DEPARTMENT OF VETERANS AFFAIRS (VA) OFFICE OF INFORMATION & TECHNOLOGY (OI&T)



ENTERPRISE PROJECT MANAGEMENT OFFICE (EPMO)

TASK EXECUTION PLAN (TEP) | TAC-17-42242 T4NG-0250 VISTA ADAPTIVE MAINTENANCE

## DUNS: 078393180 CAGE: 6PE96

Submitted by:

## AbleVets LLC

15049 Conference Center Dr Suite 250

Chantilly, VA 20151

## August 17, 2017

**Dr. Wyatt Smith, CEO**

Phone: 571.379.2745

Ema[il: wyatt.smith@ablevets.com](mailto:wyatt.smith@ablevets.com)

This document includes data that shall not be disclosed outside the Government and shall not be duplicated, used, or disclosed in whole or in part for any purpose

other than to evaluate this proposal. If, however, a contract is awarded to this offeror as a result of or in connection with the submission of this data, the Government shall have the right to duplicate, use, or disclose the data to the extent provided in the resulting contract. This restriction does not limit the Government’s right to use information contained in this data if it is obtained from another source without restriction. The data subject to this restriction are contained in all indicated sheets. AbleVets Proprietary.

***AbleVets agrees with all terms, conditions, and provisions included in this RFP and we take no exceptions.***

**Table of Contents**

1. [Understanding the Problem 1](#_bookmark0)
   1. [Identified Uncertainties and Resolutions Proposed 2](#_bookmark2)
2. [The Veteran Integrated Care Services (VICS) Architecture and Development 3](#_bookmark4)
   1. [Generation of Native Data Models from FileMan 4](#_bookmark6)
   2. [Refine Native Data Models for VICS 4](#_bookmark7)
   3. [Development of Common Utility Services 6](#_bookmark10)
   4. [Enable a Portable VICS Interface 6](#_bookmark12)
   5. [Emulate RPCs 6](#_bookmark13)
   6. [Emulate Virtual Patient Record RPC 7](#_bookmark14)
   7. [Use Node.js Compatible, Cloud-based NoSQL Data Storage 7](#_bookmark15)
   8. [Develop an RPC Router 7](#_bookmark16)
   9. [Implement Backward Synchronization Service 8](#_bookmark17)
3. [Patient Data Entry VICS (RTEP B.1.1, PWS 5.2.1) 8](#_bookmark18)
4. [Outpatient Pharmacy Computerized Physician Order Entry VICS (RTEP B.1.2, PWS 5.2.2)](#_bookmark22)

[.............................................................................................................................................. 11](#_bookmark22)

1. [Automated Testing of All Emulation, including Comprehensive Regression Test Suite](#_bookmark26) [(RTEP B.1.3, PWS 5.5.5) 15](#_bookmark26)
   1. [Begin with VistA Domain Test Suites (PWS 5.2, 5.2.1.2, 5.2.2.6, RTEP B1.3) 15](#_bookmark27)
   2. [Encompass Dual Use 15](#_bookmark28)
   3. [Sniff RPC Traffic 15](#_bookmark29)
   4. [Support Synchronization Testing for IOC (PWS 5.5.5.1) 16](#_bookmark30)
   5. [Configure a Node.js-enabled Test VistA (PWS 5.2.1.3, 5.2.2.7) 16](#_bookmark31)
   6. [Implement Security Testing 16](#_bookmark32)
2. [Initial Operating Capability Support (RTEP B.1.4) 16](#_bookmark33)
3. [National Deployment and Retirement (RTEP B.1.5) 17](#_bookmark34)
4. [Security Enhancement 18](#_bookmark36)
5. [Estimated Level of Effort (RTEP B.1.6) 18](#_bookmark37)

List of Exhibits

[Exhibit 1: Foundational Components of the Overarching Technical Approach 1](#_bookmark1)

[Exhibit 2: Identified Uncertainties and Proposed Resolutions 2](#_bookmark3)

[Exhibit 3: VICS Architecture Components 4](#_bookmark5)

[Exhibit 4: Guidelines for Refining Relevant Classes of the Common Native Model 5](#_bookmark8)

[Exhibit 5: VICS Data Model Categorization 5](#_bookmark9)

[Exhibit 6: Common Utility Services and Descriptions 6](#_bookmark11)

[Exhibit 7: Detailed Approach to RTEP B.1.1 Requirements 9](#_bookmark19)

[Exhibit 8: Patient Data Entry Services Approach 9](#_bookmark20)

[Exhibit 9: Patient Data Entry Services Features and Detailed Approach 10](#_bookmark21)

[Exhibit 10: Detailed Approach to RTEP B.2 Requirements 11](#_bookmark23)

[Exhibit 11: Approach to PWS 5.2.2 12](#_bookmark24)

[Exhibit 12: VA-Specific Information in Outpatient Prescriptions 14](#_bookmark25)

[Exhibit 13: Detailed Approach to RTEP B.1.5 17](#_bookmark35)

[Exhibit 14: Level of Effort 19](#_bookmark38)

# Understanding the Problem

VistA was developed over a 35-year span in the Massachusetts General Hospital Utility Multi- Programming System (MUMPS) programming language. While the VistA functionality is valuable and reflects end-user input, it is a complex myriad of inter-woven functions. VistA lacks the structure, architectural patterns, and test frameworks that have been standards in software since the mid-1990s.

None of the current interfaces to VistA are web compatible, and they lack the security controls expected today, resulting in VistA’s vulnerability to code injection and other attacks. Such interfaces include Remote Procedure Calls (RPCs) used by the Computerized Patient Record System (CPRS) and other graphical clients, the Virtual Patient Record (VPR) interface used by Joint Legacy Viewer (JLV) and other clients, and the Roll and Scroll Terminal interface. VistA lacks a built-in, web-compatible, secure interface to support modern clients.

The VA runs 130 separate VistAs across its network, each with different configurations. This inefficient infrastructure results in significant operating costs and creates challenges for migration to a centralized electronic health record (EHR) system.

Team AbleVets can develop a Veteran Integrated Care Service (VICS) architecture for incrementally migrating functionality off of the 130 VistAs to more secure, backward-compatible national services written in a modern programming language. Such migration will enhance security and reduce maintenance costs and facilitate the Department of Veterans Affairs (VA) move to a new enterprise-wide, cloud-based EHR system.

Key elements of Team AbleVets’ overarching technical approach for the VistA Adaptive Maintenance (VAM) Project include those outlined in [**Exhibit 1**](#_bookmark1).

## Exhibit 1: Foundational Components of the Overarching Technical Approach

|  |  |
| --- | --- |
| **Key Elements of Overarching Technical Approach** | |
|  | The majority of all staff resources have significant and recent experience with the surfacing and migration of legacy data in Department of Defense (DoD) and VA projects, using NPM-installable Node.js. We are the only offeror who can provide this level of expertise on Day 1 of contract award, specific to achieving the needs and goals of the project. The remaining staff will be trained by subject- matter experts (SMEs) with such experience, prior to award and/or project onboarding, on all aspects of the project relevant to their assigned role(s). This approach provides 100% of our team with direct relevant expertise and training. This is ***a significant strength of our approach and benefits VA*** with the most experienced staff for meeting the project objectives. |
|  | Team AbleVets, through its exclusive partners, HRG Technologies LLC (HRG) and Caregraf, not only brings the deepest and most experienced VAM team, but also brings an immediate and direct coordination capability with VistA sites. This enables unified coordination and understanding of site-specific nuances across the 130 VistA instances. This avoids duplication of effort, reduces coordination and management overhead, and eliminates performance delays. |

|  |  |
| --- | --- |
| **Key Elements of Overarching Technical Approach** | |
|  | Team AbleVets also provides a ‘force multiplier’ through our capabilities because we have direct experience with the development and sustainment of JLV and enterprise Health Management Platform (eHMP), and transition research and migration development in the DoD Transition Application Program Support (TAPS) project and the related VA VistA Data Project (VDP), which grew out of TAPS. As such, we bring relevant insights, understanding, and expertise to the project. |

This response addresses the requirements outlined in the Request for Task Execution Plan (RTEP) and Performance Work Statement (PWS) for the project. Team AbleVets will demonstrate how to create a VICS that surfaces and migrates clinical functionality and business logic from VistA into a data model-driven, NPM-installable, Node.js-based national service with no legacy MUMPS code dependencies. Such migration will enhance the security of VistA data and to enable a smoother transition to a commercial cloud-based EHR solutions enterprise wide.

This response provides a description of a VICS architecture covering the common structure and environment of every VICS. This is followed by (1) details about the in-scope VICS called for development in the project; (2) the Team AbleVets approach to automated testing in this project; and (3) the security enhancements that a move to VICS will provide. Later sections detail the contents of an integrated Initial Operational Capability (IOC) and release and deployment support.

## Identified Uncertainties and Resolutions Proposed

**Exhibit 2: Identified Uncertainties and Proposed Resolutions**

|  |  |
| --- | --- |
| **Identified Uncertainty** | **Resolutions Proposed** |
| How to surface specific VistA/CPRS clinical function and business logic for migration to national services | “RPC Sniffing” will be used to capture sequences of RPC traffic between CPRS and VistA. “Snapshotting” will establish changes in the data store (FileMan) and process stack of VistA caused by these sequences. This combination will drive the development of (1) Regression Test Suites and (2) Node.js- based Data Models to fully and formally capture VistA behavior for the first time. These same test suites and models will be used to build and test national services. |
| How to emulate specific VistA clinical function and business logic in a national service with an industry- standard, model-driven, secure service interface with no legacy MUMPS code dependencies while keeping CPRS operational | An RPC Router will seamlessly and securely redirect some CPRS RPCs to national services and pass the remainder on to the appropriate VistA. Model-backed, NPM-installable, Node.js-based national services will support relevant RPCs over their interfaces and synchronize with functionality still running in VistA. |
| How to address existing security vulnerabilities and provide enhanced Veteran | Routing RPCs will (1) introduce precise auditing of all VistA RPC traffic for the first time, while re-implementation will (2) remove vulnerabilities from all emulated RPCs and (3) enable |
|  |  |

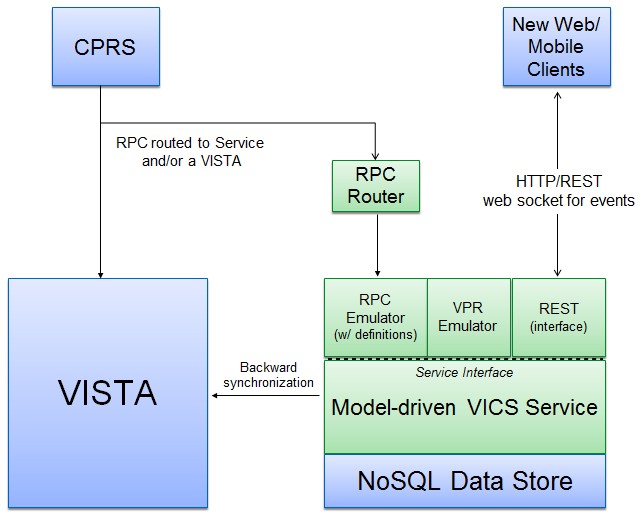
|  |  |
| --- | --- |
|  |  |
| **Identified Uncertainty** | **Resolutions Proposed** |
| data security | elevated access control. End-to-end (4) encryption of RPC traffic will guard against traffic interception. |
| How to provide a foundation for efficient migration from VistA to a commercial  cloud-based, enterprise-wide EHR solution | 130 VistAs with different configurations need to be migrated to a single national EHR. National services provide a clear migration path by (1) centralizing VA operations and (2) isolating the Veteran-specific parts of VA care. Operationally,  (3) the interfaces of these new services may be ported directly onto a commercial off-the-shelf (COTS)-based national HER, and their configurations may be re-used. |
| How to mitigate the complexities of 130 separate VistA data models and site- by-site configurations | The types of data used by CPRS RPCs are the same in all VistAs and will be isolated in the refined data models of national services. Configurations intrinsic to a site, including the rights and preferences of providers, the nature of locations, and variations in services, will be surfaced from VistAs and housed in refined form in national services. |

# The Veteran Integrated Care Services (VICS) Architecture and Development

Unlike VistA, which has few common motifs or utilities, each VICS will follow common patterns and maximize the use of shared utilities. The VICS architecture specifies the common structure and environment for every VICS.

The PWS and RTEP list similar requirements for all four in-scope VICS, including model development, RPC emulation, and VistA Synchronization. Such shared requirements are addressed in this architecture and development section.

[**Exhibit 3**](#_bookmark5)shows the components and applications involved when a VICS replaces the functionality of a VistA domain. The components depicted in green (RPC Router, RPC Emulator, VPR Emulator, a REST interface, a service interface, and the VICS Service) are those that Team AbleVets will develop in the project. Each component will be in Node.js-based, NPM-installable JavaScript and developed without MUMPS.



## Exhibit 3: VICS Architecture Components

VICS development will involve the tasks detailed in the following sections.

## Generation of Native Data Models from FileMan

Most of a VistA’s Native Data Model (Native Model) is defined in the Data Dictionary of FileMan, VistA’s proprietary, MUMPS-based data store. Additional model definitions are in FileMan configuration files that define triggers, indexes, and keys.

In this project, a Native Model will be represented in JSON/JavaScript that surfaces the model’s classes, properties, triggers, and indexes in a web-standard, Node.js-compatible, computable form. This representation will be automatically generated from the FileMan Data Dictionary and additional configuration files. The Native Model of a typical VistA has more than 2,500 classes and 65,000 properties.

There are as many Native Models as there are VistAs since each deployed VistA has unique types of data. All 130 VA VistAs, however, have a common subset of data to support national clients and applications. This subset, referred to as a Common Native Model in this document, is the foundation for the data model for each VICS to be developed in the project.

## Refine Native Data Models for VICS

The Common Native Model represents more than 35 years of incremental development that produced numerous redundancies and inconsistencies. By contrast, for use by modern clients and to ease RPC emulation, the data and model of a VICS must be clear and easy to use. To accomplish this, Team AbleVets will build a VICS Data Model that will refine relevant classes of the Common Native Model using the guidelines detailed in [**Exhibit 4**](#_bookmark8).

## Exhibit 4: Guidelines for Refining Relevant Classes of the Common Native Model

|  |  |
| --- | --- |
| **Guideline** | **Description** |
| Create Global Properties for Common Concepts | For example, a pointer to a Patient object will be called “patient” whether that property belonged to a Prescription or Allergy or Problem object. In the incrementally built Native Model, such a property could be called “patient\_name”, “pname”, “allergyPatient” and a range of other variations. |
| Remove Deprecated Properties | For example, the Native Model for Allergy has a property to denote whether a paper chart was “marked”. This came from the time of paper charts and is no longer used. |
| Use Common Forms where Appropriate | This differs from the incrementally made Native Model where Problem object removal involves marking it as “hidden” while an Allergy is removed by marking it as “entered in error”. |
| Avoid Redundancy | The Native Model has many cases of the same data in different places or data spread over many objects, but should be organized into a single object. |
| Add Missing Properties | For example, unlike Problem or Vital or most other data, Allergy in Native Model does not record where an allergy observation was made. Location can be added to an Allergy VICS Model as an optional property for use by newer clients. |

A VICS Data Model will share the same data model format (JSON/JavaScript) as the Common Native Model. Each VICS Data Model class will be sufficiently defined to allow for automatic translation between any VICS object and the native objects it refines. Such automatic translation guarantees backward compatibility between FileMan resident data and equivalent VICS data and will support selective write-back to VistA of VICS data.

Besides adding clarity and enabling automatic translation to and from FileMan data, each VICS Data Model will categorize information to enable novel security policies and to highlight the Veteran-specific nature of VA data.

## Exhibit 5: VICS Data Model Categorization

|  |  |  |
| --- | --- | --- |
| **Categorization** | **Examples** | **Enables** |
| Patient data versus metadata | A patient’s vital measurement from the definition of a vital type (“height”, “weight”) | Novel Access Control policies that focus on controlling access to patient information |
| VA-specific data versus standard data | Stop codes, eligibility information, health factors, service-rated disabilities, and period of service | Measurement of how much VA data is specific to VA and not covered by industry-wide standards such as FHIR or by COTS software |

## Development of Common Utility Services

A VICS does not re-implement common functionality but reuses utilities and common services it shares with other VICS.

Team AbleVets will develop Common Utility Services, including those defined in [**Exhibit 6**](#_bookmark11).

## Exhibit 6: Common Utility Services and Descriptions

|  |  |
| --- | --- |
| **Utility Service** | **Description** |
| Change Events | A client can register to be informed of changes to data in the system (e.g., that a Problem has been entered for a patient). |
| User Authentication | Support for “single sign on” across VICS and VistA |
| Audit Trails | Tracking who is accessing or changing information in the system as well as where and when that activity occurred |
| Time Management | Support for time zones. While each VistA is in one zone, a single national VICS spans time zones. |
| Location Management | Including the workload (“DSS” and “Credit Stop”) support intrinsic in VistA’s broad concept of location. A VICS must be aware not just of physical locations, but will need to replicate the “personal locations” configured in all 130 VistAs. |
| Identifier Normalization | Mapping between local identifiers specific to one of 130 VistAs into new or pre-existing nationally unique identifiers |
| Terminology/Concept Management | Covering the VistA Lexicon, Drug file, Allergen files and other concept “meta data” from VistA |
| Parameter Access | Some VistA domain functionality is heavily affected by parameter settings. Appropriate parameters need to be available to equivalent VICS. |

## Enable a Portable VICS Interface

A client of a VICS will not have direct access to its objects. Instead, it will use a “VICS Interface” that provides selective access to a set of service operations. Invoking an operation such as “create observed allergy” would lead to multiple changes across many objects.

For new browser and mobile clients, a VICS Interface will be accessible over REST. Traditional clients such as CPRS and JLV access the interface indirectly through RPC and VPR Emulators described below.

Leveraging Common Utility Services, Team AbleVets will deliver NPM-packaged, Node.js implementations of VICS Interfaces that manage VICS Data Model defined objects.

As required by the PWS, every VICS Interface can be re-implemented (“ported”) over “a commercial cloud-based EHR solution”. After portation, all VICS client access will work as before. CPRS will still work based on emulation, and new clients will work over REST.

## Emulate RPCs

In order to support CPRS and other RPC-using clients, a VICS needs to implement relevant RPCs in a process called RPC Emulation. RPC Emulation is a thin layer over the service interface of a VICS that performs the following functions:

* + - Allows the VICS to be tested using the same “VistA Domain Test Suite” used to establish the behaviors of traditional VistA. Such a “dual-use” test suite ensures that the VICS fully reproduces traditional behaviors (see “Automated Testing” below);
    - Prevents code injection and other security problems present in VistA’s RPC implementations by enforcing a machine-readable RPC Definition that covers all aspects of an RPC. Such a comprehensive definition will significantly enhance VistA’s under- defined RPC Definition file, 8994. Besides enhanced security, such a definition will allow the automatic generation of improved RPC documentation;
    - Categorizes RPCs into those that READ system data and those that CHANGE the system in some way. It also distinguishes those affecting a patient’s record (“Patient RPC”) from those dealing with metadata such as lists of drugs or locations (“Meta RPC”). These distinctions present an opportunity for new security policies that enforce greater control on access without requiring changes in traditional clients such as CPRS; and
    - Distinguishes utility RPCs, e.g., database locking and calculation performing RPCs.

## Emulate Virtual Patient Record RPC

The Virtual Patient Record (VPR) RPC is an RPC in name only. Unlike a traditional RPC that performs a very particular action and ranges over a small subset of VistA data, the VPR RPC allows a wide range of patient data to be queried. A typical RPC would be emulated by one VICS. The VPR RPC will be emulated in pieces across each VICS. This emulation will be implemented in a “VPR Emulator”, which will emulate the XML form of the VPR, the only form used by JLV and other production VA applications.

## Use Node.js Compatible, Cloud-based NoSQL Data Storage

Every VICS will use a shared Node.js-compatible, mainstream JSON document store (“VICS Data Store”). The first implementation will use MongoDB, a mainstream choice for such stores with multiple cloud-hosting options. During the project, support will be added for native cloud vendor stores such as Microsoft Azure’s Cosmos DB and Amazon’s DynamoDB.

FileMan is a noSQL store. Backed by Native Model definitions, its data can be transferred with full fidelity to and from the VICS Data Store. Such capability is needed in VICS development to rehouse relevant metadata from local VistAs to a VICS, to seed a VICS with functional data stored in VistAs, and, where necessary, to write data back to local VistAs.

## Develop an RPC Router

Currently any RPC invoked by CPRS goes directly to the VistA of that CPRS. Migrating parts of VistA to a set of VICS requires development of an RPC Router to sit between CPRS and VistA. Such a router will direct relevant RPCs to a VICS while passing all other RPCs to one of 130 VistAs.

An NPM-installable, Node.js RPC router will also have the following capabilities:

* + - Processing any RPC supported by the VistA RPC Broker Protocol;
    - Auditing not only VICS-supported RPCs but all RPCs sent by CPRSs, which is missing from VistA today, a significant addition to Veteran data security;
    - Centralizing the enormous volume of “utility” RPCs repeatedly called by CPRS. Such RPCs tell the time or configure a screen size. Centralizing would remove a burden from local VistAs without changing CPRS in any way and would reduce the resources required to service RPC traffic;
    - Executing a single RPC in both a VistA and a VICS; and
    - Separating Client Parameter Management from VistA.

To facilitate the quantification of the RPCs, the VAM project must emulate, as called for in PWS 5.4.1, the RPC Router and “sniff” all RPCs involved in different CPRS activities.

## Implement Backward Synchronization Service

VistA is a tangled web of interdependent functionality. As VICS is deployed, all functionality left behind in VistA must continue to operate. To enable this continued operation, VistA must be synchronized with functionality migrated to a VICS upon which it continues to be dependent.

Interdependencies in VistA include document macros that pull data from other VistA functions, Clinical Reminder indexes (PXRMINDX) that index the data of other functions, data written to temporary storage (^TMP) for use by other functions, and protocol handlers that allow functions to listen to activity in each other.

A service will be developed to provide such backward synchronization. It will be a pure JavaScript, Node.js-based plugin that runs inside a VistA. It will leverage Intersystem Cache support for in- process Node.js and be deployed as one or more NPM-based, VistA patch.

Synchronization will be enabled by Common Utility Service support for Change Events and write- back support in VICS objects.

# Patient Data Entry VICS (RTEP B.1.1, PWS 5.2.1)

The VistA Adaptive Maintenance Project calls for three “Patient Data Entry” VICS: Allergy, Vitals, and Problem. All three will be assembled following the VICS Architecture described in **Section 3** of this document. VICS for other Patient Data Entry domains, e.g. Skin Test, Immunization, and Health Factors, could follow the same approach.

RTEP Section B.1.1 lists seven explicit requirements for Patient Data Entry Services, all of which are also detailed in **Section 2: VICS Architecture and Development** or **Section 5: Automated Testing of All Emulation, including Comprehensive Regression Test Suite**.

## Exhibit 7: Detailed Approach to RTEP B.1.1 Requirements

|  |  |  |
| --- | --- | --- |
|  | **RTEP B.1.1 Requirement** | **Detailed Approach** |
| a. | FileMan data modeling using web-standard technologies and representation | Addressed in **Section 2: VICS Architecture** *(*[*Generation*](#_bookmark6)[*of Native Data Models from FileMan*](#_bookmark6)and [*Refine Native*](#_bookmark7)[*Data Models for VICS*](#_bookmark7)*).* |
| b. | Distinguishing VA-specific from generic healthcare patterns | Addressed in **Section 2: VICS Architecture** *(Refine Data Models for VICS)* and, where applicable, in the descriptions of individual Patient Data Entry Services in the [**Exhibit 8**](#_bookmark20). One example is Treatment Factors for Problems. |
| c. | Implementing MUMPS emulation using JavaScript/ Node.js-driven, model-driven replacement | RPC Emulation is addressed in **Section 2: VICS Architecture** *(*[*Emulate RPCs*](#_bookmark13)*).* The exclusive use of JavaScript/Node.js is addressed throughout **Section 2: VICS Architecture**. |
| d. | Comprehensive analysis of logic of CPRS client and its RPC interface | Surfacing CPRS’s behaviors through its use of RPCs, individually and in transactions, is addressed in **Section 5: Automated Testing**. |
| e. | Operationalizing JSON models on NoSQL data stores | Addressed in **Section 2: VICS Architecture** *(*[*Use*](#_bookmark15)[*Node.js Compatible, Cloud-based NoSQL Data Storage*](#_bookmark15)*).* |
| f. | Analysis of the VPR RPC interface | Addressed in **Section 2: VICS Architecture** *(*[*Emulate*](#_bookmark14)[*Virtual Patient Record RPC*](#_bookmark14)*)* and in **Section 5: Automated Testing***.* |
| g. | A final solution that has no legacy MUMPS dependencies | The exclusive use of JavaScript/Node.js is addressed throughout *Section III: VICS Architecture*.  JavaScript/Node.js is also the medium for the [*Implement*](#_bookmark17)[*Backward Synchronization Service,*](#_bookmark17)which synchronizes from VICS back to VistA. |

PWS 5.2.1 adds the explicit requirements detailed in [**Exhibit 8**](#_bookmark20)for all three Patient Data Entry services that are all addressed in the VICS Architecture above.

## Exhibit 8: Patient Data Entry Services Approach

|  |  |  |
| --- | --- | --- |
|  | **PWS 5.2.1 Requirement** | **Detailed Approach** |
| 1 | A node.js-based, NPM-installable VICS and Reference Implementation that supports all RPCs used by CPRS for Allergy/Vital/ Problem entry and retrieval and equivalent REST-based calls to enable new clients.  (Contractor’s solution shall replace the CPRS Remote Procedure Call [RPC] interface and any dependent CPRS RPCs (such as the Virtual Patient Record [VPR] RPC) | The exclusive use of NPM/Node.js/JavaScript is addressed throughout **Section 2: VICS Architecture**, including in the statement that “Every component is in Node.js-based, NPM- installable, pure JavaScript modules.”  The “Reference” or Service implementation is addressed in **Section 2: VICS Architecture**  *(*[*Enable a Portable VICS Interface*](#_bookmark12)*).* |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | **PWS 5.2.1 Requirement** | **Detailed Approach** |
|  |  | CPRS Access is addresse[d in Emulate RPCs](#_bookmark13)*;* VPR access is addressed in *Emulate VPR RPC*; New Client access over REST is addressed in *A Portable VICS*. |
| 2 | Regression Test Suite to cover all CPRS Allergy/Vital/Problem interactions. | Addressed in **Section 5: Automated Testing** |
| 3 | Configuration of a VistA Test System to contrast and test against the Allergy/Vitals/ Problem VICS | Addressed in **Section 5: Automated Testing**  (*Node.js-based Test VistA).* |
|  | The Contractor shall reroute the service layer to a node.js-supporting, industry- standard database.  RPCs will be routed to the centralized service, while maintaining backwards compatibility with the FileMan Data Dictionary. | Addressed in **Section 2: VICS Architecture** *(*[*Develop a RPC Router,*](#_bookmark16)[*Implement*](#_bookmark17)[*Backward Synchronization Service,*](#_bookmark17)and [*Refine Native Data Models for VICS*](#_bookmark7)*).* |
|  | CPRS shall continue to perform as before, but against single instances of centralized services, permitting the retirement of the equivalent function in the 130 VistA systems. | Addressed in **Section 2: VICS Architecture** *(*[*Emulate RPCs*](#_bookmark13)and [*Implement Backward*](#_bookmark17)[*Synchronization Service*](#_bookmark17)*).* |
|  | Contractor’s solution shall also address issues that arise with a centralized service including location, time management, and synchronization. | Addressed in **Section 2: VICS Architecture** ([*Development of Common Utility Services*](#_bookmark10)and [*Implement Backward Synchronization*](#_bookmark17)[*Service*](#_bookmark17)*).* |

The VICS Architecture maximizes the automation, shared structure, and common utilities for VICS. However, every VICS has some unique features. [**Exhibit 9**](#_bookmark21)addresses features specific to the project’s Patient Data Entry services.

## Exhibit 9: Patient Data Entry Services Features and Detailed Approach

|  |  |
| --- | --- |
| **VICS** | **Detailed Approach** |
| Allergy | VistA uses the Document package to add signing functionality to Observed Allergies. This necessitates partial support for document RPCs in the Allergy VICS.  Parts of Pharmacy being left in VistA rely on drug allergies. Backward Synchronization will be required.  The protocol handler “GMRA SIGN-OFF ON DATA” allows VistA packages to register for changes in Allergies. This may need to be supported by Backward Synchronization. |

|  |  |
| --- | --- |
|  |  |
| **VICS** | **Detailed Approach** |
| Vitals | The Vitals Domain in VistA is a heavy user of VistA Parameters to configure its operation and the screens of CPRS. The VICS will be a heavy user of Parameter Service, one of the Common Utilities Services.  Documents in VistA use Macros to pull in structured data—“latest vitals” is a popular macro. The Backward Synchronization Service must ensure that relevant data still appears in local VistAs so that this and similar macros continue operating. |
| Problem | Problems in VistA use many terminologies—VA Lexicon, Provider Narrative, ICD10CM, and SNOMED. All must be supported in a Common Utility Service for terminologies. Problems in VistA support “Treatment Factors”, which include Veteran-specific conditions such as Agent Orange. This is an example of a VA-specific concept that a VICS must support.  Though not a regular occurrence, the provider-entered “point of visit” can reference a problem. Until Patient Care Entry (PCE) Data Entry is moved to VICS, this feature will require problem list synchronization back to VistA.  Problems are key inputs for “Clinical Reminders”, which are driven by custom PXRMINDX indexes. These indexes and data used for reminder generation will need to be synchronized by the Backward Synchronization Service. |

# Outpatient Pharmacy Computerized Physician Order Entry VICS (RTEP B.1.2, PWS 5.2.2)

The project calls for one Computerized Physician Order Entry (CPOE) VICS, Outpatient Pharmacy. It will be assembled following the VICS architecture described above. VICS for other CPOE domains, e.g., Lab and Radiology, could be developed following the same approach.

RTEP Section B.2 lists six explicit requirements for the Outpatient Pharmacy Service that are common with that in RTEP Section B.1 for the three Patient Data Entry VICS. A seventh requirement from RTEP Section B.1 appears here again because it is required for the CPOE VICS as well. Most requirements are addressed in the VICS Architecture section above.

## Exhibit 10: Detailed Approach to RTEP B.2 Requirements

|  |  |  |
| --- | --- | --- |
|  | **RTEP B.2 Requirement** | **Detailed Approach** |
| a. | FileMan data modeling using web-standard technologies and representation | Addressed in **Section 2: VICS Architecture** *(Generation of Native Data Models from FileMan* and *Refine Data Models for VICS).* |
| b. | Distinguishing VA-specific from generic healthcare patterns | Addressed in general in **Section 2: VICS Architecture**  *(Refine Data Models for VICS).*  Also see *VA-Specific Data in VistA Pharmacy* below. |
| c. | Implementing MUMPS emulation using | RPC Emulation is addressed in **Section 2: VICS Architecture** *(Emulate RPCs).* The exclusive use of JavaScript/Node.js is addressed throughout the |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | **RTEP B.2 Requirement** | **Detailed Approach** |
|  | JavaScript/Node.js-driven, model-driven replacement | architecture described in **Section 2**. |
| d. | Comprehensive analysis of logic of CPRS client and its RPC interface | Surfacing CPRS’s behaviors and use of RPCs, individually and in transactions, is addressed in **Section 5: Automated Testing***.*  Pharmacy creates the requirement to analyze Pharmacist and Drug Check (MOCHA) interactions not covered by RPCs or CPRS. |
| e. | Operationalizing JSON models on NoSQL data stores | Addressed in **Section 2: VICS Architecture** *(Use Node.js Compatible, Cloud-based NoSQL Data Storage).* |
| f. | A final solution that has no legacy MUMPS dependencies | The exclusive use of JavaScript/Node.js is addressed throughout the architecture description in **Section 2: VICS Architecture**. JavaScript/Node.js is even the medium for the Backward Synchronization Service that synchronizes from a VICS back to VistA. |
| - | Analysis of VPR interface | Called for in RTEP B.1 but missing for B.2. As the PWS notes, VPR accesses Pharmacy Data and must be supported by the Outpatient Pharmacy VICS.  Addressed in **Section 2: VICS Architecture** *(Emulate Virtual Patient Record RPC)* and in **Section 5: Automated Testing***.* |

PWS 5.2.2 adds explicit requirements described in [**Exhibit 11**](#_bookmark24).

## Exhibit 11: Approach to PWS 5.2.2

|  |  |  |
| --- | --- | --- |
|  | **PWS 5.2.2 Requirement** | **Detailed Approach** |
| 1 | Develop a node.js-based, NPM-installable Outpatient Pharmacy VICS for Outpatient Pharmacy that allows a CPRS Provider and typical VA Pharmacist to manage medication orders (Prescriptions)  (Contractor’s solution shall replace the CPRS Remote Procedure Call [RPC] interface and any dependent CPRS RPCs (such as the Virtual Patient Record [VPR] RPC) | The structure and components of every VICS is addressed in **Section 2: VICS Architecture**, including in the statement that “Every component is in Node.js-based,  NPM-installable, pure JavaScript modules”.  CPRS Access in general is addressed in **Section 2: VICS Architecture** (*Emulate RPCs* and *Emulate VPR RPC)*; New Client access in general is addressed in [*Enable a*](#_bookmark12)[*Portable VICS Interface,*](#_bookmark12)and a series of new clients are listed in *IOC Support.*  Also see *Pharmacist Interaction Support*  below. |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | **PWS 5.2.2 Requirement** | **Detailed Approach** |
| 2 | Isolate and document VA-specific from general purpose functionality in VA Outpatient Pharmacy | Addressed in general in **Section 2: VICS Architecture** *(Refine Data Models for VICS)* and also see *VA-Specific Data* below. |
| 3 | Ensure CPRS works as now for VA providers managing outpatient medications | Ensuring successful emulation is addressed in **Section 5: Automated Testing** *(Begin with VistA Domain Test Suites).* |
| 4 | Demonstrate, via a web client, a capability that allows VA Pharmacists to manage outpatient medications and to illustrate Pharmacist interactions | See **Section 7: IOC Support**, where this is listed as one of the new web clients of the project.  Also see *Pharmacist Interaction Support*  below. |
| 5 | Isolate the drug checking interface called from VistA to allow it to be used in the VICS service layer | See *Drug Check (MOCHA) Service* below. |
| 6 | Provide a Regression Test Suite that covers all CPRS-based Provider and roll and scroll based pharmacist interactions. | See **Section 5: Automated Testing** *(Begin with VistA Domain Test Suites),* which notes support for pharmacist activity.  A *Pharmacist Test Suite* is also called out in  *Pharmacist Interaction Support* below. |
| 7 | Deliver a VistA configured for Pharmacy for testing and contrasting the VICS- based Pharmacy service to the current VistA Pharmacy. | See **Section 5: Automated Testing** (*A Node.js based Test VistA)*  The Test VistA will need to support the use of a MOCHA interface for Order checks. |
|  | Demonstrate that VistA’s pharmacy can be replaced by third party solutions without affecting veteran care and the integrated veteran patient record now visible to VA providers and Pharmacists | See **Section 6: Initial Operating Capability Support**, which details an integrated demonstration. |

The features discussed in the following sections are specific to the Outpatient Pharmacy VICS and will be addressed by Team AbleVets in the course of the project.

## General Order and Specific Pharmacy Order

In VistA, CPOE domains are handled by a generic Order Package, which is specialized by domain- specific packages for Outpatient Pharmacy, Inpatient Pharmacy, Lab, Radiology, and other order entry functions. The generic package communicates with specific packages within VistA using internal Health Level Seven (HL7) v2 messaging.

To prepare for addition of CPOE VICS beyond Outpatient Pharmacy, this service will maintain VistA’s concept of a generic “Order” specialized for different domains.

## State Transitions and Roles

A CPOE function is not simple data entry. It is a dialog between parties in which an Order goes through a series of states. VistA distinguishes creating, signing, canceling, replacing, editing, dispensing, and discontinuing a Pharmacy Order. Multiple providers and pharmacists may be involved. The Outpatient Pharmacy VICS will support all such transitions and roles.

## General Purpose RPCs

For Pharmacy, the RPC Router will handle general purpose RPCs. Routing will need to examine RPC arguments as well as RPC names. While some RPCs are specific to Pharmacy, many others are shared with other CPOE functions such as Radiology, Inpatient Pharmacy, or Labs. For example, one RPC, ORWDXA ISACTOI, checks to see whether a Pharmacy Orderable Item is available, but it also checks Radiology Orderable Items. Another RPC, ORWDX SAVE, may create many types of orders for a patient, not just Pharmacy orders. When used for Pharmacy operations, a router must send these RPCs to the VICS, but other uses will go to VistA as before.

## VA-Specific Data (RTEP B.2 b/PWS 5.2.2 2)

The VA-specific information shown in [**Exhibit 12**](#_bookmark25)appears in Outpatient Prescriptions.

## Exhibit 12: VA-Specific Information in Outpatient Prescriptions

|  |  |  |
| --- | --- | --- |
| **Prescription Property** | **Explanation** | **Percentage Seen** |
| CMOP event | VA’s Consolidated Mail Outpatient Pharmacy involved | 63% involved CMOP |
| Billing eligibility indicator | What type of Veteran-specific eligibility did a patient have? | 34% were categorized as VETERAN |
| IB number | Integrated Billing Record reference | 31% tied into VA’s billing package |
| Copay transaction type | Co-payment information | 32% noted a type of Co pay |
| Service connected | Was the Prescription for a Service-related condition? | This was set (not always to true) in 18% of Prescriptions |
| Agent orange exposure | Was a Veteran exposed to Agent Orange? | Set in 3% of Prescriptions |

As [**Exhibit 12**](#_bookmark25)shows, VistA Outpatient Pharmacy supports VA-specific billing, prescription mailing, and Veteran conditions. The data model of the VICS developed by Team AbleVets will support such VA-specific data. Team AbleVets’ Backward Synchronization service will ensure continued operation of billing and other packages that remain behind in VistA.

## Pharmacist Interaction Support (PWS 5.2.2 1/5.2.2 4/5.2.2 6)

Support for VistA’s oldest user interface, the roll and scroll terminal interface, is hard coded into CPOE. As noted in the PWS, pharmacists use a roll and scroll interface. Since the Roll and Scroll interface used by pharmacists and others will NOT be emulated, Team AbleVets will develop the Outpatient Pharmacy VICS to provide the following:

* A service interface with REST support for pharmacist interactions,
* A new browser-based client for invoking operations specific to a pharmacist (PWS 5.2.2 4), and
* A regression test suite (“Pharmacist Test Suite”) for its support of Pharmacists (PWS 5.2.2 6).

## Drug Check (MOCHA) Service (PWS 5.2.2 5/ 5.2.2 7)

The Medication Order Checking Health Care Application (MOCHA) service will be encapsulated in a Node.js/JavaScript Application Programming Interface (API) to support the Outpatient Pharmacy VICS.

# Automated Testing of All Emulation, including Comprehensive Regression Test Suite (RTEP B.1.3, PWS 5.5.5)

The project calls for a Test VistA and a series of domain-specific and cross-domain regression test suites. Team AbleVets will accomplish automated testing using the techniques described in the sections below.

## Begin with VistA Domain Test Suites (PWS 5.2, 5.2.1.2, 5.2.2.6, RTEP B1.3)

VistA lacks authoritative specifications and the regression test suites expected in modern software, which are often larger in size than the software they test.

Before creating a VICS for a domain, a regression test suite (“VistA Domain Test Suite”) will be created to formally capture that domain’s behavior. Whereas most development projects build out test suites as new software grows, building a VICS, a reproduction of largely untested, pre-existing functionality, will start with building a test suite for that functionality.

These suites will primarily be “RPC Test Suites” reproducing the ways CPRS and JLV use individual RPCs and sequences of RPCs. A major exception will be testing how a pharmacist interacts with VistA, which, as the PWS points out, is neither through CPRS nor its RPCs. Reproducing a pharmacist’s behavior will involve a combination of “terminal session”, API, and HL7 v2 tests interleaved with RPC tests.

## Encompass Dual Use

Once complete, a VistA Domain Test Suite will not only capture functionality to be reproduced in a VICS, but, running over a VICS, that same suite will test the effectiveness of that VICS as a reproduction. Such “Dual Use” is key; separate tests for VistA function and its emulation couldn’t ensure common behavior.

Dual Use extends the requirement for “no legacy MUMPS code dependencies” to test suites. Node.js-based VICS need Node.js-based test suites. Because of in-process Node.js support in InterSystems Cache, the same JavaScript-based suite will run natively against equivalent VistA functionality and over Node.js/JavaScript VICS.

## Sniff RPC Traffic

The RPC router described in **Section 2: VICS Architecture** will be used to capture RPC forms and transactions. Before emulation is built, a router will act as a sniffer, spying on RPC traffic between CPRS or JLV and VistA.

Captures of traffic can also be reused for performance testing, contrasting the performance of native VistA to that of a VICS for the same sequences of RPCs, a non-functional requirement in PWS 5.5.5.2. A VICS must maintain or exceed VistA’s current performance.

## Support Synchronization Testing for IOC (PWS 5.5.5.1)

Beyond domain testing, regression tests will be developed for the synchronization of migrated domains and domains still running in VistA. This functionality is described above in *VICS Architecture/Backward Synchronization Service* and below in **IOC Support**.

## Configure a Node.js-enabled Test VistA (PWS 5.2.1.3, 5.2.2.7)

A Test VistA will be based a metadata-only clone of a production VistA if one is provided by VA. Otherwise, it will be based on FOIA VistA. It will be configured to support the following:

* + - A “test hospital” with appropriate locations and users. Many locations will be allocated to particular providers to reflect the workload accreditation (DSS or Stop Codes) configured in a typical VistA.
    - Package and metadata configurations for all Data Entry domains (Allergy, Problem, Vital) and affected secondary domains. FOIA VistA has a basic working configuration for these domains.
    - Package, metadata, and service interface configurations for Pharmacy and affected secondary domains. FOIA VistA lacks settings for the pharmacy package, including a full range of orderable items. These will be added to the system to make it functional.
* Veteran patients to cover the VA-specific designations that affect project domains. For example, patient treatment factors affect the Problem domain.

It will also be able to do the following:

* Support Node.js-based components such as the Backward Synchronization Service and VistA Domain Test Suites,
* Have the ability to return to its original state after either test runs or demonstrations,
* Be packaged as an easy-to-install Virtual Machine, and
* Be usable as a component of Continuous Integration development.

The same “Dual Use” approach being used for test suites will be used for system configuration. Rather than configuring VistA using its roll and scroll interfaces, a JSON-using, Node.js-based mechanism will be employed, and the same mechanism will be used to configure the metadata needed in VICS. For example, VICS need the same location information known to VistAs.

## Implement Security Testing

See the Security Enhancement section below.

# Initial Operating Capability Support (RTEP B.1.4)

A fully integrated demonstration (“IOC Demonstration”) by Team AbleVets will meet the requirement set forth in RTEP B.1.4. The IOC Demonstration will contain the following:

* A Node.js supporting VistA, fully configured for all domains being emulated and secondary domains impacted by their migration to VICS—for more, see the description of “Test VistA” described in **Section 5: Automated Testing**;
* A Node.js-based VICS Server, hosting the Patient Data Entry and CPOE VICS delivered during the project;
* A Node.js-based RPC router;
* A Backward Synchronization Service running inside VistA;
* An instance of CPRS with support for all emulated and effected domains;
* A basic web client for Pharmacists—this is called for in PWS 5.2.2. (4);
* Browser-based management clients for viewing and controlling:
  + the VICS Server
  + the RPC Router including audit and performance logs maintained by the router
  + internal VistA activity and the state of its database
* An authentication feature via a single sign on service;
* A real or mockup MOCHA service; and
* Any other beyond-VistA services expected by Outpatient Pharmacy.

To show that CPRS continues to operate after migration to VICS, the demonstration will do the following:

* Support use cases agreed with VA that establish all expected behaviors from CPRS, VistA, and VICS;
* Run CPRS against VistA alone showcasing expected behaviors;
* Run CPRS in the same way (“continuity of service”) against a combination of (1) a VICS Server with in-scope Patient Data Entry and CPOE functionality, and (2) a VistA with that functionality retired;
* Show pharmacy use cases involving the Pharmacist Web Client and MOCHA Service;
* Use browser-based management clients to show activity of the VICS Server, RPC Router and VistA; and
* Show increased CPRS security as a result of migration to a VICS Server.

# National Deployment and Retirement (RTEP B.1.5)

Readying VistA migration to VICS for national deployment is required by RTEP B.1.5. The Team AbleVets solution is set forth [**Exhibit 13**](#_bookmark35).

## Exhibit 13: Detailed Approach to RTEP B.1.5

|  |  |
| --- | --- |
| **Requirement** | **Detailed Approach** |
| PWS 5.7: Support deployment… into a VA-provided, FedRAMP approved, industry-standard, commercial cloud-based production environment | All commercial cloud environments support pure JavaScript/ Node.js NPM installable modules.  The ability to migrate to a native, cloud-based JSON data store is addressed in **Section 2: VICS Architecture** *(Use Node.js Compatible, Cloud-based NoSQL Data Storage).* |
| RTEP B.5: indicate its approach to retirement of that service in the 130, de-centralized VistA instances, while maintaining full continuity of service in the CPRS client | Addressed in particular by **Section 2: VICS Architecture**  *(RPC Router* and *Backward Synchronization Service).*  The specific steps required to migrate will be documented as part of **Section 6: Initial Operating Capability Support***.* |

# Security Enhancement

Section 5.2 of the PWS requires the provision of enhanced Veteran data security via (1) access control, (2) auditing, and (3) RPC content encryption so that CPRS is adequately secured well beyond the level currently available in VistA. It also requires any security vulnerabilities that do not meet the enhanced security requirements. Team AbleVets will enhance security by including the following in its development:

## RPC Emulation to Remove Vulnerabilities

CPRS is a well-behaved client, but, written for a different era, many of the VistA RPCs it uses are not secured against nefarious activity by illicit clients. For example, in other projects, Team AbleVets has observed the following:

* The MUMPS implementation of Problem creation RPC (“ORQQPL ADD SAVE”) allows unauthorized arbitrary changes to the system;
* The special RPC, “XWB GET VARIABLE VALUE” could be used inappropriately to get any data in the system; and
* The support for “Reference Parameters” in the MUMPS RPC Broker itself allowed logging a client to login with the credentials of another client.

RPC Emulation provides an opportunity to fully secure the RPC interface against illicit clients without affecting CPRS. Vulnerabilities will be comprehensively described for VA as soon as they are found.

## VICS Models and New Access Control

VICS introduce the ability to apply novel patient-centric access control policies. As the data model of a VICS will clearly differentiate patient from other metadata, these services will introduce the ability to apply novel, patient-centric access control, all without affecting CPRS. This enhanced level of control will be covered in regression test suites.

## Encryption of Traffic

Traffic between the RPC router and VICS and between new clients and VICS over REST will be encrypted, and the effectiveness of that encryption will be tested. Were the VA to add traffic encryption into and out of CPRS, Team AbleVets will add support to the router and test that encryption as well.

# Estimated Level of Effort (RTEP B.1.6)

[**Exhibit 14**](#_bookmark38)depicts the estimated level of effort (LOE) for Team AbleVets’ approach for PWS sections 5.2, 5.3, 5.5, 5.6, and 5.7 and all subparagraphs (5.X.X). The LOE table includes labor categories (LCATs) and associated hours for AbleVets and our proposed team members for the base and option periods throughout the period of performance. Note—per PWS 5.5, “the development of capabilities to support adaptive maintenance shall be consistent with the objectives set by PWS 5.2”. As such, PWS 5.4–5.6 depict the required LOE to complete the objectives and requirements outlined in PWS 5.2 (and all subparagraphs). Likewise, for PWS 5.2, the LOE depicted represents the sum of the LOE for PWS 5.4–5.6. Lastly, we have included the LOE for PWS 5.1 and 5.4 to provide a comprehensive view of the required LOE for the project.

## Exhibit 14: Level of Effort

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***PWS Tasks*** | | | | | | | | | | | | | | | | |
| **Company/LCAT** | **5.1** | **5.3** | **5.4** | **5.5.1** | **5.5.2** | **5.5.3** | **5.5.4** | **5.5.5** | **5.5.6** | **5.6.1** | **5.7** | **5.7.1** | **Total** |  | **5.2.1** | **5.2.2** |
| **Base Hours** | | | | | | | | | | | | | | | | |
| **AbleVets** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Program Manager | 1,880 |  |  |  |  |  |  |  |  |  |  |  | **1,880** |  | - | - |
| Cyber Security Engineer, Sr. |  | 94 | - |  |  |  | 376 |  | 940 |  |  |  | **1,410** |  | 658 | 658 |
| Project Analyst |  | 470 | 470 |  | 470 |  |  |  |  | 470 |  |  | **1,880** |  | 705 | 705 |
| System Engineer |  | - | - |  |  |  | - |  |  |  |  |  | **-** |  | - | - |
| Systems Engineer, Sr. |  | 188 | - |  |  |  | 1,692 |  |  |  |  |  | **1,880** |  | 846 | 846 |
| Test Engineer, Sr. |  | 94 | 94 |  |  |  |  | 1,692 |  |  |  | - | **1,880** |  | 893 | 893 |
| Technical Writer/Editor |  | 188 | 188 | 188 | 188 | 188 |  | 470 |  | 2,350 | - |  | **3,760** |  | 1,786 | 1,786 |
| QA Manager |  | 94 | 188 |  | 188 |  |  | 1,410 |  |  |  |  | **1,880** |  | 893 | 893 |
| Development Manager (CSM) |  | 188 | 376 | 188 | 188 | 940 |  |  |  |  |  |  | **1,880** |  | 846 | 846 |
| **Caregraf** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Developer, Sr. |  | 158 | 105 | 105 |  | 683 |  |  |  |  |  |  | **1,050** |  | 446 | 446 |
| Functional Area Analyst, Sr. |  | - | 833 |  | 833 |  |  |  |  |  |  |  | **1,665** |  | 833 | 833 |
| **HRG** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Developer |  | 960 | - |  | 960 | 3,840 |  |  |  |  |  | - | **5,760** |  | 2,400 | 2,400 |
| Developer, Expert |  | 192 | 192 | 288 | 288 | 960 |  |  |  |  |  |  | **1,920** |  | 864 | 864 |
| Developer, Sr. |  | 960 | - |  | 960 | 3,840 |  |  |  |  |  | - | **5,760** |  | 2,400 | 2,400 |
| Functional Area Analyst, Sr. |  | - | 125 |  | 125 |  |  |  |  |  |  |  | **250** |  | 125 | 125 |
| **Total Base** | **1,880** | **3,586** | **2,571** | **769** | **4,200** | **10,451** | **2,068** | **3,572** | **940** | **2,820** | **-** | **-** | **32,855** |  | **13,695** | **13,695** |
| **OP1 Hours** | | | | | | | | | | | | | | | | |
| **AbleVets** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Program Manager | 1,880 |  |  |  |  |  |  |  |  |  |  |  | **1,880** |  | - | - |
| Cyber Security Engineer, Sr. |  | 94 | - |  |  |  | 376 |  | 940 |  |  |  | **1,410** |  | 658 | 658 |
| Project Analyst |  | 235 | 235 |  | 235 |  |  |  |  | 235 |  |  | **940** |  | 353 | 353 |
| System Engineer |  | 188 | - |  |  |  | 1,692 |  |  |  |  |  | **1,880** |  | 846 | 846 |
| Systems Engineer, Sr. |  | 188 | - |  |  |  | 1,692 |  |  |  |  |  | **1,880** |  | 846 | 846 |
| Test Engineer, Sr. |  | 125 | 125 |  |  |  |  | 125 |  |  |  | 125 | **501** |  | 125 | 125 |
| Technical Writer/Editor |  | 94 | 94 | 94 | 94 | 94 |  | 235 |  | 1,175 | 235 |  | **2,115** |  | 893 | 893 |
| QA Manager |  | 94 | 188 |  | 188 |  |  | 1,410 |  |  |  |  | **1,880** |  | 893 | 893 |
| Development Manager (CSM) |  | 188 | 376 | 188 | 188 | 940 |  |  |  |  |  |  | **1,880** |  | 846 | 846 |
| Caregraf |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Developer, Sr. |  | 144 | 96 | 96 |  | 624 |  |  |  |  |  |  | **960** |  | 408 | 408 |
| Functional Area Analyst, Sr. |  | - | 750 |  | 750 |  |  |  |  |  |  |  | **1500** |  | 750 | 750 |
| HRG |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Developer |  | 960 | - |  | 960 | 3,120 |  |  |  |  |  | 720 | **5,760** |  | 2,040 | 2,040 |
| Developer, Expert |  | 135 | 135 | 203 | 203 | 675 |  |  |  |  |  |  | **1,350** |  | 608 | 608 |
| Developer, Sr. |  | 960 | - |  | 768 | 3,552 |  |  |  |  |  | 480 | **5,760** |  | 2,160 | 2,160 |
| Functional Area Analyst, Sr. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **OP1 Total** | **1,880** | **3,405** | **1,999** | **581** | **3,386** | **9,005** | **3,760** | **1,770** | **940** | **1,410** | **235** | **1,325** | **29,696** |  | **11,425** | **11,425** |