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."[1)] 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.[2)] and the current version is 10. About every 10 years, Microsoft release another major release of Windows. [3)]. 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[4)]. 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 1 developed by the Industrial Internet Consortium[5)] 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.

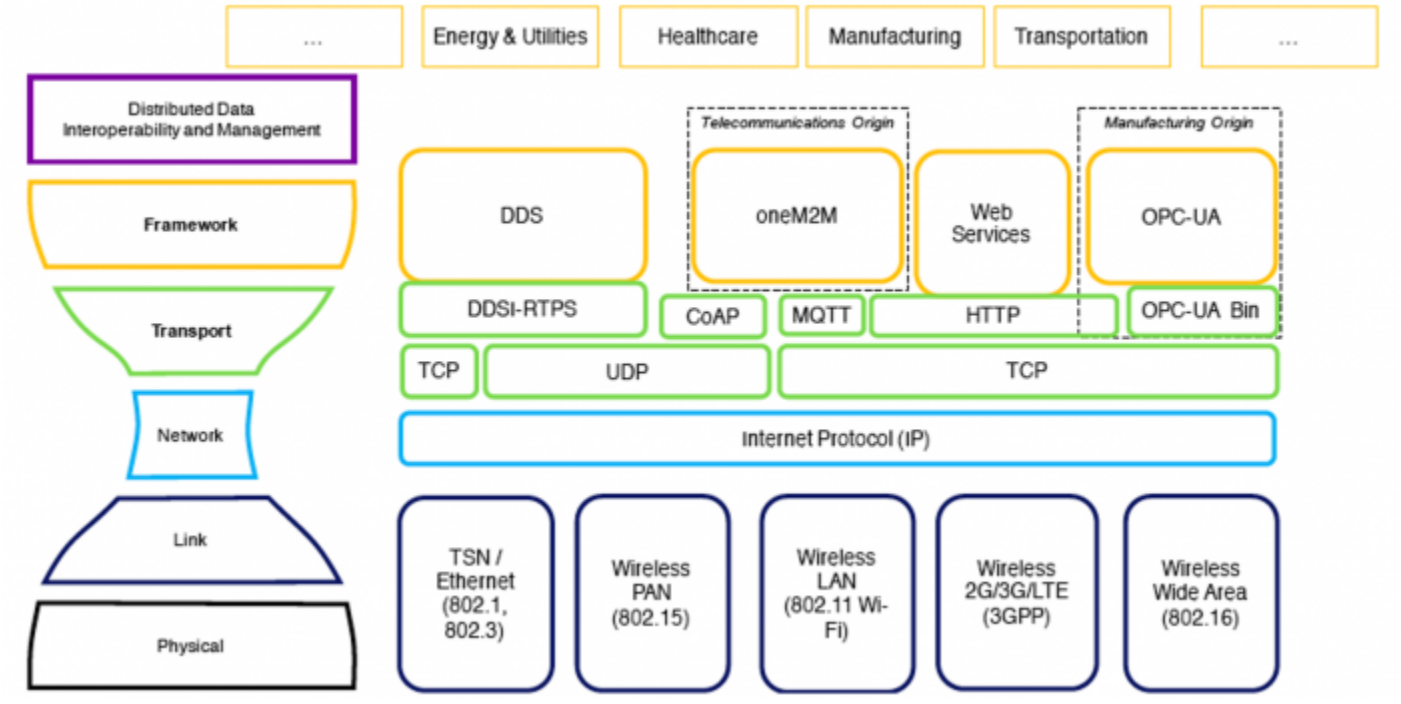

Figure 1: The Industrial Internet Consortium's Connectivity Framework layer provides the foundation for the interoperable transfer of common structured data across systems and domains [7)]

> **Note:** Also see 2.3.4.2.2 Data-in-Motion for a further discussion of the DIDO Layers.

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[6)].

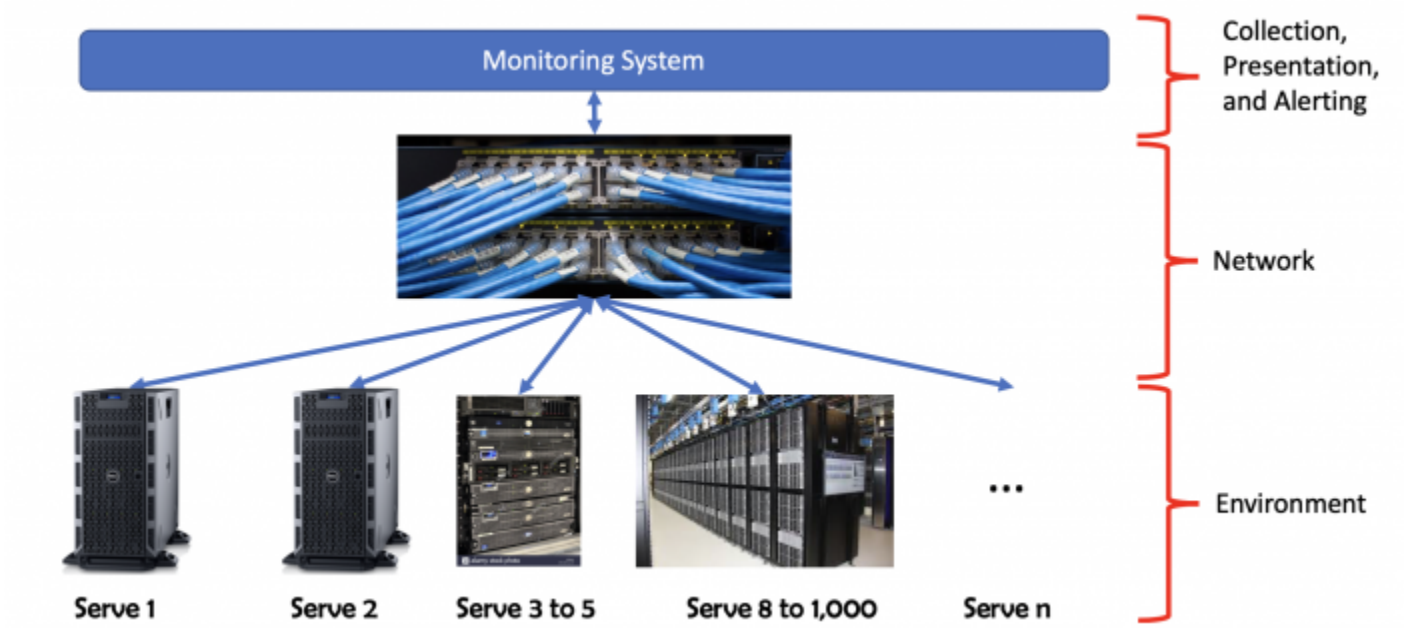

Figure 2: Three major monitoring layers representing sample design of monitoring solution

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 1: Some common logging levels used in applications[7)]

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 [8)] 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 2: 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[9]. There are any spinoffs and derivatives of ZeroMQ. Here are a few reported in Wikipedia[10]

- *In 2012, two of the original developers forked ZeroMQ as Crossroads I/O.*[11][12]
- *Martin Sustrik has started nanomsg,*[13] *a rewrite of the ZeroMQ core library.*[14]
- *In 2012, Dongmin Yu announced his pure Java conversion of ZeroMQ, JeroMQ.*[15]
- *This has inspired further full-native ports of ZeroMQ, such as NetMQ for C#*[16]*.*
- *March 2013, Pieter Hintjens announced a new draft of the ZMTP wire-level protocol bringing extensible security mechanisms to ZeroMQ*[17]*.*
- *Martin Hurton implemented the CurveZMQ authentication and encryption mechanism*[18] *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

1)
Software Evolution, Blog Post, Mitopia Technologies, https://mitosystems.com/software-evolution/
2)
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
3)

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/
4)

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/
5)

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/
6)

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
7)

Log4j - Logging Levels, Log4J, Accessed 18 July 2020, https://www.tutorialspoint.com/log4j/log4j_logging_levels.htm
8)

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
9)

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
10)

ZeroMQ, Wikipedia, Accessed 20 July 2020, https://en.wikipedia.org/wiki/ZeroMQ
11)

ZeroMQ and Crossroads I/O: Forking over trademarks. LWN.net. Retrieved 14 July 2012.https://lwn.net/Articles/488732/
12)

Crossroads I/O. Retrieved 14 July 2012, http://www.crossroads.io/
13)

nanomsg. Retrieved 8 June 2013 http://nanomsg.org/
14)

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
15)

jeromq - java pojo zeromq, zeromq-dev mailing list, Retrieved 23 May 2013, http://lists.zeromq.org/pipermail/zeromq-dev/2012-August/018265.html
16)

NetMQ, GitHub, Retrieved 23 May 2013, https://github.com/zeromq/netmq
17)

Securing ZeroMQ: draft ZMTP v3.0 Protocol, Hintjens.com, Retrieved 23 May 2013,

http://hintjens.com/blog:39
18)

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