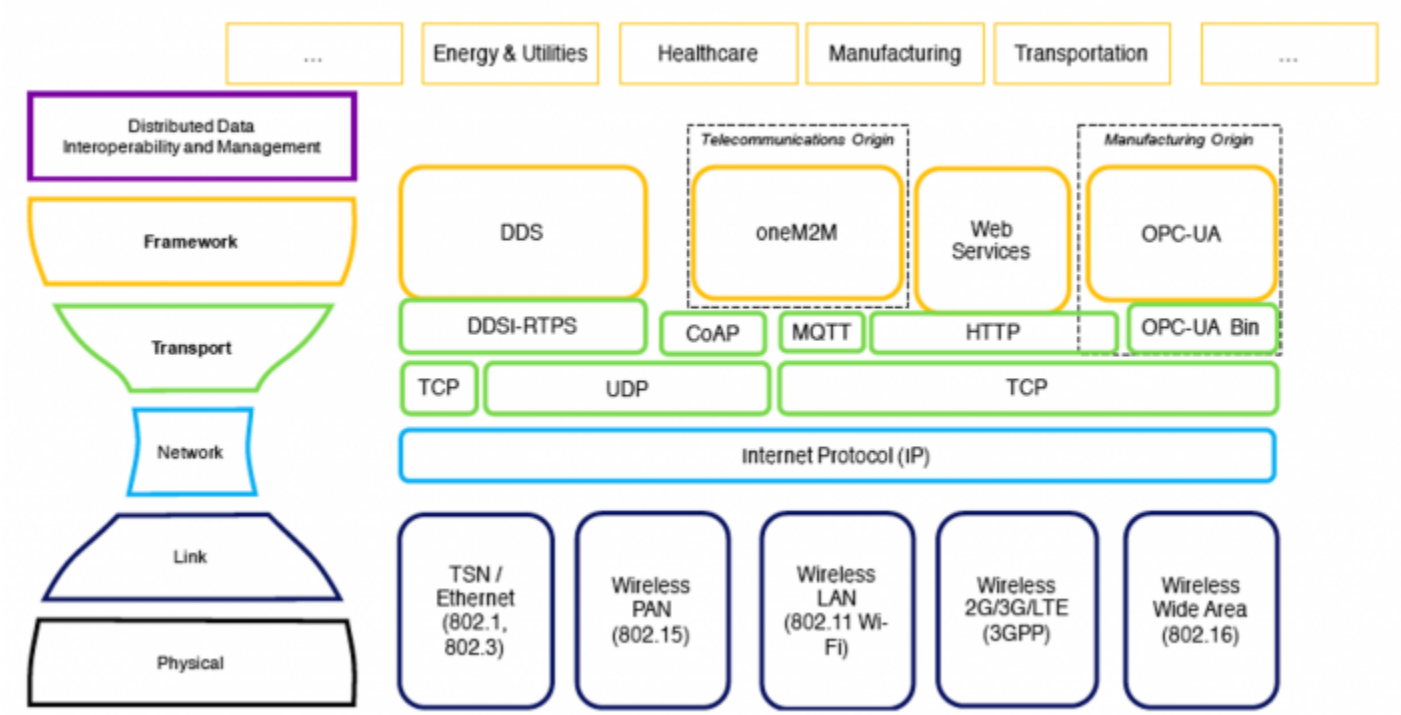


Figure 47: The Industrial Internet Consortium's Connectivity Framework layer provides the foundation for the interoperable transfer of common structured data across systems and domains [7)]

System Monitoring

Return to the Top

The fundamental requirement for manageability of any system is the collection of data about the system. This is often done with Monitoring Software specifically designed for this task. However, the task of monitoring complex, distributed systems is often difficult and beyond the scope of any particular product. The best place to start is to think of the monitoring in terms of layers. There are considered three layers[224)].

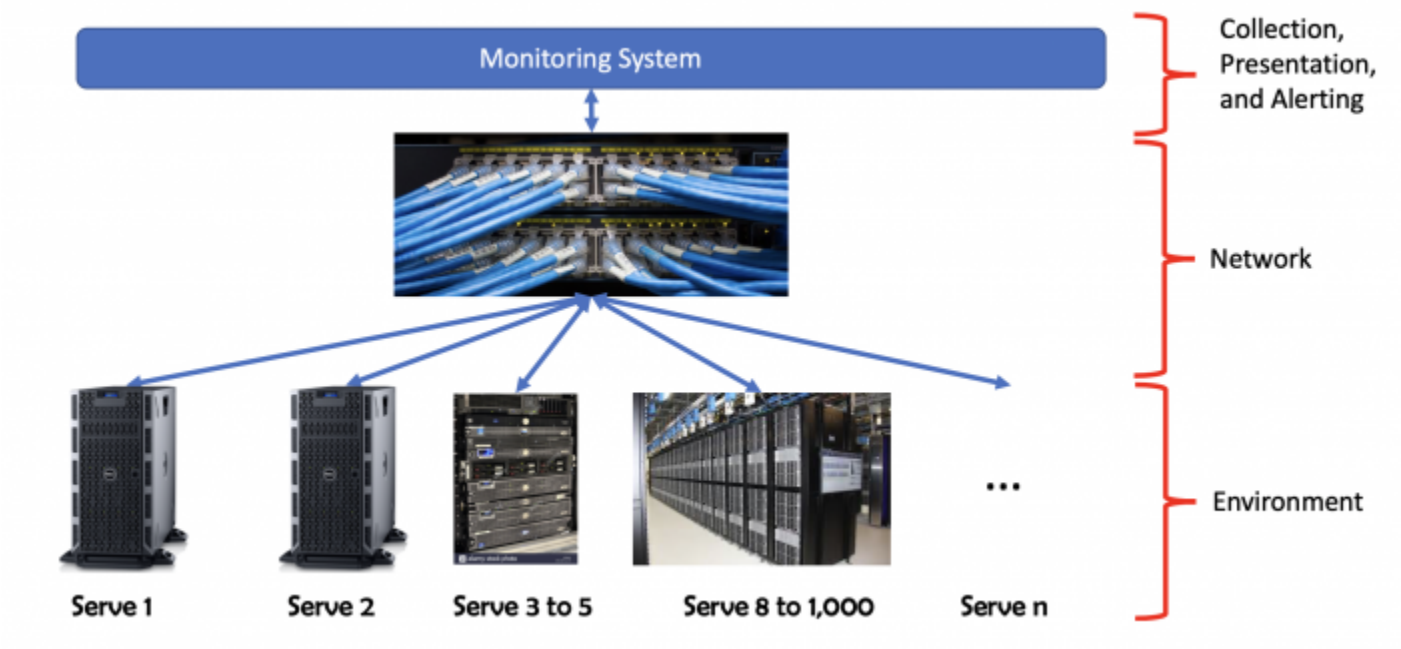


Figure 48: Three major monitoring layers representing sample design of monitoring solution

System Logging

Return to the Top

The Data Monitoring is the use of Data Logging to collect and store data for analysis to discover trends or record the events and actions of an application, a system, or a network. This allows for tracking of interactions using messages. Some of the commonly used logging levels are:

Table 18: Some common logging levels used in applications[225)]

Logging Level	Description
debug	Designates fine-grained informational events that are most useful to debug an application.
trace	Designates information about the flow of the execution or threads in an application.
info	Designates informational messages that highlight the progress of the application at coarse-grained level.
warn	Designates potentially harmful situations.
error	Designates error events that might still allow the application to continue running.
fatal	Designates very severe error events that will presumably lead the application to abort.

System Management

Return to the Top

Project Management Software is software used for project planning, scheduling, resource allocation and change management. It allows project managers (PMs), stakeholders and users to control costs and manage budgeting, quality management and documentation and also may be used as an administration system. Project management software is also used for collaboration and communication between project stakeholders. These tools can help throughout the system lifecycle, from requirement analysis through

sun-setting the system. However, in a distributed system, these tools can play a significant role in determining the health of each node, the network and the overall system of nodes.

Although the publication on Software Metrics for Predicting Maintainability [226] is a bit dated, many of the ideas of capturing metrics to measure Maintainability are still relevant today. By studying these metrics and understanding the formulas and the parameters used in the formulas, a lot of insight can be provided in explaining positive and negative Manageability traits.

- **Note:** Many of these metrics were originally targeting for Systems (or projects) that used the Waterfall Model which has been replaced for many systems (or projects) with Agile Models and DevOps. Many of these metrics can and should be applied to each Sprint to make sure the qualities of the system are maintainable and manageable. In a Distributed Application (ĐApp or DApp), word module can be replaced with subsystem or Node.

Table 19: Description of Mectrics for Maintainability of Systems (or Projects) [10]

5.1	UR	Un-referenced Requirements	The number of original requirements not referenced by a lower document in the documentation hierarchy.
5.2	NR	Non-Referencing Items	The number of items not referencing an original requirement.
5.3	M-MC	Module Coupling	A measure of the strength of the relationships between modules.
5.4	M-MS	Module Strength	A measure of how strongly related are the elements within a module.
5.5	HK-IF	Information Flow	A measure of the control flow and data flow between modules.
5.6	R-IF	Integrated Information Flow of Rombach	A measure of inter-module and intra-module complexity based on information flow.
5.7	KPL-IF	Information Flow by Kitchenham et al	A measure of inter-module complexity inspired from Henry & Kafura's information flow metric. Since Kitchenham et al experienced some difficulties in understanding the definition of flows provided by Henry & Kafura, they formulated a new set of definitions.
5.8	IF4	Information Flow Complexity	A measure of inter-module complexity based on information flow
5.9	CA-DC	Design Complexity of Card & Agresti	A measure of inter-module and intra-module complexity of a system based on fan-out, number of modules and input/output variables
5.10	COCO	Cocomo Inspired Metric	A selection of appropriate adjustment factors of the intermediate Cocomo metric
5.11	v(G)	Cyclomatic Complexity Number	The number of independent basic paths in a program.
5.12	knots		The number of crossing lines (unstructured `goto` statements) in a control flow
5.13	RLC	Relative Logical Complexity	The number of binary decisions divided by the number of statements
5.14	Vcd	Comments Volume of Declarations	Total number of characters found in the comments of the declaration section of a module. The declaration section comprises comments before the module heading up to the first executable statement of the module body.

5.15	Vcs	Comments Volume of Structures	Total number of characters in the comments found anywhere in the module except in the declaration section. The declaration section comprises comments before the module heading up to the first executable statement of the module body.
5.16	Ls	Average Length of Variable Names	Mean number of characters of all variables used in a module. Unused declared variables are not included.
5.17	LOC	Lines of Code	The number of lines in the source code excluding blank lines or comment lines.
5.18	E	Software Science Effort	An estimation of programming effort based on the number of operators and operands. It is a combination of other Software Science metrics.
5.19	DAR	Documentation Accuracy Ratio	A verification of the accuracy of the CEI Spec, RS and SDD with respect to the source code.
5.20	SCC	Source Code Consistency	The extent to which the source code contains uniform notation, terminology and symbology within itself.

Vendor Lock-in Issues

Return to the Top

A major management issue for many projects is the avoidance of Vendor Lock-In. Vendor lock-in restricts the options available to a system (or project) because of the dependency on sole-source proprietary technology, solution or service provided by a single vendor or vendor partner. This technique can be disabling and demoralizing because customers are effectively prevented from switching to alternate sources for the technology, solution or service making the customer-vendor relationship one sided.

Vendor Lock-In reduces the ability of manage costs over the life expectancy of the system (or project) or avoid risks when a vendor of a product ceases to maintain a critical component of the system (or project) or even when the product ceases to exist.

This can be partially mitigated through the use of Open Source Software (OSS), however, remember, just because something is OSS, does not mean there is not a vendor. Additionally, if the OSS software is deprecated or evolves in a divergent way from the requirements of the target system (or project), then the responsibility for the care and maintenance of the OSS has to be covered by the system (or project). However, there are OSS solutions which also adhere to standards such as the Data Distribution Service (DDS) vendor Object Computing Incorporated (OCI) .

An example of an OSS offering that has suffered from a calamity is XeroMQ. Even though ZeroMQ is still around and being used, the ZeroMQ OSS effort was shaken by the death of its prime moving force Pieter Hintjens who died[227)]. There are any spinoffs and derivatives of ZeroMQ. Here are a few reported in Wikipedia[228)]

- *In 2012, two of the original developers forked ZeroMQ as Crossroads I/O.*[229)230)]
- *Martin Sustrik has started nanomsg,*[231)] *a rewrite of the ZeroMQ core library.*[232)]
- *In 2012, Dongmin Yu announced his pure Java conversion of ZeroMQ, JeroMQ.*[233)]
- *This has inspired further full-native ports of ZeroMQ, such as NetMQ for C#*[234)].
- *March 2013, Pieter Hintjens announced a new draft of the ZMTP wire-level protocol bringing*

extensible security mechanisms to ZeroMQ[235]*.*

- *Martin Hurton implemented the CurveZMQ authentication and encryption mechanism*[236] *in the core library shortly afterwards.*

Not recognizing or managing the risks due this kind of unfortunate occurrence to the system (or project) might be expedient, but is in many ways irresponsible for systems (or projects) with a long lifespan. An alternative to the risks of OSS or proprietary vendor lock-in is the selection components that are standards based from a Standards Organization that offers a wider spectrum of vendors to choose from.

DIDO Specifics

Return to the Top

To be added/expanded in future revisions of the DIDO RA

219)
Software Evolution, Blog Post, Mitopia Technologies, https://mitosystems.com/software-evolution/
220)
The History of Windows Operating Systems, Vengie Beal, Webopedia, 2 Sugust 2018, Accessed 16 July 2020, https://www.webopedia.com/DidYouKnow/Hardware_Software/history_of_microsoft_windows_operating_system.html#windows-1
221)
When will Microsoft end support for your version of Windows or Office?, Ed Bott, 10 April 2018, ZDNet, Accessed 16 July 2020, https://www.zdnet.com/article/when-will-microsoft-pull-the-plug-on-your-version-of-windows-or-office/
222)
Six Years Since World Launch, IPv6 Now Dominant Internet Protocol for Many, Internet Society, 6 June 2018, Accessed 16 July 2020, https://www.internetsociety.org/news/press-releases/2018/six-years-since-world-launch-ipv6-now-dominant-internet-protocol-for-many/
223)
IIC Connectivity Framework defines IIoT network architecture for scalable interoperability, Industrial Embedded Systems, 18 July 2020, http://industrial.embedded-computing.com/articles/iic-connectivity-framework-defines-iiot-network-architecture-for-scalable-interoperability/
224)
Tools for Distributed Systems Monitoring, Kufel, Łukasz, 1 December 2016, Foundations of Computing and Decision Sciences, Vol 41: 10.1515/fcds-2016-0014, https://www.researchgate.net/publication/311863266_Tools_for_Distributed_Systems_Monitoring/citation/download
225)
Log4j - Logging Levels, Log4J, Accessed 18 July 2020, https://www.tutorialspoint.com/log4j/log4j_logging_levels.htm
226)
Software Metrics for Predicting Maintainability, Marc Frappier, Stan Matwin, and Ali Mili, University of Ottawa, Canadian Space Agency, 1994, References 20 July 2020,

http://www.dmi.usherb.ca/~frappier/Papers/tm2.pdf
227)

The Life, Ideas, and Legacy of Pieter Hintjens (from ZeroMQ to “A Protocol for Dying”), Medium, Evan SooHoo, 23 September 2018, Accessed 20 July 2020, https://medium.com/@evan_soohoo/the-life-ideas-and-legacy-of-pieter-hintjens-from-zeromq-to-a-protocol-for-dying-fc1673caeaa7
228)

ZeroMQ, Wikipedia, Accessed 20 July 2020, https://en.wikipedia.org/wiki/ZeroMQ
229)

ZeroMQ and Crossroads I/O: Forking over trademarks. LWN.net. Retrieved 14 July 2012.https://lwn.net/Articles/488732/
230)

Crossroads I/O. Retrieved 14 July 2012, http://www.crossroads.io/
231)

nanomsg. Retrieved 8 June 2013 http://nanomsg.org/
232)

Why should I have written ZeroMQ in C, not C++ (part I) , 250bpm, Martin Sústrik, 10 May 2012, Accessed 20 July 2020, http://250bpm.com/blog:4
233)

jeromq - java pojo zeromq, zeromq-dev mailing list, Retrieved 23 May 2013, http://lists.zeromq.org/pipermail/zeromq-dev/2012-August/018265.html
234)

NetMQ, GitHub, Retrieved 23 May 2013, https://github.com/zeromq/netmq
235)

Securing ZeroMQ: draft ZMTP v3.0 Protocol, Hintjens.com, Retrieved 23 May 2013, http://hintjens.com/blog:39
236)

CurveZMQ - Security for ZeroMQ , CurveZMQ, Accessedd 20 July 2020, http://curvezmq.org/

4.3.5.4 Software Manageability Issues

Return to Manageability

About

Over the last few decades, many advances have been made in terms of Open Source Software (OSS) which has to change the way software was developed, released and used. During this time period, the traditional Waterfall Model of System and software development has also been largely supplanted with the Agile Model. Many of these changes also have to do with the evolution of systems (or projects) from being Greenfield to Brownfield development and from a “build the world” attitude towards “integrate and glue the world” mindset.

Successful OSS development and adoption not only has to produce products which are solid, strong and robust but also must meet the needs of a Community of Interest (CoI) that has coalesced around a single minded, purpose built, functionality (i.e., Apache Tomcat application server, PostgreSQL Database, Node.js an asynchronous event-driven JavaScript runtime, Docker containerized apps, Kubernetes orchestration engine for containers, etc.). Many of the OSS products are part of many of the successful projects today.

However, it is not good enough to just write software and make it publicly available. At the heart of these successful efforts are the well governed, focused, supporting CoIs. There is a desire from almost all systems (or projects) to join the OSS trend but unfortunately, the need for strong governance and rigorous methodology is minimized or skipped in the name of expediency. Fortunately, there is an organization which can help with this called Talk Openly Develop Openly (TODO) (not to be confused with a `to-do`).

TODO organization, though focused on OSS, has written a series of white papers that are well worth studying and using even it your system (or project) is not OSS. One of these papers which is particularly germane to Software Manageability is Tools for managing open source programs [1)]. It is beyond the scope of this document to try to recreate the full content of this white paper. It does present a list of many of the tools available for managing software and how to use them. Here is the table of content from the document:

- Why you need special tools for open source program management
- How to select and plan your tools
- Elements of a basic toolset
- Tools for managing source code
- Tools for tracking project health
- Tools for communications and collaboration
- Tools for corporate-scale GitHub management

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

1)

Tools for managing open source programs, Talk Openly Develop Openly (TODO), Accessed 20 July 2020, https://todogroup.org/guides/management-tools/

4.3.6 Usability

Return to Non-Functional Requirements

About

Usability is defined by ISO/IEC 25010:2011 SQuaRE -- System and Software Quality Models as the degree to which a product or system can be used by Stakeholder (i.e., specified users) to achieve specified goals within a specified context.

Goals

The goals of usability are[237)]:

1. **Effectiveness** - The accuracy and completeness with which users achieve specified goals
2. **Efficiency** - The resources expended in relation to the accuracy and completeness with which users achieve goals.
3. **Satisfaction** - The comfort and acceptability of use.

Sub-Characteristics

This characteristic is composed of the following sub-characteristics[238)]:

- ***Appropriateness Recognizability*** *- Degree to which users can recognize whether a product or system is appropriate for their needs.* (1)
- ***Learnability*** *- Degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use.* (1)
- ***Operability*** *- Degree to which a product or system has attributes that make it easy to operate and control.* (1)
- ***User Error Protection*** *- Degree to which a system protects users against making errors.* (1)
- ***User Interface Aesthetics*** *- Degree to which a user interface enables pleasing and satisfying interaction for the user.* (1)
- ***Accessibility*** *- Degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.* (1)

See:

- ISO/IEC 25010:2011 SQuaRE -- System and Software Quality Models
- https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/61-usability
- ISO/IEC 9241-210:2019 Ergonomics of human-system interaction

Metrics

Return to the Top

Usability as a characteristic is often considered a subjective quality and left to “interpretation”, however, there are metrics which use to quantify these sub-characteristics. Before we delve into the definition of the specific metrics, it is important to understand why we need metrics rather than just rely on intuitive evaluations.

A core reason to collect Usability Metrics is to provide data about a stakeholder's understanding of a product's usability rather than the developer's understanding of usability. When the two understandings (i.e., interpretations) converge everyone is happy resulting in a way forward. That result may be to either continue in the same direction or to have a reassessment of the user's needs.

The metrics must quantify that the system meets the goals of the overall system:

1. The Effectiveness Metrics of the communication between the system and the users
2. The Efficiency Metrics of the users use of the system to accomplish their work
3. The Satisfaction Metrics of the users that the sub-characteristics of the system are met.

Ultimately, the primary objective of usability metrics for evaluating a system or product is properly engineered (i.e., neither under- or over-engineered).

DIDO Specifics

Return to the Top

To be added/expanded in future revisions of the DIDO RA

237)

Justin Mifsud, Usability Metrics – A Guide To Quantify The Usability Of Any System, Accessed 18 November 2020, https://usabilitygeek.com/usability-metrics-a-guide-to-quantify-system-usability/

238)

International Organization for Standardization (ISO), Usability, ISO25000, Accessed: 17 November 2020, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/61-usability

4.3.6.1 Effectiveness Metrics

Return to Usability

About

There are two ways to calculate effectiveness:

- Count the success of completing tasks
- Count the errors while completing tasks

Using Task Completion Rates

Return to Top

Effectiveness measures the ratio of tasks completed versus those that are not completed. It is usually reported as a simple percentage such as 80% of the tasks are completed. The data is collected using a simple binary (i.e., yes = '1' and no = '0') for each user for each task.

Mathematically:

$$Effectiveness = \frac{\sum_{user=1}^{numberOfTesters} Completed\,Tasks\,by\,user}{\sum_{user=1}^{numberOfTesters} Total\,Tasks\,assigned\,to\,user} \times 100\,\%$$

Here is an example where the system uses 10 tasks and all the tasks are completed by 5 users. The results are:

user	Completed Tasks	Incomplete Tasks	Total Tasks	Effectiveness
user 1	8	2	10	**80%**
user 2	9	1	10	**90%**
user 3	10	0	10	**100%**
user 4	5	5	10	**50%**
user 5	6	4	10	**60%**
total	**38**	**12**	**50**	**76%**

This efficiency ratio is referred to as the **Completion Rate** or **Fundamental Usability Metric**. As a result of its simplicity and ease of understanding, the **Completion Rate** is a popular way of reporting efficiency. However, in order to determine the efficiency, a working system with real users needs to be available. This is a major advantage of using the Agile Model since the efficiency can be calculated during each Sprint and used as a guidance for future sprints.

Using Error Rates

Return to Top

Another way to understand an application or system efficiency from a usability perspective is to count the errors instead of counting the success of completing tasks. Every error can have more consequences requiring more work to “undo” thus negatively impacting Effectiveness and efficiency. The following is a partial list that users can make while completing tasks[239] :

- **Unintended actions** - using a `next` operation without intending.
- **Slips entering information** - pressing a `return escape` key accidentally
- **Mistakes in data entry** - twiddling the digits in a phone number or social security number or having autocorrect inadvertently “correct” an entry
- **Omissions in data entry** - forgetting login as an existing user rather than creating a new user, skipping filling in age

Recording each instance of an error along with a description can help refine the system to prevent these errors in the future. For example, “user entered last name in the first name field”.

In many ways, this metric can be considered similar to the Task Complete Rate metric but instead of counting completed tasks the errors are counted. Note, many errors can occur in accomplishing a single task. For example, a data entry form required for one task can have errors associated with each field. Each of these errors can have a negative impact on the overall efficiency.

Another enhancement to this metric wold be to assign a weight to the error. For example, data entry errors are less dramatic and important than unintended actions. Therefore, unintended actions would receive a higher weighting than data entry.

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

239)
Jeff Sauro, 10 Essential Usability Metrics, Measuring U, 30 November 2011, Accessed 19 November 2020, https://measuringu.com/essential-metrics/

4.3.6.2 Efficiency Metrics

Return to Usability

About

Efficiency is measured in terms of the time it takes to complete a task from when the task is initiated to when it is successfully completed. The units used to record the time must be uniform for all tasks (i.e., milliseconds, seconds, minutes, etc).

Mathematically The time taken to complete a task can then be calculated by simply subtracting the start time from the end time:

$$Task\ Time = End\ Time - Start\ Time$$

There are two ways to calculate Efficiency:

- Time-Based Efficiency
- Overall Relative Efficiency

Time-Based Efficiency

Return to Top

In this calculation, the quotient from the division of the success of a task (either one or zero) divided by the time to accomplish a task is an indicator of the efficiency of the task. For example, if a tasks was not successful, then the success of the task is zero and the efficiency is zero. If the task is successful and it takes one minute to accomplish the task, then the efficient is one (i.e., 1/1). If it takes two minutes to accomplish the task, then the efficiency is one half (i.e., 1/2 = .5).

To calculate the Time-Based Efficiency for all tasks and all users, the following equation applies:

$$Time\ Based\ Efficiency = \frac{\sum_{i=1}^{R} \sum_{j=1}^{N} \frac{n_{i,j}}{t_{i,j}}}{N \times R}$$

Where:

- **N** : The total number of tasks (goals)
- **R** : The number of users
- $\mathbf{n_{i,j}}$: The result of task i by user j; if the user successfully completes the task, then Nij = 1, if not, then Nij = 0
- $\mathbf{t_{ij}}$: The time spent by user j to complete task i. If the task is not successfully completed, then time is measured till the moment the user quits the task

Justin Mifsud[1] provides an excellent example of how for calculating time-based efficiency:

Suppose there are 4 users who use the same product to attempt to perform the same task (1 task). 3 users manage to successfully complete it – taking 1, 2 and 3 seconds respectively. The fourth user takes 6 seconds and then gives up without completing the task.

Taking the above equation:

N = The total number of tasks = 1
R = The number of users = 4

User 1: Nij = 1 and Tij = 1
User 2: Nij = 1 and Tij = 2
User 3: Nij = 1 and Tij = 3
User 4: Nij = 0 and Tij = 6

Placing the above values in the equation:

$$Time\ Based\ Efficiency = \frac{\left(\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{0}{6}\right)}{1\times 4} = 0.46\ \frac{goals}{\text{sec}}$$

Overall Relative Efficiency

Return to Top

The overall relative efficiency uses the ratio of the time taken by the users who successfully completed the task in relation to the total time taken by all users. The equation can thus be represented as follows[240)]:

$$Overall\ Relative\ Efficiency = \frac{\sum_{i=1}^{R}\sum_{j=1}^{N} n_{i,j}\times t_{i,j}}{\sum_{i=1}^{R}\sum_{j=1}^{N} t_{i,j}} \times 100\%$$

Justin Mifsud[1] provides an excellent example of how for calculating Overall Relative Efficiency.

Assume there are 4 users who use the same product to attempt to perform the same task (1 task). 3 users manage to successfully complete it – taking 1, 2 and 3 seconds respectively. The fourth user takes 6 seconds and then gives up without completing the task.

Taking the above equation:

N = The total number of tasks = 1
R = The number of users = 4

User 1: Nij = 1 and Tij = 1
User 2: Nij = 1 and Tij = 2
User 3: Nij = 1 and Tij = 3
User 4: Nij = 0 and Tij = 6

Placing the above values into the equation yields the following:

$$Overall\ Relative\ Efficiency = \left(\frac{\left(\left(1\times1\right)+\left(1\times2\right)+\left(1\times3\right)+\left(1\times6\right) \right)}{\left(1+2+3+6\right)} \right) \times 100 = 50\,\%$$

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

240)

UI Designer, Efficiency, Accessed 19 November 2020, http://ui-designer.net/usability/efficiency.htm

4.3.6.3 Satisfaction Metrics

Return to Top

About

Usability Metrics are generally done through standardized questions designed to capture a the user's sentiments about the application, product or system. The survey's pose questions to the users and provide a scale of acceptability the user chooses in assessing a particular attribute. The most common scale is based on the Likert Scales originally proposed in 1032 [241)].

Figure 49 gives a few of the Scales that Lickert defined. There are more available here:

Scale	Attitude / Sentiment				
Agreement	**Strongly Disagree**	**Disagree**	**Undecided**	**Agree**	**Strongly Agree**
Frequency	**Never**	**Rarely**	**Sometimes**	**Often**	**Always**
Importance	**Unimportant**	**Slightly Important**	**Moderately Important**	**Important**	**Very Important**
Quality	**Very Poor**	**Poor**	**Fair**	**Good**	**Excellent**
Likelihood	**Almost Never True**	**Usually Not True**	**Occasionally True**	**Usually True**	**Almost Always True**
Score	1	3	3	4	5

Figure 49: The Lickert Scale

There are two ways that user satisfaction can be measured:

- **Task Level Satisfaction** - The Task Level Satisfaction is made at the end of each task attempted by the user. Note, a task may be attempted but it may not be completed. Therefore, it is important to record not just the attitude or sentiment about the task, but also the status of the task when the user takes the survey.
- **Test Level Satisfaction** - Similar to the **Task Level Satisfaction**, Test Level Satisfaction is conducted at the end of a Test which can be comprised of multiple tasks. Therefore, in order to properly assess the Test Level, an evaluation of the Task assessments also needs to be made. For example, a test assessment might be low because some of the tasks were assessed as poor.

ISO also provides some guidance in how to assess User Satisfaction. See:

- ISO 10001:2018 Quality management — Customer satisfaction — Guidelines for codes of conduct for organizations
- ISO 10002:2018 Quality management — Customer satisfaction — Guidelines for complaints handling in organizations
- ISO 10003:2018 Quality management — Customer satisfaction — Guidelines for dispute resolution

external to organizations
- ISO 10004:2018 Quality management — Customer satisfaction — Guidelines for monitoring and measuring

- **Note:** For more information, see: https://blog.ansi.org/2018/07/customer-satisfaction-iso-10002-quality/#gref

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

241)

Saul McLeod, Likert Scale Definition, Examples and Analysis , Simply Psychology, 2019, Accessed 20 November 2020, https://www.simplypsychology.org/likert-scale.html

4.3.7 Performance

Return to Non-Functional Requirements

About

Performance is the ability of a system to accomplish the required functionality at or under the required specification limits. The limits are generally provided relative to time (i.e., so-many transactions per second, so-many updates per millisecond, so-many recorded entries per second, etc.) The specifications can also include accuracy, precision, precision or even efficiency of other dependent systems as requirements. The following are some examples of performance specifications:

- Time for a data-entry window to appear
- Number of units produced in a given amount of time
- Time to react to a given event
- Amount of energy required to perform an activity
- The amount of computing resources required (i.e., Central Processing Unit (CPU), Random access memory (RAM), Read-Only Memory (ROM), Storage Device, Bandwidth, etc)
- Time to process a computational task such as compression-decompression, encryption-decryption, generate a Unique ID (UID), serialize-deserialize, calculate an area, calculate a new trajectory, etc.
- Time to process a storage task such as store records, index the records, retrieve records, etc.
- Time to access memory or cache such as direct memory access(DMA), the hit ratio
- Time to transfer data between components of a computer such as from memory to Central Processing Unit (CPU), from CPU to graphics card, etc

Performance can also be viewed from the following perspectives:

- 4.3.7.1 Platform Performance
- 4.3.7.2 Application Performance
- 4.3.7.3 Network Performance

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

=-
=-

4.3.7.1 Platform Performance

Return to Performance

About

Platform Performance is concerned with the abilities of a computer system matched with an Operating System (OS) to meet or exceed the requirements of a specific system (or project). The platform could be a real system or a virtual system running locally or in the cloud. Often this comes down to the cost of the hardware and OS, but this can also include social responsibility requirements such as energy consumption, geopolitical concerns, or even ethics which prevent the use of certain products.

Figure 50 shows the salary of a Senior Software Engineer in the USA in July 2020, around $105,000. This is the take home pay, not the fully burdened cost.

Figure 50: Average Sr. Software Engineer / Developer / Programmer Salary - June 2020[242)]

Figure 51 shows the cost of either owning a Server or using a Infrastructure-as-a-Service (IaaS) solution. The Total Cost of Ownership (TCO) cost is about 10 times that of an IaaS.

	On-premises	Cloud	Savings $	Saving %
Year 1	39,347.18 $	3,766.80 $	35,580.38 $	90%
Year 2	9,063.19 $	3,766.80 $	5,296.39 $	58%
Year 3	9,063.19 $	3,766.80 $	5,296.39 $	58%
Year 4	9,063.19 $	3,766.80 $	5,296.39 $	58%
Year 5	39,347.18 $	3,766.80 $	35,580.38 $	90%
Year 6	9,063.19 $	3,766.80 $	5,296.39 $	58%
Year 7	9,063.19 $	3,766.80 $	5,296.39 $	58%
Total:	124,010.31 $	26,367.60 $	**97,642.71 $**	**79%**

Please note that inflation is not factored into these results

Figure 51: Average Total Cost of Ownership (TCO) versus Cloud [243)]

If your application is not performing well and it is estimated that it might take one year to upgrade the

software, you could get a senior Software Engineer for the $105,000 or you could upgrade your server. If it is a purchase, you could get almost three servers for the cost of an engineer. If you decided to use IaaS, for the cost of the engineer, you could get almost 20 servers.

It is recommended to follow the guidelines for sizing a server provided on-line.[244)245)246)]

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

242)
Payscale.com, Accessed 20 July 2020, https://www.payscale.com)
243)
Cost of server ownership: on-premise vs. IaaS, April 2019, Sophie Furnival, Sherweb, Accessed 20 July 2020, https://www.sherweb.com/blog/cloud-server/total-cost-of-ownership-of-servers-iaas-vs-on-premise/
244)
Basic Guidelines for Sizing Servers, Jason Thomas, 24 August 2017, Accessed 20 July 2020, https://www.mirazon.com/basic-guidelines-for-sizing-servers/
245)
3 secrets to right-sizing a server, TidalScale, Accessed 20 July 2020, https://www.tidalscale.com/3-secrets-to-right-sizing-a-server/
246)
Determining Your Ideal Server Size: Which Package is Right for Me?, Media Temple, 7 April 2015, Accessed 20 July 2020, https://mediatemple.net/resources/web-hosting-101/determining-your-ideal-server-size-which-package-is-right-for-me/

4.3.7.2 Application Performance

Return to Performance

About

Application Performance are measures of real-world performance and availability of applications. It is often used to describe remote and cloud computing applications, however, there is a lot of time spent on tuning applications running locally to enhance or improve their performance. For example file access time, database, infrastructure, graphics, human interfaces, network access, etc.

Most applications today are not Brownfield versus Greenfield systems. In addition, many programs rely on incorporating reusable components as part of the cost reduction strategy. Reusing components such as operating systems, databases, Access Control etc. makes a lot of sense since the functionality of these components is not an area of expertise for the developers of the system. For example, if the system is for medical devices, then designing new virtual page swap software has nothing to do with medical technology or medicine. Another example might be developing middleware software for accessing other computers on the network. This kind of development might seem “fun” but it is not within the scope of the project that is developing the system. These re-used components also have the benefit that they are used by a wide range of other projects and systems accessed by many people. Collectively the communities can find and fix software faster than it can be done locally.

I don't believe there is a company that could pay for all the testing the user communities provide. Linux is a great example.[247)]

- *About 3 to 3.5 billion, which is pretty much the number of humans on the planet who have regular access to the internet. Let’s break it down....*
- *Directly as a desktop OS? A minority of users, sadly windows is hogging over 85% of that market share - which it certainly doesn’t deserve. The remaining 10 to 15 percent is mostly Apple, who have done a very good job at creating its desktop OS based on a Unix implementation, so it’s kind of a “cousin” of Linux in that sense.*
- *If you use an Android phone or tablet, you’re using the Linux kernel and GNU software indirectly. Many Android devices will show this in the “kernel info” or “kernel version” in the settings app’s About section.*
- *If you have a router, modem, printer, scanner, or any similar device, its firmware is probably based on a small Linux or FreeBSD system.*

It is also true that this many users also marks Linux as a huge target for malicious activities, but pretending that any OS is not vulnerable to attack is a fool's game.

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

247)

Éric Nunya, 9 March 2020, Quora, ccessed 27 July 2020, [https://www.quora.com/How-many-Linux-users-are-there-in-the-world[]]

4.3.7.3 Network Performance

Return to Performance

About

Return to Top

Network Performance captures the statistical metrics and the analytical review of a network. Collectively they reflect the network's Quality of Services.

It is a qualitative and quantitative process that measures and defines the performance level of a given network. It guides a network administrator in the review, measure and improvement of network services.

There are two main ways to connect devices together:

- Ethernet (i.e., wired using network cables.)
- Wireless Fidelity (Wi-Fi) (i.e., wireless using radio signals).

Although it is possible to connect computers directly together, generally the computers connect to a Network Device such as a Router. There are a number of variables that determine the actual speed of the connection between the computers. The wired connections are as a rule faster than the wireless connection but the number, length of the quality of the network connections and the kinds of network devices and the number of devices can impact wired connection speed. WiFi connections are more susceptible to interference from electrical devices, physical objects (i.e., metal walls or cages), or environmental conditions (i.e., weather and solar flares).

An Ethernet connection is consequently more reliable especially when Shielding Network Cabling (i.e., Category 6 (Cat-6),Category 7 (Cat-7), Category 8 (Cat-8)) are used. Ethernet is almost always faster than WiFi. The fastest Ethernet speeds today top out at 10Gbps or higher, while the fastest WiFi speeds theoretically max out at 6.9Gbps, though actual speeds are much slower – usually less than 1Gbps.[248)].

Speed

Return to Top

Network speed is for the most part about acquiring the correctly sized physical Network Devices (i.e. Modem, Router, Switches, Network Cabling, etc.) to meet the demands of the system. However, there are restrictions that arise such as the need for wireless connections (i.e., WiFi, Bluetooth, ZigBee, Infrared Wireless Networking etc.), space and heat considerations (i.e., a big problems for planes, ships, labs, hospital rooms, etc.) or when the assets participating in the system are distributed and not under the control of a single source (i.e., blockchains, Distributed Ledger Technologies (DLT), supply chains, etc.).

Most Ethernet connections fall into the following categories: Wired Network and Wireless Network

The wired networks use hardware such a modems, routers, switches, cabling, etc. together.

Wireless cables connect to each and every one of the computers in the network. The cost of a wired network is lower compared to the wireless network since Ethernet, cables, and switches are not expensive. Wired LAN offers better performance compared to wireless networks.

Wired Connections

Return to Top

Table 20: Summary of Difference between CAT-5 through CAT-8[249)]

Category	Standard	Data rate	Frequency[1)]	# of Conductors
Category 5 (Cat-5)	100BASE-TX	100Mbit	100 Mhz	4 or 8
Cat-5E	100BASE-TX	1Gbit	100 Mhz Duplex	8
Category 6 (Cat-6)	EIA 568B2.1	1-10 Gbit[2)]	250 Mhz	8
Cat-6A	10GBASE-T	10 Gbit	500 Mhz	8
Category 7 (Cat-7)	10GBASE-T	10 Gbit	600 Mhz	8
Cat-7A	10GBASE-T	10 Gbit	1000 Mhz	8
Category 8 (Cat-8)	40GBASE-T	40 Gbit	1600-200Mhz Mhz	8

[1)] **Note:** 1 hertz is roughly equivalent to 1000 milliseconds, 20 killohertz is roughly equivalent to 0.05 milliseconds. See Unit Juggler.
[2)] **Note:** Depends on the length and cable type

Wireless Connections

Return to Top

Table 21: Side-by-side comparison of wireless routers[250)]

	Netgear Nighthawk X10 AD7200	Asus RT-AC86U AC2900	Linksys EA6350 AC1200+	TP-Link Archer C7 AC1750	Trendnet AC2600	TP-Link AC2300	Linksys WRT32X
Top Theoretical Speed	4600 Mbps on 60 GHz	2167 Mbps on 5GHz	867 Mbps on 5 GHz	1300 Mbps on 5 GHz	1733 Mbps on 5 GHz	1625 Mbps on 5 GHz	2600 Mbps on 5 GHz

Bandwidth

Return to Top

Bandwidth is defined as the bandwidth data carrying capacity of a network channel or the entire network. Bandwidth is measured by the bit-rate of the network transmission capacity. This is sometimes thought of as the network channel's data transfer speed. Bandwidth can be used to describe wired, wireless or even data buses.

A bit represents a single binary digit either '0' or '1'. The '0' or '1' generally represent yes/no, true/false, on/off, or up/down/ It does not necessarily equate a '0' with false and a '1' with true. When transmitted over a network, the data is sent as a stream of bits (not bytes). A byte is generally used to signify a unit of memory or storage (i.e., RAM or ROM) that usually is eights bits long (wide) and is the smallest number of bits used to represent a character in the original ASCII character set used by most most computers.

Bandwidth is measured as bits per second and is used as a denominator of bits (i.e., kilobits, megabits). When bandwidth is used to describe a network connection (i.e., switch, server or router), it is generally in megabits, however, when it is used to describe the data flowing into the connection then bandwidth referred to as traffic and could be measured in either megabits per second (Mb/s or Mbps) or megabytes per second (MB/s or MBps). Although the nomencalture is subtle, it is important to be aware of the difference. An inadvertent misunderstanding could result in a error of magnitude 8 (i.e., 1 byte = 8 bits).

Since the megabytes figure will be larger than the megabits figure (equation to follow shortly) most industry service providers like to give a total transfer based on this figure – however most bandwidth providers use megabits.

$$1 byte = 8 bits$$

Some examples:

Table 22: Some examples of converting MegaBits to MegaBytes

Mega Bytes per second	bits per Byte	MegaBits per second
8 MBps	*8	64 Mbps [1)]
9 MBps	*8	72 Mbps
10 MBps	*8	80 Mbps
20 MBps	*8	160 Mbps

[1)] **Note:** there are two kinds of units listed: Mbps (Mega BIT per second) and MBps (Mega BYTE per second)

> *Today, many cable ISPs are capable of delivering internet speeds over 1 Gigabit per second. That's 1 billion bits per second! Not everyone needs this much speed today (Netflix reports that a connection speed of 25 Megabits per second is all that's required to stream Ultra HD content), but cable ISPs see a future of virtual reality, telehealth, driverless cars, and an internet of things. In that environment, speed requirements are going to increase. Regardless of whether it's necessary today, ISPs are preparing their networks for the needs of the future. So, while we'll likely always measure speed in bits and data volume in bytes, the consistency and speed at which those bits are delivered over the internet will surely rise.*[251)]

In addition to the Internet Service Provider (ISP) bandwidth limitations, it is also important to remember that there are local hardware components to the network. For example, Network Quality of Service (QoS) Policies.

Network Quality of Service (QoS)

Return to Top

Quality of service (QoS) captures the metrics used to measure the overall performance of a computer network as experienced by participants (i.e., computers, processes, devices, etc) in network.

Internet Protocol (IP) networks, QoS is particularly focused on setting priorities for packet traffic and reserving resources rather than the QoS of the Network Services (i.e, DNS). It helps establish overall priorities for applications, users, or data flows, or to guarantee a certain level of performance to a data flow. For example, setting Voice over IP (VOIP) as a priority over texting.[252] Some of the common metrics used to describe Network QoS are:

- Throughput
- Latency
- Packet Loss
- Jitter
- Bit Error
- Download Speed
- Upload Speed

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

248)
What is the difference between a WiFi and Ethernet connection?, Spectrum Enterprise, Accessed 27 July 2020, https://enterprise.spectrum.com/support/faq/network/what-is-the-difference-between-wifi-and-ethernet-connection.html

249)
How to Tell CAT Data Cables Apart, Horst Messerer, Machine Design, 8 January 2016, Accessed 24 July 2020, https://www.machinedesign.com/automation-iiot/cables-connectors-enclosures/article/21834690/how-to-tell-cat-data-cables-apart

250)
Connect at Speed with the Fastest Routers, Speedtest, PCMag, Accessed 21 July 2020, https://www.speedtest.net/about/knowledge/fastest-routers

251)
Why Do We Use Bits to Measure Internet Speed but Bytes to Measure Data?, The Internet and Television Association (NCTA), 21 July 1017, Accessed 20 July 2020, https://www.ncta.com/whats-new/why-do-we-use-bits-measure-internet-speed-but-bytes-measure-data

252)
Andrew Froehlich, The Basics Of QoS, 15 August 2016, Accessed 27 July 2020,

https://www.networkcomputing.com/networking/basics-qos

4.3.8 Interoperability

Return to Non-Functional Requirements

About

https://www.isko.org/cyclo/interoperability#3.1

Interoperability is a characteristic of a product or system whose interfaces are completely understood to work with other products or systems, present or future, in either implementation or access, without any restrictions. There are different levels of interoperability. As one moves up the interoperability levels, the degree of interoperability increases in difficulty. The most difficult is the semantic level, in which two assets can communicate with little or no prep work using common detailed formal, machine readable Ontologies including a vocabulary of terms.

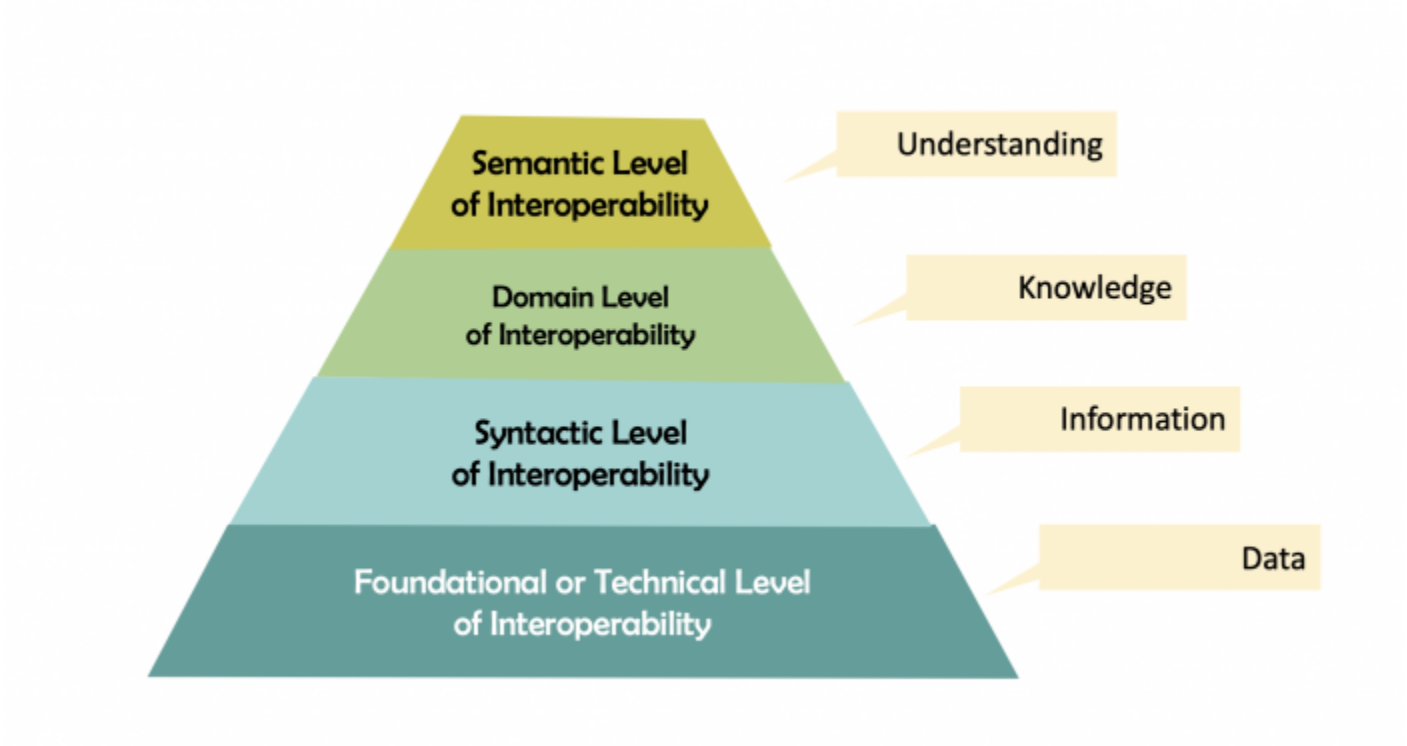


Figure 2: The Automation Pyramid and Interoperability

Foundational or Technical Level

return to Top

The *Foundational* or *Technical Interoperability Level* is concerned with the most fundamental and basic kind of interoperability. It provides the “foundation” for all the higher levels of interoperability. It establishes the technical aspects of peer-to-peer interoperability that prevail regardless of hardware, operating system, or middleware platforms. The Foundational or Technical Interoperability Level is

defined in terms of two perspectives: Data and System.

- From the Data perspective, there is a clear, shared expectation for the contents, context and meaning of that data (e.g., Big-Endian or Little-Endian, or size of an integer, or character set used)
- From the system perspective the differences in computer technology are non-existent (e.g., 16-Bit, 32-Bit and 64-Bit processors. Reduced Instruction Set Computer (RISC) versus Complex Instruction Set Computer (CISC) computers, etc.). In a Distributed System, the network protocols are another part of the Foundational or Technical Level of interoperability (e.g., Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) over Internet Protocol address (IP address), etc.).

Syntactic Level

return to Top

The *Syntax Level* is concerned with the correct combination and sequence of the elements in a language and, in this context, as it applies to Data Structure or a Programming Language. For example, in English, we do not say `ball red large plastic`, we would say `large, red plastic ball`. The first form is syntactically incorrect, the second form is not. Interoperability at the syntactical Level means two different systems can exchange information and be structurally equivalent. The interpretation of the syntactical data can be different (i.e., big to an ant is different than big to an elephant). The difference in interpretation is a semantic difference (see Semantic Level).

Domain or Structural Level

return to Top

The *Domain Level* is knowledge of a specific, specialized subject area, discipline or field. For example, there is specific domain knowledge in Chemistry, Organic Chemistry, Physics, Mathematics, or Statistics. A specific example might be the use of the word `round`. Round has different meanings in the context of Poker, Math, Cheese, and the Military. This is in contrast to general knowledge, or domain-independent knowledge which, in general, refers to the common usage of a word, sometimes referred to as the dictionary usage of a word. The use of the word Domain or Domain knowledge is used within general purpose disciplines such as history, law, systems engineering or even computer science. People with knowledge in a generic area (i.e., law) might have specific domain knowledge in computers, electronics, computers or even specific computer architectures such as Reduced Instruction Set Computer (RISC).

Semantic Level

return to Top

The *Semantic Level* is concerned with the meaning, relationship and restrictions on data and information described in the Domain or Structural and Syntactical Levels. At this level, computer systems exchange information unambiguously. The information exchanged might not be identical but there is a shared understanding between the two systems about the meaning of the information. Some examples of

Semantic Interoperability are described by the ability of:[6]:

- Two independent programs with reasoning capability to arrive at the same conclusions from the same data
- An information system, without additional human intervention, to perform the tasks for which it was designed through the exchange of relevant data with other systems[7], independently of how and what purpose this data was originally collected, stored and managed.
- One system to use data from an external source (without prior planning) and still able to output useful (though not necessarily perfect) results.

DIDO Specifics

return to Top

To be added/expanded in future revisions of the DIDO RA

6)
Ontology Taxonomy Coordinating WG / OntacGlossary, Accessed 10 July 2020, http://colab.cim3.net/cgi-bin/wiki.pl?OntologyTaxonomyCoordinatingWG%2FOntacGlossary

7)
Note: mediated by formally specific definitions and axioms

4.3.9 Elasticity

Return to Non-Functional Requirements

About

Cloud Elasticity also known as Elasticity "is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible."[255]. A primary motivation behind Elasticity is to save money by not investing in Infrastructure-as-a-Service (IaaS) that is not used or under used. It also saves natural resources since heating and air conditioning are not used on resources that are on standby[256].

The following are the various strategies used to achieve elasticity:

- **Cost-aware criteria**: The default is to assume that there is a firm fixed price for IaaS providers, however, some providers allow for spot pricing schemes (i.e., Amazon) which can allow users to tap into IaaS excess capacity. This excess capacity is there so that the IaaS provider can meet the Service Level Agreements (SLAs) guaranteed to all customers.
- **Power-aware cost function**: Using the power required to meet the application's needs and little more, i.e., using off-peak power consumption only.
- **Multiple classes of requests**: Allow applications to be segmented into categories based on the need for service. For example, customers' requests for service from the application can be divided into three categories: High Priority for performing financial transactions; Medium Priority for those making product inquiries; Low priority for simple browsing.
- **Scaling multiple applications**: Allow an application to be broken up into smaller applications whose functionality and services are orchestrated.

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

255)
Nikolas Roman Herbst, Samuel Kounev and Ralf Reussner, Elasticity in Cloud Computing: What It Is, and What It Is Not, Accessed on 11 August 2020, https://sdqweb.ipd.kit.edu/publications/pdfs/HeKoRe2013-ICAC-Elasticity.pdf

256)
Rui Han, Investigations into Elasticity in Cloud Computing, November 2013, Accessed 12 August 2020, https://arxiv.org/ftp/arxiv/papers/1511/1511.04651.pdf

4.3.10 Scalability

Return to Non-Functional Requirements

About

Scalability is the ability of a system to accomplish more work while maintaining the quality (i.e., without degradation) of the products produced or services provided by the system. There are different ways to calculate the work produced or performed by the system, which usually depends on the kind of product produced or services rendered.

For software systems, here are some of the common metrics used to quantify the products produced or the services provided:

- Number of transactions or events per unit of time (e.g., 5,000 credit card transactions a second[257])
- Number of tweets processed per unit of time (e.g., 6,000 tweets a second[258])
- Number of hours of videos uploaded per minute (e.g., 500 hours of fresh video per minute[259])
- Number of searches per day (e.g., 3.5 billion searches per day[260]).

Scalability is about being able to increase the output of products and services without major disruptions, interruptions or increased costs. Often, because of the Economies of Scale, the estimates for the costs should actually decline.

Two valid approaches to achieve Scalability are:

- Scaling Up: An approach generally applied to centralized or decentralized systems or products. It amounts to just adding more resources with more powerful versions. This method works until you have reach the limits on the availability of more sources power, for example, the speed of the Central Processing Unit (CPU), the amount of available memory or even the network capacity or speed. In decentralized systems(i.e., Cloud Computing) scaling can be either Vertical or Horizontal.
- Scaling Out: An approach usually associated with a Distributed System, which by its nature, allows for more Network Nodes, replicated with prepacked applications (i.e., Distributed Application (ĐApp or DApp)), that can be added with minimal overhead cost. In essence, adding more nodes offering redundant products and services. Alternatively, one can divide up the problem by functionality. For example, if accessing customer data is the bottleneck, then add more nodes with which to access the same data. If access to the actual data is the bottleneck, adding replications to the data store is recommended.

DIDO Specifics

Return to Top

To be added/expanded in future revisions of the DIDO RA

257)

Ryan Vlastelica, Why bitcoin won't displace Visa or Mastercard soon Market Watch, 18 December 2017, Accessed 10 August 2020, https://www.marketwatch.com/story/why-bitcoin-wont-displace-visa-or-mastercard-soon-2017-12-15

258)

Kit Smith, 60 Incredible and Interesting Twitter Stats and Statistics, Bandwidth, 2 January 2020, Accesssed 10 August 2020, https://www.brandwatch.com/blog/twitter-stats-and-statistics/

259)

James Hale, More Than 500 Hours Of Content Are Now Being Uploaded To YouTube Every Minute, Tubefilter, 7 May 2019, Acessed 10 August 2020, https://www.tubefilter.com/2019/05/07/number-hours-video-uploaded-to-youtube-per-minute/

260)

Maryam Mohsin, 10 Google Search Statistics You Need to Know in 2020 [Infographic], Oberlo, 3 April 2020, Accessed 10 August 2020, https://www.oberlo.com/blog/google-search-statistics

4.4 Assessing Requirements

Return to Requirements

An important part of having functional and non-functional requirements is access a particular effort to determine if the effort complies with the requirements. The approach is slightly different between Functional and non-functional requirements.

For Functional Requirements, it is important to think about what requirements need to be covered for a particular effort. For example, which Networks need to be supported? (Wired Network, Wireless Network. Bluetooth, ZigBee, Near-Field-Communication (NFC), etc).

Non-functional Requirements often have a large impact on the components that are selected as part of the infrastructure. For example, if a component is not scalable, the product built using that component is probably also not scalable.

The following sections are meant as aids in helping evaluate a project Functional and Non-FUnctional Requirements.

4.4.2 Non-functional Requirements Assessment

Return to Assessment

- Link to the DIDO-RA Google WorkBook

Assessing the Alternatives

Return to Top

Non-Functional requirements are by nature, hard to measure and hard to assess for compliance. The complexity of the assessment problem is compounded because products or systems as well as the environment they operate within are not static. Therefore, an assessment that is done today, may no longer be accurate in the future. When assessing Commercial Off-The-Shelf (COTS), Government Off-The-Shelf (GOTS), Modified Off-The-Shelf (MOTS) or NATO Off-The-Shelf (NOTS) or the inclusion of Open Source Software (OSS) the potential for changes (particularly enhancements) needs to be assed also.

One way to accomplish this is to provided an assessment which is weighted for the “ease of implementation” for new features. For example, a system or product may not have done much to support 4.3.4.1 Confidentiality, but the vendor of the product must determine that it is an easy upgrade to add it to the product. On the other hand, the support for **Confidentiality** might be extremely difficult. Sometimes, the feature may be easy to solve but requires time and money to accomplish. As a potential stakeholder, they can direct resources to help overcome this shortfall.

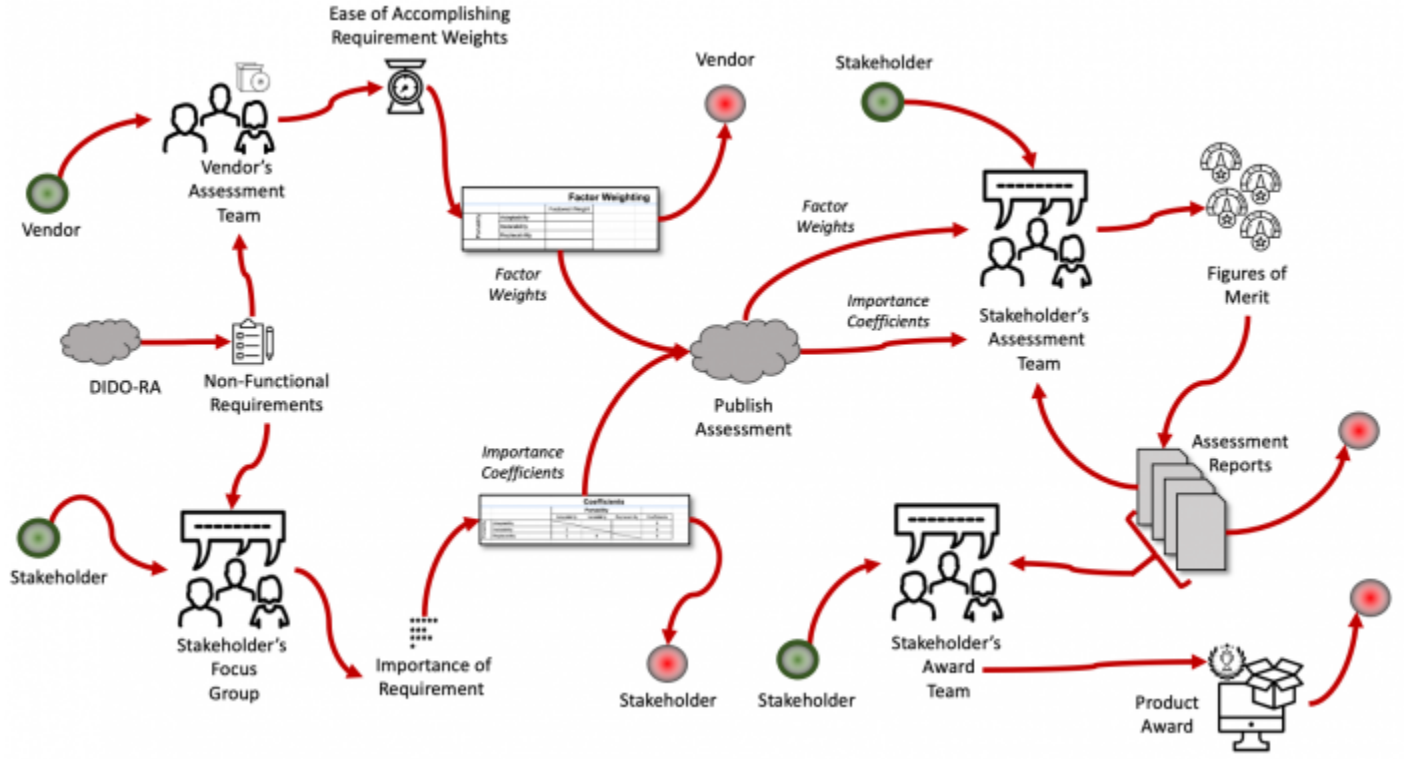


Figure 53: An overview of the Assessment Process

The Vendor's Assessment

Return to Top

The Vendor's assessment team must discuss with each other and use the DIDO-RA workbook to determine the weight of the requirement.

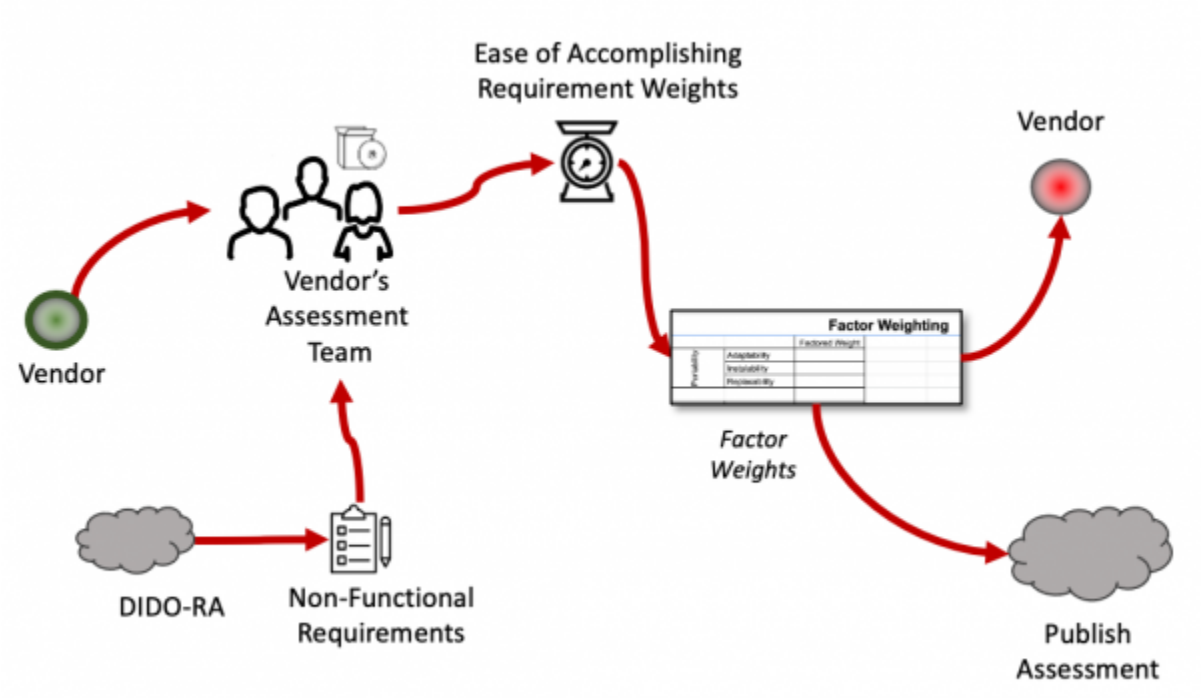
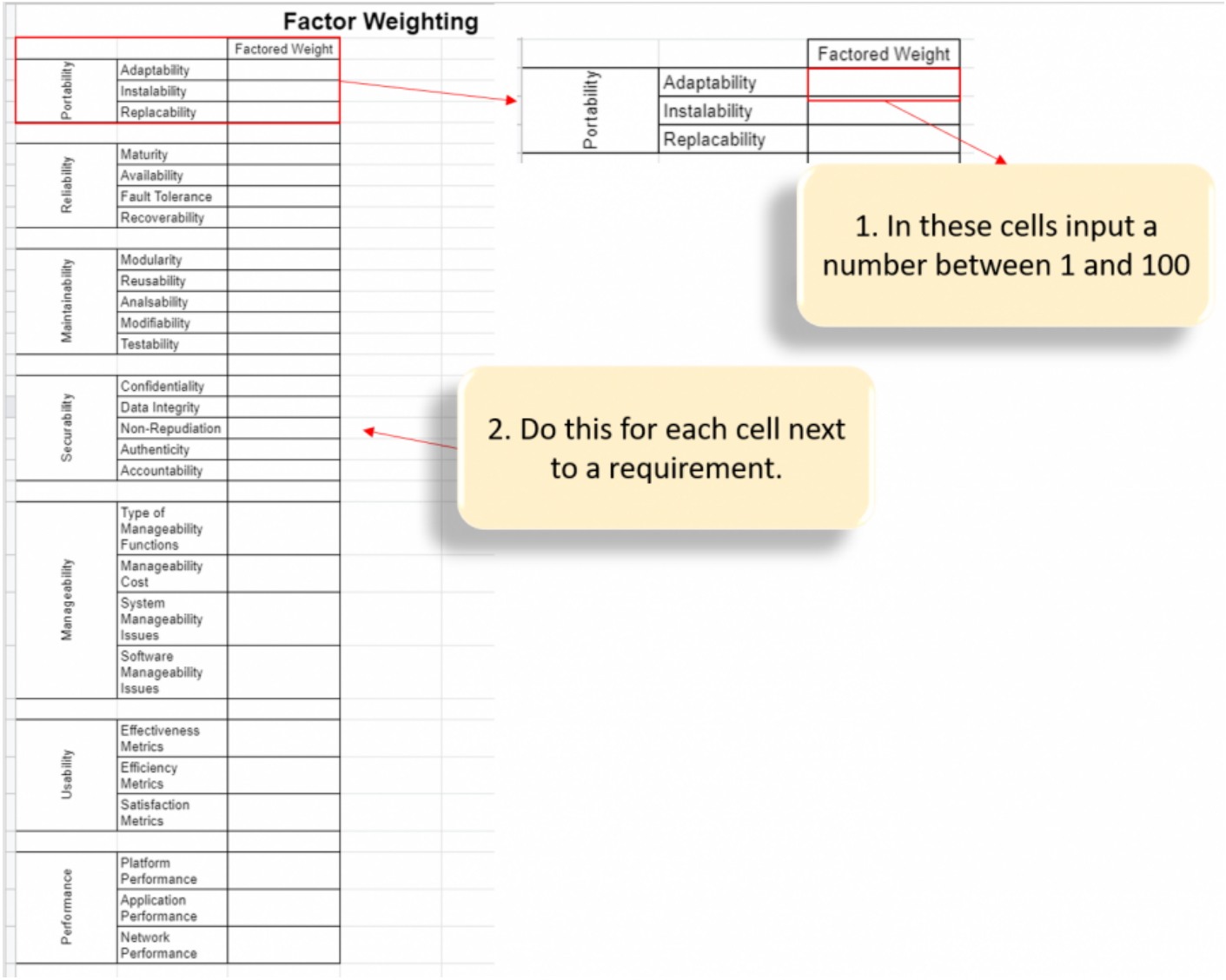
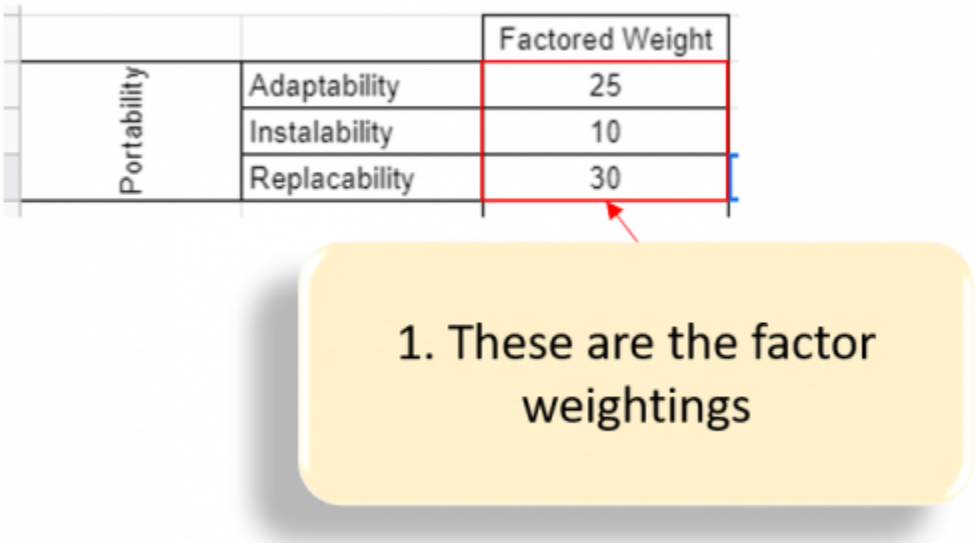


Figure 54: The Vendor's Assessment Team Activities

This is done by weighing each property, related to the Requirement, by giving a score between 1-100. 1 representing an easy development process, and 100 being impossible now, merely very difficult in a couple years.





Publish the assessment, and notify the vendor..

The Stakeholder's Focus Group

Return to Top

The Vendor's stakeholder assessment team must create a focus group of stakeholder's. This group is used in conjunction with the stakeholder assessment team to determine the importance of the Requirement and potentially direct more or less resources to this area.

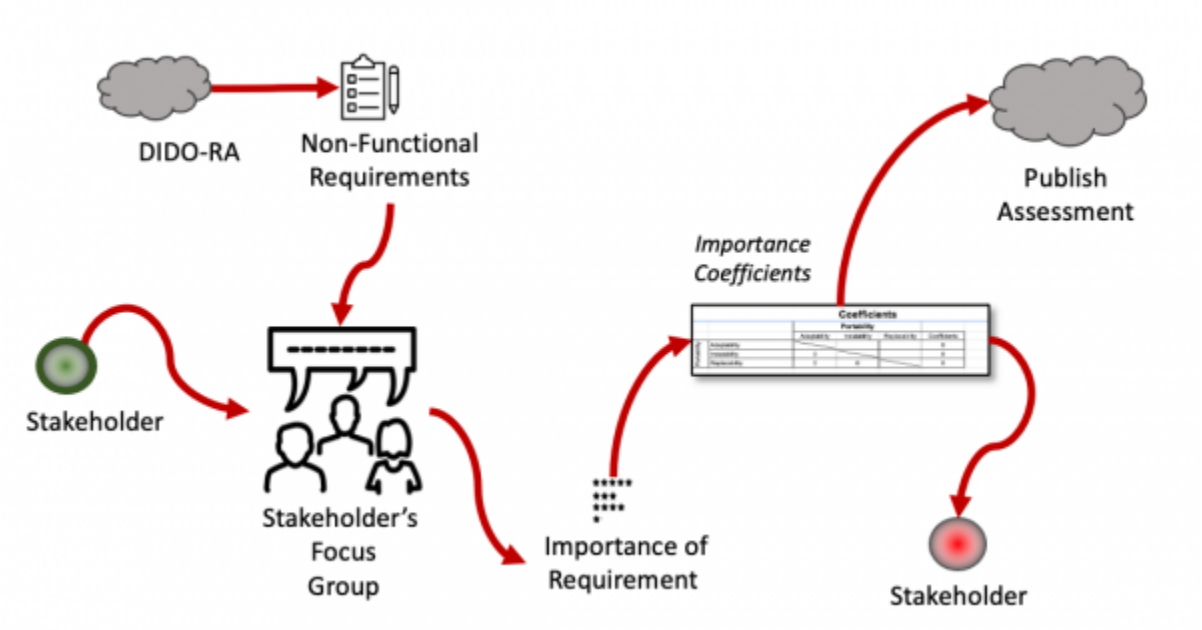
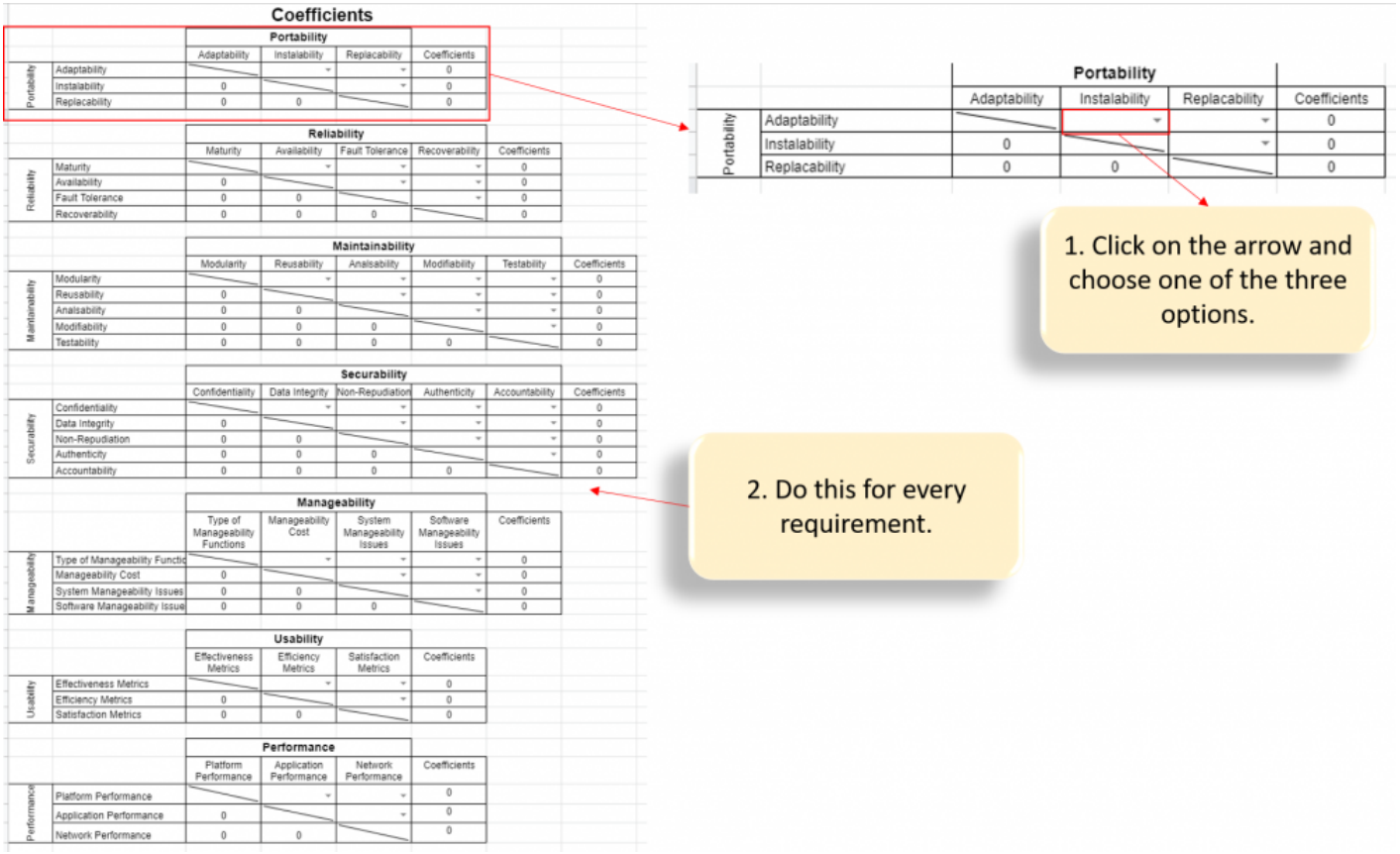


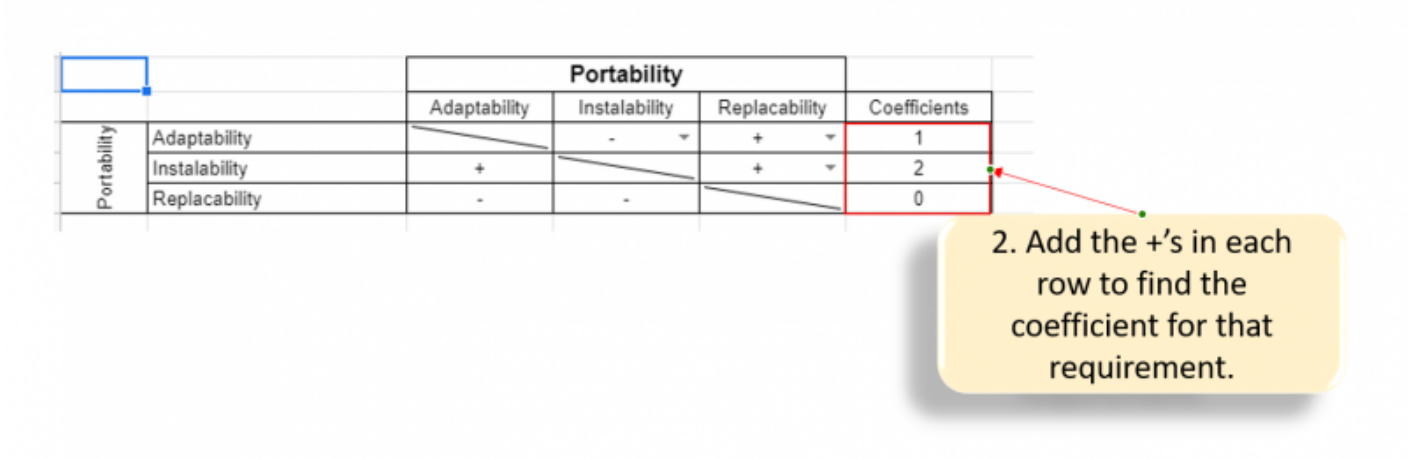
Figure 57: The Stakeholder's Focus Group Activities

This is done by rating each property related to the requirement with either a '+' option representing a 'more important' status, or a '-' option representing a 'less important' status. On the Coefficient sheet choosing a '+' will cause a '-' to be put in the opposite cell in the table. EX: cell(1,2)='+' means

cell(2,1)='-'. Also properties can't be compared to themselves, these cells have lines drawn through them to represent this.



At the end of filling out each table. Each row is counted for '+', the number of '+' in a row is equal to the coefficient for that property.



When completed publish the assessment and notify the stakeholder's.

The Stakeholder's Assessment Team

Return to Top

Once the assessments and importance coefficients are set, the DIDO-RA is passed to the stakeholder's assessments team. Here they must take the factor weighting and the coefficients to create the Figure of Merit (FoM). These figures are then published to assessment reports that are given to the Vendor, and/or given back to the stakeholder's assessment team for further review.

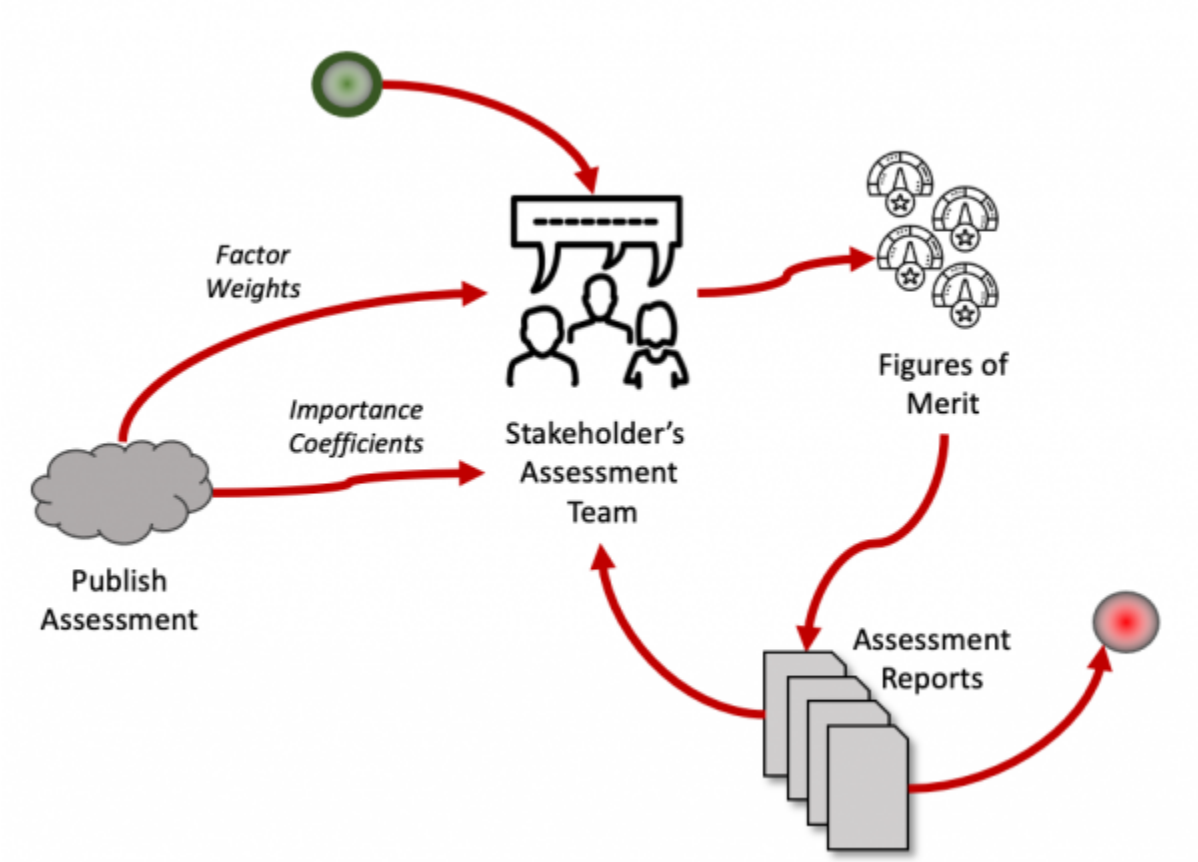
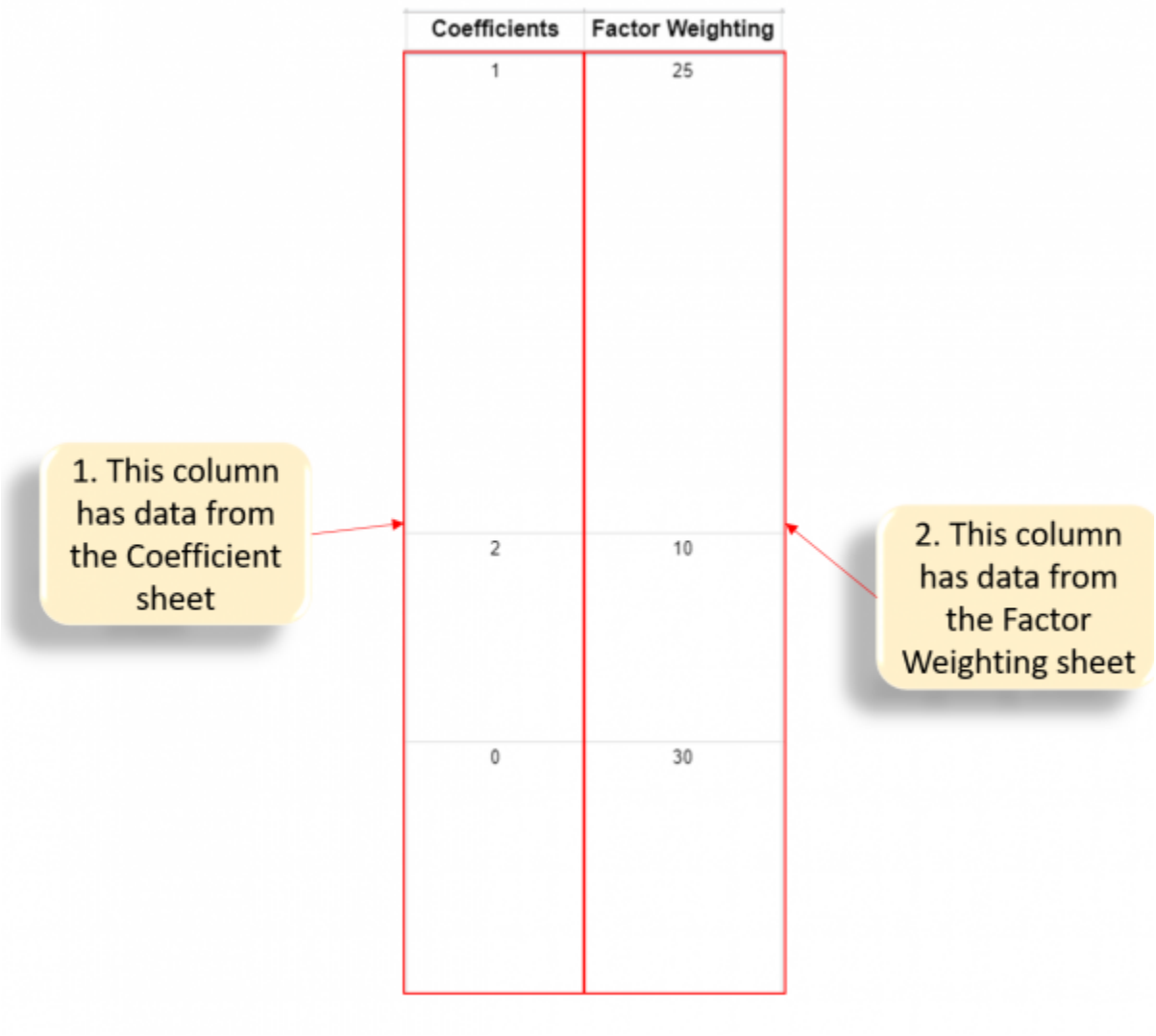
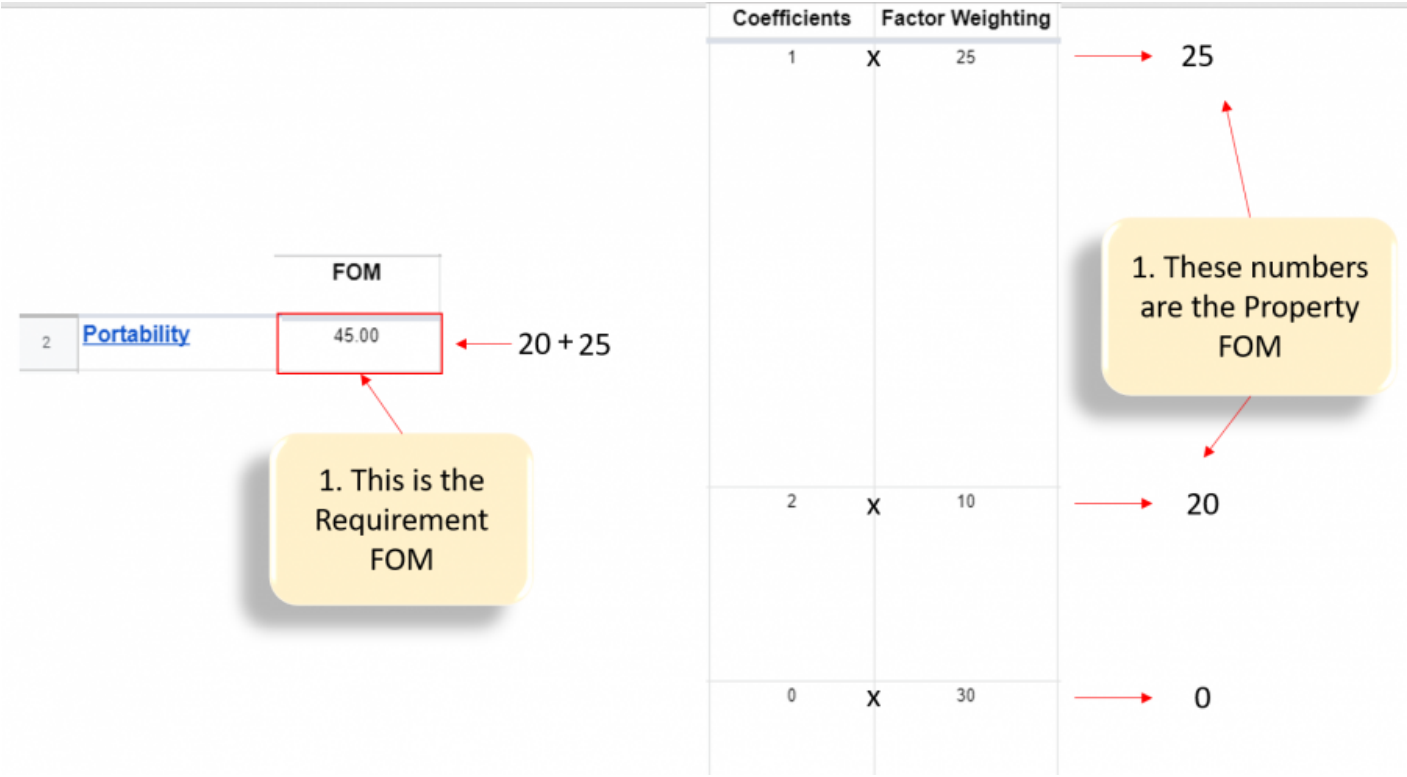


Figure 60: The Stakeholder's Assessment Team Activities

In the DIDO-RA workbook after the Coefficient/Factor Weight sheets are filled out, those values associated with the requirement, are then sent to the corresponding Requirement Sheets in the workbook.



This data is then used to create the FOM(Figures of Merit). The formula to calculate the FOM = (C1)(FW1) + (C2)(FW2)+ (Cn)(FWn) for each property. These Property FOMs are then added into a Requirement FOM.



NOTE: Comparing Requirement FOMs from one vendor to another will cause a skewed comparison. To compare between vendors you must break the Requirement FOMs into its components. The components of Requirement FOM are the Property FOMs.

The Stakeholder's Award Team

Return to Top

The stakeholder's receive the Assessment Reports as well, if they deem the results satisfactory they will choose the product just reviewed.

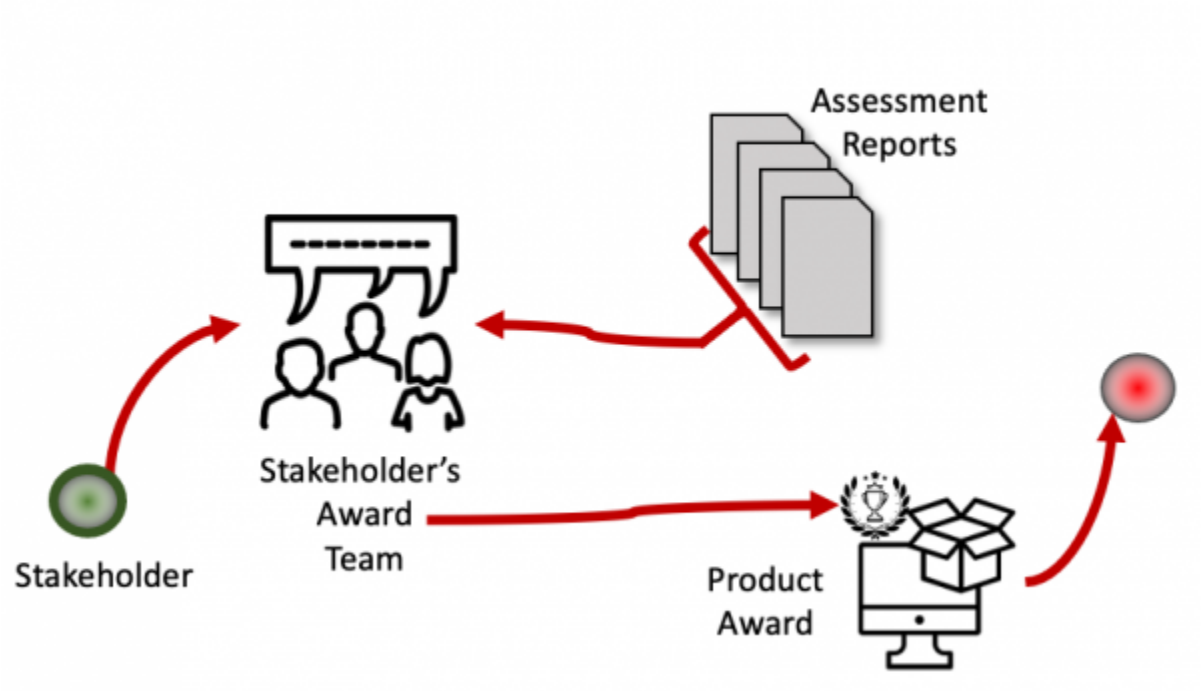


Figure 63: The award is usually your software is chosen to be used.

Appendices

Return to Reference Architecture (RA)

- Appendix A: Glossary of Terms Related to DIDO
- Appendix B: Standards Organizations
- Appendix C: Hardware Architectures
- Appendix D: Operating Systems
- Appendix E: Tools
- Appendix F: DDS Quality Of Service
- Appendix G: Tests
- Appendix H: Acronyms
- Appendix I: Cognitive Model
- Appendix J: Governance Model

Appendix A: Glossary of Terms Related to DIDO

Return to Reference Architecture (RA) or Return to Appendices

The following terms are used by DIDO in various documents, web pages, etc.

0-9	!-*	A	B	C	D	E	F	G	H
I	J	K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z		

Appendix B: Standards Organizations

Return to Reference Architecture (RA) or Return to Appendices

The DIDO RA recognizes two categories of standard bodies:

- Technical Standards Bodies
- de facto Standard Bodies

Technical Standards Bodies

return to Standards Area

Technical standards are developed by Standards Organizations [261]. Standards organizations are organizations whose primary activities are to develop, coordinate, promulgate, revise, amend, reissue, interpret, or otherwise produce technical standards intended to address the needs of a group of affected adopters.[262]

The DIDO RA provides data sheets for technical standards developed by the following organizations:

- Apache Software Foundation (ASF)
- ECMA International
- Institute of Electrical and Electronics Engineers (IEEE)
- Internet Engineering Task Force (IETF)
- International Organization for Standardization (ISO)
- International Telecommunications Union (ITU)
- National Institute of Standards and Technology (NIST)
- Organization for the Advancement of Structured Information Standards (OASIS)
- Object Management Group (OMG)
- Open Source Initiative (OSI)
- World Wide Web Consortium (W3C)

261)
also known as standards body, Standards Developing Organization (SDO), or Standards Setting Organization (SSO)

262)
Intercloud: Solving Interoperability and Communication in a Cloud of Clouds, Appendix A, by Ken Owens, Monique Morrow, Venkata Josyula, Jazib Frahim, Publisher: Cisco Press Release Date: June 2016 ISBN: 9780134189239,https://learning.oreilly.com/library/view/intercloud-solving-interoperability/9780134189239/app01.html

de facto Standards Bodies

return to Standards Area

A *de facto* Standard is something that is used so widely that it is considered a standard for a given application although it has no official status. [263)] A *de facto* Standard is generally controlled by a corporation (Microsoft, IBM, Google, Amazon, etc) or group (Apache, Linux, Mozilla, etc).

The DIDO RA provides data sheets for de facto standards developed by the following:

Corporate Projects

- Amazon
- Apache Software Foundation (ASF)
- Apple
- Bitcoin
- Consortium for Information & Software Quality (CISQ)
- Ethereum
- Google
- IOTA
- Linux Foundation
- Microsoft
- Oracle
- Talk Openly Develop Openly (TODO)

Individual Projects

- GIT (Revision Control)
- InterPlanetary File System (IPFS)
- Jenkins (Continuous Delivery)
- Jira (Bug tracking system)
- Participating in Open Source Communities
- ZeroMQ Distributed Messaging
- ZeroMQ Message Transport Protocol (ZMTP)

263)

https://whatis.techtarget.com/definition/de-facto-standard

Appendix C: Hardware Architectures

Return to Reference Architecture (RA) or Return to Appendices or 4.2.1.1 Hardware Platform

Gezelter[264] defined Hardware architecture as follows:

> *"Computer architecture" refers to the underlying physical and logical structures of the computer system. In the case of a computer system, this includes instruction set, numeric sizes and representations, and how the system connects to external devices (interrupts or polling).*
>
> *While in the theoretical sense, most architectures are functionally equivalent, it does not mean that some architectural choices make some applications and algorithms easier or more efficient to implement. An architecture without support for floating point can be extended in software to perform the needed functions, but it will generally be slower than a hardware implementation. In the interfacing device area, interrupts make it more efficient to overlap device operation with computing. Such operations can be done without interrupts, but it is far more complex.*
>
> *In summary, computer architecture defines the underlying structure of the computing system. It governs how the various elements interact.*

The following Hardware Architectures need to be considered when designing distributed systems. It is not important that every distributed system supports all these kinds of architectures, but that the inclusion and elimination of some of the architectures is an overt act. The Goal during the functional requirements specification process is to establish the subset of Hardware Architectures supported. This does not mean that the set is static and, thus, cannot be changed. It does mean; however, that changes require careful consideration and planning.

- C.1 Embedded Systems
- C.2 Servers
- C.3 Desktops
- C.4 Handheld Computers
- C.5 Supercomputers
- C.6 Network Devices

264)
Gzelter, Robert; The Graduate Center, CUNY, Quora, 2018, Accesssed 8 December 2020, https://www.quora.com/What-is-the-importance-of-computer-architecture-in-a-computer-system

Appendix D: Operating Systems

Return to Reference Architecture (RA) or Return to Appendices

Create a Operating System datasheet for **OS Name** →		Add page

- Android
- Apstra
- Azure Real Time Operating System (or Azure RTOS)
- Azure Sphere OS
- balenaOS
- Blackberry QNX
- CentOS
- Chromium OS
- Cisco Digital Network Architecture (Cisco DNA)
- Cisco Internetwork Operating System (IOS)
- Cisco IOS XR
- Cisco NX-OS
- ClearOS
- CloudReady
- ExtremeXOS
- FreeBSD
- FreeRTOS
- IBM i
- iOS
- Junos operating system (Junos OS)
- LynxOS RTOS
- Nokia X Software Platform
- Open Network Linux
- OpenServer
- Oracle Linux (OL)
- Oracle Solaris
- Red Hat Enterprise Linux (RHEL)
- SANtricity Software Operating System (OS)
- SCO UnixWare
- SUSE Linux Enterprise Server (SLES)
- TrueNAS
- Ubuntu Linux
- Windows
- Windows IoT
- Windows NT
- Windows Server
- Windows XP

Appendix E: Tools

Return to Reference Architecture (RA) or Return to Appendices

As with all projects, DIDOs require tools to support their development, management, and operation. The following lists of tools are meant to be informative and by no means exhaustive.

Community Tools

The following tools are used to aid in the development of software, particularly the development of Open Source Software (OSS).

- Tools: Archiving and Release Management
- Tools: Bug and Issue Tracking
- Tools: Code Reviews
- Tools: Contributor License Agreements (CLA)
- Tools: GitHub Management at Corporate Scale
- Tools: Logging Tools
- Tools: Open Source Paradigm
- Tools: Project Quality
- Tools: Source Code Scanning and License Compliance
- Tools: Tracking Project Health

Network Traffic Analysis Tools

- Tools: Network Traffic Analysis

Tools for communications and collaboration

- Tools TBD

Tools for corporate-scale GitHub management

- Tools TBD

Appendix F: DDS Quality Of Service

Return to Reference Architecture (RA) or Return to Appendices

To Be Updated

About

These are the Quality of Service (QoS) parameters defined in the main Data Distribution Service (DDS) specification.

DDS Quality of Service Parameters

The following list of DDS QoS parameters are listed in the order they appear in the DDS Specification rather than in Alphabetical order. (Source: DDS Section 2.2.3 Supported QoS)

1. User Data
2. Topic Data
3. Group Data
4. Durability
5. Durability Service
6. Presentation
7. Deadline
8. Latency Budget
9. Ownership
10. Ownership Strength
11. Liveliness
12. Time Based Filter
13. Partition
14. Reliability
15. Transport Priority
16. Lifespan
17. Destination Order
18. History
19. Resource Limits
20. Entity Factory
21. Writer Data Lifecycle
22. Reader Data Lifecycle

Appendix G: Tests

Return to Reference Architecture (RA) or Return to Appendices

See: OMG: Test Information Interchange Format (TestIF)

Test Name	Description
Acceptance Testing	**Acceptance Testing** is a testing technique performed to determine whether or not the software system has met the requirement specifications. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users. There are various forms of acceptance testing: • User acceptance Testing • Business acceptance Testing • Alpha Testing • Beta Testing Source: https://www.tutorialspoint.com/software_testing_dictionary/acceptance_testing.htm
Black Box Testing	**Black Box Testing** is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products. Black box testing can be done in the following ways: 1. Syntax Driven Testing – This type of testing is applied to systems that can be syntactically represented by some language. For example- compilers and language that can be represented by context free grammar. In this, the test cases are generated so that each grammar rule is used at least once. 2. Equivalence partitioning – It is often seen that many type of inputs work similarly so instead of giving all of them separately we can group them together and test only one input of each group. The idea is to partition the input domain of the system into a number of equivalence classes such that each member of class works in a similar way, i.e., if a test case in one class results in some error, other members of that class would also produce the same error. Source: https://www.geeksforgeeks.org/software-engineering-black-box-testing/
End-to-End Testing (E2E Testing)	**End-to-End Testing (E2E testing)** refers to a software testing method that involves testing an application's workflow from beginning to end. This method basically aims to replicate real user scenarios so that the system can be validated for integration and Data Integrity. Essentially, the test goes through every operation the application can perform; to test how the application communicates with hardware, network connectivity, external dependencies, databases, and other applications. Usually, E2E testing is executed after functional and system testing is complete. Source: https://www.browserstack.com/guide/end-to-end-testing

Test Name	Description
Integration Testing	**Integration Testing** is performed to test individual components to check how they function together. In other words, it is performed to test the modules which are working fine individually and do not show bugs when integrated. It is the most common functional testing type and performed as automated testing. Generally, developers build different modules of the system/software simultaneously and don't focus on others. They perform extensive black box and white box functional verification, commonly known as unit tests, on the individual modules. Integration tests cause data and operational commands to flow between modules which means that they have to act as parts of a whole system rather than as individual components. This typically uncovers issues with UI operations, data formats, operation timing, API calls, database access, and user interface operations. Source: https://www.simform.com/functional-testing-types/
Interface Testing	**Interface Testing** is defined as a software testing type that verifies whether communication between two different software systems is done correctly. A connection that integrates two components is called an interface. This interface in the computer world could be anything like an Application Programming Interface (API), web services, etc. Testing of these connecting services or interfaces is referred to as Interface Testing. An interface is actually software that consists of sets of commands, messages, and other attributes, which enable communication between a device and a user. Source: https://www.guru99.com/interface-testing.html
Regression Testing	**Regression Testing** is re-running tests for Functional Requirements and Non-Functional Requirements to ensure that previously developed and tested software still performs after a change. If not, that would be called a **regression** as far as functionality. Changes that may require regression testing include bug fixes, software enhancements, configuration changes, and even substitution of electronic components. As regression test suites tend to grow with each found defect, test automation is frequently involved. Sometimes a change impact analysis is performed to determine an appropriate subset of tests (non-regression analysis). Source: https://en.wikipedia.org/wiki/Regression_testing
Sanity Testing	**Sanity Testing** is a subset of regression testing. Sanity testing is performed to ensure that code changes are working properly. Sanity testing is a stopgap to check whether testing for a build can proceed or not. The focus of the team during sanity testing process is to validate the functionality of the application, not to perform detailed testing. Sanity testing is generally performed on builds where the production deployment is required immediately, like a critical bug fix. Source: https://www.geeksforgeeks.org/sanity-testing-software-testing/

Test Name	Description
Smoke Testing	**Smoke Testing** is performed on a 'new' build given by developers to the QA team to verify if the basic functionalities are working or not. It is one of the important functional testing types. This should be the first test to be done on any new build. In smoke testing, the test cases chosen cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionality of the system is working fine. Source: https://www.simform.com/functional-testing-types/
Unit Testing	**Unit Testing** ensures that each part of the code developed in a component delivers the desired output. In unit testing, developers only look at the interface and the specification for a component. It provides documentation of code development as each unit of the code is thoroughly tested standalone before progressing to another unit. Unit tests support functional tests by exercising the code that is most likely to break. Source: https://www.simform.com/functional-testing-types/
White Box Testing	**White Box Testing** techniques analyze the internal structures of the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing, clear box testing, or structural testing. Source: https://www.geeksforgeeks.org/software-engineering-white-box-testing/

Appendix H: Acronyms

Return to Reference Architecture (RA) or Return to Appendices

Acronym	Meaning
AON	All-Or-None
API	Application Programming Interface
APP	Application
ASF	Apache Software Foundation
ASIC	Application-Specific Integrated Circuit
BFT	Byzantine Fault Tolerance
BIP	Bitcoin Improvement Proposal
CISQ	Consortium for Information & (or IT) Software Quality
CLA	Contributor License Agreements
CIL	Common Intermediate Language
CLI	Command Line Interface
CLR	Common Language Runtime
CM	Case Management
CM	Configuration Management
CoI	Community of Interest
CPU	Central Processing Unit
ĐApp	Đistributed Application
DApp	Distributed Application
DaaS	Data as a Service
dBFT	Delegated Byzantine Fault Tolerant
DBMS	Database Management Systems
DDS	Data Distribution Service
DIDO	Distributed Immutable Data Objects
DIL	Disconnected, Intermittent and Limited networks
DLT	Distributed Ledger Technology
DM	Data Model
DNS	Domain Name System
DoD	U.S. Department of Defense
DPoS	Delegated Proof of Stake
DSS	Digital Signature Standard
ECMA	European Computer Manufacturers Association
ECMAScript	European Computer Manufacturers Association Scripting Language Specification
ERC	Ethereum Request for Comment
EIP	Ethereum Improvement Proposal
EVM	Ethereum Virtual Machine
FDIC	Federal Deposit Insurance Corporation (FDIC)
FIGI	Financial Instrument Global Identifier

Acronym	Meaning
FOK	Fill-Or-Kill
GDPR	General Data Protection Regulations
GMS	Google Mobile Services
GPU	Graphical Processing Unit
gRPC	Google Remote Procedure Call
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IA	Identification and Authentication
IaaS	Infrastructure as a Service
ICO	Initial Coin Offering
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IIOT	Industrial Internet of Things
IP	Intellectual Property
IP	Internet Protocol
IPFS	Interplanetary File System
IS or InfoSec	Information Security
ISO	International Organization for Standards
IT	Information Technology
ITU	International Telecommunications Union
JiT	Just-in-Time
JVM	Java Virtual Machine
K8	K8s or Kuberenetes
KYC	Know-Your-Customer
LMT	Limit Order
LSB	Linux Standard Base
MA	Mission Assurance
MKT	Market Order
MIT	Market If Touched
MIT	Massachusetts Institute Of Technology
MQ	Message Queue(MQ)
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
ODM	Ontology Definition Metamodel
OMG	Object Management Group
OpenJS	open source JavaScript
OpenMAMA	Open Middleware Agnostic Messaging API
OS	Operating System
OT	Operational Transforms
OSI	Open Systems Interconnection
OSS	Open Source Software

Acronym	Meaning
OWL	Web Ontology Language
P2P	peer-to-peer
P&P	Policy and Procedure
PaaS	Platform as a Service
PIM	Platform Independent Model
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
Protobuf	Google Protocol Buffer
PSM	Platform Specific Model
RA	Reference Architecture
RDF	Resource Description Framework
RFP	Request For Proposal
RFC	Request for Comment
RFI	Request for Information
REST	Representational State Transfer
RDBMS	Relational Database Management Systems
RESTful	RESTful API
RPC	Remote Procedure Call
RSS	Really Simple Syndication
SaaS	Software as a Service
SAML	Security Assertion Markup Language
SCAP	Security Content Automation Protocol
SfA	Safety Assurance
SPDX	Software Package Data Exchange
SPV	Simple Payment Verification
SQuaRE	Systems and software Quality Requirements and Evaluation
STP	Stop Order
STP	Straight-through Processing (StP)
STPLMT	Stop Limit Order
SW	Software
SwA	Software Assurance
SysA	System Assurance
SysML	System Modeling Language
TCP	Transmission Control Protocol
SI	System Integrator
SIG	Special Interest Group
SDO	Standards Developing Organization
SoS	System of Systems
SPARQL	Simple Protocol and RDF Query Language
SSO	Standards Setting Organization
TODO	Talk Openly Develop Openly

Acronym	Meaning
UAF	Unified Architecture Framework
UDP	User Datagram Protocol
UID	Unique Identifier
UUID	Universal Unique Identifier
UML	Unified Modeling Language
VM	Virtual Machine
W3C	World Wide Web Consortium
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language
XSLT	XSL Transformations
ZMTP	ZeroMQ Message Transport Protocol

Appendix I: Cognitive Model

Return to Reference Architecture (RA) or Return to Appendices

- **Note:** The following is based on an Engineering Governance Model developed at US Navy SPAWAR[265].

The Cognitive Model abstractly represents human cognition defined in the dictionary as[266]:

1. *The mental process of knowing, including aspects such as awareness, perception, reasoning, and judgment*
2. *That which comes to be known, as through perception, reasoning, or intuition; knowledge*

Cognition roughly maps to the Information Science and Knowledge Management DIKW (Data, Information, Knowledge and Wisdom) hierarchies[267].

Table 23 highlights the alignment of the two models to each other.

Table 23: Mapping Cognitive Aspects to DIKW Hierarchy

Cognitive Aspects	DIKW Hierarchy
Awareness	Data
Perception	Information
Reasoning	Knowledge
Judgment	Wisdom

In addition to the basic layers in the DIKW Hierarchy, Russell Ackoff[268] and Milan Zeleny[269] propose an additional layer between Knowledge and Wisdom. Ackoff refers to it as *Understanding*. Zeleny adds one more layer above Wisdom called *Enlightenment*. For the purposes of governance, there does seem to be a need for an *Understanding Layer* to the hierarchy. However, adding an *Enlightenment Layer* when referring to governance always seems to elicit smiles.

The result is termed the Cognitive Model instead of the original DIKW (or DIKUW) Hierarchy for a several reasons. The word hierarchy implies an order or precedence and this hierarchy always starts with data. This is a useful concept when thinking in terms of Information Science and Knowledge Management, which generally tries to organize and classify large amounts of data and extract *wisdom* or, in Zeleny's case, even *enlightenment*. In governance, the hierarchy is applicable in both directions (i.e., from *Wisdom* to *Data* and from *Data* to *Wisdom*). Another problem with the hierarchical approach is that although the relationship of data to wisdom, in some cases, is many-to-one (i.e., many pieces of data contribute to a single piece of wisdom), the reality is that the relationship is more of a network where one piece of data may ultimately be part of many pieces of wisdom.

The acronym DIKW (or DIKUW, etc.) is not very pronounceable and the term specifically captures the model as we currently understand it. Consequently, as our understanding of the model evolves, as with the acceptance of having an *Understanding Layer*, the name of the model must also change.

The Cognitive Model is depicted below in Figure 32. It has the five layers of the original DIKUW Hierarchy, and this view is from Wisdom to Data. However, the model could just as well be presented in the reverse,

starting with Data and ending in Wisdom. The directionality through the model is inconsequential and reflects the higher level human cognitive process.

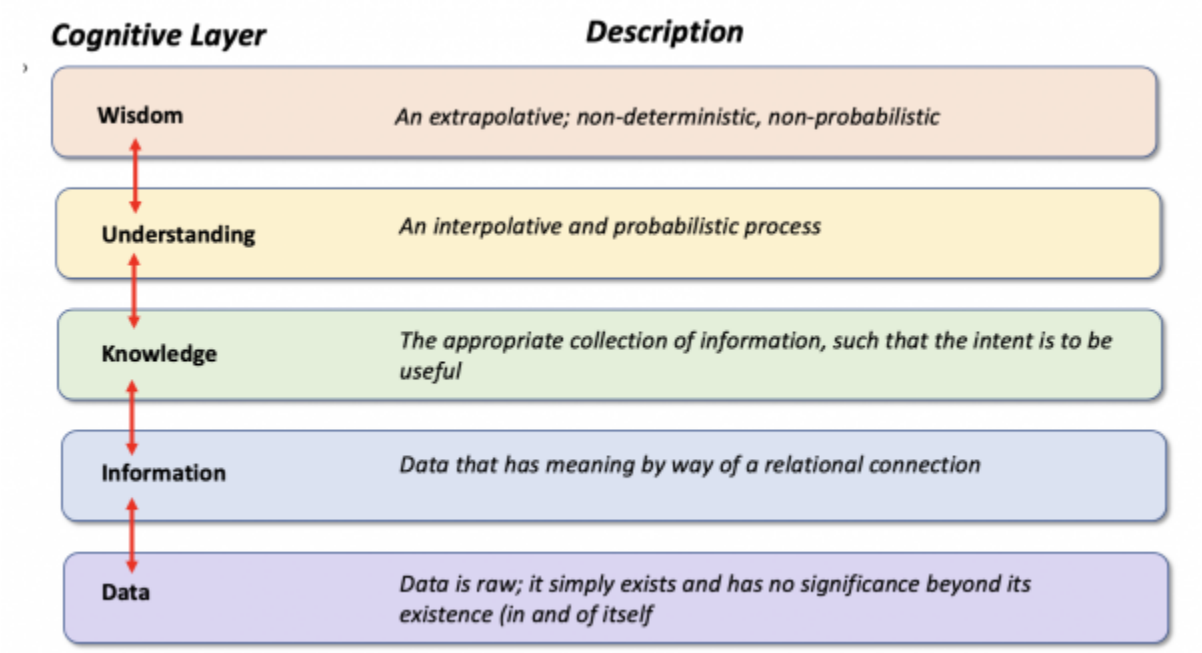


Figure 64: Cognitive Model

Bottom-Up Cognitive Model Example

Return to Top

The simple Bottom-Up Cognitive Model example presented in Figure 65 illustrates how bottom-up cognition applies in our lives, usually as part of analytical processes. It is bottom-up because the process described starts with Data and ends with Wisdom. At the Cognitive Data Layer, a temperature of 100° means little. Adding that the temperature is in degrees Fahrenheit provides a bit more data; however, it still has little relevance until the temperature is put in the context of a person's temperature and becomes Information. Adding that information with other information like the normal temperature for a person is 98.6° Fahrenheit starts to provide us some Knowledge of the situation. This knowledge, combined with other knowledge, allows us to understand that the person has flu. The final step is adding this knowledge with what we already know about the individual allowing a decision that the temperature is not serious and that the solution is to take two aspirin and call the doctor in the morning if symptoms persist. In reality, there is more data than information, more information than knowledge, etc.

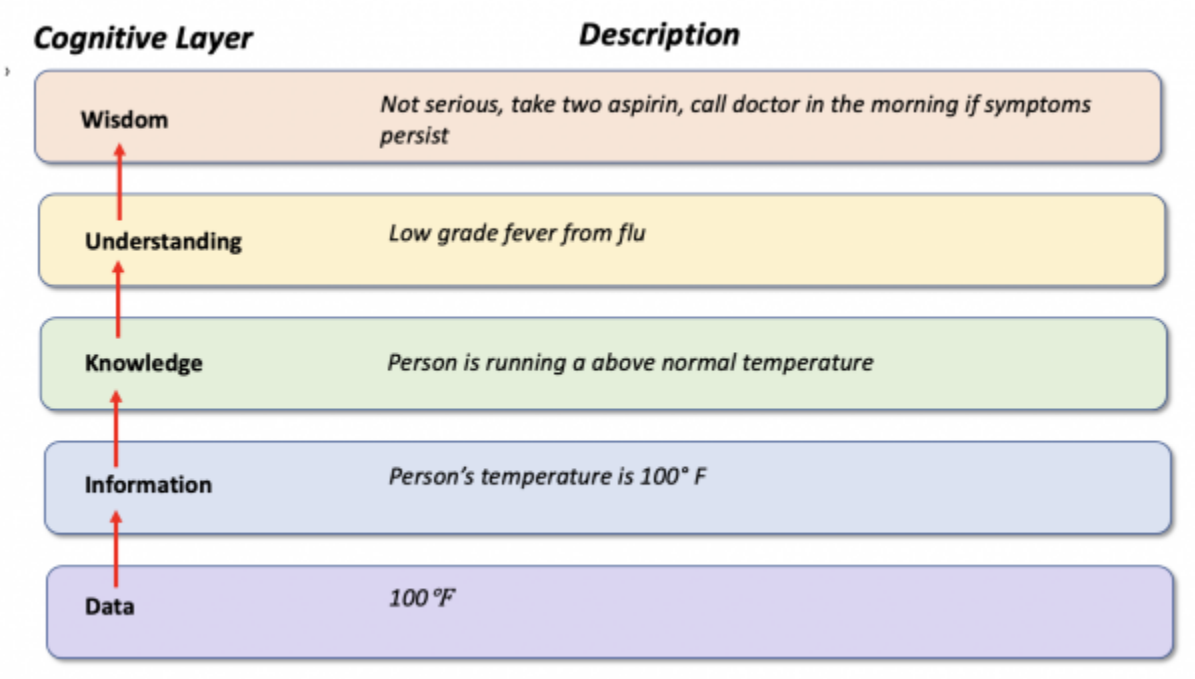


Figure 65: Example of Cognitive Model – Data to Wisdom

Top-Down Cognitive Model Example

Return to Top

The simple Top-Down Cognitive Model example presented in Figure 66 illustrates how top-up cognition applies in our lives, usually as part of educational or regulatory processes. It is top-down because the process described starts with Wisdom and ends with Data. At the Cognitive Wisdom Layer, there needs to be a uniform policy to protect people at risk from influenza. To support this policy (i.e., Wisdom), there are many different kinds of things to understand, such as people can be immunized against flu. As a part of the Understanding, there is Knowledge that vaccines are made from eggs. This leads to the need to disseminate Information that people who are allergic to eggs can not use the vaccine and ultimately the collection of Data (i.e., evaluation criteria) about egg allergies from people receiving the vaccine.

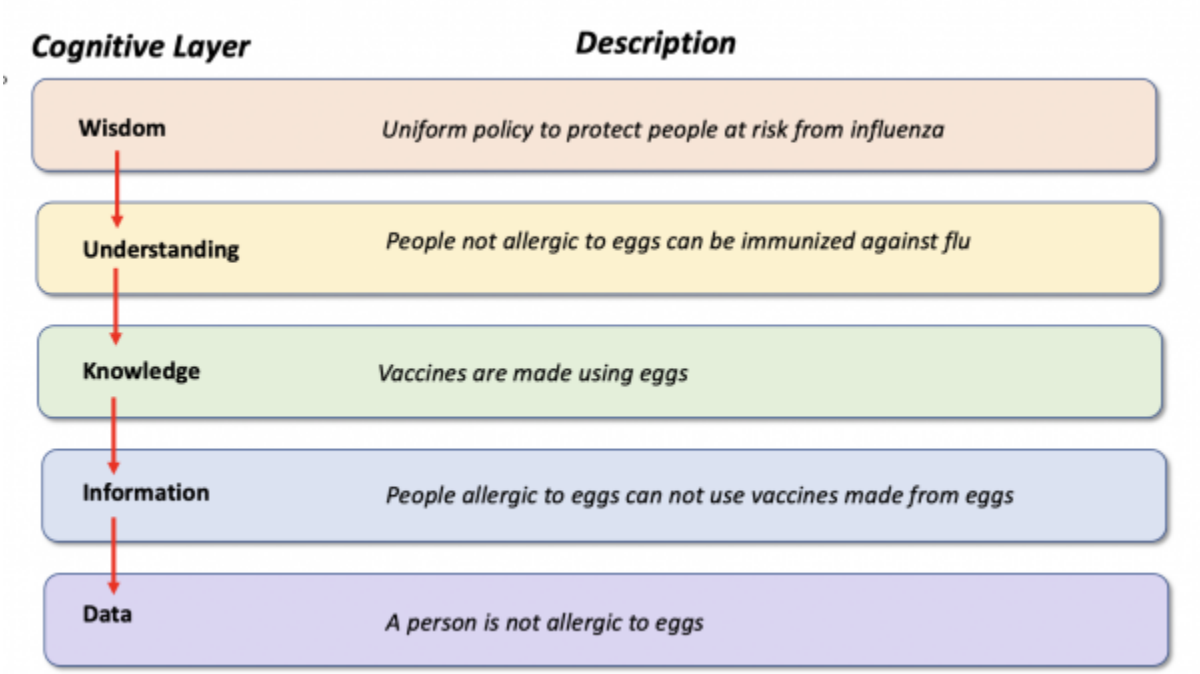


Figure 66: Example of Cognitive Model – Wisdom to Data

265) Stavros, Robert W. and Albrant, Jeremiah; Engineering Governance, SPAWAR, October 9, 2007,

266) ((American Heritage Dictionary, Accessed 10 February 2021, http://dictionary.reference.com/help/ahd4.html

267) Nikhil Sharma, The Origin of the "Data Information Knowledge Wisdom" Hierarchy, 2008 https://www.researchgate.net/publication/292335202_The_Origin_of_Data_Information_Knowledge_Wisdom_DIKW_Hierarchy

268) Ackoff, Russell; 1981, pp. 15-16

269) Zeleny, M., ed. (1981) Autpoiesis: A theory of living organization, vol.3, New York, New York, American Elsevier

Appendix J: Governance Model

Return to Reference Architecture (RA) or Return to Appendices

Fundamental Governance Model

- **Note:** The following is based on an Engineering Governance Model developed at US Navy SPAWAR[270)].

Governance is not a synonym for government or for regulations; rather, governance is the *process* governments use to *interpret* and *use* regulations.

> *Governance is that separate process or certain part of management or leadership processes that make decisions that define expectations, grant power, or verify performance. Frequently a government is established to administer these processes and systems.* [271)]

This definition offers three aspects as to what comprises governance:

- Making decisions that define expectations
- Granting power
- Verifying performance

The first aspect of governance conveys regulation, the second aspect conveys execution, and the third aspect conveys compliance, as represented in the following model.

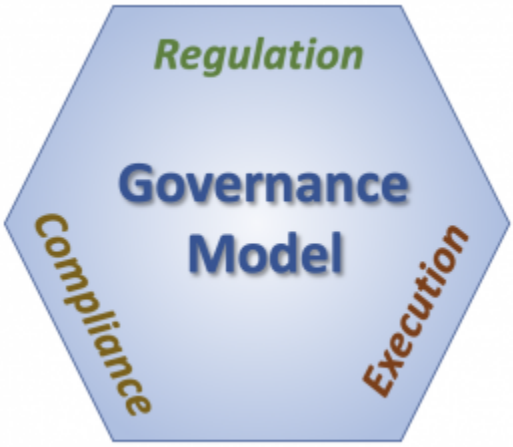


Figure 67: General Governance Model

Good governance is comprised of all three components in equal and sometimes opposing aspects. It is meaningless to have regulation without execution or execution without compliance. In other words, regulation indicates what needs to be done, execution is actually doing it, and compliance is making sure it is done correctly.

Regulation

Return to Top

Regulations are formal, codified authoritative rules. They are adopted by a public regulatory agency and are usually interpretations of statutes passed by a legislative body.

A regulation as a legal term is a rule created by an administration or administrative agency or body that interprets the statutes setting out the agency's purpose and powers, or the circumstances of applying the statute. A regulation is a form of secondary legislation which is used to implement a primary piece of legislation appropriately, or to take account of particular circumstances or factors emerging during the gradual implementation of, or during the period of, a primary piece of legislation.[272)]

Figure 68: Regulation Aspect of Governance Model

An example of the regulatory aspect of governance is a body that creates statutes such as the U.S. Congress or a state legislature, an agency that creates or enforces statutes such as the U.S. Internal Revenue Service (IRS), or civilian or commercial groups that create and promote standards such as the Object Management Group (OMG).

Compliance

Return to Top

Compliance ensures that regulations are met through observation, measurement or testing. Good governance cleanly and effectively separates the responsibility for creating regulations from the enforcement of these regulations. This does not mean that regulations can be developed in a vacuum; they must be written to be enforceable using objective measures of compliance. Therefore, the line between regulation and compliance is not fixed and rigid but needs to be negotiated with validation of regulations from the compliance aspect.

Figure 69: Compliance Aspect of Governance Model

Examples of the Compliance Aspect of Governance is the auditing functionality of the IRS and the independent verification and validation (IV&V) functionality within the DoD.

Execution

Return to Top

Execution is the aspect of governance charged with actually fulfilling formal, codified authoritative rules specified by regulation to those specifications provided by compliance. The responsibility for executing the regulation rarely, if ever, falls on the legislative body or those responsible for enforcing compliance with the regulation. Without execution, the other aspects of Governance are meaningless. Consequently, any discussion of governance must include the Execution Aspect.

Figure 70: Execution Aspect of Governance Model

Examples of the Execution Aspect are the individuals that file their tax forms with the IRS and the personnel who actually create the functionality needed by a DoD Program, Project, or Initiative.

270)

Stavros, Robert W. and Albrant, Jeremiah; Engineering Governance, SPAWAR, October 9, 2007,
271)

Adapted from Wikipedia: Governance; accessed 9 July 2007
272)

Adapted from Wikipedia: Regulation, accessed 9 July 2007 https://en.wikipedia.org/wiki/Regulation]]