Comprehensive Child Welfare Information System

Technical Bulletin #3:

Modular Design and Review Guidance

July 3, 2018

This technical bulletin (TB) provides title IV-E agencies that choose to develop a Comprehensive Child Welfare Information System (CCWIS) with information on a modular design approach and guidance on the assessment of modular design during the CCWIS automated function review.

TABLE OF CONTENTS

1. Purpose of the Technical Bulletin

This technical bulletin (TB) provides title IV-E agencies[1] that choose to develop a CCWIS with information on a modular design approach and guidance on the assessment of modular design during the CCWIS automated function review.

2. Background

The Administration for Children and Families (ACF) replaced the regulations for Statewide/Tribal Automated Child Welfare Information Systems (S/TACWIS) with the CCWIS regulation on August 1, 2016. The regulations for CCWIS at 45 CFR 1355.50-1355.59 include specific design requirements. In particular, section 1355.53(a)(1) emphasizes modern modular software design. In contrast, the regulations for S/TACWIS did not require a modular design approach, which contributed to the development of monolithic systems that were costly and difficult to maintain, enhance, and modernize. The purpose of this TB is to provide agencies that choose to develop a CCWIS with information on the concept of modular design within the context of the CCWIS requirements and guidance on the assessment of modular design during the CCWIS automated function review.

We included design requirements at section 1355.53 to promote efficiency and economy in federal investments in child welfare technology. Unless exempted under certain conditions, CCWIS development activity must follow a modular design that separates the business rules from core programming, simplifies the language of system documentation, and adheres to a development standard. By building the system with severable components, it will enable states and tribes to share and reuse core functional modules and develop affordable and adaptable systems.

The modular design requirement at 45 CFR 1355.53 reads as follows:

> *(a) Except as exempted in paragraph (b) of this section, automated functions contained in a CCWIS must:*
>
> > *(1) Follow a modular design that includes the separation of business rules from core programming;*
> >
> > *(2) Be documented using plain language;*
> >
> > *(3) Adhere to a state, tribal, or industry defined standard that promotes efficient, economical, and effective development of automated functions and produces reliable systems; and*

---

[1] Title IV-E agency is defined at 45 CFR 1355.20

> *(4) Be capable of being shared, leveraged, and reused as a separate component within and among states and tribes.*

> *(b) CCWIS automated functions may be exempt from one or more of the requirements in paragraph (a) of this section if:*

> *(1) The CCWIS project meets the requirements of section 1355.56(b) or (f)(1); or*

> *(2) ACF approves, on a case-by-case basis, an alternative design proposed by a title IV-E agency that is determined by ACF to be more efficient, economical, and effective than what is found in paragraph (a) of this section.*

Requiring a modular design makes it easier for states and tribes to improve their CCWIS systems when replacing or upgrading modules.

This document contains examples of system implementation techniques that ACF has included in order to provide clarity in its expectations of modularly designed CCWIS automated functions. Developing a child welfare information system according to the examples in this document does not guarantee that the resulting information system complies with all of the CCWIS regulations. Only an ACF CCWIS Review may determine whether a child welfare information system is in compliance with the CCWIS regulations.

3. Information

Definitions

Modularity is a design approach to building computer software by breaking up complex functions into separate, manageable, independent modules. Modular design is defined in the Department of Health and Human Services' Medicaid regulations[2] as, "A modular, flexible approach to systems development include[es] the use of open interfaces and exposed application programming interfaces; the separation of business rules from core programming, available in both human and machine readable formats." Module is defined in State Medicaid Letter #16-010 as,[3] "A packaged, functional business process or set of processes implemented through software, data, and interoperable interfaces that are enabled through design principles in which functions of a complex system are partitioned into discrete, scalable, reusable components." **ACF adopts the definition in the 81 FR 3011, Jan. 20, 2016 version of the**

---

[2] Modular Design is defined at 42 CFR 433.112(b)(10)
[3] State Medicaid Letter #16-010 is available here: https://www.medicaid.gov/federal-policy-guidance/downloads/smd16010.pdf

regulation at 42 CFR 433.112(b)(10) as the definition of modular design, and the definition in the August 16, 2016 State Medicaid Letter #16-010 as the definition of a module, in CCWIS.[4]

Modularity

Unlike the design approach that title IV-E agencies have often used in the past to build S/TACWIS systems, a CCWIS that is designed modularly promotes timely and cost-effective development of a system. Modular design of automated functions in CCWIS systems (or technological designs determined to be more efficient than modular design) is essential for agencies to meet CCWIS requirements. Furthermore, the modular design requirement also modernizes child welfare information system regulations while acknowledging that new technologies more efficient than modular design may be proposed through the waiver process outlined at 45 CFR 1355.53.

Initially, modular development may result in increased development costs. However, the savings realized by decreased operational costs of well-designed systems and the reusability of these modules should offset the potential initial development cost increase. The benefits, including cost savings, of a modular approach are so significant, that a 2010 U.S. Chief Information Officer's "25 Point Implementation Plan to Reform Federal IT"[5] recommended that federal agencies only fund major information technology projects if they used a modular approach. This document is included in the References section below.

The CCWIS modular design requirement requires separating business rules from core programming (see section 1355.53(a)(1)). In the requirement, core programming may be code other than business logic, such as code for interfaces and data access layers. Software systems that are broken up into modules follow a principle of separation of "concerns" or tasks. The separation of concerns creates layers within programming, with each layer specializing in the type of task that it performs. In a modular design approach, a module is typically comprised of three layers: presentation (user interface), business logic (rules), and data access.

Consider the example of an agency modifying its intake policy to change the emergency response number of hours. The agency changes a business rule and calculation in the automated intake function; however, it does not affect the user screens or database elements. In this example, only the business rules layer needs modification, leaving the data access and user interface unchanged. If the agency had combined this automated function's business rules

---

[4] ACF maintains the interpretive discretion to define the terms for CCWIS purposes.
[5] U.S.CIO "25 Point Implementation Plan to Reform Federal IT" is available here:
https://www.dhs.gov/sites/default/files/publications/digital-strategy/25-point-implementation-plan-to-reform-federal-it.pdf

with its core programming, then the interdependencies, with the user interface and the database, would increase the time to make the modification.

Modular design approach applies to both the internal design of each module as well as the system as a whole. Software systems made up of "loosely-coupled modules" are systems where there is little overlap between the function of each module. Systems built in this way are quicker to develop and deliver to end users. Projects that implement systems in this way can more easily reuse modules within the same system or in other systems. Loosely coupled modules allow developers to release updates to end users quicker than in a non-modular system, where interdependencies and tightly coupled functionality require the recompiling of the entire non-modular system before releasing it to the end user.

Application Programming Interface (API)

Interfaces between modules allow them to interoperate with other modules. In software development *application programming interfaces* (APIs) clearly define the method of communication between software components.

APIs can facilitate module reuse within the same system or in other systems. Developers may use APIs to build and update a shared infrastructure whenever they discover a task common to multiple modules. For example, if several modules must access a database, then all modules communicate with a data access API, and the data access API is the only module that interoperates with the database.

Agencies electing to modernize a legacy S/TACWIS or non-S/TACWIS application, where the existing backend database is only set up to "talk" to the existing client user interface, should consider implementing an API module as a means to delivering new functionality to users. In this case, an API module could allow new services to use existing data; handle requests to the existing database; decide whether data needs to go into the existing legacy database or to a new CCWIS database; and provide a modern software framework for new modules to exchange data. In this way, new modules may interoperate with existing modules by adhering to the data access API and API standards. Standard security procedures built into the data access API module can further protect agency data from access by unauthorized modules/automated routines.

Benefits of Modular Design Approach

1. Modular design separates business rules from core programming resulting in:
   - Lowered risk – working with specialized layers of code limits errors due to simplicity in the design;
   - Lowered cost – developing specialized layers of code generally results in fewer maintenance costs in a finished product;

- Shortened development time – working with specialized layers of code is easier, especially in complex systems;
- Shortened testing time – working with specialized layers of code limits the number of test scenarios that need to be run to fully test the code;
- Improved maintainability – each layer of code can be modified without the need to modify any other layer; and
- Improved reusability – core programming or business rules can either be packaged and shared together or separated and shared individually.

2. A modular approach to system design creates efficiencies such as the following, which improve upon results from the development of heavily entangled code (sometimes described as spaghetti code):

- Documentation efficiencies – developers have an easier time documenting functions that perform a narrow set of tasks;
- Development efficiencies – developers have an easier time coding the necessary functionality in a way that protects it from being affected by unrelated changes;
- Maintenance efficiencies – developers can more easily find specific functionality within the code when making edits; furthermore, developers may more easily locate the appropriate place in the code to make changes when adding or modifying functionality within a program;
- Testing efficiencies – because modular approach eliminates dependencies, there is no need to make changes to code outside of the module being tested, which reduces the amount of testing to ensure that no unforeseen issues arise; and
- Use efficiencies – end users experience fewer and less impactful bugs, as smaller sections of code are affected by a bug when one is discovered, leading to less downtime and fewer workarounds.

3. Modular design has the potential to deliver reliable software, quickly and for less cost. Designing a modular system allows developers to test and ensure the functionality and reliability of individual modules, rather than waiting until a complete system is functional. This means developers can test earlier rather than later, reducing risk and the impact of potential errors.

4. Modular projects may leverage other modular approaches, such as those that may be found in commercial off-the-shelf (COTS) products, other agency systems, or Software-as-a-Service (SaaS) solutions. Agencies can mix and match between developing new software, which might be expensive, and using existing COTS, agency, or SaaS software.

5. Modular systems using industry-standard APIs and data formats (for example, JSON, SOAP, REST and XML) allow vendors to compete on features and functionality in specific modules. For example, states and tribes might decide that their best option is to create their own case management module, as commercial options might not meet their

needs. Over time, though, commercial options may improve and a modular system would make it easier to replace the state/tribe-designed case management module with a commercial option in the future.

6. Modular design makes it easier for title IV-E agencies and vendors to collaborate and reduce duplicated work. For example, a smaller vendor might concentrate on developing a search module for a state or tribe that another state or tribe could then integrate into their child welfare system solutions.

7. Modular design broadens the opportunity to use and share open source code so that, for example, an open source API that makes a title IV-E agency's legacy DB2 mainframe database available to modern web-based services could be used by another title IV-E agency with a different database management system, for example, ADABAS, to support modern web-based services.

8. If the state/tribe owns the software, then the title IV-E agency can easily upload it to the federal repository where other states and tribes may choose to download and use it.

9. Modular design lends itself to a more robust system architecture approach, and more clearly draws attention to points of intersection where data exchanges may be beneficial, both within the CCWIS environment and with external partners. We recommend considering the use of best practice approaches, such as the National Human Services Interoperability Architecture (NHSIA) and the National Information Exchange Model (NIEM), for exchanges with external partners as a means of reaching agreement on how to define those interfaces and shared data requirements.

Modularity in New and Transitioning CCWIS Systems

Modular design allows states and tribes to choose the software that meets their specific needs. States and tribes are free to choose as much or as little functionality as is required, without committing them into a particular vendor or technology when technology inevitably improves.

For new child welfare information systems, modular design will allow states or tribes to build their systems in the way that is best suited to their needs instead of a one-size-fits-all approach. For example, a transfer system (an existing child welfare information system from another state or tribe) might come with functionality that the receiving state or tribe does not need. If the state or tribe that built the original system developed it using a modular approach, the next state or tribe can choose the individual modules that they want to use and upgrade those as desired.

Most, if not all, existing child welfare information systems were developed after the S/TACWIS regulations were published in 1993. As a product of available technologies at the time of implementation, and the constraints of S/TACWIS regulations themselves, these child welfare information systems have tended to be monolithic, meaning, their software is one large

application that is not split into modules. Often developed as spaghetti-code, they are comparatively more difficult to maintain than modern modularly developed systems.

Private and public sector software development teams successfully use the "strangler pattern" technique[6] to upgrade large, monolithic information systems. In this technique, developers gradually replace functions of the old system with new modular functions. When moving from an existing child welfare information system, one of the first modules typically developed is an API module that "talks" to the database that the existing child welfare information system uses, and delivers a modern web-based interface that new modules can be built upon. With a completed API module, projects are free to develop or install new modules, so that they deliver new pieces of functionality sooner, rather than all at once, after a longer period of time. As developers implement new modules, they 'wall-off' or 'strangle' legacy functions. After 'strangling' all of the functionality of the old system, developers retire the old system.

In some child welfare information systems that already have web interfaces, states or tribes might still be interested in upgrading back-end functionality with modular automated functions. If these child welfare information systems support open source APIs, then it is significantly easier for those agencies to add on new functionality alongside their existing systems.

Here are some examples of modularly designed automated functions that may exist in a CCWIS:

- An intake function for case workers to record incoming information about a client so that they can make a determination (for example, opening a case and assigning it to a case worker), search existing data and create and assign new cases;
- A case management function for caseworkers to use throughout the lifetime of a case so that they can deliver and manage appropriate child welfare services, manage case files and attachments, provide calendaring, to-do list and reminder functions;
- A resource management function for caseworkers to find and update information about child welfare resources so they can deliver and manage appropriate child welfare services;
- A licensing function to receive license applications, manage rule violations/complaints, issue and revoke licenses;
- An eligibility function to support the title IV-E eligibility determination process;
- A reporting function to automatically create and upload required federal reports and provide dashboards and reporting for state and tribal staff;
- A data exchange function specialized in a particular entity's need for data (for example, court data);

---

[6] Agencies are not required to use the Strangler Pattern to develop child welfare information systems. A brief presentation on the Strangler Pattern is available on the 18F website: https://modularcontracting.18f.gov/resources/

- A financial management function to support service approvals, payments, and tracking; and
- A public portal function for foster youth, families and stakeholders to upload documents, communicate with assigned workers and access relevant and appropriate case information.

Exception

While ACF encourages title IV-E agencies to upgrade large monolithic applications to modularly designed systems, section 1355.53(b)(1) provides agencies 'transitioning' a former S/TACWIS or non-S/TACWIS application to a CCWIS with a time-limited waiver to this requirement. During the CCWIS transition period (section 1355.56), title IV-E agencies with operational systems meeting the requirements of sections 1355.56(b) or 1355.56 (f)(1) may request a waiver to the CCWIS design requirement for the operational components of their system that they have elected to transition to an otherwise eligible CCWIS compliant application. The CCWIS transition period ends July 31, 2018. To receive a waiver from the modular design requirement for the legacy application, the title IV-E agency must notify ACF prior to the end of the CCWIS transition period that the agency will 'transition' an eligible system as a CCWIS and commit that the finished application will meet all other CCWIS requirements.

Additionally, section 1355.53(b)(2) allows title IV-E agencies to propose an alternative to the modular design requirement. If ACF determines that the alternative approach is more efficient, economical, and effective than original CCWIS modular design requirements, it may grant a waiver to this requirement. The exemption request under section 1355.53(b)(2) is limited to an agency's adoption of an advanced technological approach more efficient than the CCWIS modular design approach.

4. Assessment of Modular Design in a CCWIS Review

As described in 1355.55, ACF will review, assess, and inspect the planning, design, development, installation, operation, and maintenance of each CCWIS project on a continuing basis, in accordance with APD regulations in 45 CFR 95 Subpart F, to determine the extent to which the project meets the regulations in §§1355.52, 1355.53, 1355.56, and, if applicable, 1355.54. As part of this review process, ACF will review automated functions for adherence to CCWIS design requirements at 45 CFR 1355.53. Conditions for modularity and interoperability require specific focus on ensuring the integrity and interoperability of the automated functions and cohesiveness of the various automated functions incorporated into the CCWIS. The target outcome for the CCWIS modular design should be to accommodate ever-evolving solutions for the title IV-E agency's business requirements and the successful integration of the chosen solutions and infrastructure into a seamless functional CCWIS.

A CCWIS review of an automated function, for example case management, will include a review of its modular design. During the review, ACF will assess the case management automated function for its general adherence to modular design requirements defined at section 1355.53(a) and further described in this document. At a detailed level, ACF's assessment of CCWIS modularity will determine:

- That business rules are separated from core programming. Module configuration may vary between deployments and may include information such as credentials and domain names, which all need to be externalized from the module code.[7]
- That use settings, such as limits or formatting options, are set through configuration options to maximize reuse potential between agencies. More robust techniques include the encoding of business process logic and flow as an external model which is imported by the module at startup.
- That the module is documented using plain language that is easy to read and understand.
- That developers conducted testing and verification for each module independently of system integration testing.
- That developers assessed the accuracy of an exchange and response between modules to confirm that the integration of the module with the system does not adversely affect the functionality and operation of the CCWIS as a whole or individual modules. ACF may review the test cases and test plans to determine the impact.
- That exchanges are well documented and open, which will allow vendor-independent integration of modules into an overall business solution. Our assessment may include the review of APIs. For example, when reviewing the documentation of the module, we may review the versioning control system used to manage an API.[8]
- That exchanges between modules preserve confidentiality, integrity and availability of the information, using techniques such as end-to-end encryption for sensitive information or a deployment architecture that can detect and repair hardware or software failures.
- That the module can be shared as an independent component.

---

[7] The agency may consider adopting techniques such as those described by the twelve-factor app, available here: https://12factor.net/

[8] Versioning Control Systems, such as Semantic Versioning, available at https://semver.org/, provide a means of tracking changes in software builds in an effort to reduce the instances of bugs. Such techniques address which changes are "backwards compatible", meaning they will still work with older versions, and which are not. By assigning "major" and "minor" version numbers, a developer can signal the compatibility of the change with other functionality, where major versions are typically not backwards compatible and minor versions are compatible. Additionally, a "patch" version typically consists of bug fixes and is backwards compatible.

ACF's review process of the CCWIS will follow a modular approach. ACF may conduct reviews of individual automated functions or modules periodically throughout the CCWIS development lifecycle. This will allow the implementation team to correct and change course as needed, rather than discovering flaws during a CCWIS review at the very end of the project.

5. Conclusion

Modularity in CCWIS systems will support the technology needed to deliver modern child welfare services and position IV-E agencies to be able to continuously upgrade the application to meet user needs. CCWIS systems can be large and complex. A modular approach manages complexity by breaking the CCWIS into segmented automated functions; enables faster user delivery by allowing parallel work to be performed due to decreased interdependencies; and accommodates uncertainty (such as changing policies) because extracting and changing system code can be accomplished without disrupting the entire system. Modular systems are more flexible. They are also easier and cost less to design, develop, implement, modify and enhance than non-modular systems.

(ACF will make more information available on ACF's review of CCWIS automated functions as a whole on its Division of State Systems website.)

6. References

- Medicaid Information Technology Architecture 3.0: https://www.medicaid.gov/medicaid/data-and-systems/mita/index.html
- Medicaid Enterprise Certification Toolkit: https://www.medicaid.gov/medicaid/data-and-systems/mect/index.html
- State Medicaid Director Letter #16-010: https://www.medicaid.gov/federal-policy-guidance/downloads/smd16010.pdf
- Defining MMIS Modularity: A Common Approach for Successful Implementations: https://downloads.conduent.com/content/usa/en/white-paper/defining-mmis-modularity.pdf
- CCWIS Notice of Proposed Rulemaking: https://www.gpo.gov/fdsys/pkg/FR-2015-08-11/pdf/2015-19087.pdf
- Microsoft.com - Design Concepts: Modularity: https://msdn.microsoft.com/en-us/library/ff921069(v=pandp.20).aspx
- ICF White Paper "Modular Software Development – The Federal Government Challenge"
- OMB M-13-08: https://obamawhitehouse.archives.gov/sites/default/files/omb/memoranda/2013/m-13-08.pdf
- U.S.CIO "25 Point Implementation Plan to Reform Federal IT": https://www.dhs.gov/sites/default/files/publications/digital-strategy/25-point-implementation-plan-to-reform-federal-it.pdf
- DSS Webinars: "Modular Procurement and Agile Development", "Modular Design in CCWIS Systems", and "Challenges and Benefits of Modular System Development": https://www.acf.hhs.gov/cb/research-data-technology/state-tribal-info-systems/training
- Kendall & Kendall. (2002). *Systems Analysis and Design: the Fifth Edition*
- Semantic Versioning: https://semver.org/
- The Twelve Factor App: https://12factor.net/
- The 18F office of the General Services Administration: https://18F.gsa.gov/ and https://guides.18f.gov/

Attachment A: Guidance on Administration for Children and Families CCWIS Automated Function Review: Assessment of Modular Design

An Administration for Children and Families (ACF) review of CCWIS automated functions will incorporate an assessment of CCWIS modular design. As discussed in this document, adherence to modular design principles promotes efficiently constructed CCWIS systems that have lowered initial automated function development time, that do not require excessive effort to operate or upgrade, and that promote reusability in other systems.

During the assessment of CCWIS modular design, ACF may ask agencies to perform a self-assessment; run any automated assessment tools developed by the agency and show outputs, such as a report from a code quality assessment tool; engage in follow-up discussion with federal system architects; or demonstrate remediation enhancements to the functional area or individual modules in a subsequent automated function review. ACF will also require title IV-E agencies to upload completed functions and associated documentation to the Federal software repository as required at 45 CFR 1355.52(h).

Below is a list of sample topics and questions that ACF reviewers may consider and ask as a part of the assessment of modular design. Text in italics provides additional information and in some cases, best practices. Agencies should not consider this list of topics and questions to be all-inclusive; rather, ACF intended the list of topics and questions to help agencies understand the requirements that ACF is assessing during a review of modular design. We encourage you to send in your comments and questions through the CCWIS mailbox at CCWIS.Questions@acf.hhs.gov.

Topics and Questions – Version 1

1. Is the agency project a transitional CCWIS project or a new build?

2. Is the code under review a CCWIS Automated Function, and if so, which one? What does the code do? What is the scope of this code?

3. Describe the high level system architecture used that the module fits in.

4. Is there an overall system architecture design or planning documentation to guide development?

5. Does the design allow for implementation of modified business rules without incurring significant software development?

6. Demonstrate how the business rules are separated from core programming:
    - Is the applicable code separate and distinguishable from other code? How is the code distinguishable from other code? Does it operate based on a set of inputs and produce outputs? Please demonstrate.
    - Is the applicable code manageable? Can the agency change the code without significant changes to other code external to the module?
    - Is the applicable code independent from other code? Can the code operate if it were pulled from the rest of the application and placed in a different application? Does the applicable code require proprietary code, or a specific platform, in order to function? Please describe/demonstrate how. *This relates to the idea of coupling. Loosely coupled modules are more independent from each other than tightly coupled modules. Complete independence would mean modules are loosely coupled, platform agnostic and require no proprietary code. CCWIS modules are not required to be completely independent.*

7. Provide documentation to show or demonstrate that the agency wrote module documentation (for example, comments in code) using plain language.

8. What state, tribal, or industry-defined standard does the module follow?

9. If applicable, can the agency share, leverage, and reuse the module as a separate component within the system, *for example, a search engine*?

10. Can the agency share, leverage, and reuse the module as a separate component within and among states and tribes? *This does not mean that the module/function is 'Plug-and-play.' Title IV-E agencies that use a module/function from another system may need to make changes to the code.*

11. Is the module/function and related documentation ready to be shared through the federal software repository to facilitate reuse (as permissible)?

12. Has the agency defined the module at the appropriate level of granularity to balance development requirements with timely delivery of high quality end-user functionality?

13. Is there an adequate definition of system modules, their business functions, and interfaces? *This question asks about the information provided in the Advance Planning Document, such as the Automated Function Checklist and system design documents.*

14. Describe the interface/API requirements for data sharing between the module and other modules. *Every module will have some form of a public interface to be useful. The ACF review makes no assumptions or recommendations about application architecture: these interfaces could be in the form of a library loaded into a single process or more commonly these days a web API connecting multiple processes. ACF expects that agencies build interfaces so that they are so understandable that potential adopters can evaluate whether a module might be a good fit for their system and how to call the module functionality when integrating it into their system.*

15. Describe the security measures taken that affect the applicable code. *A challenge when constructing a system using modules is that the developer is pulling in code from someone else that may introduce security vulnerabilities into your system. One way of lowering that risk is to continually analyze modular code for defects and then analyze the assembled system for defects in case the process of combining two modules that are individually sound introduces vulnerabilities when the two are used together. Static code analysis (analyzing code without actually executing the program) can highlight issues in code that are not identified through other means of testing. Commercial solutions are available; open source solutions such as SonarQube https://www.sonarqube.org/ are also popular alternatives.*

16. What messaging architecture does the code use (client-server, publish-subscribe or "pub/sub", etc.)? *This is not a requirement as the agency could have modules that they run within a single CCWIS application, but distributed systems are increasingly common. Pub/Sub is a technique that can drastically reduce the complexity of multi-application integration and increase the reliability by absorbing traffic spikes and temporary outages of components. The focus of the recommendations has been within a single application (functions/configuration/ versioning). There are also the higher-level architecture questions as to exactly how a system composed of many modules/applications will work together to perform a useful service. Messaging, APIs and other means of cross process communication are needed if the service is composed of many applications.*

17. Does the agency use a versioning control system to manage code and if so, does the agency version final releases? *To share software between groups, final releases using release versions of all dependencies are important to create reproducible builds. The release should have an associated version name or number.*

    If the agency versions its code, does the versioning follow a standard? If so, what standard does the agency follow? *If build numbers follow a scheme such as semantic versioning https://semver.org/ then other groups can assume they can safely upgrade between minor*

versions without encountering a breaking change. If the agency does not follow a semantic versioning standard then you have to assume that any update may break existing code.

18. Does the code use an API for its inputs and outputs? Please provide more details for the API. *Recommendations:*
    - *API should be versioned*
    - *REST (preferred), SOAP - use either in preference to rolling your own standard*
    - *API should support organization-mandated content-types (JSON, XML, Avro, etc.)*
    - *Consider implementing caching, quota/rate limiting, scalability*

19. If so, are the APIs versioned? *Similar to the need for release version identifiers for code, APIs should be versioned. There are a number of ways this can be accomplished so no specific guidance is provided here, other than there should be an awareness and a plan to version public APIs.*

20. Monitoring/Observability: Is the code observable? *In monitoring systems constructed of modules, consider operation needs as a part of system design. Design workloads that provide insight to the operating status, customer behavior, and customer experience enabling issue response and the identification of areas for improvement.*

    ACF recommends that agencies document modules in some form of centralized logging system. Otherwise developers could spend significant amounts of time researching problems to see why the system may not be performing as expected or is difficult to operate. If the modules are libraries loaded into a single application then developers just have to know the correct interface or library to which they need to log. If the application is a distributed system then logs will probably be spread across a number of machines and it is common for the developers to ship logs to a centralized service from which the entire system can be monitored for problems. So for example, in cloud-enabled operations, which can provide advantages over physical resources in system design, a specific vendor example would be the ability to log to a centralized logging service like Amazon's AWS CloudWatch.

    Identifying success metrics that can be used to measure the behavior of the workload against business and customer expectations can be challenging. When designing a module an agency is usually forced to make some trade-offs between complexity and functionality, which should be documented by the producer. Other projects need to consider these trade-offs before reusing the code. Depending on the needs of the project, the agency reusing the code might have requirements that exceed the capabilities of a given module. As an example, a COTS search engine/service may have several options for search services built with different underlying technologies, which an agency would evaluate to pick one over the

other based on some combination of their capabilities, limitations and cost. The documentation on the design limitations makes it easier for another solution architect to pick the service or module that solves a given problem.

Monitoring includes providing insights into workload behavior. Building instrumentation into the system design to enable understanding of what is going on within the system and measure performance across individual components is highly recommended.

Monitoring includes providing insights into customer behavior. Building instrumentation into the system design to enable understanding of how the customer uses the system and the quality of their experience is highly recommended.

21. Is the module's configuration externalized? *Agencies should externally configure module functionality so that deployments to different environments do not require code changes. If there are configurable settings for modular behavior or limits that need to be changed when integrating a module into a larger system, it is much simpler to change a configuration file or environment variable instead of having to make code changes.*

Some interpreted languages (meaning, the code does not need to be compiled) can have a "constants" file that is in effect a configuration file as all of the rest of the code reads those settings and the module does not have to be recompiled if the values change.