Carnegie Mellon University
Software Engineering Institute

Openness Assessment Checklist Method for Complex Systems

Bart Hackemack
Bryce L. Meyer

May 2020

TECHNICAL NOTE
CMU/SEI-2020-TN-001

Software Solutions Division

http://www.sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100

DM20-0391

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Table of Contents

List of Figures

List of Tables

Acknowledgments

The authors owe a great debt of gratitude to the U.S. Space Force Space and Missile Center (SMC) Tools, Applications, and Processing (TAP) Laboratory personnel for support in helping develop this method.

The authors would also like to thank Andrea Amram and Aaron Volkmann for their assistance in refining the scope and direction of the effort that led to this Technical Note.

Executive Summary

The Openness Assessment Checklist Method (OACM) for complex systems is a method to determine if a software-based system is constructed to enable reuse or replacement to the service or module level, by examining compliance to open systems architecture/modular open systems architecture principles. Openness in an architecture context implies the ability to break out portions of the system for reuse, replacement, or relocation, using well-understood interfaces and standards that have a broad user community.

Openness for software systems principles include use of open, well-understood standards and protocols; services that are contained or compartmentalized; well-documented and implemented code, manuals, and designs that allow a variety of users, developers and maintainers; licensing that allows a diversity of users; and quality construction to minimize breakage if components are swapped, rehosted, or used in an unintended but authorized manner. The method in this Technical Note (TN) uses a three-pronged approach: find the modules and interfaces of the system or system of systems, confirm observations and examine code to confirm and document openness boundaries, then—using the knowledge of design and boundaries—complete the checklist (see Section 2.6 and Appendix A).

Deviation cases are used as a vehicle to complete the checklist. Once complete, the results are summarized and reported. The checklist is derived from best practice and published openness guidance. Introspection using this approach benefits the code developers and system in addition to those requesting the review.

Abstract

Openness in the software systems sense is a term loaded with both potential and argument. This technical note examines the definitions of Open Systems, Modular Open Systems, and Openness as applied to software dependent systems, then proposes a method to examine and determine the state of openness in a system. Based on best practice and experience, there are five principles that have a payoff to software systems and their maintainers: use of open, well-understood standards and protocols; services that are contained or compartmentalized; well-documented and implemented code, manuals, processes, and designs that allow a variety of users, developers, and maintainers; licensing that allows a diversity of users and maintainers; and quality construction to minimize breakage if components are swapped, rehosted, or (re)used in an unintended but authorized manner. This technical note proposes and describes the Openness Assessment Checklist Method (OACM) for complex systems, to assess systems against these principles. The method herein uses the determination of the boundaries of openness for the system components and a checklist that finds specific qualities of the software in the openness realm.

1 Introduction

Openness in the software systems sense, and especially in the Department of Defense sense, is rife with both argument and promise. Argument because the terminology is variable between organizations and uses, though defined when applied to systems architecture. Many SEI Blog entries and panels (see References) are devoted to at least explaining the best practices and core ideas for software architectures.

Historically, the term *open systems* derives from physics. It entered the software world with the advent of operating systems (such as UNIX) that allow many parties to create and develop code against a known and well-understood platform. The open source movement enhanced this concept, as did modular electronics architectures such as those for personal computers. Anyone can develop an application or component as long as they follow the standards used by the system.

The benefits included the reduction of vendor lock, enhanced competition, and increased ability to upgrade or repair. In the cybersecurity realm, the ability to quickly patch software is a must-have to ensure security of the system and its data. From the open source movement, another principle emerged: licensing that allows reuse and modification. Needless to say, large government acquisitions and internet backbone hardware buys are plagued with proprietary restrictions in both design solution and legalese. Such restrictions also limit the ability to include government off-the-shelf (GOTS) and commercial off-the shelf (COTS) software in new acquisitions, and the ability of the government to move maintenance of legacy systems to either government or support contractor shops.

Therefore, the move to openness benefits buyers, but it also might benefit sellers too. Allowing a degree of openness allows a larger developer base, a very large community of troubleshooters, and extensions that allow increased market avenues. That said, it takes rules and policy to force vendors to play along. As a result, the U.S. Department of Defense (DoD) settled on the concept of Modular open systems architecture (MOSA), and it was implemented in law: "A MOSA is the DoD preferred method for implementation of open systems, and it is required by United States law. Title 10 U.S.C. 2446a.(b), Sec 805 states all major defense acquisition programs (MDAP) are to be designed and developed using a MOSA that

- employs a modular design that uses major system interfaces between a major system platform and a major system component, between major system components, or between major system platforms;
- is subjected to verification to ensure major system interfaces comply with, if available and suitable, widely supported and consensus-based standards; and
- uses a system architecture that allows severable major system components at the appropriate level to be incrementally added, removed, or replaced throughout the life cycle of a major system platform to afford opportunities for enhanced competition and innovation" [10 U.S. Code § 2446a].

Given this impetus, a clear way to find the openness of a system is a high demand area, and this technical note is based upon experience in performing assessments for DoD programs, though extensible to commercial systems as well.

2 Methodology

2.1 Overall Approach

The method uses a three-pronged approach: find the modules and interfaces of the system or system of systems, confirm observations and examine code to confirm and document openness boundaries, then using the knowledge of design and boundaries, complete the checklist (see Appendix A). Once complete, the results are summarized and reported.

2.2 Team Composition

While any team, internal or external can use this process, ideally the team is outside, and highly experienced on the whole. A team of three is likely ideal, with two very senior software experts with a deep and broad knowledge of the software technologies used on the system, and understanding in theory of the mission of the system, and one more junior notetaker and listener. The junior participant should also have some knowledge of technology and mission, but also adds a benefit by innocence, adding an alternative view of results and conclusions, and an interface to younger system developers.

2.3 Getting in the Door

Before executing the phases below, the team will need approval from both overall leadership and at least tacit agreement from developers and engineers. Construct a letter similar to the sample letter (Appendix B) that defines the scope of the effort (i.e., who will ask what of whom, time and cost, and expected results). The concept should not be one of inquisition and test, but rather a comfortable introspection where leadership gets the report, but the engineers get a little wisdom from the team, and a chance to bounce concepts off those with a broader set of experience.

2.4 First Phase

The first phase involves examining a series of design, version, and architectural documents for the system in examination, with initial contact with system architects, developers and customer. In this stage, a lay of the land should describe what the module and service structure of the system are, what typical scenarios and use cases were used to define the system, intended system and module uses, and interfaces between modules and at the system edges. Emerging from this first phase are lists of modules, interfaces, and loose system flows in use. This should also bound the study itself, to set expectations for results, and resource commitment.

The study team will need to assure the program team that the purpose is not to destroy their system or get them fired. Trust is essential to gaining the true picture of the system and cannot really be down directed. As a result the study team should have a calm demeanor and be very knowledgeable of the system in general and technologies involved, and show a respect for the beauty of the design in examination. Do not call the baby ugly in the first encounter or show a sourness that discourages full and open conversation. Study team members as a result should have at least one

member with a similar academic and work experience history as the architects and engineers in the system, and likely have an earlier career stage member who can interact with earlier career stage developers. Bridging the trust gap is again key.

Get a starter set of digital data in a virtual contact session with the leadership and key contractor leads. Even if the program is classified, the contact should be able to provide links and unclassified documents that provide terminology and top-level scenarios. These artifacts are not to form the study, but to educate the study team. This session can lead to a scoping round to establish how the budget for the study will be spent and what artifacts and presentations will be made by which date.

The budget of the study might be established before the study starts, but it is important to provide a scoping presentation. It is important that those paying for the study know what they will get, when they will get it, and that the study is not an eternal train to nowhere. Start with the definition of openness, open standards, modular open systems architecture, open systems architecture in general, and the differences between these concepts and open source software. These definitions may prove useful for this interchange:

- open systems architecture (U.S. Navy definition): "[The] Open Systems Architecture (OSA) approach integrates business and technical practices that yield systems with severable modules which can be competed. A system constructed in this way allows vendor-independent acquisition of warfighting capabilities, including the intentional creation of interoperable Enterprise-wide reusable components. Successful OSA acquisitions result in reduced total ownership cost and can be quickly customized, modified, and extended throughout the product life cycle in response to changing user requirements" [Navy 2013].
- modular open systems architecture (as per DoD Defense Standardization Program): "A Modular Open Systems Approach (MOSA), formerly known as Open Systems Architecture or Open Systems Approach, can be defined as a technical and business strategy for designing an affordable and adaptable system. A MOSA is the DoD preferred method for implementation of open systems, and it is required by United States law. 10 U.S. Code § 2446a., Sec 805 states all major defense acquisition programs (MDAP) are to be 'designed and developed using a MOSA that
  - Employs a modular design that uses major system interfaces between a major system platform and a major system component, between major system components, or between major system platforms;
  - Is subjected to verification to ensure major system interfaces comply with, if available and suitable, widely supported and consensus-based standards; and
  - Uses a system architecture that allows severable major system components at the appropriate level to be incrementally added, removed, or replaced throughout the life cycle of a major system platform to afford opportunities for enhanced competition and innovation'" [DoD Std Prog 2020].
- Open source: Software where the code is released under a license that allows rights to use, distribute, and alter the code for any reason for any purpose, ideally.

- Open standards: Standards that are available for wide distribution, ideally publicly available and royalty-free; typically, but not required to be, under a non-profit consortium that accepts inputs from a wide set of users, such as ISO, IEEE, IEC, IETF, W3C.
- Openness principles: Openness for software systems principles include
  - use of open, well understood, standards and protocols
  - services that are contained or compartmentalized
  - well documented and implemented code, manuals, processes, and designs that allow a variety of users, developers and maintainers
  - licensing that allows a diversity of users and maintainers
  - quality construction to minimize breakage if components are swapped, rehosted, or (re)used in an unintended but authorized manner

Out of this set of definitions, principles emerge. Open systems architecture is not synonymous with open source. On the contrary, a system can be completely open in an architectural tense, and use no open source software. Openness in an architecture context implies the ability to break out portions of the system for reuse, replacement, or relocation, using well-understood interfaces and standards that have a broad user community. Further, to enable openness there must be solid verification and development methods to limit breakage from fixes, changes, or swap outs, which may benefit quality and security in general.

As a result, the openness assessment needs to establish where the software system can be decomposed into modular boundaries, and that it has a solid verification and issue resolution process to prevent breakage, uses well-understood standards that are known to even unforeseen users, and that documentation at all levels from source to user level enables reuse and replacement of modules.

The core artifacts in a final brief out will be

- a list of what happened when
- the openness boundary drawings with interfaces
- the checklist responses
- a list of recommendations to improve the openness of the system
- any side observations that affect the quality or security of the system

After the scoping meeting with leadership, the detailed interviews and interaction with the system technical team begins. Initial interviewees should be senior level, chief, architects and engineers. Keep the first meetings short, beginning virtually, then go in person. The first virtual call will provide the scoping received from leadership, and the conduct of the study. Ideally, start with the chief software architect or the equivalent. Their vision is why the current system looks the way it does, and their vision will shape the future direction of the system. As a result, they will know why key decisions were made, and what compromises resulted in the system under examination. At the first in-person meetings, add in chief engineers and a few working-level engineers with knowledge of code and system function. These initial meetings should build enough trust in the

development team, and vocabulary and familiarity for the investigating team, to begin more detailed phases below.

2.4.1 When to Exit Phase

Do not exit this phase until

- All parties agree on openness definitions, the scope and expected products of the study, and duration and cost.
- The investigation team and developers have committed to the schedule an provide products.
- Visit requests and clearances are transmitted and received. Access will be granted to the investigation team for the duration of the study.
- The development team, including architects and engineers, and investigating team has enough understanding that a level of trust is established.

2.5 Second Phase

The second phase takes the modules and interfaces, and finds the minimal boundaries in the system where the core functions could be broken out and reused. The checklist is filled out using artifacts and deviation case analysis.

Each boundary will enclose a set of modules (services/applications) that accomplish the to-be reused function (or service) set, and interfaces at the boundary edge (see Figure 1). The core question is what boundaries define the most useful level to break up the system into reusable functional parts. Find the largest composition of modules that define the minimal open interface set. This creates the draft boundary definition. It would be tempting to just rely on interviews and documents. However it is important to see how the system works in reality by observing user tests and acceptance, and to examine source code and DevOps structures, possibly sitting in on Agile meetings as they occur.

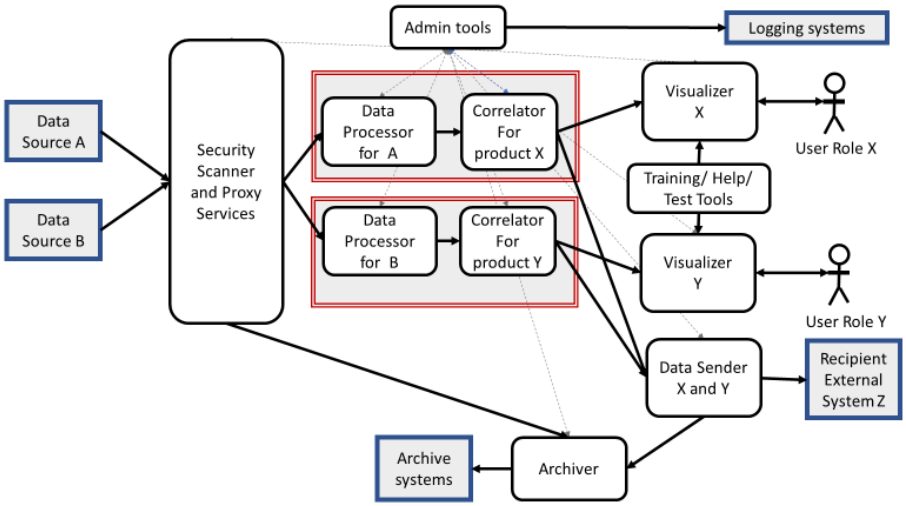


Figure 1: Example Openness Diagram. The system's services and applications are aggregated into the minimal functional sets to assess openness. The double line areas indicate that the Data Processor and Correlator A-->X and B-->Y could be bounded together.

At these boundaries, determine

1. which protocols are at the base of data interchange
2. whether interfaces are known and defined
3. what environment exists at a minimum to instantiate the bounded function

Again, the key principle is the ability to relocate each bounded area into an arbitrary system.

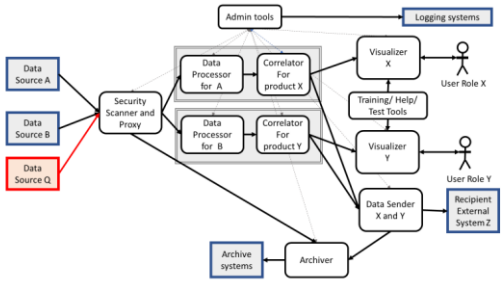
A second principle asks if the bounded area can be instantiated again in the system under examination, possibly in a container, virtual machine, or physical machine, and still complete the core scenarios of the overall system.

The checklist starts assuming the bounded areas, then uses a deeper approach to more fully pull forth assessments against specific openness criteria. The checklist examines seven areas of openness: Boundaries and Breakout, Logging, Coding, Licensing, Documentation, Configuration Management, and Users. These areas and questions are derived from three sources: The Navy Fact File for open systems architecture [Navy 2013], the SEI document *Open System Architectures: When and Where to be Closed* [Firesmith 2015], and the 2007 paper *Foundations of Open: Evaluating Aspects of Openness in Software Projects* by Waugh Partners & OSS Watch [Waugh 2007]. These works can be used, along with the wisdom and best practice of experts. Armed with the checklist, which looks at areas and code examination using deviation cases, the questioning effort will show the current state of the system at least at a cursory level. Deviation cases in this sense are scenarios or partial scenarios, at the design and user level, that employ what-ifs beyond the typical use of the system.

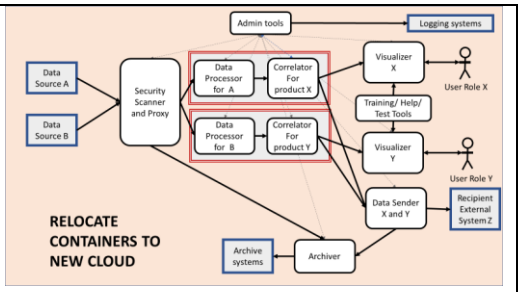
At least these cases should be examined (see Table 1):

1. adding additional data sources or formats
2. relocation or containerization of the system in a different platform or cloud, including relocation of bounded areas to other systems
3. taking various bounded components and using multiple instances
4. using various components connected to other recipient systems, or adding recipients
5. adding functions

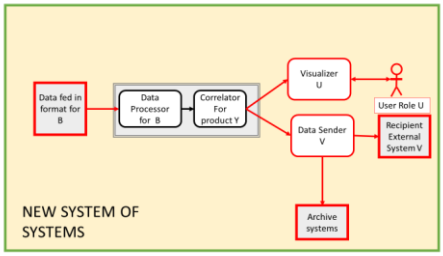
Table 1: Examples of Deviation Cases



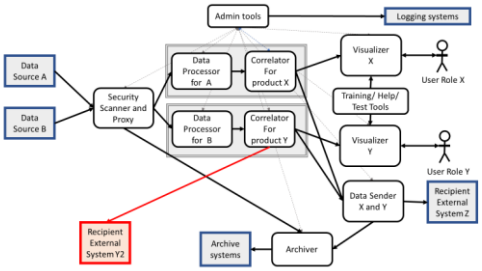
Added Data Source



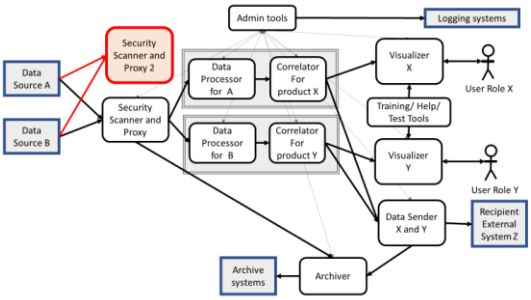
Relocation (of system) to different platform



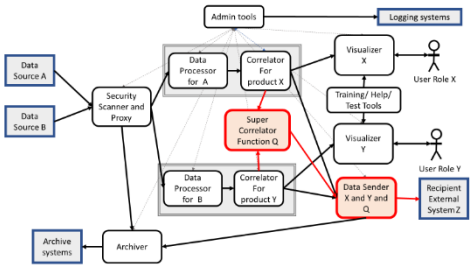
Using various components connected to other recipient systems



Added recipient system



Taking various bounded components and using multiple instances



Adding functions

Effort should include how workflow, configuration, documents, and processes are stimulated by the case. How does walkthrough of these cases in code and diagrams answer the checklist questions? Document results in the checklist, noting issues revealed.

2.5.1 When to Exit Phase

Do not exit this phase until the following are in place

- The investigation team has enough artifacts and notes to complete the checklist, and draw and establish the openness boundaries.
- Format of the final report and brief out are established. Distribution of each is established, with draft approvals.

2.6 Third Phase

The third phase documents the results of the examinations by assembling these artifacts: diagrams showing boundary definitions, the filled-in checklist, and a summary table showing system compliance with the openness approach. Note that a numeric score is not provided. Rather, the product is a loose gauge of how the system is open, and what should change to make the system more modular and open. A final result should include recommendations to improve the openness of the system and any other suggestions to improve the system or process.

The checklist itself (a blank checklist is in Appendix A) has seven areas: Boundaries and Breakout, Logging, Coding, Licensing, Documentation, Configuration Management and DevOps, and Users and Administration. Each area is linked to the openness principles established above.

This checklist can be, and should be, tailored after dialog and examination in the first two phases. Each project may want deeper looks at code requiring code scanners, or in-depth examinations of the container or virtual machine structure, or addition of areas from test events or reviews. Fitting the checklist to the project can better target results.

2.6.1 Area: Boundaries and Breakout

This area assesses the system's architecture to find out if there is at least a high--level way to bound services or modules that might be reusable/swappable. These questions are answered against data collected in the Second Phase, above, and check against Openness Principle #2.

2.6.1.1 Are there clear usability boundaries and services? Are they discrete?

2.6.1.2 Is data passed between services in the system, and from the system to external systems, using well documented, understood, and clearly available published standards and protocols?

2.6.1.3 Are there naming standards and data guides for data passed between boundaries and to external systems?

2.6.1.4 Are there containers or virtual machine instantiations of services?

2.6.1.5 Can multiple instances of each service run simultaneously? In the same machine?

2.6.1.6 Are there shared resources or software components that sit at lower levels that prevent bounded areas from isolation?

2.6.1.7 Have there been successful tests that demonstrate boundaries for services and modules? Did they include off nominal scenarios, virtual structures, diversity of processing, and reuse?

2.6.2 Area: Logging

This area covers the ability to both collect rich information on the status and issues of each module or service, the ability to parse logs using a known method and standard, and the ability to reuse and forward logs to other systems using standard protocol. Checks against Openness Principles #1 and #3 above.

2.6.2.1 Can logs be parsed by external systems, and readable by humans?

2.6.2.2 Are they centralized at bounded areas or alternatively for decomposed services?

2.6.2.3 Are they consistently timestamped?

2.6.2.4 Are they provided using a well-understood method and protocol, and usable by commonly available monitoring systems and tools?

2.6.2.5 Is logging consistent between modules and services in the system?

2.6.2.6 Do logs include at least error/traps, performance, utilization, and security?

2.6.2.7 Are they controllable by usage (i.e., test/debug vs. normal use)?

2.6.3 Area: Coding

This area checks the system's implemented design. While ideally tools could be used to obtain heavy coverage for large designs, the intent is to get a sampling using the deviation cases. Have a developer walk through the code for each deviation case, showing comments, variables, and data sets. Look at configuration files, interface files, and module and service descriptions. Be alert for

anything in the source code that looks like it may be hard to disentangle in the event of modification. Could you look at the code and understand what it does and the libraries it needs? Do all named items follow a naming standard? These questions check Openness Principles #1, #2, #3, and #5.

2.6.3.1 Is the source code in a commonly used language appropriate for application, using commonly available and/or provided libraries?

2.6.3.2 Is it well-commented in a consistent manner to allow external parties to debug?

2.6.3.3 Are the libraries used well understood, and are they all proprietary or from well-understood sources? Are they standard libraries?

2.6.3.4 Are variables and object named in a consistent, readable manner that can be associated to usage and function to trace code?

2.6.3.5 Is there a controlled list of government off-the-shelf (GOTS) and commercial off-the-shelf (COTS) and versions? Are the GOTS and COTS items available to external users?

2.6.4 Area: Licensing

This area checks to see if

- a variety of developers can reuse the code at the boundaries at a minimum
- users can instantiate for further uses
- maintainers or developers from a broader community can modify and maintain the code

These questions check Openness Principle #4.

2.6.4.1 Is the licensing of all functions for each bounded function/service understood and well documented for re-instantiation, reuse, and relocation?

2.6.4.2 Are there modules or services that contain code or libraries that require additional, proprietary licenses?

2.6.4.3 Are there additional licenses required for unintended uses?

2.6.5 Area: Documentation

This area checks to see if there is

- documentation for the design
- versions of components with protocols and standards in use
- required environment to run for each bounded area
- maintainers' guides
- administrators' guides
- users' manuals

A rich set of manuals for at least the bounded areas will help a variety of users and maintainers understand how each component works, which protocols are implemented, reused components, and GOTS and COTS required. Special attention should be paid to interface documentation and version documents.

These questions check Openness Principle #3.

2.6.5.1 Are all modules/services, at a minimum, at openness boundaries, documented for the following: use, black-box troubleshooting, interfaces and protocols, performance requirements and limitations, and known issues?

2.6.5.2 Are there directions and help documents that describe use of the services at the boundaries at a minimum?

2.6.6 Area: Configuration Management and DevOps

This area checks the to see if the state of all the components of the current design is known, that updates and patches from both the development team and especially further test and user groups, can be tracked and implemented in an orderly manner. In Agile processes, are inputs from external test groups and users entering the backlogs, then on into code?

These questions check Openness Principles #3 and #5.

2.6.6.1 Are there clear, understandable methods for external users to suggest fixes and changes?

2.6.6.2 Are changes, updates, and issues reported via a method available to other users?

2.6.6.3 Are changes, issues, and updates tracked using a consistent method and process? Using a well-known tool?

2.6.6.4 Is there a body to review and implement changes and fixes from the larger community?

2.6.6.5 Will this body include representation beyond the initial developer?

2.6.6.6 Are there forums to show use examples beyond those originally envisioned?

2.6.7 Area: Users and Administration

This area checks the system's ability to set user roles and types, and restrict behavior at least at the bounded areas to controlled and limited resources, interfaces, and protocols. Ideally, a relocated module can fit into the user control methods of the recipient system, since the module's user administration follows well-understood and widely known standardized implementations and methods.

These questions check Openness Principles #1, #2, and #3.

2.6.7.1 Are there published methods to control use and access to the bounded services?

2.6.7.2 Can user types in bounded services be controlled by external systems?

2.6.8 When to Exit Phase

Do not exit this phase until the following are in place:

- boundary diagram is completed, and agreed to by the development team
- checklist is completed, with referencing documents and notes to back up answers to each checklist item, and that development team ideally agrees with or at least understands checklist answers
- report is drafted is ready for review and edit
- briefing is created in draft and ready for initial review

2.7 Reporting and Brief-Out

Given diagrams and the completed checklist, the assessment team should have a sense of the openness of the system. This is not a graded exercise or a yes-or-no response. The team will need to prepare both a written report and a presentation, but—importantly—be prepared to provide recommendations to improve the openness of the system and expected tradeoffs in implementation.

While not required, ideally the team should refine the report by bouncing sections and conclusions against the chief-level engineers interviewed in Phases 1 and 2. This prevents misunderstanding about the design and documents and provides the engineers a chance to show documents that were assumed missing or clarify answers to check-list questions. It is also critical to show the engineers/developers the openness boundaries with interfaces and see if they agree with or can improve the diagram.

At a minimum, the report should include the following:

- scope
- assumptions with openness definition as used
- events and interaction
- openness boundary diagram
- a table summarizing each of the seven checklist sections and results
- the completed checklist with sources for findings
- recommendations to improve openness

The table of checklist sections with findings and the openness diagram are the two most important sections.

The presentation will include the same sections, with tables for each section and bullets for results and findings. The assessment team will need to prepare and rehearse possible questions and answers, especially if the checklist responses indicate lower openness. When given, the presentation should include one of the engineer representatives to offer details and expand on recommendations. The presentation, like the report, should not be presented as an assault on the system or its developers and engineers. Rather, the presentation and report present opportunities to tune the system and enhance the system for a wider userbase and customer base. It is deadly critical to prevent a shouting match or stimulate a fight-or-flight response. Whoever asked for the study, the system's developers and engineers, and the assessment team are all acting to improve the system in the end, so are on the same team. This is why nothing in the report or briefing should be a surprise to the engineers or developers.

Be prepared for follow-on. Based on the authors' past experience using this method, it is very likely there will be follow-up to explore areas for improvement in method and improvement for the system. Notes for future efforts using the OACM will help better define openness in general and the associated open systems architecture body of work. More likely, however, the assessment team will be tapped to provide wisdom to improve the system assessed. The assessors now have a deeper knowledge of the system from a fresh perspective, and working with management and developers can shape trade-offs for requirements in future releases. If so, it is likely deeper reviews such as the Quality Attribute Workshop (QAW) and Architecture Tradeoff Analysis Method (ATAM) can help, too.

3 Uses and Guidelines

This is just the beginning of the reviews that might benefit the program and system. Deeper examinations such as ATAMS, QAWs, or other reviews may provide more detailed guidance to improve the quality of the system. Further, this method is not a cybersecurity review, but can indicate the need for a deeper examination in the cybersecurity sense, such as the various techniques from the SEI's CERT Division. The most likely outcome is the desire to increase the breadth and scale of testing of the system, especially at contained service boundaries. As stated before, OACM is not meant as a tool to get a yes-or-no answer. Therefore, it may be unsuited for choosing between system proposals, though could be used as a bolt-on to begin investigation in this context. The best use is to examine a system that has a history, but is looking to expand usage or scale, or possibly to harvest portions to add to newer systems or existing systems of systems frameworks.

4 Conclusions and Recommendations

As stated throughout this Technical Note, openness definitions are arguable and fluid, and may change in the future. For contractors in the U.S. Department of Defense (DoD), the MOSA definition is met largely in the checklist. Other developers might find the methods herein applicable to enhancing their own efforts and steering development to allow business opportunities in reuse of components. Many business cases can be closed using parts of a software system, even if the part is provided gratis, since this part can preview a larger package, or open inside sales opportunities for consulting or further elements.

This Technical Note is not an end-all approach and should be used as in combination with other more formal approaches. Nor does this note force Agile or DevOps since the checklist can be tailored for the approach in use. One of the best uses for this Technical Note is to steer movements from a legacy system to a new system, where architecture is still somewhat fluid, but there is code and a minimal working system with a user base. The authors expect that the methods in this note will lead to the development of improved definitions and better checklists and assessment methods.

Appendix A Blank Checklist

Goals of Openness:

Openness: Openness can be defined as open systems architecture (OSA) or open source. Open systems architecture is well defined in industry, and a good definition from the U.S. Navy can be found here:

https://www.navy.mil/navydata/fact_display.asp?cid=2100&tid=450&ct=2

"An Open Systems Architecture (OSA) approach integrates business and technical practices that yield systems with severable modules which can be competed. A system constructed in this way allows vendor-independent acquisition of warfighting capabilities, including the intentional creation of interoperable Enterprise-wide reusable components."

Another (from the SEI blog) is here:

https://insights.sei.cmu.edu/sei_blog/2015/10/open-system-architecture-when-and-where-to-be-closed.html

1. "It is modular, being decomposed into architectural components that are cohesive, loosely coupled with other components (and external systems), and encapsulate (hide) their implementations behind visible interfaces.

2. Its key interfaces between architectural components conform to open interface standards (that is, consensus based, widely used, and easily available to potential users).

3. Its key interfaces have been verified to conform to the associated open interface standards."

To reach OSA, a system must have boundaries with understood standards open to the community of potential developers (Criteria #1).

Open standards are standards in this construct that are published (Criteria #2A), well understood by a larger community (which may be DoD or beyond) and supported by the community (Criteria #2B).

Ideally, given this type of openness unanticipated users could implement uses for the system (or bounded modules) just knowing the standards used. Openness in this sense does not mean open source. A system can be completely described by an open systems architecture and have no open source software.

Openness Principles: Openness for software systems principles include

1. use of open, well understood, standards and protocols
2. services that are contained or compartmentalized

3. well-documented and implemented code, manuals, processes, and designs that allow a variety of users, developers, and maintainers
4. licensing that allows a diversity of users and maintainers
5. quality construction to minimize breakage if components are swapped, rehosted, or (re)used in an unintended but authorized manner

Pre-work:

1. What are the core services and applications of the system? Are they represented in code? Are there boundaries that can be established that represent the limits of openness?
2. Are resources localized and controlled by each service?
3. What languages are used by the services?
4. For each service (i.e., boundary), what are the minimum inputs and conditions to run? What dependencies exist between services?
5. What are core use cases? Are there scenarios that can reuse services?

Checklist:

1. Boundaries and breakout:
    a. Are there clear usability boundaries and services? Are they discrete?
    b. Is data passed between services in the system, and from the system to external systems, using well-documented, understood, and clearly available published standards and protocols?
    c. Are there naming standards and data guides for data passed between boundaries and to external systems?
    d. Are there containers or virtual machine instantiations of services?
    e. Can multiple instances of each service run simultaneously? In the same machine?
    f. Are there shared resources or software components that sit at lower levels that prevent bounded areas from isolation?
    g. Have there been successful tests that demonstrate boundaries for services and modules? Did they include off nominal scenarios, virtual structures, diversity of processing, and reuse?
2. Logging:
    a. Can logs be parsed by external systems and readable by humans?
    b. Are they centralized at bounded areas or alternatively for decomposed services?
    c. Are they consistently timestamped?

d. Are they provided using a well-understood method and protocol, and are they usable by commonly available monitoring systems and tools?

e. Is logging consistent between modules and services in the system?

f. Do logs include, at least, error/traps, performance, utilization, and security?

g. Are they controllable by usage (i.e., test/debug vs. normal use)?

3. Coding (if the source code is to be provided)

a. Is the source code in a commonly used language appropriate for application, using commonly available and/or provided libraries?

b. Is it well commented in a consistent manner to allow external parties to debug?

c. Are libraries used well understood, and are they all proprietary or from well-understood sources? Are they standard libraries?

d. Are variables and object named in a consistent, readable manner that can be associated to usage and function to trace code?

e. Is there a controlled list of xOTS and versions? Are the xOTS items available to external users?

4. Licensing:

a. Is the licensing of all functions for each bounded function/service understood and well documented for re-instantiation, reuse, and relocation?

b. Are there modules or services that contain code or libraries that require additional, proprietary licenses?

c. Are there additional licenses required for unintended uses?

5. Documentation:

a. Are all modules/services at a minimum at openness boundaries and documented for use, black-box troubleshooting, interfaces and protocols, performance requirements and limitations, and known issues?

b. Are there directions and help documents that describe use of the services at the boundaries at a minimum?

6. Configuration management and DevOps:

a. Are there clear, understandable methods for external users to suggest fixes and changes?

b. Are changes, updates, and issues reported via a method available to other users?

c. Are changes, issues, and updates tracked using a consistent method and process? Using a well-known tool?

d. Is there a body to review and implement changes and fixes from the larger community?

    i. Will this body include representation beyond the initial developer?

    ii. Are there forums to show use examples beyond those originally envisioned?

7. Users and admin

a. Are there published methods to control use and access to the bounded services?

b. Can user types in bounded services be controlled by external systems?

Appendix A Sources:

Firesmith, Donald. Open System Architectures: When and Where to be Closed (blog post). SEI Blog. https://insights.sei.cmu.edu/sei_blog/2015/10/open-system-architecture-when-and-where-to-be-closed.html

Navy Fact File. *Open Systems Architecture (OSA).* https://www.navy.mil/navydata/fact_display.asp?cid=2100&tid=450&ct=2

Foundations of Open: Evaluating aspects of openness in software projects. Waugh Partners and OSS Watch. Version 2.0. October 4.

Appendix B Sample Scope Letter

SUBJECT: Openness Assessment

Duration: xx April 20XX to xx July 20XX

Expected effort: two people, 500 labor-hours effort

Travel: Four trips, one to scope and establish review, another to examine architecture, another to examine code, and a final trip to brief results. These may be virtual if security concerns allow.

The Openness Assessment Checklist Method (OACM) for complex systems is a qualitative assessment that allows a quick assessment of the openness of the architecture of a software intensive system. The assessment examines these principles:

1. use of open, well-understood standards and protocols
2. services that are contained or compartmentalized
3. well-documented and implemented code, manuals, processes, and designs that allow a variety of users, developers and maintainers
4. licensing that allows a diversity of users and maintainers
5. quality construction to minimize breakage if components are swapped, rehosted, or (re)used in an unintended but authorized manner

The result will be definitions of openness boundaries for the system (i.e., the minimal aggregations that can be broken out and reused or replaced) and a completed checklist of criteria that also indicate areas of quality or security that may merit further reviews by other groups. The OACM does not replace formal methods, nor is it an unfriendly inquisition to weigh judgement on architects or developers.

This review will require a friendly technical dialog leading to benefit for both requestor and developers. Therefore, the team for the OACM will require access to the system and documents, people architecting, using, and developing the system in question, and direction from leadership to shape the assessment. Attached is a blank checklist and the Software Engineering Institute's Technical Note describing the process. Thank you in advance for your support and faith in our efforts.

Very respectfully,

<<SIGNED>>

References/Bibliography

URLs are valid as of the publication date of this document.

[AcqNotes 2020]
Systems Engineering, Modular Open Systems Approach, AcqNotes 2020. http://acqnotes.com/acqnote/careerfields/modular-open-systems-approach

[AiDA-MITRE 2019]
AiDA - MITRE Corporation. *Build a Modular Open Systems Approach (MOSA).* 2020. https://aida.mitre.org/blog/2019/03/26/build-a-modular-open-systems-approach-mosa/

[Firesmith 2015]
Firesmith, Donald. Open System Architectures: When and Where to be Closed. 2015. https://insights.sei.cmu.edu/sei_blog/2015/10/open-system-architecture-when-and-where-to-be-closed.html

[DoD Std Prog 2020]
Defense Standardization Program: Modular Open Systems Architecture (MOSA), Defense Standardization Program Office, Fort Belvoir VA. Found at https://www.dsp.dla.mil/Programs/MOSA/

[10 U.S. Code § 2446a]
United States Code Title 10 § 2446a. Requirement for modular open system approach in major defense acquisition programs; definitions, found at: https://www.govinfo.gov/app/details/USCODE-2016-title10/USCODE-2016-title10-subtitleA-partIV-chap144B-subchapI-sec2446a

[INCOSE 2020]
INCOSE. *Open and Closed Systems*. International Council on Systems Engineering. https://www.incose.org/about-systems-engineering/system-and-se-definition/more-systems

[Meyer 2015]
Meyer, Bryce. OSA: 4 Best Practices for Open Software Ecosystems. *SEI Blog*. November 17, 2015.

[Munira 2018]
Munira, Hussan; Runesona, Per; and Wnukb, Krzysztof. A theory of openness for software engineering tools in software organizations. *Information and Software TecWnukbhnology*. Volume 97, May 2018. Pages 26-45.

[Navy 2013]
Navy Fact File. *Open Systems Architecture (OSA).* December 18, 2013. Naval Sea Systems Command, Washington, D.C. 20376.
https://www.navy.mil/navydata/fact_display.asp?cid=2100&tid=450&ct=2

[SEI Blog 2020]
SEI Blog. Subject Area: Open Systems Architecture (Six entries). 2020.
https://insights.sei.cmu.edu/sei_blog/open-systems-architectures/

[SEBoK 2019]
Systems Engineering Body of Knowledge (SEBoK). Open System. SEBoK v. 2.1.
https://www.sebokwiki.org/wiki/Open_System_(glossary)

[Waugh 2007]
Waugh Partners and OSS Watch. *Foundations of Open: Evaluating Aspects of Openness in Software Projects*. Volume 2.0. October 4, 2007.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	May 2020	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Openness Assessment Checklist Method for Complex Systems	FA8702-15-D-0002

6. AUTHOR(S)

Bart Hackemack, Bryce L. Meyer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2020-TN-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100	n/a

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Openness in the software systems sense is a term loaded with both potential and argument. This technical note examines the definitions of Open Systems, Modular Open Systems, and Openness as applied to software dependent systems, then proposes a method to examine and determine the state of openness in a system. Based on best practice and experience, there are five principles that have a payoff to software systems and their maintainers: use of open, well-understood standards and protocols; services that are contained or compartmentalized; well-documented and implemented code, manuals, processes, and designs that allow a variety of users, developers, and maintainers; licensing that allows a diversity of users and maintainers; and quality construction to minimize breakage if components are swapped, rehosted, or (re)used in an unintended but authorized manner. This technical note proposes and describes the Openness Assessment Checklist Method (OACM) for complex systems, to assess systems against these principles. The method herein uses the determination of the boundaries of openness for the system components, and a checklist that finds specific qualities of the software in the openness realm.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Openness, open systems, modular open systems, Openness Assessment Checklist Method	33

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102