4.2.7 Manageability

Return to Non-Functional Requirements

- ☐ **[char]Please Review**

Manageability is most important during the second half of a 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 hard to manage is described as fragile since the smallest of changes can have dire consequences on the systems functionality.

> *Manageability directly influences a system's reliability, availability, security, and safety, thus being 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 to 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.* [1)]

Types of Manageability Functions

Return to the Top

NI (formerly National Instruments) defines four basic manageability functions [2)]:

Table 1: 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 might also metrics on the system as a collection of Nodes. • 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.

Kinds of Management	Description
Configuration and Control	During the startup and sometimes during operations, the systems needs to obtain configuration details and provide control over to the systems 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, 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.
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	

[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.*

Manageability Costs

Return to the Top

> *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 [3].

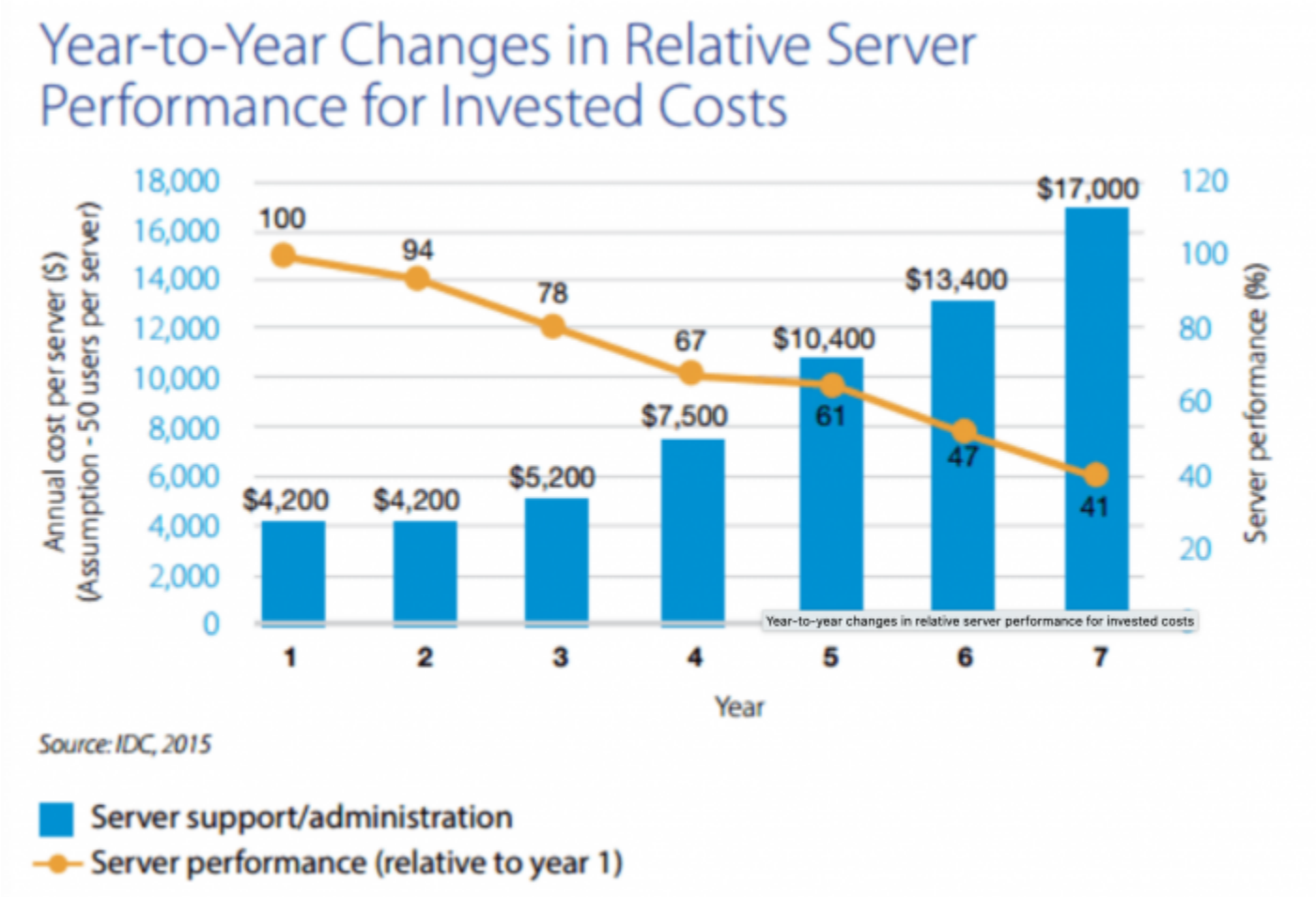

Figure 1: Year-to-Tear Changes in Relative Server Costs

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 2 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.

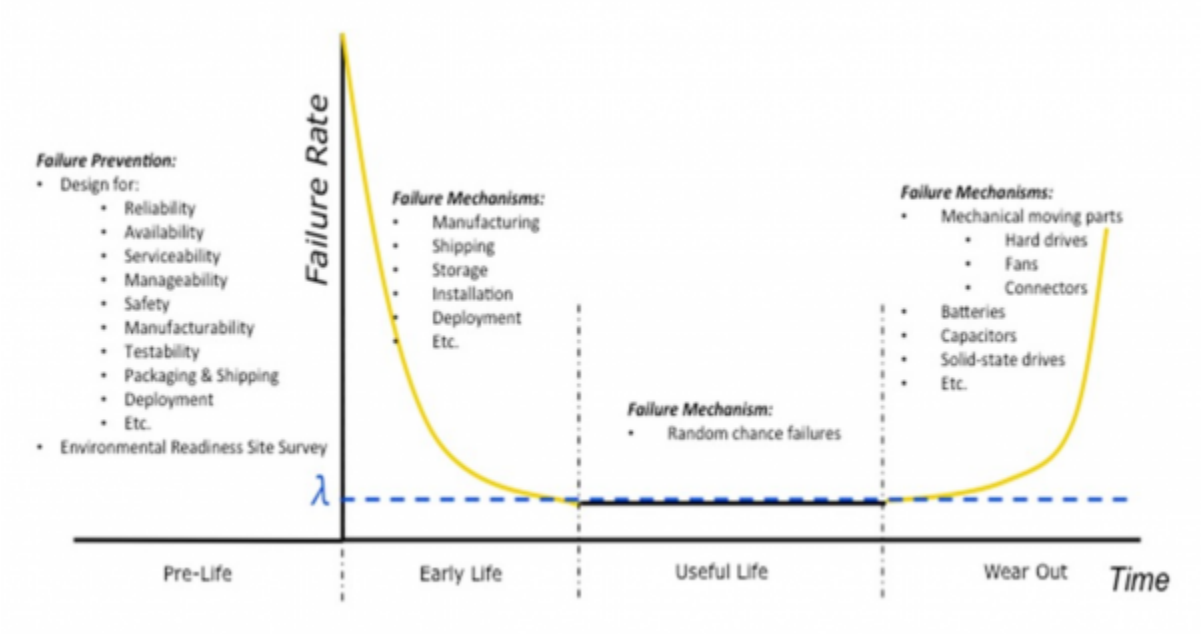

Figure 2: The Bathtub Curve [1)]

System Manageability Issues

Return to the Top

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 that for extremely large complex programs (i.e., over a million Lines of Code – LOC) the average climbs as high as 12-14 years.”[4)] 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.[5)] and the current version is 10. About every 10 years, Microsoft release another major release of Windows. [6)]. 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[7)]. 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 3 developed by the Industrial Internet Consortium[8] 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 managed. Each of these components has its own unique System Lifecycle which evolves independently of the target system.

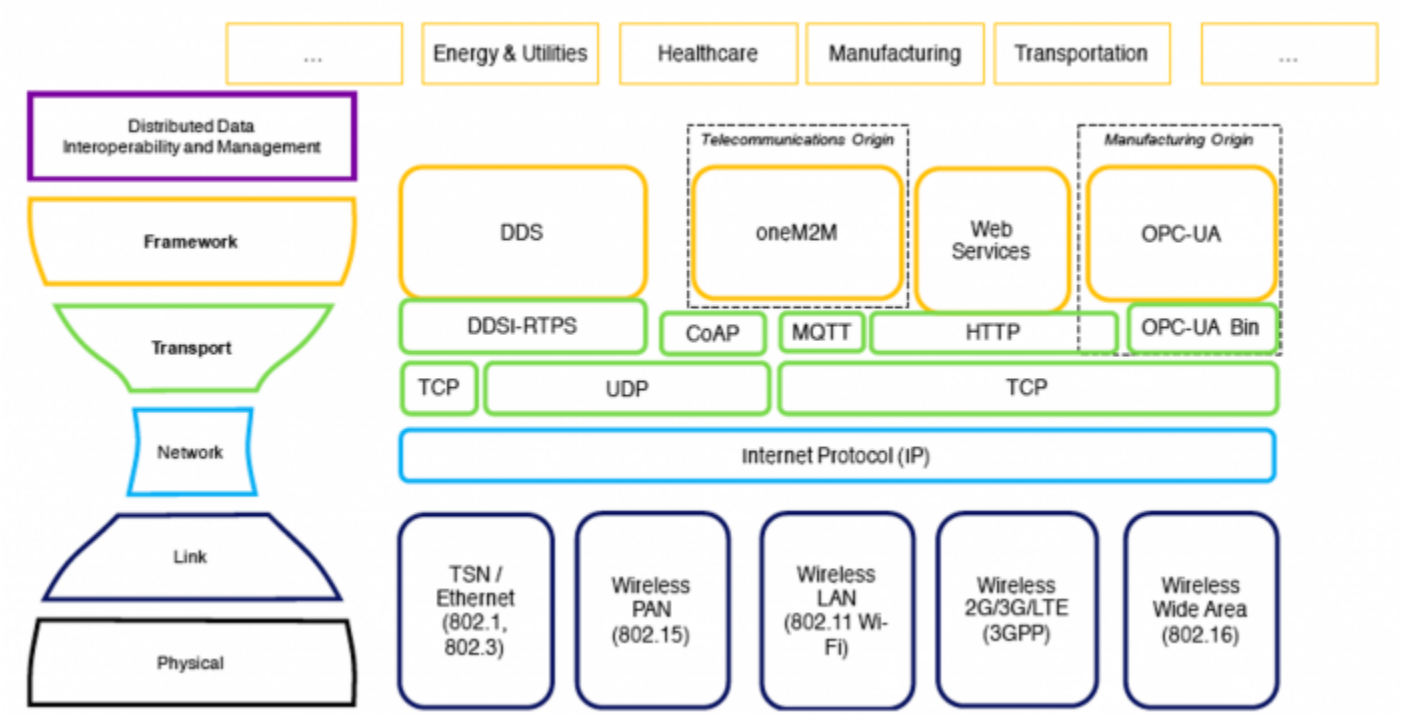

Figure 3: The Industrial Internet Consortium's Connectivity Framework layer provides the foundation for the interoperable transfer of common structured data across systems and domains [7]

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[9].

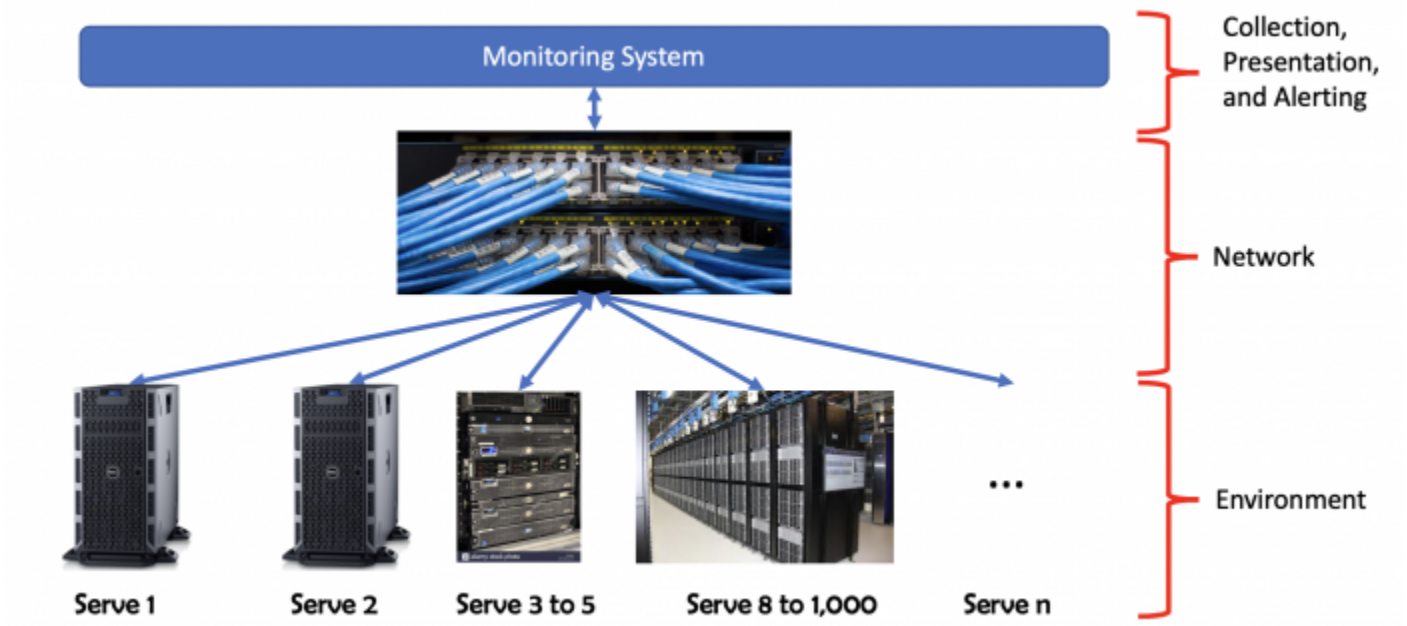

Figure 4: Three major monitoring layers representing sample design of monitoring solution

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 a application, a system, network. This allows for tracking of interactions using messages. Some of the commonly used logging levels are:

Table 2: Some common logging levels used in applications[10)]

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 requirements analysis through to sun-setting the system. However, in a distributed system, these tools can play a significant role in determining the health of the each of the nodes, the network and the overall system of nodes.

Although the publication on Software Metrics for Predicting Maintainability [11] 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 3: 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[12]. There are any spinoffs and derivatives of ZeroMQ. Here are a few reported in Wikipedia[13]

- *In 2012, two of the original developers forked ZeroMQ as Crossroads I/O.*[14)15)]
- *Martin Sustrik has started nanomsg,*[16)] *a rewrite of the ZeroMQ core library.*[17)]
- *In 2012, Dongmin Yu announced his pure Java conversion of ZeroMQ, JeroMQ.*[18)]
- *This has inspired further full-native ports of ZeroMQ, such as NetMQ for C#*[19)]*.*
- *March 2013, Pieter Hintjens announced a new draft of the ZMTP wire-level protocol bringing extensible security mechanisms to ZeroMQ*[20)]*.*
- *Martin Hurton implemented the CurveZMQ authentication and encryption mechanism*[21)] *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 chose from.

Software Manageability Issues

Return to the Top

Over the last few decades, many advances have been made in terms of Open Source Software (OSS) which has to change the way that software was developed, released and used. During this same 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, its 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 [22)]. 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

DDS Specifics

Return to the Top

Data Distribution Service (DDS) can not solve all of a systems(or projects) Maintainability issues, however, by DDS's design, it can eliminate or reduce the Manageability issues that could arrise from using DDS.

Table 4: DDS role in helping Manageability

Kinds of Management	Description
Health Monitoring, Logging, and Alerting	Although there are currently no DDS standards for that directly supports System Monitoring, each of the DDS Vendors have sets of tools which can be used for that purpose. These tools include • Development and troubleshooting including specialized network sniffers, modeling tools, and code generators • System monitoring and administration including terminals, shared memory management tools, recorders and replayers • Functional, systems and performance testing • Federated Discovery • Bridges to other Message-Oriented Middleware (MOM) products • Topic aggregators
Configuration and Control	DDS uses a standardized Discovery process which eliminates most of need for configuration. Some DDS Vendors offer specialized or advanced tools that aid in tuning DDS configurations and discovering performance issues
Deployment and Updates	DDS Extensible and Dynamic Topic Types for DDS (DDS-XTypes) allows for planned evolution of the Datatypes within a ddsapplication. For example, adding or removing fields in a Data Structure, changing the basic type from an int16 to an int32, etc.
Asset Discovery and Inventory	DDS automatically registers all Data Writer and Data Reader allowing them to be discovered. It is possible

1)

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
2)

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
3)

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)

Software Evolution, Blog Post, Mitopia Technologies, https://mitosystems.com/software-evolution/
5)

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
6)

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/
7)

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/
8)

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/
9)

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
10)

Log4j - Logging Levels, Log4J, Accessed 18 July 2020, https://www.tutorialspoint.com/log4j/log4j_logging_levels.htm
11)

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
12)

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
13)

ZeroMQ, Wikipedia, Accessed 20 July 2020, https://en.wikipedia.org/wiki/ZeroMQ
14)

ZeroMQ and Crossroads I/O: Forking over trademarks. LWN.net. Retrieved 14 July 2012.https://lwn.net/Articles/488732/

15)

Crossroads I/O. Retrieved 14 July 2012, http://www.crossroads.io/

16)

nanomsg. Retrieved 8 June 2013 http://nanomsg.org/

17)

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

18)

jeromq - java pojo zeromq, zeromq-dev mailing list, Retrieved 23 May 2013, http://lists.zeromq.org/pipermail/zeromq-dev/2012-August/018265.html

19)

NetMQ, GitHub, Retrieved 23 May 2013, https://github.com/zeromq/netmq

20)

Securing ZeroMQ: draft ZMTP v3.0 Protocol, Hintjens.com, Retrieved 23 May 2013, http://hintjens.com/blog:39

21)

CurveZMQ - Security for ZeroMQ , CurveZMQ, Accessedd 20 July 2020, http://curvezmq.org/

22)

Tools for managing open source programs, Talk Openly Develop Openly (TODO), Accessed 20 July 2020, https://todogroup.org/guides/management-tools/