Commonwealth of Virginia

Enterprise Technical Architecture (ETA)

Application Domain Report

Version 1.2 April 5, 2017

Prepared by:

Virginia Information Technologies Agency
ETA Application Domain Team

(This Page Intentionally Left Blank)

Application Domain Team Members

Pamela Tauer	Virginia Department of Transportation
Peter Rotscheid	Virginia Department of Health
Ted Maxwell	VITA, Business Services
Tim Mauney	VITA, Business Services
Rusty Waterfield (Higher Education Representative)	Old Dominion University
Scott Somerhalder	Virginia.gov
Tony Shoot/Jim Dart	Northrop Grumman
Todd Kissam (Team Facilitator)	VITA, Enterprise Architecture

The Application Domain team began its work by delineating the team's goals, objectives, and scope of work. Discussions included how the domain interfaces with other architecture domains, present and future directions, and how often the information provided in this document is to be updated. The team also reviewed input from publications and individuals with specialized knowledge. The results of the team's efforts, research and deliberations are provided throughout this document.

Application Domain Report: Version History		
Revision	**Date**	**Description**
1.0	07-10-2006	Initial
1.1	07-01-2006	Update necessitated by changes in the Code of Virginia and organizational changes in VITA. The changes are administrative. There are no substantive changes to the principles, recommended practices or requirements.
1.2	04/05/2017	Update to remove a link to an non-existent reference.

Review Process

Technology Strategy and Solutions Review

The domain report was reviewed and approved by the Manager of the Enterprise Architecture Division.

Online Review

Participation of all Executive Branch agencies was encouraged through a review and comment period via VITA's Online Review and Comment Application (ORCA). Technology businesses and the general public were also actively encouraged to use ORCA to provide comments. All comments were considered and many resulted in modifications to the final document. Additionally, the Domain team provided the reviewers with responses to their comments.

Identifying Changes in This Document

- See the latest entry in the revision table above
- Vertical lines in the left margin indicate the paragraph has changes or additions.
    - Specific changes in wording are noted using italics and underlines; with italics only indicating new/added language and italics that is underlines indicating language that has changed.
- Note that page header dates vary throughout the document depending on when or if some portion of a particular chapter/section has been updated.

The following examples demonstrate how the reader may identify requirement updates and changes:

EXA-R-01 **Technology Standard Example with No Change –** The text is the same. The text is the same. The text is the same.

EXA-R-02 **Technology Standard Example with Revision –** The text is the same. *A wording change, update or clarification is made in this text.*

EXA-R-03 ***Technology Standard Example of New Standard –*** *This standard is new.*

(This Page Intentionally Left Blank)

Table of Contents

Executive Summary of Application Domain

The Application Domain Report is written to assist business and technical leaders in state agencies and central services in making sound decisions related to agency application development and support. A well-defined ETA Application Domain will enable the Commonwealth of Virginia to leverage the most value from its agency software solution assets. This application domain report includes the technology topics and the components needed to provide agencies with a foundation of development and support platforms, tools, processes, practices and requirements that can implement business processes and meet the Commonwealth's ever changing business needs. The topics include Enterprise System Design, Application Acquisition, Development and Support Platforms, Software Engineering, Geospatial Technologies, and Enterprise Applications

The Commonwealth relies heavily on computer applications to support agency business operations. The agencies' business processes often must change in response to both legislation and new demands from citizens. Unfortunately, the Commonwealth's computer applications can not always respond to these changes in an effective and efficient manner because many current applications are either monolithic or two-tier client/server applications.

Many of the Commonwealth's current applications/solutions were developed independently using different languages and tools. The ability to communicate with other applications or systems or to adapt to changes in the business processes generally was not a design requirement. This architectural approach has adversely impacted the Commonwealth's business in three ways:

- Additional cost and time needed to modify existing applications to support changing business requirements
- Difficulty in integrating applications to share common services and data
- Extra expense to develop, use, and maintain new applications because there is little reuse of code between applications

Application development tools, methodologies and technology are now available that can help address these problems. Examples include:

- Reuse of Code
- Integration tools/Middleware
- New User Interface Options:
- N-tier Service-Oriented Architecture (SOA).

Although having a single software development/support product of each type might be the ideal, the reality is that agencies have unique application needs. The application requirements, technical and economic environments of each business unit will have a strong influence upon tool/product choices. Over time, agencies will group related tools into a limited number of standardized technology stacks. These "stacks" will provide agencies with cost-effective development solutions for N-tier, Collaborative, Business Intelligence (BI) or Analytical Technology and other solutions.

Overview

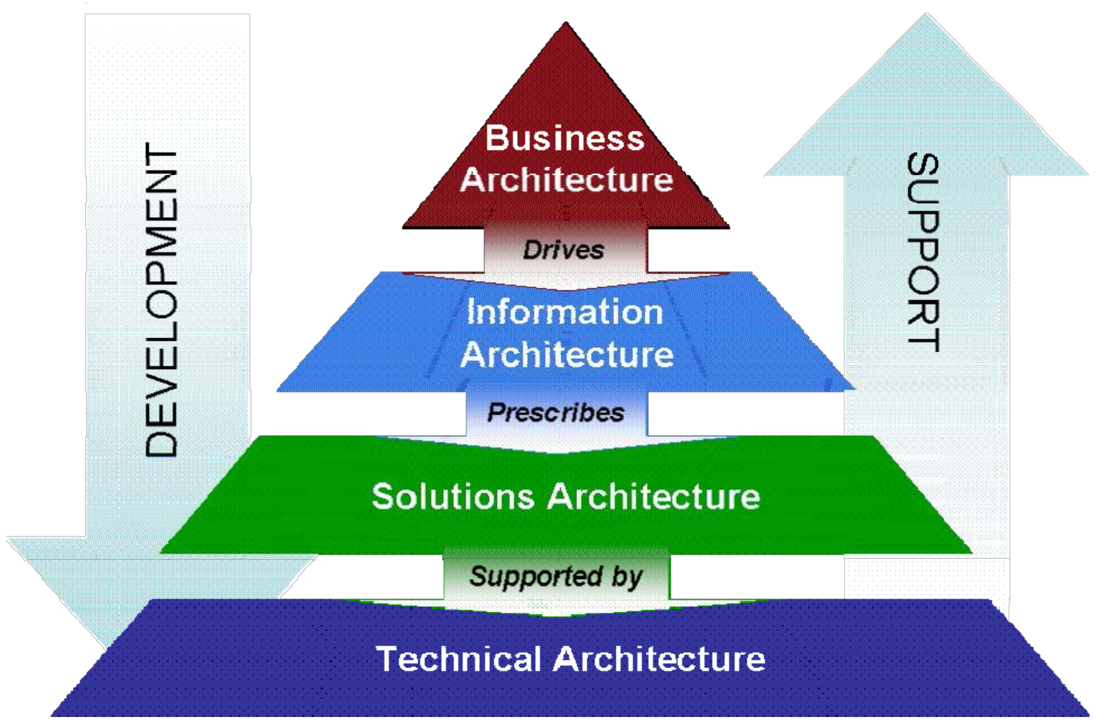
The Commonwealth’s Enterprise Architecture is a strategic asset used to manage and align the Commonwealth’s business processes and Information Technology (IT) infrastructure/solutions with the State’s overall strategy.

The Enterprise Architecture is also a comprehensive framework and repository which defines:

- the models that specify the current (“as-is”) and target (“to-be”) architecture environments,
- the information necessary to perform the Commonwealth’s mission,
- the technologies necessary to perform that mission, and
- the processes necessary for implementing new technologies in response to the Commonwealth’s changing business needs.

The Enterprise Architecture contains four components as shown in the model in Figure 1.

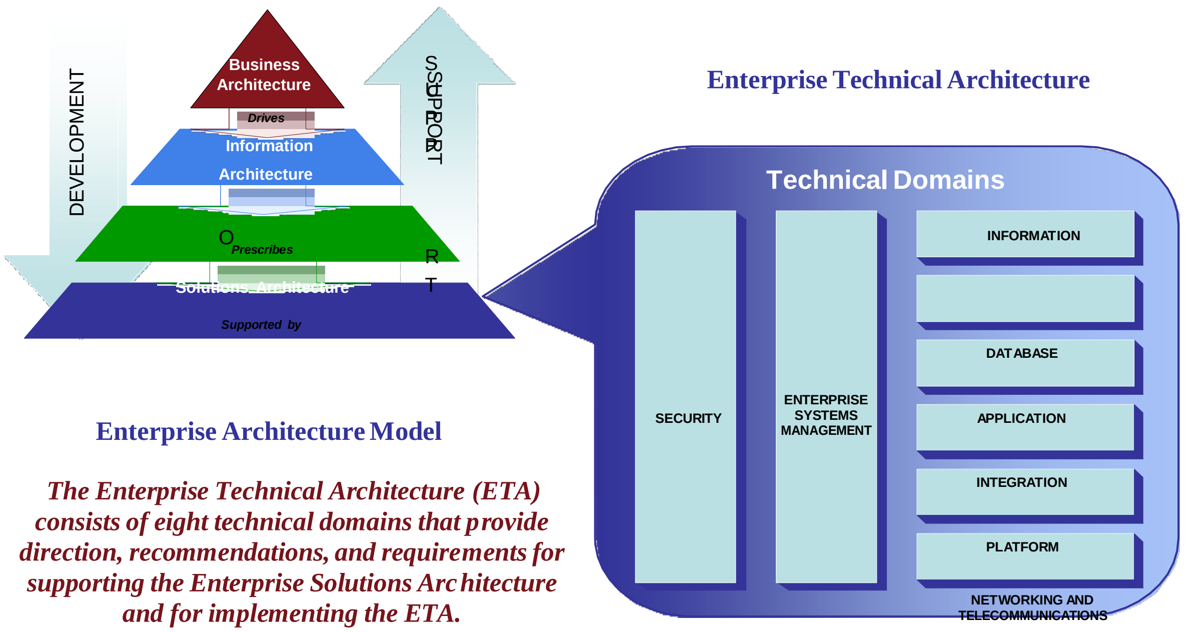
Figure 1
Commonwealth of Virginia Enterprise Architecture Model



The Business Architecture drives the Information Architecture which prescribes the Solutions Architecture that is supported by the Technical (technology) Architecture.

The Enterprise Technical Architecture (ETA) shown in Figure 2 consists of eight technical domains that provide direction, recommendations and requirements for supporting the Solutions Architecture and for implementing the ETA. The ETA guides the development and support of an organization's information systems and technology infrastructure.

Figure 2
ETA Relationship to the Enterprise Architecture


DEVELOPMENT
Business Architecture
Drives
Information Architecture
Prescribes
Solutions Architecture
Supported by
SUPPORT
Enterprise Architecture Model
The Enterprise Technical Architecture (ETA) consists of eight technical domains that provide direction, recommendations, and requirements for supporting the Enterprise Solutions Architecture and for implementing the ETA.
Enterprise Technical Architecture
Technical Domains
SECURITY
ENTERPRISE SYSTEMS MANAGEMENT
INFORMATION
DATABASE
APPLICATION
INTEGRATION
PLATFORM
NETWORKING AND TELECOMMUNICATIONS

Each of the domains is a critical piece of the overall ETA. The Networking and Telecommunications and Platform Domains address the infrastructure base and provide the foundation for the distributed computing. The Enterprise Systems Management, Database, Applications, and Information Domains address the business functionality and management of the technical architecture. The Integration Domain addresses the interfacing of disparate platforms, systems, databases and applications in a distributed environment. The Security Domain addresses approaches for establishing, maintaining, and enhancing information security across the ETA.

This report addresses the Enterprise Technical Architecture Application Domain and includes requirements and recommended practices for Virginia's agencies[1, 2].

---

[1] This report provides hyperlinks to the domain report Glossary in the electronic version. In the electronic and printed versions, the hyperlinks will have the appearance established by the preferences set in the viewing/printing software (e.g., Word) and permitted by the printer. For example, the hyperlinks may be blue and underlined in the screen version and gray and underlined in the printed version.

[2]The Glossary entry for agency is critical to understanding ETA requirements and standards identified in this report and are ~~is~~ repeated here. **State agency or agency -** Any agency, institution, board, bureau, commission, council, or instrumentality of state government in the executive branch listed in the appropriation act. ETA requirements/standards identified in this report are applicable to all agencies

This report was developed by the Application Domain team, which was commissioned to identify domain related requirements and recommendations. Identified requirements and technology product standards from this domain report ~~will~~ *were* be combined with requirements and technology product standards from other technical domain reports into a single ETA Standard in 2006 for review and acceptance by the Information Technology Investment Board (ITIB). *In 2010 the ITIB was replaced by the Information Technology Advisory Council (ITAC).*

Concerning local governments, courts, legislative agencies, and other public bodies, while they are not required to comply with a requirement unless the requirement is a prerequisite for using a VITA service or for participating in other state-provided connectivity and service programs, their consideration of relevant requirements is highly recommended. This architecture was designed with participation of local government and other public body representatives with the intent of encouraging its use in state and local interconnectivity efforts.

including the administrative functions (does not include instructional or research functions) of institutions of higher education, unless exempted by language contained in a specific requirement/standard.

Commonwealth of Virginia: To-Be ETA

The to-be Enterprise Technical Architecture envisioned for the Commonwealth will be one where the Commonwealth's citizens and other customers who wish to access Virginia services will do so by utilizing an Enterprise Portal via standard web browsers.

Where appropriate, these online government services will be developed, delivered and supported using a Service-Oriented Architecture (SOA) based on open and industry standard solutions. Selected legacy applications will be exposed to the SOA using web services.

The SOA will be supported by an Enterprise Service Bus that provides Orchestration and Choreography Services to the agencies.

Central integration and coordination will be managed by an Integration Competency Center (ICC) that supports agency needs such as: asynchronous message queuing and persistence.

Large complex *mission critical* applications that need to be reliable, scalable, secure and highly available will be n-tiered and will utilize business rule and workflow engines.

Enterprise application software for the core government administrative business functions will be consolidated and the underlying business processes modernized. An Application Management Center of Excellence will service and manage the new enterprise applications that replace existing legacy and silo-based applications.

Data will be exchanged among systems, agencies, institutions of higher learning, localities, the federal government, and partners using XML based standards such as the Global Justice XML Data Model and the National Information Exchange Model.

The number and types of software tools and products used by the Commonwealth will be decreased to reduce complexity. This will create the opportunity for agencies to refocus their current in-house IT resources to achieve higher levels of expertise on the fewer required products resulting in, among other benefits, a lower dependence on outside contractors.

Agency software applications and customer services will be delivered and supported by an IT infrastructure that will:

- Be responsive, agile, modular, scalable, reliable, secure, and highly available (24x7)
- Utilize ITIL (IT Infrastructure Library) best practices
- Have extensive and proactive technology refreshment
- Utilize a shared services model for technology delivery
- Have a single secure state-wide network and Intranet
- Have a state-of-the-art data center and back-up facility
- Consolidate agency servers in their most cost-effective locations

- Unify statewide electronic mail services
- Employ innovative procurements, supplier partnerships, and financing arrangements to fund, expedite, and ensure the performance of future initiatives
- Provide a statewide customer care center
- Improve the cost performance of IT utilized by the Commonwealth

Transition:
The Commonwealth will transition from silo-based, application centric and agency centric information technology investments to an enterprise approach where applications are designed to be flexible. This allows agencies to take advantage of shared and reusable components, facilitates the sharing and reuse of data where appropriate, and makes the best use of the technology infrastructure that is available.

The implementation of the to-be architecture will take some time. It is not the intent of the Commonwealth to force agencies to replace their existing systems. The migration to the to-be architecture will occur as Agencies consider new information technology investments or make major enhancements/replacements to their existing systems. It is important to note that the Commonwealth ETA is not static; it needs to continue to evolve to support changing business strategies and technology trends.

Rationale:
Agencies can achieve the following benefits resulting from the Commonwealth's implementation of the ETA:

- Better responsiveness to changing business needs and rapidly evolving information technologies.
- Greater ease of software application integration and application interfacing.
- Easier secure access to data and information to enable interagency collaboration and sharing.
- Increased levels of application interoperability within the Commonwealth, with other states and municipalities, and with the Federal government.
- Increased sharing and re-use of current information technology assets.
- Faster deployment of new applications.
- Reduction in costs required to develop, support and maintain agency applications.

Definition of Key Terms

All of the Application Domain ETA standards and requirements considered to be critical components for implementing the Commonwealth's ETA are included in this report.

The report presents three forms of technical architecture guidance for agencies to consider when planning or when making changes or additions to their information technology:

- Requirements – mandatory enterprise technical architecture directions. All requirements are included within the ETA Standard.
- Technology Component Standard Tables - indicate what technologies or products that agencies may acquire at a particular point in time. These are mandatory when acquiring new or replacing existing technology or products. All technology component standard tables are included within the ETA Standard.
- Recommended Practices - provided as guidance to agencies in improving cost efficiencies, business value, operations quality, reliability, availability, decision inputs, risk avoidance or other similar value factors. Recommended Practices are optional.

The following terminology and definitions are applicable to the technology component standard tables presented in this report:

■ **Strategic:**

This technology is considered a strategic component of the Commonwealth's Enterprise Technical Architecture. It is acceptable for current deployments and shall be used for all future deployments.

■ **Emerging:**

This technology requires additional evaluation in government and university settings. This technology may be used for evaluative or pilot testing deployments or in a higher education research environment. Any use, deployment or procurement of this technology beyond higher education research environments requires an approved *Commonwealth Enterprise Technical Architecture Exception*. The results of an evaluation or pilot test deployment should be submitted to the **VITA Technology Strategy and Solutions: Policy, Practice and Architecture Division** for consideration in the next review of the Enterprise Technical Architecture for that technology.

■ **Transitional/Contained:**

This technology is not consistent with the Commonwealth's Enterprise Technical Architecture strategic direction. Agencies may use this technology only as a transitional strategy for moving to a strategic technology. Agencies currently using this technology should migrate to a strategic technology as soon as practical. A migration or replacement plan should be included as part of the Agency's IT Strategic Plan. New deployments or procurements of this technology require an approved *Commonwealth Enterprise Technical Architecture Exception*.

■ **Obsolescent/Rejected:**

This technology may be waning in use and support, and/or has been evaluated and found not to meet current Commonwealth Technical Architecture needs. Agencies shall not make any procurements or additional deployments of this technology. Agencies currently using this technology should plan for its replacement with strategic technology to avoid substantial risk. The migration or replacement plan should be included as part of the Agency's IT Strategic Plan.

Agency Exception Requests

Agencies that desire to deviate from the requirements or the technology component standards specified in this report shall request an exception for each desired deviation and receive an approved *Enterprise Technical Architecture Change/Exception Request Form* prior to developing, procuring, or deploying such technology or not complying with a requirement specified in this report. The instructions for completing and submitting an exception request are contained within the *Commonwealth Enterprise Architecture Policy*.

(This Page Intentionally Left Blank)

Application Scope

The Commonwealth relies heavily on computer applications to support agency business operations. The agencies' business processes often must change in response to both legislation and new demands from citizens. Unfortunately, the Commonwealth's computer applications can not always respond to these changes in an effective and efficient manner because many current applications are either monolithic or two-tier client/server applications.

Many of the Commonwealth's current applications/solutions were developed independently using different languages and tools. The ability to communicate with other applications or systems or to adapt to changes in the business processes generally was not a design requirement. This architectural approach has adversely impacted the Commonwealth's business in three ways:

1. Additional cost and time needed to modify existing applications to support changing business requirements
2. Difficulty in integrating applications to share common services and data
3. Extra expense to develop, use, and maintain new applications because there is little reuse of code between applications

Application development tools, methodologies and technology are now available that can help address these problems. Examples include:

- Reuse of Code: Units of code previously duplicated in many applications can be packaged into components or services for reuse in different applications.
- Integration tools/Middleware: Shared software allows applications to communicate with each other, access data residing on different platforms, and access shared services.
- New User Interface Options: There is an expanding array of user interface options - including Web browsers, personal digital assistants (PDAs), and interactive voice response units (IVRs).
- N-tier Service-Oriented Architecture (SOA): In the n-tier SOA, applications are partitioned into discrete functional units called "services." Each service implements a small set of related business rules or function points. If a business rule must be modified to support changing business requirements, only the service that implements that business rule is impacted. The remainder of the application remains intact. The SOA comprises loosely coupled (joined), highly interoperable application services that interoperate over different development technologies. The services are very reusable because the interface definition is defined in a standards compliant manner.

The ETA Application Domain provides agencies with a foundation of development and support platforms, tools, processes, practices and requirements that can implement business processes and meet the Commonwealth's ever changing business needs.

Overall Application Domain Scope

The domain report represents the work and decisions of the 2005-2006 Application Domain Team. The domain team identified six technology topics:

1. Enterprise System Design
2. Application Acquisition
3. Development and Support Platforms
4. Software Engineering
5. Geospatial Technologies
6. Enterprise Applications

The following hierarchy shows how each of these topics was sub-divided into technology areas (components) and the priorities [] for development:

[1]:	Team selected components to be addressed first
[2]:	Components that are expected to be addressed within 2 years
[EA]:	Components that are expected to be addressed as part of the CGI-AMS (PPEA) Enterprise Application initiative

Enterprise System Design Topic Components:

- Service-Oriented Architecture (SOA): Implementation and Governance [1]
- Standards-based service-oriented development model (SODA) [2]
- Enterprise Artifact Repository [1]
- Application Interfaces [1]

Application Acquisition Topic Components: [1]

Development and Support Platforms Topic Components:

- Enterprise Framework Platform [1]
- Wireless/Mobile Platform - Java 2 Platform, Micro Edition (J2ME) [2]
- Collaborative Platform [1]
- Development Languages [1]
- Coding Guidelines and Standards [1]
    - J2EE Guidelines [2]
    - .NET Guidelines [2]
- Integrated Development Environment (IDE) [2]
- Plug-ins: Application Program Interface (API)/Protocol [2]
- Object Relational (OR) Mapping [2]
- Application Platform Servers/Enterprise-Scope Application Platform Suites [2]
- Web Portal (includes Enterprise) [2]

Software Engineering Topic Components:

- Software Development Methodologies [1]
- Application Architecture and Design [1]
- Modeling [1]
  - Business Process Execution Language (BPEL) and other business modeling tools [2]
- Business Rules [1]
- Reusable Components [1]
- Presentation/Interface: [2]
  - Hyper Text Markup Language (HTML)
  - Dynamic/Server-Side Display
  - Content Rendering:
    - Dynamic HTML (DHTML)
    - eXtensible HTML (XHTML)
    - Cascading Style Sheets (CSS)
  - Wireless/Mobile/Voice:
    - Wireless Markup Language (WML)
    - XHTML Mobile Profile (XHTMLMP)
      - Voice XML (VXML)
- Configuration Management [1]
- Test Management [1]
- Defect Tracking [2]
- Change Management [2]
- Deployment Management [2]
- Requirements Gathering: Functional and Non-functional [2]
- Requirements Management and Traceability [2]
- Schedule and Task Management [1]

Geospatial Technologies Topic Components:

Mapping/Geospatial/Elevation/GPS (GIS):

- Integrated Development Environment (IDE)
- Geospatial Data Development Standards
- Database/Geospatial Metadata
- Utilities
- Reporting and Analysis
- Global Positioning Systems (GPS)

Visualization: [2]

- Graphing/Charting

Enterprise Applications Topic Components:

- eCommerce [1]
- Human Resources [EA]
- Financial Management [EA]
- Assets/Materials Management [EA]
- Human Capital/Workforce Management [EA]
- Customer Relationship Management [EA]

- Customer Initiated Assistance [EA]
- Supply Chain Management [EA]
- Document Management [2]
- Authentication/Single Sign-on (SSO) [EA]
- Search Engines [2]
- Audio and Video Conferencing [EA]

Scope of this Report

This report will address all of the components identified above having a priority of one [1].

As-Is Application Architecture

Data on over 1,600 agency applications was compiled as part of the 2003 and 2004 due diligence effort in preparation for negotiating partnerships with several companies interested in helping the Commonwealth modernize its infrastructure. Unfortunately, this data did not include specific enough information on all of the tools used to support agency software development. There was some data collected on agency use of application development languages:

Application Development Language	*Reported 2003 Agency Usage in Applications*
ASP	145
Assembler	5
C/C++	28
Clipper	7
COBOL	53
Cold Fusion	33
Infobasic	9
JAVA/JAVA Script	86
MAPPER	28
MS Access	36
MS Visual Basic/VB Script	286
Natural	46
Perl	8
PL/SQL	79
Powerbuilder	25

Over the next year, the Commonwealth will collect data on agency (excluding higher education) use of software tools to create, maintain and publish a complete as-is inventory.

To-Be Application Architecture

Although having a single software development/support product of each type might be the ideal, the reality is that agencies have unique application needs. The application requirements, technical and economic environments of each business unit will have a strong influence upon tool/product choices. Over time, agencies will group related tools into a limited number of standardized technology stacks. These “stacks” will provide agencies with cost-effective development solutions for N-tier, Collaborative, Business Intelligence (BI) or Analytical Technology and other solutions.

Future Application Domain Initiatives

The next version of this report and associated documents is expected to address:

- J2EE and .NET Guidelines
- Geospatial Technologies

The completed Software Tool Inventory will be analyzed by the domain team in setting the priorities of which of the above priority 2 components will be addressed in the future versions of this report. The Enterprise Applications Initiative progress will also drive when those technology components are addressed.

(This Page Intentionally Left Blank)

Domain-wide Principles, Recommended Practices and Requirements

The following principles, recommended practices and requirements pertain to all components, in all situations and activities related to the ETA Application Domain. Component specific principles, recommended practices and requirements are discussed in the next section of the report.

Domain-wide Principles

There were no domain-specific principles identified by the Application Database Domain team in addition to the principles identified in the "*Commonwealth of Virginia Enterprise Architecture – Conceptual Architecture*".

Domain-wide Recommended Practices

The following nine domain-wide recommended practices were identified:

APP-RP-01: **Adopt TCO Model for Applications and Technologies –** The Commonwealth and agencies should adopt a total cost of ownership model for applications and technologies which balance the costs of development, support, training, disaster recovery and retirement against the costs of flexibility, scalability, ease of use, and reduction of integration complexity.

Rationale:

- Leads to higher quality solutions.
- Enables improved planning and budget decision-making.
- Reduces the IT skills required for support of obsolete systems or old standards.
- Simplifies the IT environment.

APP-RP-02: **Software Acquisition Policy –** Any agency that purchases software should develop and implement a Software Acquisition Policy that covers the following:

- Determination of what documentation is necessary as evidence of licensing for each type of software the organization owns.
- Designation of centralized, safe location(s) for license documentation.
- Delegation of the responsibility and accountability for purchasing new software, maintaining records and updating the agency software inventory.
- Storage of the original documentation.

- Storage of the original media.

APP-RP-03: **Software Use Policy –** Agencies should develop and implement a Software Use Policy that covers the following:

- The agency's rules for downloading, installing, and using software titles.
- The review of the terms and conditions for each license to ensure proper usage.
- A software installation authorization process.

APP-RP-04: **Enterprise Shared Services –** The Commonwealth (excluding higher education) should create, deploy and support Enterprise Shared Services such as:

- Messaging: Enterprise Service Bus - Use of ESB provides an assured delivery mechanism that eliminates the need for developers to code for potential network failures
- Portal: Content publishing and management - provides users with a single point of access for all Commonwealth services. In addition, a consistent presentation layer greatly enhances the government application user experience.
- Security: Identity Management - eliminates the need for each application to perform authentications and maintain identity repositories
- Publish/Discover: Web Services Registry - run time service reuse only requires finding a service in the Registry and binding to it. This eliminates interoperability problems that typically impede code reuse, such as compiler versions, platforms and programming languages.

APP-RP-05: **Support Risk-Mitigation –** Agencies should maintain vendor or equivalent quality level support or have a risk-mitigation strategy for all software tools and hardware used to develop or support Commonwealth and/or agency "*mission critical applications*".

APP-RP-06: **Vendor Dependency Risk Mitigation –** The agencies should create and maintain a vendor dependency risk-mitigation strategy for all vendor dependent "*mission critical applications*".

Rationale:

- Applications that incorporate vendor specific product technologies risk becoming dependent on the vendor.
- Vendor dependent applications require that developers must maintain an in-depth knowledge of the vendor's products and planned future product changes.

APP-RP-07: **Service Level Agreement –** Agencies should implement a Service Level Agreement (SLA) and track/report SLA performance measurements for each deployed "*mission critical application*".

APP-RP-08: **Open Standards –** Agencies should select open-standards based products, tools, designs, applications, and methods where appropriate to reduce integration and infrastructure complexity.

Rationale:

- The use of standard interfaces and products that adhere to open standards will help reduce the complexity associated with the IT environment.
- Use of open standards-based products reduces the need to develop custom solutions to make components interoperable, thus reducing time and cost of developing and supporting new systems and upgrades.
- Costs associated with help desk support, training and total cost of ownership can also be reduced through the reduction in the complexity of the information infrastructure.
- Less complex structures and better integration means easier information access and sharing, encouraging use of the resources.
- Risks associated with system implementation and upgrades will be reduced. Applications will behave in a logically consistent manner across user environments.

APP-RP-09: **Industry Standards and Mainstream Technologies –** Agencies' applications/solutions should use, industry standards and industry-proven "state-of-the-art" mainstream technologies.

Rationale:

- Avoids dependence on weak vendors.
- Reduces risks.
- Ensures robust product support.
- Enables greater use of commercial-off-the-shelf solutions.
- Allows flexibility and adaptability in product replacement.

Domain-wide Requirements

The following four domain-wide requirements were identified:

APP-R-01: **Security, Confidentiality, Privacy and Statutes –** Agencies shall implement applications/solutions in adherence with all security, confidentiality and privacy policies and applicable statutes.

Rationale:

- Safeguards sensitive and proprietary information.

- Enhances public trust.
- Enhances the proper stewardship over public information.
- Ensures the integrity of the information.

APP-R-02: **Software Tools Version/Release Support –** The version/release levels of all software tools used for development and support of Commonwealth and/or agency "*mission critical applications*" shall have vendor or equivalent quality level support available.

Rationale:

- Unsupported software is no longer being updated to fix newly discovered security vulnerabilities or other problems that occur due to environmental changes.

APP-R-03: **Disaster Recovery and Business Continuity Planning –** An assessment of business recovery requirements is mandatory when acquiring, developing, outsourcing, or making major enhancements to "*mission critical applications*". Based on that assessment, appropriate disaster recovery and business continuity planning, design and testing shall take place.

Rationale:

- The pressure to maintain availability will increase in importance. Any significant visible loss of system stability could negatively impact our image.
- Continuation of business activities without IT is becoming harder.
- Application systems and data are valuable State assets that must be protected.

APP-R-04: **Maintain Software Tools Inventory –** VITA shall collect data on agency (excluding higher education) use of software tools, maintain an up-to-date inventory, and perform research in order to create a more effective and efficient environment in support of the Application Domain.

Rationale:

- No current agency software tool use data exists for most tool categories.
- The Commonwealth can negotiate lower prices based on larger quantities of products purchased from fewer vendors.
- Reducing the number of products can result in the availability to the agencies of higher levels of support that can be provided with the same or less resources.

Application Domain Technical Topics

The Application Domain defines all the technology components for Agency development and support of applications/solutions. The following discussion of each component identifies various Principles, Recommended Practices, Requirements and/or Product Standards. Requirements are conditions which must be met, i.e. are required and Product Standards are specifications for the use of specific hardware and software relative to the particular component:

The Application domain includes the following six technology topics:

1. Enterprise System Design
2. Application Acquisition
3. Development and Support Platforms
4. Software Engineering
5. Geospatial Technologies
6. Enterprise Applications

Enterprise System Design

Enterprise System Design refers to a collection of technologies, practices, requirements and standards that can assist the agencies in the design of solutions that can meet the Commonwealth's ever changing business needs.

The Enterprise System Design topic includes the following components:

- Service-Oriented Architecture (SOA): Implementation and Governance
- Standards-based Service-Oriented Development Model (SODA): To be addressed in future versions of this report
- Enterprise Artifact Repository
- Application Interfaces.

Service-Oriented Architecture (SOA): Implementation and Governance

In a Service-Oriented Architecture (SOA) environment, nodes on a network make resources available to other participants in the network as independent services that the participants access in a standardized way. Unlike traditional object-oriented architectures, a SOA comprises loosely coupled (joined), highly interoperable application services. Because these services interoperate over different development technologies (such as Java and .NET), the software components become very reusable due to the virtue of the interface definition being defined in a standards compliant manner (Web Service Definition Language [WSDL]). This also encapsulates and hides the vendor/language specific implementation from the calling client/service. SOA provides a methodology and framework for documenting enterprise capabilities and supports both integration and consolidation activities.

SOA-based composite applications will enable the Commonwealth to integrate business-critical processes with existing applications and systems. To gain the agility, flexibility and efficiency that SOA enables, these services and composite applications must be accessible and controlled across the enterprise.

The Commonwealth needs to implement a SOA as a foundation for Enterprise Applications and agency developed solutions for in-scope agencies. A key to successful implementation is SOA Governance.

SOA Governance is the ability to *ensure* that all of the independent efforts (whether in the design, development, deployment, or operations of a Service) come together to meet the enterprise SOA requirements.[3]

Rationale:
Well designed and consistently enforced policy and governance procedures are critical to the success of SOA, along with effective communication and collaboration. Gartner defines *policy*, in the context of SOA, as a set of guidelines, rules, regulations or requirements to be enforced on services. Examples are *security policies*, such as access and authentication; *management policies*, such as performance, monitoring and availability; *development policies*, such as development-language requirements; *routing policies*, such as content-based routing; *transformation policies*, based on document types or partner profiles; and *correct usage policies*, such as sequencing resources. [4]

An example of the importance of formalized policy, process, and governance procedures is in the design of new “public” services, those services with a reasonable chance of being shared or reused across multiple domains. Remaining services are sometimes called "private" services. When SOAs grow to more than 50 services (for example, by patching together two different SOA pilots on related sections of the IT infrastructure), their growth cannot be managed informally anymore, and must be disciplined (typically by an agreed governance mechanism, such as an ***Integration Competency Center*** [ICC]) to foster reuse and avoid duplication of services.[5] An example of the detailed functions of a centralized ICC are shown in the **Appendix A – SOA Centralized Implementation and Governance Model**.

Implications:
The Enterprise Architecture group (which consists of members from across state agencies and VITA’s ICC) will establish the enterprise SOA vision and the SOA reference architecture. A central integration group such as an ICC is typically established and

[3] *SOA Governance* (Source: **SOA Governance,** WebLayers, Inc. 238 Main Street, 4th Floor, Cambridge, MA 02142)

[4] *SOA Governance: Frameworks, Registries and Policy Enforcement*. **Gartner.** L. Frank Kenney and Daryl Plummer. 5-7 December 2005 JW Marriott Grande Lakes Orlando, Florida

[5] *The ICC and SOA Governance: Managing a Successful Integration Project.* Paolo Malinverno, Gartner Research

empowered within an organization as the approach to structured application integration matures. For “public” or shared services, agencies, localities and business partners may utilize VITA’s SOA backbone (see section on Enterprise Service Bus in Integration Domain Report).

In initial SOA projects, collaboration and communication can be managed informally. However, in order to obtain wide spread adoption and realize the full benefits, rigorous processes and strong governance are essential. Development and governance are typically coordinated through a central ICC.

Reuse is such a fundamental element of "value" around SOA that developer incentives must evolve from a project-centric focus on quantity of code to a more strategic focus on the number of services reused across projects and/or the number of reusable services created. Adjustments in developers' compensation typically goes a long way, possibly revamping HR policies to create more visibility, increase non-monetary compensation for good collaborators, provide for revised orientation and training to change the (traditional "I built it") culture in most developer communities, and so on. Focused coordinated organizational efforts – involving HR, application development (AD) management and key business stakeholders – to change the inherent behavior and culture of the developer community are a key ingredient of success.

In addition to development, deployment and architecture issues, client organizations must be prepared to address design, maintenance and management of an SOA, all of which are critical to project success.

The ownership of services must be defined upfront, including identifying which groups, individuals or roles develop and maintain specific service interfaces and/or service implementations.[6]

Principle:

The following one principle was identified:

APP-P-01: **Re-configure Existing Application Functionality as Reusable Business Processes –** Where cost-effective, agencies should support the transition to a SOA by re-configuring existing application functionality as reusable business processes.

Rationale:

- Allows the acceleration integration.
- Permits business process innovation while leveraging legacy investments.

Recommended Practices:

The following seven recommended practices were identified:

[6] *SOA Governance: Frameworks, Registries and Policy Enforcement.* L. Frank Kenney and Daryl Plummer. Gartner Research

APP-RP-10: **SOA Center of Excellence –** The Commonwealth should explore creating a SOA Center of Excellence to establish common standards, skills, and architecture migration strategies for information integration and reuse in support of an Enterprise-wide SOA implementation. The SOA Center of Excellence should provide guidance on SOA issues such as:

- The use of central seed-funding mechanisms to subsidize initial costs for reusable integration platforms in order to gain long-term cost advantages to the Commonwealth.
- Building a centralized portfolio of reusable development assets (services, components, objects, and adaptors) to reduce agency development cycles and costs. How testing services must be done.
- How teams cooperate to deploy SOA systems.
- What roles must exist and how to ensure that reuse occurs.
- How to achieve effective service composition.
- How to govern orchestrated services.
- Design for extensibility and reuse.
- Deciding which new functionality should be exposed as a service.
- Loosely coupling services to support broad interoperability when requirements change.
- Designing appropriate modularity and granularity of services.
- Encapsulating business processes into well-defined, self-contained, course-grained services.
- Providing interoperable access to published services.
- Accessing services through standardized, platform-neutral, self-describing, well-structured, and extensible messages.
- Separating the service interface from its implementation.
- Describing services using a standard format.
- Publicizing and discovering services using standard service registries.
- Utilizing standard protocols for exchanging messages and data between services.

APP-RP-11: **Governance of New or Shared Services –** To ensure reduced integration costs and complexities, limit liabilities such as security, and to effectively compete in the marketplace, the Commonwealth should govern the design, development, deployment, and operations of any new, and or shared, services across the enterprise. The SOA Center of Excellence should also perform Enterprise-wide architecture and design review during the design phase of any application (excluding higher education) that will either use or create web services. The SOA Center of Excellence would be responsible for establishing web service granularity, testing and deployment requirements as well as the marketing of reusable services to agencies and partners.

Rationale:

- After an agency has built a standard solution, the SOA Center of Excellence's final task is to internally merchandise these as preferred solutions, ensuring that other agencies are aware of the favorable speed and cost economics of reusing the existing solution.
- SOA Center of Excellence's review can also evaluate large and complex applications that that are under development to ensure that the solutions will be:
    - *Scalable:* The application can handle potentially higher traffic loads in the future.
    - *Extensible:* The application can serve as the platform for the development of future functionality.
    - *Secure:* The application ensures user privacy.
    - *Available:* The application is operational and accessible as required to meet customer needs.

APP-RP-12: **Centralized Governance Model –** A centralized governance model is recommended for entities planning to implement "public" or shared services, likely to be reused across the enterprise.

APP-RP-13: **SOA Centralized Operations Model –** A SOA Centralized Operations model should be implemented.

Rationale:

- It will be less costly to invest in SOA operations staff and a SOA infrastructure in a single data center.
- It will be far easier to manage shared web services that are all running in the same environment, and then managing web services that are spread out across multiple data centers. Centralized operations means configuring only one set of network devices (firewalls, routers, etc.) and one set of application platforms.
- It will be much easier to troubleshoot performance, availability, and scalability problems if all shared services are running in a single environment.
- Security will be more manageable if "circles of trust" are established between services in the same environment.
- Backup and disaster recovery will also be simplified.

APP-RP-14: **Service Contract Policies –** Service performance contracts should be published by a producing department for a given web service. Consuming organizations would build their applications around these contracts. The contract process itself would be established by the Governance portion of the

SOA. Actual service performance would be monitored by the individual data centers. For enterprise public service applications, performance could be monitored by VITA's ICC infrastructure. In that scenario, VITA's ICC and the publisher would collaborate with the producing development organization to fix any problems.

APP-RP-15: **Centralized Integration Competency Center (ICC) –** A centralized Integration Competency Center (ICC) should be built and empowered within an organization as the approach to structured application integration matures. [7] The Integration Competency Center (ICC) and the enterprise architecture group typically establish the SOA vision and the SOA reference architecture. An example of a centralized implementation and governance model organization is shown in *Appendix A– SOA Centralized Implementation and Governance Model.*

APP-RP-16: **Service Location Transparency.** Service location should be transparent to applications looking up services in a shared Registry.

Rationale:

- Improves code mobility because services can be moved to different machines, or to external providers.

Requirements:

The following four requirements were identified:

APP-R-05: **Implement SOA –** Agencies excluding higher education shall create and implement the centralized architectural review processes that are needed to support and control SOA implementation ensuring that all services built conform to standards, are interoperable, non-duplicative, and reusable where possible.

APP-R-06: **SOA Support of .NET and J2EE (Java Platform Enterprise Edition) –** The Commonwealth's SOA for in-scope agencies shall support both .NET and J2EE Enterprise Framework Platforms.

APP-R-07: **SOA Center of Excellence Review of Developed Applications –** VITA, together with other executive branch agencies, shall create recommended practices

[7] *The ICC and SOA Governance: Managing a Successful Integration Project*. Gartner Research, Paolo Malinverno

and requirements to implement the SOA Center of Excellence enterprise level (state-wide excluding higher education) architectural design review and architectural governance of agency developed new applications that are large-scale, complex, use/create web services, or can potentially share business processes with other agencies.

APP-R-08: **SOA Center of Excellence Review of COTS (Commercial off-the-shelf) –** VITA, together with other executive branch agencies, shall create Enterprise level (state-wide excluding higher education) architectural review recommended practices and requirements to support agency's review/selection and implementation of COTS based solutions that implement Enterprise-wide Applications or cross-cutting functions (such as accounting, facilities management or procurement).

Enterprise Artifact Repository

As part of a systems acquire/develop decision, agencies should first consider the reuse of existing applications and system components/artifacts. To be successful, a state-wide library (repository) of reusable components and artifacts must be implemented and maintained.

Designers can build flexible, scalable, and extensible applications by using components as application building blocks, similar to building cars on an assembly line. Using previously built and tested components in different ways or with new components can accelerate the design, development, and delivery of new applications. Sharing of components across applications can also eliminate significant duplicate design and test efforts.

There are two strategies for reuse:

1. Opportunistic reuse: using assets that were not designed to be reused or are reused in a manner for which they were not designed
2. Systematic reuse: using assets which were purposefully designed, built, and managed to be reused

Systematic reuse has several advantages:

- Responsiveness: accelerates and streamlines project delivery
- Return on Investment (ROI): reduces solution delivery costs and provides only those assets that produce the best business advantage
- Quality: ensures that only quality assets will be reused

Both reuse strategies require an implemented Enterprise Artifact Repository with supporting practices and processes to be successful.

Requirement:

The following requirement was identified:

APP-R-09: **Implement Enterprise-wide Artifact Repository –** The Commonwealth shall select, deploy and maintain an Enterprise-wide Artifact Repository to support implementation of a SOA and create recommended practices and processes that support and encourage agency use of the Repository.

Application Interfaces

Interfaces define the capabilities of communicating, transporting and exchanging information through a common dialog or method. Delivery channels enable the information to reach the intended destination, whereas Interfaces allow the interaction to occur based on a predetermined framework. A Web-Services User Interface (WSUI) uses a simple schema for describing a "component" that can be used in a portal to call backend SOAP and XML services. WSUI uses XSLT style sheets to construct user-facing views to enable users to interact with the services.

In n-tier applications, changes in business rules normally do not require changes in interface code. Interfaces may need updating for other reasons. Examples include when changes occur in another computer system that interfaces with that application or when users need a graphical user interface instead of a character-based interface for that application.

Recommended Practices:

The following two recommended practices were identified:

APP-RP-17: **Interfaces Should Utilize Web-services –** All interfaces for newly developed or purchased (COTS) applications, reusable components and services should utilize web-services developed to industry/open standards.

Rationale:

- All interfaces should be based on an industry-defined set of open standards to limit the potential for vendor dependency and to reduce development complexity.
- Reuse is a key goal of service-oriented architectures (SOA)
- Ease of reuse can be maximized by developing and designing open interfaces based on industry standards.

APP-RP-18: **Interfaces Should be Message-Based –** The interfaces between separate application systems should be *message-based*; this applies to both internal and external systems.

Rationale:

- The use of messaging is important for enforcing the architecture principle of logical partitioning and boundaries.
- Enables rapid response in maintenance and enhancement activities as required by changes in business processes.
- Messaging technology simplifies integration efforts.
- Messaging technology allows for transparency in locations, databases, and data structures.

Application Acquisition

The choice of a systems acquisition method (buy/build decisions) should take into account the functional characteristics of the proposed systems. The agencies should first consider the reuse of existing applications and system components. If no components exist, purchased solutions (COTS) should be explored. Applications or systems that can provide automation of agency core business functions that have unique processes, yield competitive advantages, or have demonstrable cost savings and/or enhanced value should be the only candidates for in-house development by the Commonwealth. The Application Acquisition topic is not broken down into components.

Principle:

The following one principle was identified:

APP-P-02: **Acquisition Method Choice –** The choice of a systems acquisition method (buy/build decisions) should take into account the functional characteristics of the proposed systems. The agencies should first consider the reuse of existing applications and system components. If no components exist, purchased solutions (COTS) should be explored. Applications or systems that can provide automation of agency core business functions that have unique processes, yield competitive advantages, or have demonstrable cost savings and/or enhanced value should be the only candidates for in-house development by the Commonwealth.

Rationale:

- Use and availability of effective packaged solutions is increasing.
- Using tested solutions reduces risks.
- Reduces the total cost of ownership.
- The more you're "like" everyone else (e.g., same standard, same systems), the easier it is to share with others.

Commercial off-the-shelf (COTS) is a term for software or hardware products that are ready-made and available for sale to the general public. They are often used as alternatives to in-house developments or one-off government-funded developments (government off-the-shelf [GOTS]). The use of COTS is being mandated across many government and business programs because they may offer significant savings in procurement and maintenance.

Recommended Practices:

The following six recommended practices were identified:

APP-RP-19: **Use COTS when Cost-effective and Beneficial –** Systems and components of systems should be implemented using commercial off-the-shelf (COTS) products when they represent the most cost-effective (total cost of ownership) and beneficial solution.

Rationale:

- The use of COTS products is potentially more cost-effective and efficient than other approaches because of reduced development, implementation, maintenance, and training costs.
- The use of COTS solutions offers the promise of reduced development time, increased development productivity and improved system quality.
- Buying existing commercial services may provide the best value solution for parts of work processes.

APP-RP-20: **COTS Escrow –** All agency contracts with COTS solution vendors should require that the solution’s source code and documentation be placed in escrow. It is recommended that the contact also make provision for the agency inspection (vendor supervised) of the source code and documentation upon request.

APP-RP-21: **Prefer COTS Web-services/SOA –** Newly acquired COTS solutions that utilize web services and support a Service-Oriented Architecture (SOA) are preferred over those COTS solutions that do not.

APP-RP-22: **COTS Web-services Access to Data –** Newly acquired COTS solutions should include web services that provide access to the applications data.

APP-RP-23: **Customize COTS to Meet Business Needs –** Commercial off-the-shelf (COTS) should be customizable to meet business needs either by the purchasing agency or by the vendor as a paid-for-service.

APP-RP-24: **Limit COTS Customizations –** Agencies should limit and/or isolate any customizations made to COTS solutions. All of the customizations should be fully documented.

Rationale:

- Isolating COTS customizations improves the ability to upgrade and move to new releases.
- Customization of COTS solutions can represent a significant burden in the management, implementation and support of technical infrastructure.
- Customizing COTS solutions can add value/functionality to the business that may outweigh the burden and proper documentation of the customization can minimize the negative impact.

Requirements:

The following two requirements were identified:

APP-R-10: **Evaluate COTS as Alternative –** Commercial off-the-shelf (COTS) solutions shall be evaluated and documented as part of an Alternatives Analysis of systems acquisition methods for all Enterprise-wide Applications and cross-cutting functions (such as accounting, facilities management or procurement).

APP-R-11: **COTS Documentation –** All *"mission critical"* COTS solutions shall have their application components and configurations fully documented.

Development and Support Platforms

The complexity, size, lifespan, and performance requirements of agency developed applications/solutions vary greatly. Development and Support Platforms provide the agencies with distinct approaches to address different application needs/ requirements.

These approaches can be implemented by the following development platforms:

- Enterprise Framework Platform – supports n-tier development of service-oriented architecture for large-scale or complex applications that need to support high-volume usage and/or long life spans.
- N-tier Visual-based Tool Development Platform – supports applications that are not large-scale, complex and do not require high-volume usage and/or long life spans. Generally developed by Business Analysts by using visual-based tools that provide automated code generation.
- Collaborative Platform – many business' needs do not require scalable or highly available solutions. These needs often can be met by Workflow and Forms Automation tools.

~~Please see "*N-Tier Application Development with Microsoft.NET* "by Karim Hyatt for an excellent introduction to N-Tier Architecture. http://www.microsoft.com/belux/msdn/nl/community/columns/hyatt/ntier1.mspx~~

Application development tools are critical to the development and support of applications. Regardless of the tools used, it is important to design each tier to be portable across platforms. Tool limitations can, however, impact tradeoffs in an application's design/architecture. The architecture should determine the tool selection, not the other way around.

There are three approaches for selecting tools to develop n-tier applications:

Best of breed. Use separate, specialized tools for each application tier. Use middleware to support communications between the different tiers.

Front end/back end. Two different tools are used: a specialized user interface development tool and an integrated tool set that provides middleware for the business rules and data access tiers. The middleware must support communications between the user interface and other two tiers.

Integrated. Integrated tool sets are used that generate code for all tiers of the application. These tools provide the middleware necessary to support communications between all application tiers.

N-tier service-oriented application architectures require additional types of tools.

- Repositories (libraries) to keep track of business rules that have been automated by components.
- Software management tools to provide version control, configuration management, and software distribution services.

Recommended Practices:

The following two topic-wide recommended practices were identified:

APP-RP-25: **Agency Development and Support Platforms –** An agency's architectural strategy should define the available development and support platforms and the process for selecting the appropriate platform. Examples of development and support platforms are:

- Enterprise Framework Platform – supports n-tier development of service-oriented architecture for large-scale or complex applications that need to support high-volume usage and/or long life spans.
- N-tier Platform – supports applications that are not large-scale, complex and do not require high-volume usage and/or long life spans. Generally developed by Business Analysts by using visual based tools that provide automated code generation.
- Collaborative Platform – many business needs do not require scalable or highly available solutions. These needs can be often met by Workflow and Forms Automation tools.

APP-RP-26: **Agency Standardized Technology Stacks –** Agencies should develop a limited number of standardized application development toolsets ("standardized technology stacks" or "hardened" architectural patterns). Technology Stacks serve as reusable technology packages that contain a complete (A - Z) integrated/best-of-breed set of development tools, frameworks and platforms needed to develop and support an application/solution. Examples:

- N-tier Technology Stack
- Collaborative Technology Stack
- Business Intelligence (BI) or Analytical Technology Stack

Objectives:

- Improve how the Commonwealth designs systems in order to have modern, common, and effective platform reuse.
- Improve how the Commonwealth converse/collaborate in the design and sharing of experiences/practices/services.
- Improve how we articulate the benefits of systems built on platforms that are composed of standards, Patterns and services.

Rationale:

- Standardized system analysis, design, and development tools will help IT managers create a consistent and repeatable systems development life cycle, resulting in a predictable and efficient process that will improve over time.
- An eventual standardized development environment will result in long-term cost savings and elimination of redundancies in data management and systems design.
- Standard tools will ensure continuity in the systems development life cycle across contractor transitions. Contractors will be required to use the standard Agency tools rather than their own preferred tools.
- Toolsets that align with the agency architecture and development philosophy will better support end-to-end design, development, and deployment of applications.
- Historically, project teams selected tools first, and then had to live with the architecture those tools supported. That led to the problem of the tools driving the architecture, and thus the business, rather than the business requirements mandating the tools.
- Reduces total cost of ownership.
- Produces higher quality solutions
- Avoids disjointed steps in the software engineering process
- Helps automate proper technical documentation
- Documents metrics that facilitate process improvement

The Development and Support Platforms topic includes the following components:

- Enterprise Framework Platform
- Collaborative Platform
- Development Languages
- Coding Guidelines and Standards
- The following will be addressed in future versions of this report
    - Wireless/Mobile Platform - Java 2 Platform, Micro Edition (J2ME)
        - J2EE Guidelines
        - .NET Guidelines
    - Integrated Development Environment (IDE)
    - Plug-ins: Application Program Interface (API)/Protocol
    - Object Relational (OR) Mapping
    - Application Platform Servers/Enterprise-Scope Application Platform Suites
    - Web Portal (includes Enterprise)

Enterprise Framework Platform

Java 2 Platform Enterprise Edition (J2EE) and .NET are the two dominant distributed computing architecture frameworks. J2EE provides portability of a single language (Java) over multiple operating systems and hardware platforms. .NET supports a wide range of languages but is primarily tied to the Microsoft Windows operating system and Intel hardware.

Recommended Practices:

The following three recommended practices were identified:

APP-RP-27: **Recommended Enterprise Frameworks –** The Commonwealth's recommended Enterprise Frameworks are .NET and J2EE.

APP-RP-28: **Use of Enterprise Frameworks –** Agencies new large, complex applications (either developed or purchased COTS solutions), that are anticipated to have high usage volumes and/or long life spans should utilize an Enterprise Framework in the development of applications and services.

Rationale:

- Frameworks provide an efficient, distinct, reusable, and unified software infrastructure.
- Frameworks include presentation services, server-side processing, session management, business logic framework, application data caching, application logic, persistence, transactions, security, and logging services for applications.
- Frameworks provide platforms, tools, and programming environments for developing multi-tiered distributed applications.
- Frameworks provide support for the creation of Web services.

APP-RP-29: **Use of N-tier Architecture –** Agencies large, complex applications (either developed or purchased COTS solutions), that are anticipated to have high usage volumes and/or long life spans should utilize an n-tier architecture and support a future implemented service-oriented architecture (SOA).

Rationale:

- While many problems inherent in the State's existing monolithic and two-tier applications could be overcome by implementing applications with a three-tier architecture, the Commonwealth will be better served by an n-tier service-oriented architecture.
- N-tier applications are easily modified to support changes in business rules.
- N-tier applications are highly scalability, availability, and manageability.
- An n-tier architecture offers the best performance of any application architecture.
- Any combination of user interfaces (e.g., character, graphical, web browser, and telephone interfaces) may be implemented in an n-tier application.
- N-tier applications are less expensive to build and maintain because much of the code is pre-built and shared by other applications.
- Enables simplification of the environment and geographical independence of servers.
- Takes advantage of modular off-the-shelf components.
- Reuse will lower costs and maintenance efforts.

- Allows for leveraging skills across the enterprise.
- The ability of an n-tier application to adapt to changing business needs ensures longevity of the applications useful life.

Collaborative Platform

Collaborative Platform consists of Workflow Automation and Forms Automation.

Workflow Automation: the automated management of the flow of material, information, and knowledge through a well-defined process. The core of workflow is the automation of information-based tasks and activities, but workflow processes must also be able to keep track of non-electronic objects through their association with electronic identification (for example a barcode). Knowledge is captured in the rules that are embedded in the automated processes during its creation. These rules include schedules, priorities, routing paths, authorizations, security and the roles of all the people and computer systems in the process. The development environment for workflow includes analysis and design tools that facilitate change through rapid deployment of workflow applications and reuse of process elements.

Forms Automation: defines the set of capabilities that support the creation, modification and usage of physical or electronic documents used to capture information within the business cycle. Forms automation includes:

- the use of workflow technologies for the automatic routing of electronic forms (e-forms) among a group of people responsible for processing them;
- the initiation of fully automated processing of information contained on e-forms;
- the validation or insertion of data on e-forms via database look-up;
- combinations of these three functions;
- plus audit trail creation,
- exception notifications and archive functions.

Workflow standards developed by the Workflow Management Coalition are expected to provide interoperability between workflow software and applications as well as between different workflow systems. As is normally the case, international standards are not consistently implemented from one product to the next. Careful assessment of integration issues will be necessary especially to facilitate decisions for multi-agency event-driven systems.

Principle:

The following one principle was identified:

APP-P-03: **Workflow Systems Conformance –** Agencies should implement workflow systems that conform to the interface specifications of the Workflow Management Coalition (WfMC).

Rationale:

- The Workflow Management Coalition (WfMC) has established a framework for workflow standards. This framework contains interoperability and communication standards that enable allow multiple workflow products to coexist and inter-operate. This framework includes five interface specifications:

1. Process Definition Tool.
2. Workflow Enactment Services.
3. Workflow Client Applications.
4. Invocation of Native Applications.
5. Workflow Package Interoperability

Recommended Practices:

The following four recommended practices were identified:

APP-RP-30: **Provide Basic Collaboration Services –** An agency-standardized set of basic collaboration services should be provided to all employees as required to meet business needs.

Rationale:

- Increases productivity.
- Reduces costs of maintenance.
- Provides the basis for multi-agency or Enterprise-wide business initiatives.
- Provides for universal employee access to information.
- Leverages the investments made in technology.

APP-RP-31: **Selection of Workflow Automation Tools –** Agencies should select workflow automation tools that provide monitoring of work-in-process and reporting of production statistics.

Rationale:

- Workflow technologies should support the capability to collect, analyze, and report metrics.
- Workflow automation tools can be used to identify inefficiencies and bottlenecks within the workflow/process.
- Workflow automation tool features can be used to identify problems allowing management the opportunity to balance and modify the workflow.

APP-RP-32: **Document Workflow Through Use Cases –** Agencies should define and document business and workflow processes through use cases.

Rationale:

- Workflow automated business processes need to be defined clearly, concisely, and unambiguously.
- Use cases describe workflow processes from a business perspective.
- Use cases document agreed upon business goals, participants, and outcomes of a business process/workflow.
- Use cases provide definition of the automated and manual processes required by the workflow.

APP-RP-33: **Classify Collaborative System Content –** When designing collaborative systems (e.g. document management, workflow), the content that will move through the system should be classified according to applicable statutes, policies and regulations pertaining to availability, retention and security.

Rationale:

- Information in collaborative systems is another type of Commonwealth information that must be managed according to the same principles of stewardship as structured data.
- The Commonwealth must minimize the exposure and liability of mismanaging information stored in collaborative systems.
- To make information easily shared, it must be classified.

Development Languages

There have been thousands of different programming languages and new ones are created every year. Every language has its strengths and weaknesses. For example, FORTRAN was (and still is) a particularly good language for processing numerical data, but it does not lend itself very well to organizing large programs. Pascal was very good for writing well-structured and readable programs, but it is not as flexible as the C programming language. C++ embodies powerful object-oriented features, but it is complex and difficult to learn.

The Commonwealth will continue to use specialized development languages as required to meet special needs (example: FORTRAN for engineering applications). With the exception of these special needs applications, in-house development should use languages that are consistent with the creation of SOA n-tier solutions on Enterprise Framework Platforms such as .NET and J2EE.

Technology Component Standard

The technology component standard table below provides strategic technology directions for agencies that are acquiring languages used in the development of new large, complex applications that are anticipated to have high usage volumes and/or long life spans.

Table APP-S-01: Languages used in developing new large, complex applications anticipated to have high usage volumes and/or long life spans Technology Component Standard
Strategic:
Java, Visual Basic, C++, VB.NET Fortran (for engineering applications only)
Emerging:
Transitional/Contained:
Cobol, Power Builder, PL/SQL, Delphi, MAPPER (BIS, Cool Ice)
Obsolescent/Rejected:
Assembler, C, Clipper, Basic, PL/1
Exception History:

Coding Guidelines and Standards

Coding Guidelines and Standards (also called programming style or code convention) describe conventions for writing source code in a given programming language.

Recommended Practice:

The following recommended practice was identified:

APP-RP-34: **Adopt Coding Standards –** Agencies should adopt coding standards for all widely-used languages and platforms.

Rationale:

- Coding standards make debugging and maintenance easier.

Examples:

- Naming conventions: variables, constants, data types, procedures and functions.
- Code flow and indentation.
- Error and exception detection and handling.
- Source code organization, including the use of libraries and include files.
- Source code documentation.

Requirement:

The following one requirement was identified:

APP-R-12: **J2EE and .NET Guidelines –** The Commonwealth shall research and publish recommended practices supporting agency development of applications/solutions using J2EE and .NET Enterprise Frameworks.

Software Engineering

Software Engineering is the application of best-practice processes and methods of design to the development and maintenance of software applications/solutions. Software engineering covers not only the technical aspects of building software systems, but also development management issues, such as testing, modeling and versioning.

Principles:

The following two principles were identified:

APP-P-04: **Adopt Consistent Software Engineering Practices –** The Commonwealth shall create, adopt and utilize consistent software engineering practices and methods based on accepted industry standards.

Rationale:

- Reduces training costs.
- Leads to benchmarks for measurement.
- Enables improved quality assurance.
- Facilitates the reuse of programming modules and code.

APP-P-05: **Deploy Event-driven Applications –** The Commonwealth should deploy application systems that are event-driven (driven by *business events)*, employing a real-time processing methodology versus batch processing.

Rationale:

- Increases adaptability.
- Business processes are a series of business events.
- Business process changes involve the adding, removing, or changing of business events.
- Increases linkage to the business.
- Mirrors the actual business environment.
- Easier to realign IT when change occurs.
- Real-time event processing supports rapid response to business events and an up-to-date data environment.
- Real-time event processing is essential to 7 days a week and 24 hours a day operations, ensuring that customers have access to current data on an as-needed basis.

Recommended Practice:

The following one recommended practice was identified:

APP-RP-35: **Fully Documented Requirements –** Systems/solutions should be created based on fully documented requirements that are reviewed and approved by the business owner/sponsor. These documented requirements should be maintained throughout the software development life cycle.

Requirements:

The following four requirements were identified:

APP-R-13: **Commonwealth Web and Accessibility Standards –** Public-facing and Web applications (Intranet and Internet) shall comply with Commonwealth Web and Accessibility Standards as applicable.

Rationale:

- Persons with disabilities will be able to use all applications.
- A consistent user interface can improve user productivity by supporting integrated use of applications.
- A consistent user interface also facilitates training staff on how to use new applications. Ease of use will improve productivity.
- Training will be less costly and time consuming.
- Staff will intuitively learn new applications.
- Applications using a consistent interface will be more readily accepted and used.
- A consistent user interface promotes application portability and facilitates development of future applications.

APP-R-14: **Public Web Applications Browser Independent –** Agency public-facing web-based solutions shall be browser independent (the functionality of the application can not be restricted to a single browser).

Rationale:

- Utilizing browser specific features can limit the solution audience and can lead to vendor dependencies.
- Solutions developed for a specific browser will have higher support costs when other browsers or version upgrades are required in the future.

APP-R-15: **Maintain Application Code Documentation –** All newly developed applications shall have their code documented. This documentation shall be maintained throughout the product life cycle.

APP-R-16: **Accessible and Transferable Repositories –** All electronic repositories of source code, metadata, development artifacts, models, documentation, etc. shall have their contents accessible either by an export facility or direct access method. This ability is required to allow the repository contents to be transferred from one methodology or tool to another as needed.

The Software Engineering topic includes the following components:

- Software Development Methodologies
- Application Architecture and Design
- Modeling
- Business Rules
- Reusable Components
- Configuration Management
- Test Management
- Schedule and Task Management
- The following will be addressed in future versions of this report
    - Business Process Execution Language (BPEL) and other business modeling tools
    - Presentation/Interface
    - Defect Tracking
    - Change Management
        - Deployment Management
    - Requirements Gathering: Functional and Non-functional
    - Requirements Management and Traceability

Software Development Methodologies

Software development methodologies provide a framework that is used to structure, plan, and control the process of developing an information system. The advancement of new technologies has evolved new methodologies that allow both rapid and flexible delivery. Methodologies can be divided into several groupings:

Linear:	SDLC and Waterfall
Iterative:	Prototyping, Spiral, Rapid Application Development
Rapid Response:	Extreme Programming and Adaptive Software Development
Parallel:	Alternative Path

The **linear** methodology group represents the more traditional model and is best suited for projects exhibiting the following characteristics:

1. Clear project objectives
2. Stable project requirements
3. Knowledgeable user
4. No immediate need to install
5. Inexperienced team members
6. Fluctuating team composition
7. Less experienced project leader
8. Need to conserve resources
9. Strict requirement for approvals

The **Iterative** approach provides both for prototyping or a more complex method such as spiral.

The Rational Unified Process (RUP) is an iterative software development process created by the Rational Software Corporation, now a division of IBM. The RUP is not a single concrete prescriptive process, but rather an adaptable process framework. As such, RUP describes how to develop software effectively using proven techniques. While the RUP encompasses a large number of different activities, it is also intended to be tailored, in the sense of selecting the development processes appropriate to a particular software project or development organization. The RUP is recognized as particularly applicable to larger software development teams working on large projects.

Prototyping lets users work with a small-scale mock up of their system, experience how it might function in production, and request changes until it meets their requirements.

1. Project objectives are unclear
2. Functional requirements are changing
3. User is not fully knowledgeable
4. Immediate need to install something
5. Experienced team members (particularly if the prototype is not throw-away)
6. Stable team composition
7. Experienced project leader
8. No need to absolutely minimize resource consumption
9. No strict policy or cultural bias favoring approvals
10. Analysts/users appreciate business problems involved, before they begin project
11. Innovative, flexible designs that will accommodate future changes are not critical

Where **Spiral** model is highly customized to each project, and thus is more complex.

1. Risk avoidance is a high priority
2. No need to absolutely minimize resource consumption
3. Project manager is highly skilled and experienced
4. Policies or cultural bias favor approvals
5. Project might benefit from a mix of other development methodologies
6. Organization and team culture appreciate precision and controls
7. Delivery date takes precedence over functionality, which can be added in later version

Rapid Response methodologies are aimed at creating a lighter, faster, more flexible and responsive approach to development.

1. Rapid installation of the bulk of the system is not a critical goal
2. Users have ability to make rapid, binding decisions
3. Users are flexible and willing to work through many small implementations
4. Collaborative team atmosphere
5. Team with substantial system design experience
6. Experienced project leader
7. Minimal pressure to conserve resources with some model

With **Parallel** models, different approaches are tried at the same time by different individuals or teams, and then as the project progresses, less productive paths are pruned.

1. Rapid installation is a primary goal
2. Solid, experienced team
3. Strong project management
4. Excellent project-related communications
5. Stable team composition
6. Experienced project leader
7. Little pressure to conserve resources
8. Uniquely flexible development team

It is considered a best practice to use an established software methodology determined by project risk, maturity of user requirements, project leader experience, team stability and technical expertise, schedule, and budget to develop information systems. The methods and criteria provided above are not all inclusive but are provided as a launching point to assist in selection of a software development methodology.

Recommended Practice:

APP-RP-36: Chose Appropriate Development Methodology – Agencies should chose an appropriate established software development methodology based on project risk, maturity of user requirements, project leader experience, team stability and technical expertise, schedule, and budget.

Rationale:

- Software development methodologies provide a framework that is used to structure, plan, and control the process of developing an information system.
- The advancement of new technologies has evolved new methodologies that allow both rapid and flexible delivery (ie, Iterative and Rapid Response).
- Traditional linear methodologies (ie, Waterfall and SDLC) are most appropriate for projects with stable requirements and long-term implementation schedule.

Application Architecture and Design

There are five different application architectures currently within the Commonwealth:

1. Monolithic Applications

Monolithic applications are applications where the code that implements the business rules, data access, and user interface are tightly coupled together as part of a single, large computer program. A monolithic application typically is deployed on a single platform, often a mainframe or midrange computer. There are examples of monolithic applications running on smaller systems - or even distributed across multiple machines. The determining characteristic of a monolithic application is that the code is tightly coupled and highly interdependent.

Monolithic computer applications are deployed across Virginia. Since the Commonwealth provides many different services to its citizens, there are many computer applications to support those services. In most cases, these applications were developed independent of each other using different combinations of technology. For example, one agency application may use COBOL, CICS, and VSAM. Another application to support the same group of citizens may use COBOL and IMS.

Monolithic applications have several drawbacks:

- *It is costly and time consuming to modify monolithic applications.*
  Changing one piece of code that implements a business rule, accesses data, or provides an interface to users or other systems likely impacts other code in the application. When any code in a monolithic application changes the entire application must be re-tested and re-deployed.
- *It is difficult to integrate monolithic applications to share services and data.*
  Most monolithic applications do not have well-defined interfaces that can be accessed by other applications or new user interfaces.
- *There is little reuse of redundant code between monolithic applications, making it more expensive to build and maintain them.*
- Many monolithic applications contain functionality already replicated in other applications. Monolithic applications are slower and more costly to build because existing functionality must be reinvented many times. Monolithic applications are more expensive to operate, since the same data often has to be gathered, entered, and stored in many places.
- *It is difficult to have monolithic applications communicate with other applications.*
  Most existing applications do not have the ability to communicate with other applications, within an agency, and with applications in other agencies.
- *Monolithic applications can be accessed using only a single user interface.*
  Most monolithic applications were developed to be accessed via mainframe terminals. Having a single user interface is a limitation when application services need to be accessed from other user interfaces such as Web browsers or the telephone (via IVRs).

- *There is little flexibility where monolithic applications can be deployed.*
  Most monolithic applications must be deployed on a single machine type, for example a mainframe. This could be because either the software code is tightly coupled to that machine type or the mainframe is needed to get enough processing capacity to process all parts of the application: the user interface, the business rules, and the data access code.

2.Two-tier client/server applications

Some agencies have attempted to overcome the business impact of monolithic applications by adopting client/server technology for new applications. The terms "client/server", "client", and "server" are often misunderstood. Many believe that "client/server" means an application with a graphical user interface and a relational database. Neither is necessarily true. In fact, client/server applications are constructed of software "clients" that, in order to perform their required function, must request assistance - "service" - from other software components known as "servers." Middleware provides communication between the client and server.

Early client/server applications used architectures dictated by the tools used to write them. As a result, most early applications used a *two-tier client/server architecture*. The "tiers" of client/server applications refer to the number of executable components into which the application is partitioned, not to the number of platforms where the executables are deployed. Sometimes, the tiers into which the application is partitioned is called "logical partitioning", and the number of physical platforms on which it is deployed is called "physical partitioning."

In two-tier client/server architecture, application functionality is partitioned into two executable parts, or "tiers." On one model, one tier contains the code that implements a graphical user interface (GUI) and the code that implements the business rules. This tier executes on PCs or workstations and requests data from the second application tier, which usually executes on the machine where the application's data is stored. This model is referred to as *two-tier, fat client*. Though while the application has two tiers of executable code, most of the code is contained in the tier executing on the workstations - the "fat client."

Since business rules are tightly integrated with user interface code, the code that implements the business rules must be deployed on the same platform(s) as the user interface. Thus, the entire workstation-resident portion of an application must be re-deployed if a business rule or the user interface changes. If the number of workstations is high or the workstations are geographically dispersed, the maintenance costs for two-tier, fat client applications can escalate quickly.

A second model for two-tier client/server applications has much of the code that implements the business rules tightly integrated with the data access code, sometimes in the form of database stored procedures and triggers. This model is called *two-tier, fat server*.

Two-tier, fat server applications are often implemented as mainframe applications with Web browsers as user interfaces. This approach may be a useful first step to migrate to a three-tier or n-tier service-oriented application architecture. Users can enjoy the speed and ease-of-use provided by the web's graphical interface while developers update other parts of the application.

Since the business rules in two-tier applications are tightly integrated with the user interface code or data access code, two-tier client/server applications have the following drawbacks:

- *Two-tier client/server applications are difficult and expensive to modify when business requirements change.*
  The business rules tend to be monolithic. Changing a business rule may impact other business rules and the rest of the application.
- *There is little reuse of redundant code in two-tier client/server applications.*
  It is difficult to reuse business rules elsewhere (e.g., in other computer applications that require similar services or in batch processing that is part of the same application) when they are tightly coupled to each other and to the user interface (fat-client) or the data (fat-server).
- *There is little flexibility in selecting the platforms where the two-tier client/server applications will be deployed.*
  In two-tier, fat client applications, the business rules must execute on the same platform as the user interface because the code they are implemented in is tightly coupled with the interface. Likewise, in two-tier, fat server applications, the business rules can only execute on the machine that hosts the database because they are implemented either with or inside the database.
- *Users only can access two-tier client/server applications with PCs running a graphical user interface.*
  Since the user interface is graphical and requires a workstation, users with other I/O devices are excluded from using the application. These devices include existing non-graphics terminals (e.g., UNIX terminals or mainframe terminals), telephone interfaces via IVRs, and new user interface devices still evolving (e.g., PDAs and other mobile communications devices).
- *Two-tier client/server applications can be more difficult to manage than monolithic applications.*
  Changes to either business rules or the GUI often mean that the entire workstation-resident portion of the application must be redistributed and reinstalled on every workstation that uses the application. Such software distributions can be time-consuming, costly and logistically difficult to manage.

3.Three-tier client/server applications

Three-tier client/server applications are partitioned into three executable tiers of code: the user interface, the business rules, and the data access software. This does not mean that the three tiers execute on three different platforms. Although it is also possible to deploy the business rules on the same platform as the user interface in a three-tier architecture, it is not recommended because of the software management problems associated with using many or dispersed user workstations.

Three-tier client/server applications offer the following advantages:

- Three-tier client/server applications can be easier to modify to support changes in business rules.
- With three-tier client/server applications, there is less risk in modifying the code that implements any given business rule.
- Three-tier client/server applications can be made to support multiple user interfaces: character, graphical, web browser, telephones, and others.

4.N-tier Service-Oriented Architecture

Many problems inherent in the Commonwealth's existing monolithic and two-tier applications can be overcome by implementing applications with a three-tier architecture. However, large, complex projects that are anticipated to have high usage volumes and/or long life spans may be better served by an n-tier service-oriented architecture.

In the n-tier service-oriented architecture, applications are partitioned into discrete functional units called "services." Each service implements a small set of related business rules or function points. If a business rule must be modified to support changing business requirements, only the service that implements that business rule is impacted. The remainder of the application remains intact. The adaptability of applications is further enhanced by the use of an n-tier shared services architecture that segments rule processing into a series of services that can be accessed individually.

The maximum benefits of n-tier architecture are realized when many n-tier applications are deployed across the Commonwealth, sharing common software services and offering multiple user interfaces. In this environment, any application can access any service, provided the user has the proper security permissions. In n-tier service-oriented application architecture:

- Some services will be shared by applications from multiple agencies.
- Others services will be shared by applications within a single agency.
- A few, highly specialized services may be developed, at least initially, for a specific application.

Since the business rules are implemented as separate executables, any combination of business rules may run on any combination of platforms. This offers flexibility in selecting the platforms where the application components can be deployed, resulting in a high degree of scalability. As transaction loads, response times, or throughputs change, an individual service can be moved from the platform on which it executes to another, more powerful platform.

Since business rules are implemented discretely instead of being tightly integrated with the graphical user interface, changes to business rules typically do not require updates of code on the workstations accessing the application. This is very important in managing an application with many, geographically dispersed workstations.

Also, since business rules are implemented in discrete services, the same business rule can be invoked by users accessing the application from a GUI, from character terminals, from web browsers, by telephone from IVRs or by batch jobs. A separate interface tier provides programmer productivity and consistency of application behavior.
N-tier service-oriented applications offer the following key advantages:

- N-tier service-oriented applications are highly scaleable.
- An n-tier service-oriented architecture offers the best performance of any client/server.
- N-tier service-oriented applications offer the highest potential for code reuse and sharing.

The greatest strength of a service-oriented architecture is the opportunity it provides for the repeatable, rapid development of new applications.

<u>5.Web-enabled Applications</u>
There are two types of web-enabled applications. Some web-enabled applications provide information to clients in page format using HTML and XML to manage content dynamically. Other Web-enabled applications have fully interactive functionality and near real-time transaction processing capabilities.

Web-enabled applications are a special case of client-server applications where the “client” is a standard Web browser like Microsoft Internet Explorer or Firefox. The browser serves as another type of user interface (thin client) in the three-tier or n-tier application. Use of a standard Web browser as the client provides the user with a familiar, intuitive interface and significantly simplifies the process for developing and distributing the user interface.

Ideal web-enabled applications for the Commonwealth are n-tier service-oriented applications that use:

- An industry standard Web browser as the thin client;
- Intranets to provide secure access by Commonwealth employees;
- Extranets to provide restricted access by selected business partners; and
- The Internet and firewall technology to provide managed access by citizens and other interested parties.

Web-enabled applications will continue to grow in importance as a means to timely and cost effective delivery of information to the Commonwealth’s employees, business partners and citizens.

Web browsers are applications that accept text in the form of HTML/XML statements. The HTML/XML is interpreted and the file is presented on the desktop screen in web page format based on the corresponding HTML/XML. Web pages can contain hyperlinks to other documents, and multimedia such as text, images, audio and video.

The web started out as an environment for publishing static pages using HTML. Early on, the notion of enabling interactive, transaction oriented applications via the same browser became attractive since it could eliminate the need to install client software on every user's workstation. Browser technology supports the execution of programs written in scripting languages embedded in an HTML page. Browser technology also supports the execution of programs written in scripting languages including JavaScript, VBScript and others. Browsers may also support the running of Java Applets in the context of a Java Virtual Machine (may require a plug-in).

Recommended Practices:

The following ten recommended practices were identified:

APP-RP-37: **Analyze Business Processes –** Business processes will be analyzed, simplified or otherwise redesigned in preparation for and during new development, acquisition or major enhancement of information systems. The emphasis should be on process improvement, not just applying new technology to old processes.

Rationale:

- Work processes should be more streamlined efficient and cost effective.
- Work processes, activities, and associated business rules will be well understood and documented.
- Systems should no longer be developed using antiquated work processes.
- Systems should be most responsive to business needs.
- Enables E-Government initiatives.
- Potentially reduces the total cost of ownership.
- Provides better customer service.
- Facilitates the development of software using business rule based tools.

APP-RP-38: **Architected Before Designed –** Systems should be architected before they are designed. Systems architecture documentation should be verified for Commonwealth and agency Enterprise Architecture compliance.

APP-RP-39: **Object-oriented Design and Structure –** Applications should be based on object-oriented design and structure, in which objects encapsulate data structures and present a functional interface to application logic.

Rationale:

- Objects create a functional interface to data elements and permit developers to modify access methods and underlying data structures independent of the application.
- Object-oriented design supports re-use of objects across many applications and improves flexibility.

- Object-oriented design supports the reuse of application components in the development and maintenance of systems.
- It speeds development and modification and improves flexibility.
- Objects will allow for easier adaptation of business process changes.
- Objects utilize data encapsulation and permit developers to modify the underlying data structures and methods independent of the interface to the object.
- All industry leading application development tools are either object-oriented or object-based.

APP-RP-40: **Standards-compliant System Components –** Information systems should be designed and implemented using standards-compliant system components.

Rationale:

- Use of standards-based components supports incremental acquisition of systems.
- It assures that new and redesigned systems will be open. They will be upgradeable because old components can be replaced with new improved components that use the same standard interfaces.
- Standards-based components, using supported standards-compliant products, will keep product support costs manageable.

APP-RP-41: **Isolate Business Logic from Data –** Business logic should be isolated from the data by implementing a discrete data access layer. This logical boundary between business logic and data *should not be violated*.

Rationale:

- Isolated business logic can exist well beyond the lifetime of any system using it, substantially increasing ROI and encouraging component reuse.
- Agencies can maintain better control over data integrity by developing applications that allow only business rules to control access to data.
- Data cannot be managed consistently if multiple processes or users access it directly.
- A change in a database or application can potentially affect many large programs, if they are not highly partitioned.
- Partitioning isolates/minimizes change impact.
- Partitioned code is more adaptive to changes in internal logic, platforms, and structures.

APP-RP-42: **Document Application Design –** The design of all applications should be documented. This documentation includes: Object models, interaction diagrams and other design artifacts that record the structure, behavior and interfaces of application solutions.

Rationale:

- These are important deliverables of the development process that can benefit future efforts.
- The application design is an asset of the development process, facilitates extensibility and adaptability, and provides for future reuse.
- A documented design can be used as a training tool for new employees or consulting staff.

APP-RP-43: Design Platform Independent Components – Agencies should design applications, services and components that are platform independent if possible.

Rationale:

- Designers and operations support staff should make deployment decisions.
- Minimizing platform dependence builds in adaptability and scalability.
- Platforms change over time and platform dependent solutions will need to be reconfigured or redeployed.

APP-RP-44: Data Entered Once. Agencies should design application/solution so that data is entered once, and only once, as close to its source as possible.

Rationale:

- Data collection burdens for both the agencies and its customers will be reduced.
- The level of effort for managing data will be reduced.
- Duplicate and inconsistent database copies will be eliminated.
- Redundancies in collection, storage, processing, and dissemination of data will be eliminated.
- Costs will be reduced in the long term.

APP-RP-45: Facilitate Monitoring and Measurement – Applications and infrastructure components should be designed and implemented to facilitate monitoring and measurement.

Rationale:

- To assure an appropriate return on IT investments, agencies must be able to measure the performance of these investments.
- Consistent business management information will result in better investment management decisions, thus better returns on investment.

APP-RP-46: Take Advantage of Enterprise System Management Recommended Practices – Agencies should design applications, components and services so they take advantage of the Commonwealth's Enterprise System Management recommended practices.

Modeling

Modeling is the process of representing entities, data, business logic, and capabilities for aiding in software engineering. Use of object modeling principles and supporting tools are crucial to the successful implementation of large-scale object-oriented applications. Modeling tools that support industry standard UML are recommended. Ideally the tool integrates with the developer's IDE enabling roundtrip engineering between class diagrams and the code.

Recommended Practices:

The following two recommended practices were identified:

APP-RP-47: **Adopt Standard Work Process Modeling Tools –** Agencies should select and adopt standard work process modeling tools for use in business process re-engineering efforts.

APP-RP-48: **Utilize UML.** Agencies should use a standardized modeling tool that utilizes the Unified Modeling Language (UML).

Rationale:

- Aligns business requirements and application functionality.
- Modeling can identify opportunities for increased efficiency.

Business Rules

Business rules are abstractions of the policies and practices of a business organization. The business rules approach is a development methodology where rules are in a form that is used by, but not embedded in business process management systems. The Busiiness Rules Approach formalizes an enterprise's critical business rules in a language the manager and technologist understand. Business rules create an unambiguous statement of what a business does with information to decide a proposition. The formal specification becomes information for process and rules engines to run.

Business rules support agency business processes. The rules:

- automate the process,
- define what must be done, and
- define how it must be done.

Business events which are the triggers for business rules, define when it should be done.

As agency business processes change, the business rules in the applications that support the agencies also must change.

Recommended Practices:

The following four recommended practices were identified:

APP-RP-49: Assign Business Rule Responsibility to Business – Agencies should assign responsibility for defining and maintaining the integrity of business rules to the business units.

Rationale:

- IT staff is responsible for coding and administering the software that implements business rules.
- The business units are responsible for the definition and integrity of business rules, and for communicating changes in business rules to IT.
- Business subject matter experts (SME) should manage the process requirements
- Optimally every business rule should be assigned to a custodian or steward.

APP-RP-50: Business Rule Components Platform-Neutral – Business rule application components should be platform-neutral.

Rationale:

- Implement business rules in a non-proprietary, cross-platform language.
- This approach makes platform independence and portability possible.

APP-RP-51: Implement Business Rules as Discrete Components – Agencies should implement business rules as discrete components.

Rationale:

- Business rules need to be executed to ensure the correct policies are enacted governing the accuracy of related data and the execution of the actions to be performed.
- By implementing business rules as discrete components, the users can be assured of proper application of the rules.

APP-RP-52: Access Data Through Business Rules – Applications should access data through business rules.

Rationale:

- Designing applications so business rules control access to the data assures accuracy, consistency and reliability.
- Data is created and used by business processes. In computer applications, data must be created, used by, and managed by the application component that automates the business process.
- Accessing data in any way other than by business processes bypasses the rules of the module that controls the data. Data is not managed consistently if multiple processes or users access it.
- Federated data should be used wherever possible to assure data accuracy and simplify data management.

Reusable Components

A component is a loosely defined term for a software technology for encapsulating software functionality. Components must meet the following five criteria:

1. Multiple-use
2. Non-context-specific
3. Composable with other components
4. Encapsulated i.e., non-investigable through its interfaces
5. A unit of independent deployment and versioning

An artifact is a valuable, high quality software work product such as: documentation, analysis and design models, source code, interfaces, executable binaries, tools, processes, and test plans.

As part of a systems acquire/develop decision, agencies should first consider the reuse of existing applications and system components/artifacts. To be successful, agencies must be able to search for existing applications, components and artifacts that have already implemented specific business processes.

Recommended Practices:

The following four recommended practices were identified:

APP-RP-53: **Shareable Data and Process –** Systems should be designed, acquired, developed, or enhanced such that data and processes can be effectively shared, for appropriate purposes, across the Commonwealth and with our partners.

Rationale:

- Increased efficiency will better serve our customers (e.g., the public, employees, etc.).
- Redundant systems cause higher support costs.
- Ensures more accurate information.
- Shared data and processes lead to better decision-making and accountability.

APP-RP-54: **Reuse vs. Create –** Agency solutions should be built by assembling and integrating a collection of reusable, loosely coupled components and services where appropriate, rather than by creating or recreating common functionality.

Rationale:

- Provides development efficiency and deployment flexibility.
- Applications designed with reusable components can be developed rapidly and at lower cost.
- Reduces the risks associated with new applications because the quality of the reused components has already been validated.
- Tightly coupled components become problematic when they limit the capability of the application, such as in development, testing, and deployment of the application.

- Reusable components increase the productivity (reducing cost and time to market) of the application development departments within the enterprise.
- The use of proven components enhances the quality of solutions.
- The adoption by the Commonwealth of a service-oriented architecture (SOA) requires development teams to will be able to take advantage of component and service reuse.
- Components and services can exist within a business unit, an agency or across agency boundaries.

Definitions:

- Components are fine-grained encapsulated functionality to support common development efforts.
- Services are course-grained, process-centric, business functions utilized to carryout a specific task.

APP-RP-55: **Reuse or Share Cross-Functional Systems –** Agencies should implement cross-functional systems that take advantage of common software modules that may be shared and reused for similar business functions.

Rationale:

- Many agencies and business units share common work processes with similar information requirements (e.g. licensing, processing applications, making awards) and may be able to reuse applications, data, and related information technology across the Commonwealth.
- Common software modules may be reused for similar functions.
- Systems based on standard software modules can be implemented faster and with better quality than systems based on newly designed components.
- An enterprise-wide, cross-functional review will identify similar functions, thus eliminating duplicative design and development activities.
- Office work processes will need to be reengineered to support common functions and common software module usage; efficiencies may be expected as a result.

APP-RP-56: **API Provided for All Reusable Components –** A well-documented Application Programming Interface (API) should be provided for all reusable components.

Rationale:

- A documented API is how components/services and applications should communicate.
- API documentation should include parameter specifications such as: input, output, optional/required, lengths and type.
- Writing to standard API's protects applications from platform, network and database changes.

Requirement:

The following one requirement was identified:

APP-R-17: **Search for Existing Business Process –** The Commonwealth Enterprise Architecture shall evolve to incorporate a search feature that addresses the customer's need to locate existing Commonwealth/ agency (excluding higher education) solutions that implement specific business processes.

Rationale:

- In order for the agencies to be able to first consider the reuse of existing applications and system components before purchasing or developing solutions, they must be able to search for those of existing applications and system components.

Configuration Management

Configuration Management is applicable to all aspects of software development from design to delivery. It focuses on the control of all work products and artifacts generated during the development process. Version Management (a subset of Configuration Management) refers to the tracking and controlling of file versions. It includes capabilities such as labeling, branching, merging, version content comparisons, and security and permission management. An initial step on the path to Configuration and Version Management is to implement a source code repository with supporting processes.

Code management is crucial to maintain application integrity through the development and maintenance lifecycle. Ideally, code management tools would integrate with defect tracking and application build tools. The Commonwealth will be researching code management systems that can scale across the enterprise to foster an environment that supports reuse of shared components.

Requirement:

The following requirement was identified:

APP-R-18: **Source Code Repository –** All application source code shall be maintained in a repository using a formal process.

Test Management

Test Management is the consolidation of all testing activities and results. Test Management activities include test planning, designing (test cases), execution, reporting, code coverage, and heuristic and harness development.

Implementing a comprehensive testing strategy can increase the effectiveness of testing. The following are five sample testing strategy objectives with recommended approaches:

1. <u>Increase testing efficiency</u>

a. *Establish testing priorities for each project*
Priority column should be added to the User Requirement Traceability Matrix.

b. *Focus Testing where defects are most likely to exist*
Use Configuration Management in combination with the Help desk to focus testing on code modules that have the greatest number of changes, complexity, and history of reported defects.

c. *Ensure adequate testing coverage*
Use the Requirements Traceability Matrix to drive the generation of a Project Test Plan and SLDC Phase specific "sub" Test Plans.

d. *Coordinate testing between SLDC Phases.*
Implement Project Test Coordinator Role.

2. Create Testing Environment that supports greater efficiency:
   - Research current Testing Environments
   - Create Testing Environment Functionality Requirements

3. Migrate from manual to automated testing methods where cost effective:
   - Evaluate needs
   - Evaluate potential tools
   - Explore the possibility of providing shared knowledge/resources
   - Initially focus on automating Regression Testing

4. Migrate from Dynamic to Static Testing Methods
   It is more cost effective to correct defects in earlier SLDC phases.

See *Appendix B: Software Testing Types and Techniques* for more information on testing methodology.

Information on testing tools and approaches to aid in meeting Commonwealth Accessibility Standards can be found within the Web Accessibility and Template Guide (WATG) located at http://www.vadsa.org/watg.

Recommended Practice:
The following three recommended practices were identified:

APP-RP-57: **Comprehensive Testing Strategy –** Agencies should create and implement a comprehensive testing strategy that covers the entire development lifecycle. Where appropriate, the testing strategy should address:
- Functional Testing
- Unit Testing
  - Integration
- Business Cycle Testing
- Usability Testing

- Regression Testing (preferably automated)
- Load/Stress/Volume Testing
- Security and Access Control Testing
- Reliability Testing
- Configuration Testing
- Installation Testing
- User Acceptance Testing

APP-RP-58: **Automated Regression Testing Solutions –** New medium, large or complex agency applications (either developed or purchased COTS solutions) should include automated regression testing solutions where cost-effective.

Rationale:

- Agencies often have sufficient funds to purchase (COTS) or develop an application, but not enough to thoroughly test the application when updates to the solution or underlying application platform (operating system, database, application server) are required.

APP-RP-59: **Design to Test –** Application components and services should be designed so they can be tested and debugged completely with ease.

Rationale:

- Testing is a critical step in the development process.
- Application components with consistent interfaces are easier to test on an application-wide basis.
- Error handling, tracing, and check-pointing should be included.
- These functions should be implemented in the earliest phases of development.
- Testing performance, fault-tolerance, and security are also important elements of a test plan.

Schedule and Task Management

Schedule management is a critical component of project planning and control. Schedules are part of project baselines, and critical milestone completions often are important project events that are reported to stakeholders. Schedule development and control starts with the technical scope, assumptions, activity definition, logic sequencing, and duration estimation. Expert judgment based on similar projects is best applied during these early steps to determine reasonable schedules and to limit future schedule risk. Techniques which can improve project schedule performance are high performing software, network diagrams, critical path determination and analysis, mathematical analysis/ simulation, resource leveling, conditional scheduling for high risk areas, and duration compression.

Task Management defines the set of capabilities that support a specific undertaking or function assigned to an employee. Task Management tools provide automation features for managing, delivering, assigning, reminding, prioritizing, and collaborating task management and execution.

Recommended Practice:
The following two recommended practices were identified:

APP-RP-60: **Schedule Management Tool –** Agencies needing a schedule management tool should consider acquiring Microsoft Project/Microsoft Project Professional.

Rationale:

- Microsoft Project and Microsoft Project Professional are de facto standards within the Commonwealth.

APP-RP-61: **Task Management Tool –** Agencies needing a task management tool should consider acquiring Microsoft Project Server.

Rationale:

- Microsoft Project Server is a de facto standard within the Commonwealth.

Geospatial Technologies

Geospatial Technologies integrate acquisition, storing, editing, displaying, modeling, analysis, and management of spatially referenced data, i.e. data identified according to their locations.

The Geospatial Technologies topic includes the following components which: will be addressed in future versions of this report:

Mapping/Geospatial/Elevation/GPS (GIS

- Integrated Development Environment (IDE)
- Geospatial Data Development Standards
- Database/Geospatial Metadata
- Utilities
- Reporting and Analysis
- Global Positioning Systems (GPS)

Visualization:

- Graphing/Charting

Enterprise Applications

Enterprise Applications are software solutions that perform business and cross-cutting functions (such as accounting, facilities management or procurement).

The Enterprise Applications topic includes the following components:

- eCommerce
- The following will be addressed in future versions of this report
  - Human Resources
  - Financial Management
  - Assets/Materials Management
  - Human Capital/Workforce Management
  - Customer Relationship Management
  - Customer Initiated Assistance
  - Supply Chain Management
  - Document Management
  - Authentication/Single Sign-on (SSO)
  - Search Engines
  - Audio and Video Conferencing

eCommerce

Electronic commerce, EC, e-commerce or eCommerce consists primarily of the distributing, buying, selling, marketing, and servicing of products or services over electronic systems such as the Internet and other computer networks. The information technology industry might see it as an electronic business application aimed at commercial transactions. It can involve electronic funds transfer, supply chain management, e-marketing, online marketing, online transaction processing, electronic data interchange, automated inventory management systems, and automated data-collection systems. It typically uses electronic communications technology such as the Internet, extranets, e-mail, Ebooks, databases, and mobile phones.
Wikipedia, The Free Encyclopedia. 28 Apr 2006, 02:49 UTC. 1 May 2006, 19:03 <http://en.wikipedia.org/w/index.php?title=Electronic_commerce&oldid=50523647

eCommerce can be broken into four main categories:

- *B2C (Business-to-Business:* Companies doing business with each other such as manufacturers selling to distributors and wholesalers selling to retailers.
- *B2C (Business-to-Consumer):* Businesses selling to the general public typically through catalogs utilizing shopping cart software.
- *C2B (Consumer-to-Business):* A consumer posts his project with a set budget online and within hours companies review the consumer's requirements and bid on the project. The consumer reviews the bids and selects the company that will complete the project.

- *C2C (Consumer-to-Consumer):* There are many sites offering free classifieds, auctions, and forums where individuals can buy and sell thanks to online payment systems like PayPal where people can send and receive money online with ease. eBay's auction service is a great example of where person-to-person transactions take place everyday since 1995.

G2G (Government-to-Government), G2E (Government-to-Employee), G2B (Government-to-Business), B2G (Business-to-Government), G2C (Government-to-Citizen), C2G (Citizen-to-Government) are other forms of eCommerce that involve transactions with the government--from procurement to filing taxes to business registrations to renewing licenses.

Virginia.gov is part of the Virginia Information Technologies Agency (VITA) and assists other Virginia government entities in providing information services via the Internet. Virginia.gov manages the official Virginia portal at www.virginia.gov or http://www.state.va.us. See *Appendix C: Features and Benefits of the Virginia.gov Payment Portal* for additional information on Virginia.gov.

Recommended Practices:

The following twenty-four recommended practices were identified:

APP-RP-62: **Evaluate Virginia.gov Payment Portal –** The Virginia.gov Payment Portal should be evaluated as part of an Alternatives Analysis for all eCommerce systems developed or purchased by the Commonwealth/Agencies (not including higher education). The Alternatives Analysis should provide a compelling financial or business case justifying the selection of any other solution.

Rationale:

- The Commonwealth's goal is to create a single window on government with a common look and feel and consistency across all levels of government. Utilizing a single enterprise solution helps achieve this objective.
- The Virginia.gov payment portal is an enterprise solution offered through VITA, which can be used by any government entity in the Commonwealth (state, city or county).
- There are over 200 instances of the cost effective, secure Virginia.gov Payment Portal in use today in the Commonwealth" (all instances are hosted by Virginia.gov)

APP-RP-63: **Cardholder Information Security Program and Data Security Standards –** All agencies all entities deploying online payment applications should follow Cardholder Information Security Program (CISP) and Data Security Standards (DSS).

APP-RP-64: **Online Credit Card Payments –** Fulfillment type services (e.g. – online stores) should use a delayed capture approach for online credit card payments, whereby the transaction is first authorized, the funds are “held” and upon fulfillment, the funds are “captured.”

APP-RP-65: **Confirm Payment Amount –** Payment applications should include a step for the user to confirm the payment amount, including any fees. In addition, a printable receipt screen should be built into the process.

APP-RP-66: **Collect Billing Name, Address and Zip Code –** Online payment screens should collect billing name, address and zip code for increasing verification and ensuring lower merchant account discount rates.

APP-RP-67: **Delayed Capture Approach –** If using a delayed capture approach for credit card payments, online automation of the delayed capture amount should be equal to or less than the original purchase.

APP-RP-68: **Refunds –** Applications that handle refunds should limit the refunded amount to be equal to or less than the original purchase.

APP-RP-69: **Log All Activity –** Payment applications should log all activity and electronic gateway responses, even for rejected or declined transactions.

APP-RP-70: **Limit the Likelihood of Duplicate Payments –** Payment applications should develop controls to limit the likelihood of duplicate payments.

APP-RP-71: **Secure Coding Guidelines –** Payment applications should be developed based upon secure coding guidelines. (See www.owasp.org)

APP-RP-72: **Cookies –** Sensitive payment information should not be stored in cookies.

APP-RP-73: **Server Side Controls –** Server side controls should be implemented to prevent SQL injection and other bypassing of client side-input controls.

APP-RP-74: **Secure Network and Security Standards –** Payment applications should reside on a secure network, including a firewall. All routers, switches and firewall configurations should be secured and conform to security standards.

APP-RP-75: **Conduct Network Scans –** Entities offering online payment services should conduct network scans semi-annually to identify potential vulnerable points.

APP-RP-76: **Up-to-date Security Patches –** Entities offering online payments should ensure all servers involved with payment processing have the most up-to-date security patches, upgrades and anti-virus software.

APP-RP-77: **Transport Customer's Payment Securely.** Payment applications should transport customers' payment data securely and reliably. Secure Sockets Layer (SSL) using 128 bit encryption is the industry standard for transmission encryption and allows information to be sent securely and reliably over the internet. Pretty Good Privacy (PGP) File Transfer Protocol (FTP) is the recommendation for Automated Clearing House (ACH) file transfers.

APP-RP-78: **Unique Username and Complex Password –** Access to PCs, servers or databases with payment applications should require a unique username and complex password.

APP-RP-79: **Credit Card Numbers –** Full credit card numbers should not be stored in any form on any server and should be masked on any non-input screens during the transaction process (receipt screens and confirmation emails). Only the first two and the last four digits could be retained to assist with transaction tracking and customer service.

APP-RP-80: **Card Verification Code –** If using card verification code (CVC) numbers for online payment processing, they should not be stored in any form on any server and should be masked on any non-input screens during the transaction process.

APP-RP-81: **Merchant Account –** Merchant account and electronic gateway information, including logins and passwords, should be password protected and only available to limited staff with "need to know" responsibilities.

APP-RP-82: **Merchant Unique Password –** Entities offering online payment services should create unique passwords for merchant and electronic gateway online tools and change these passwords regularly.

APP-RP-83: **Revoke Terminated Employee Access –** Should an employee, who had access to payment applications or information, be terminated or quit, the employee's user accounts and passwords should be revoked as soon as practical. Merchant account and electronic gateway passwords should be changed.

APP-RP-84: **Employee Background Checks –** Background checks and investigations are strongly recommended for any employee with access to payment applications or account information.

APP-RP-85: **Encrypt ACH Transactions –** ACH transactions should be stored in an encrypted file that can only be decrypted by the bank. These files should be regularly purged from the servers.

(This Page Intentionally Left Blank)

Glossary

Following are Glossary entries pertaining to the Application Domain and required to support this document. Additional glossary definitions can be found in the ITRM Technology Management Glossary located on the VITA website here: http://www.vita.virginia.gov/projects/cpm/glossary.cfm.

Some useful public glossaries can also be found at:
Wikipedia, the free encyclopedia at http://en.wikipedia.org/wiki/Main_Page
Loosely Coupled Glossary at http://looselycoupled.com/glossary/azindex.html

Another excellent glossary can be found at: http://www.matisse.net/files/glossary.html

Agency	Any agency, institution, board, bureau, commission, council, or instrumentality of state government in the executive branch listed in the appropriation act. ETA requirements/standards identified in this report are applicable to all agencies including the administrative functions (does not include instructional or research functions) of institutions of higher education, unless exempted by language contained in a specific requirement/standard.
Business Reference Model (BRM)	In a service-oriented architecture, all business services are defined in the business reference model (BRM). The BRM is part of the Enterprise Repository. One of the key principles behind SOA is to break down business services into reusable components that can be combined and shared across the enterprise. These shared components are called web services and they are defined in the service component reference model (SRM) which is also located in the Enterprise Repository. Both the BRM and SRM are hierarchical. The exact structure of the model will be determined at design time.
Component	A readily accessible and observable aspect of a technology topic, such as Test Management is a component of the Software Engineering topic. A component is not the individual pieces such as tables, SQL scripts, etc. and other many similar pieces which make up the component.
Commercial off-the-shelf (COTS)	Commercial off-the-shelf (COTS) is a term for software or hardware products that are ready-made and available for sale to the general public. They are often used as alternatives to in-house developments or one-off government-funded developments (GOTS). The use of COTS is being mandated across many government and business programs, as they may offer significant savings in procurement and maintenance. Commercial off-the-shelf. *Wikipedia, The Free Encyclopedia*. Retrieved 18:10, January 11, 2006 from http://en.wikipedia.org

Domain	The Enterprise Technical Architecture (ETA) is typically divided into logical groups of related technologies and components, referred to as "domains". The purpose of a Domain Architecture is to provide a combination of domain principles, best practices, reusable methods, products, and configurations that represent "reusable building blocks". Thus, the Domain Architecture provides the technical components within the Enterprise Architecture that enable the business strategies and functions. Note, the Conceptual Architecture serves as the foundation for the Domain Architectures, and ensures that they are aligned and compatible with one another.[8]
Enterprise	As used in this document and generally when discussing Enterprise Architecture topics, the *enterprise* consist of all Commonwealth of Virginia agencies as defined above.
ETA	The Enterprise Architecture has business and technical components. All of the technical components taken together are called the Enterprise Technical Architecture.
ORCA	Online Review and Comment Application is a web based application managed by VITA to allow public comment and review of proposed policies, standards, and guidelines. ORCA may be accessed through the Commonwealth Project Management Web page or by pointing your Web browser to the URL http://apps.vita.virginia.gov/publicORCA.
Principles	High-level fundamental truths, ideas or concepts that frame and contribute to the understanding of the Enterprise Architecture. They are derived from best practices that have been assessed for appropriateness to the Commonwealth Enterprise Architecture.[9]
Product Standards	Are specifications for the use of specific hardware and software relative to the particular component.
Recommended Practices	Are activities which are normally considered leading edge or exceptional models for others to follow. They have been proven to be successful and sustainable and can be readily adopted by agencies. They may or may not be considered the ultimate "best practice" by all readers but for this place and time they are recommended practices and should be used and implemented wherever possible.
Requirements	Are activities that are considered strategic components of the Commonwealth's Enterprise Technical Architecture. They are acceptable activities for current deployments and must be implemented and used for all future deployments.

[8] COTS Enterprise Architecture Workgroup, "*Commonwealth of Virginia Enterprise Architecture – Common Requirements Vision*", v1.1, December 5, 2000, p 26.

[9] COTS Enterprise Architecture Workgroup, "*Commonwealth of Virginia Enterprise Architecture – Conceptual Architecture*", v1.0, February 15, 2001, p 5.

Service-Component Reference Model (SRM)	Service component-based framework that can provide—independent of business function—a “leverage-able” foundation for reuse of applications, application capabilities, components, and business services.
Topic	A topic is simply a logical subdivision of the domain. All components relevant to the Commonwealth’s Technical Architecture are included within one of the identified topics. Within the Application domain topics include Enterprise Systems Design, Application Acquisition, Development and Support Platforms, Software Engineering, Geospatial Technologies, and Enterprise Applications.

(This Page Intentionally Left Blank)

Appendix A: Example SOA Centralized Implementation and Governance Model

The following is an example of a centralized SOA Implementation and Governance model. It is largely based on the draft California SOA model referenced in the footnotes and we thank our California colleagues for sharing their work.

1) **Decentralized Service Development**
   Service components would be built and tested by individual departments (for example DMV, DSS etc). Each service would be submitted for certification to the VITA Architecture Review (VAR) Group. Upon approval, the service would need to go through the ICC data center deployment process

2) **Centralized Governance Management and Support Model**
   a) *Centralized Budget* – Effective SOA Management would require an investment in staff, hardware, software and tools. Therefore, an appropriate budget or fee based structure would need to be granted to support Governance, and maintenance of an Enterprise Repository. Initially, some outside staff consultants would likely be required to manage certification and performance labs, as well as serve as enterprise technical troubleshooters.

   b) *Service Contract Policies* - Service performance contracts would be published by a producing department for a given web service. Consuming organizations would build their applications around these contracts. The contract process itself would be established by the Governance portion of the SOA. Actual service performance would be monitored by the individual data centers. For enterprise applications, performance may be hosted and monitored by VITA's ICC infrastructure. In that scenario, VITA's ICC and the publisher would collaborate with the producing development organization to fix any problems. Consuming organizations would have a key part in revising performance contracts.

   c) *Service Certification* - Certifying web services would be a key function. All "public use" and "reusable" revised services would go through the ICC certification process to ensure that they would play nicely in the enterprise architecture. The testing would be done in the ICC Lab maintained by VITA.

   d) *SOA Policies and SOA Security* - Based on SOA Leadership input, the SOA VITA Integration Competency Center would establish and enforce SOA Policies and SOA Security Policies.

   e) *Application Services Granularity* – This is a key component in determining how manageable the SOA environment would be, as well as the degree of service reuse. If the service interfaces are too complex or if there are simply too many services, then manageability would become a real problem. Services would need to be easily composed into higher level services to achieve maximum reuse. So, careful thought and ongoing diligence would be required.

3) **Centralized Enterprise Repository**
Reference models, portfolio and application information would be managed in a centrally maintained Integration Competency Center. This includes ensuring that the repository is always available and it is regularly backed up. A guide on how to search and use the repository would also be published. The repository would have a hierarchical structure for the BRM and SRM (see glossary of terms). That is, service components would be related to their business services. The owner, description, version and interface definition of each service would be clearly stated as well as any dependencies. Extensive search capabilities would be provided to ensure easy access by developers as well as business architects.

4) **Centralized Integration Competency Center**
The ICC and the enterprise architecture group typically establish the SOA vision and the SOA reference architecture. The ICC is built and empowered within an organization as the approach to structured application integration matures. [10]

   a) Research a process for creating a state-wide repository of public and reusable services would be maintained by the VITA SOA Integration Competency Center for the purpose of locating web services. (may ultimately migrate toward UDDI)

   b) *Manage Service Reference Model* – The service reference model would have an appropriate hierarchical structure that maps to the business reference model and application and project portfolios. Additionally, service granularity would be properly managed. Provide high quality search and cross reference services.

   c) *Master Developer's Guide* - A master developer's guide would be created that states the general guidelines for developing services.

   d) *Certification Lab* – Development organizations would package their unit tested services and submit them to the ICC for certification. A certification lab would be set up and published to all development organizations so they know how their services would be tested. It would take significant collaboration to initially set up this lab as well as ongoing fine-tuning. The goal should be to ensure that services play nicely together in a distributed environment, meet their stated requirements, have stable and well defined interfaces, and meet all stated security requirements.

   e) *Performance Lab* – In cases where a service has a stated performance contract, then they need to be tested in the performance lab. For composite services, which depend on one or more base services, the lab would be able to install the

[10] *The ICC and SOA Governance: Managing a Successful Integration Project*. Paolo Malinverno, Gartner Research

composite service and test it on the live network consuming the base services in the data centers.

f) *Developer Discussion Group* – Integration Competency Center would facilitate a discussion group for service developers.

g) *Operations Discussion Group* – Integration Competency Center would facilitate a discussion group for service operators. This would likely take the form of an SOA Center of Excellence.

h) *Developer Workshops* – Integration Competency Center would design appropriate workshops for developers. At minimum, there should be a workshop for developing a base service, and one for developing a composite service. Workshops would implement best practices as defined by SOA Leadership and Governance.

i) *Compliance Reporting* – Some shared services would have contracts specifying availability, scalability, and recoverability metrics. Integration Competency Center would be responsible for reporting out of compliance services.

j) *Installation and Administration* – SOA data centers would be responsible for installing, configuring, deploying, and registering their services. They would also ensure that proper logging is turned on and review the operational logs on a regular basis. Appropriate data and files would be backed up on a regular schedule that fits the particular service. Additionally, the services would be installed and configured to meet performance, availability, and scalability requirements. A recommended best practice is to consolidate data centers.

k) *Service Inventory* – VITA's ICC would be responsible for updating the Enterprise Repository with information about all services. At minimum, this would include the service owner, version number, and who is using the service. This last data element might be dynamically updated by service monitoring tools.

l) *Incident Management* – Ultimately, for centralized services and when the new enterprise help desk is deployed, when a service problem arises, the VITA Integration Competency Center would be expected to resolve the problem in an efficient manner by coordinating the response with the agency that created the web service. There would be a single owner for each ticket placed to VITA's Customer Care Center which should increase customer satisfaction. This probably falls under the ITSM (Information Technology Service Management) umbrella.

m) *Configuration Management* – When enhancement or bug fixes are applied by a development organization, the resulting service would be versioned and resubmitted to the certification group. Upon successfully certification, the newly version component would be put into production via Enterprise Operation's configuration management policy (again, probably ITSM based).

n) *Release Management* – When we receive a request from an owner, initial services, as well as major changes to existing services, would be provided in a release package and submitted for certification. Upon approval, the release package would be deployed into production by Integration Competency Center following proper release policies.

o) *SOA Security Management* – Some services would not have inherit security requirements while others may have very stringent requirements. Services that participate in Identity, Access, and Privacy implementations would follow specific enterprise security policies. This is especially true for those services that are part of a "circle of trust". While the Governance group would determine the policies, they would be enforced by Integration Competency Center.

p) *Service Contracts* – The Integration Competency Center would be responsible for ensuring availability, scalability, and recoverability requirements are met as defined in a services contract. They would evaluate data gathered by monitoring tools to determine whether they are in or out of compliance. They would provide compliance reports to as well as be responsible for getting a service back into compliance. They would use an escalation process if they need additional help.

q) *Operational Guides* – ICC Enterprise Architects would provide operational guides detailing startup, shutdown, and service recovery procedures. They would contain configuration and deployment packaging information. A section on common error messages and typical troubleshooting procedures would also be helpful.

r) *Gather Operational Data* – A common set of tools would be specified by Enterprise Architecture Governance and Integration Competency Center. It is expected that the Integration Competency Center would be proactive in evaluating the data generated by the tools and take appropriate action when potential problems are indicated.[11]

[11] *Service-oriented Architecture (Draft).* California Enterprise Architecture Program, December 8, 2005

Appendix B: Software Testing Types and Techniques

There are two types of software testing:

1. ***Dynamic Testing***: testing the software program when it is actually running.
   Examples: Functional Testing, Load Testing, and Regression Testing.
   Dynamic Testing for a Software Project can be performed in five SLDC phases:

SDLC Phase	Dynamic Testing Type
Development	Bug Verification Testing Build Acceptance Testing Compliance/Conformance Testing Conversion Testing Interface Testing Unit Testing
System Integration Test	Bug Verification Testing Claim Testing Compliance/Conformance Testing End-to-End Testing Functional Testing Installability Testing Installation Testing Integration Testing Interface Testing Load Testing Negative Testing Operations Acceptance Testing Performance Testing Platform Configuration Testing Recovery Testing (Automated) Regression Testing Reliability Testing Risk-based Testing Security Testing Stress Testing System Testing Technical Testing
User Acceptance Testing	Bug Verification Testing Claim Testing End-to-End Testing Regression Testing Risk-based Testing Specification-based Testing User Acceptance Testing
Implementation	Parallel Testing Pilot Testing

2. ***Static Testing:*** testing performed while program is not running. Examples: Requirement Validation, Code Inspections, Design Review, General Reviews, and Audits. (Note: Static Testing may be implemented using computer-automated tools.)

   Static Software Project Testing can be performed in all SDLC phases.

SDLC Phase	Static Testing Type	Performed by
Planning		
User Requirements	Requirement Validation	Business User Development Testing
Design		
Development	Code Inspections	
System Integration Test		
User Acceptance Testing		
Implementation		

Dynamic Software Testing Types

Dynamic Testing can be performed using two different methods: black-box and white-box.

Black-box test design treats the system as a "black-box", so it doesn't explicitly use knowledge of the internal structure. Black-box test design is usually described as focusing on testing functional requirements. Synonyms for black-box include: behavioral, functional, opaque-box, and closed-box.

White-box test design allows one to peek inside the "box", and it focuses specifically on using internal knowledge of the software to guide the selection of test data. The task is to test the code or its logic with little or no regard to the specifications. All internal components need to be adequately exercised. The ultimate goal is to test every execution path through the program logic. Synonyms for white-box include: structural, glass-box and clear-box.

The following Dynamic Testing types appear in alphabetic order.

Bug Verification Testing

Bug fix verification is one of the most important and common types of regression testing that is performed. Many fixes fail, for a variety of reasons. Perhaps the developer didn't understand your bug report and fixed something else instead: perhaps the developer didn't test his fix before he checked in the code; or perhaps the fix did not make it into the build. It's also common to find new bugs while verifying bug fixes. Sometimes the fix breaks nearby functionality, or sometimes the new bug was not observable until the previous error was corrected. Sometimes bugs are fixed in such a way that you would still consider the behavior a problem (although usually not as serious a problem as the original bug

Build Acceptance Testing

A build acceptance test (sometimes also called build verification test, smoke test, quick check, or the like) is a set of tests run on each new build of a product to verify that the build is testable before the build is released into the hands of the test team. The build acceptance test is generally a short set of tests, which exercises the mainstream functionality of the application. Any build that fails the build acceptance test is rejected.

Claim Testing

Packaging, help files, and user's guide along with the software itself, are going to make some claims about what the software offers its users. Some of these claims are explicit: it will run on this platform; it will run with this much RAM; it will run on these operating systems; it will perform these tasks; it will work. Others are implicit: if there is a print function, it will print; if there is a save function, it will save; etc… The claim testing technique will identify these claims, both implicit and explicit, and validate that they are in fact accurate.

Compliance/Conformance Testing

Testing to verify that the software meets standards. For example: testing to meet browser standards could include testing the web pages for compliance to HTML/CSS/XML/XHTML layout and rendering, DOM, parsers, and JavaScript standards.

Conversion Testing

Testing to ensure that all data elements and historical data is converted from an old system format to the new system format. Most conversions of data from old applications to new ones require an automated conversion. With an automated conversion, the software that has been written to convert the data must also be tested.

End-to-End Testing

End-to-End Testing involves a coordinated effort between IT and the Users to ensure that data put into one application will move correctly to other applications. Similar to Functional Testing, firms define a suite of test scenarios, develop test cases that will validate the scenarios, and create test data to validate that the application works properly.

Functional Testing

The goal of Functional Testing is to ensure that the applications can perform correctly under all of the conditions that the application could encounter when in live processing. Define a suite of test scenarios, develop test cases that will validate the scenarios, and create test data to support the test cases. Testing using this approach means testing the functionality of the application.

*Functionality Testing **might be***:

- Installability Testing
- Reliability Testing
- Sequence (Scenario) Testing
- Specification-based Testing

*Functionality Testing **is***:

- **Impacted by Testability:** In order to test the functionality of a product, it must be testable – it must be code complete, it must install, and all branches must be available. Any area that isn't complete or cannot be examined, obviously cannot be tested for functionality.
- **Concerned with Features:** Functionality is concerned with what works and how it works – not necessarily why it works. This includes menu items and UI options.

Functional testing should cover the following aspects of the module's functionality:

- Exercise all interfaces between units within a module whether new or modified.
- Exercise each new or modified function or user input command and each command option.
- Interface as expected with each database, database table, utility, external software package or external file, mainframe, server, hardware device or other entities external to the module.
- Verify the correct generation of all error or warning or other user or log messages.

Installability Testing

Can a product be installed on a clean system, can it be installed over a previous version of itself, and can it be completely removed from a system?

Installation Testing

The package will be deployed in an environment similar to the customer environment. This will allow the company to detect any unexpected behavior of the system or application being developed. The simulation of the exact working environment of the customer is essential in order to know what bugs exist before actually sending the application to the customer and start receiving bug reports from him.

Integration Testing

The process of progressively aggregating individual system components to demonstrate proper interworking. Integration testing is aimed at exposing problems that arise when two or more components are combined. Typical problems identified in integration testing are improper call or return sequences, inconsistent data validation criteria and inconsistent handling of data objects. Three approaches are usually taken toward implementing Integration Testing:

1. **Bottom-up**: In order to test using the bottom-up approach, each component is tested individually - to do so, test drivers should be provided to simulate other components as if they exist. If the test goes fine, then the other components are linked to the ready components when they are fully implemented, everything is then tested as a whole, i.e. as one complete unit. Using this approach will require the implementation of test drivers to simulate the calls from other components. Using this approach makes it hard for detecting the interface problems between components until all the components are linked and tested.
2. **Top-Down**: A top-down test is almost the opposite of the bottom-up approach. A main skeleton is provided, which is kind of a main program, this skeleton represents the normal major flow of the execution of the program, through which the components are called. Since components are not yet ready, they are replaced with small stubs, these stubs tell the main program that these components exist, i.e. they simulate the existence of these components, but they do not perform the complete functionality required from the real component. These stubs are replaced by their corresponding components when they are implemented.
3. **Big Bang**: Simply, test each module (component) individually, then link all components together and test the project as one whole application. This method although being an easy approach is not considered good. Bugs found when all modules are linked can't be fixed easily; in fact they can't be even detected easily.

Interface Testing
It is initially important to ensure that each program passes data correctly to other programs in the system, particularly if programs have been developed off-site. The checking of this data passing is sometimes referred to as "Interface Testing".

Load Testing
Subjecting a system to a statistically representative (usually) load. In load testing, load is varied from a minimum (zero) to the maximum level the system can sustain without running out of resources or having, transactions suffer (application-specific) excessive delay. (see Performance and Stress Testing)

Negative Testing
The tester determines how many ways he/she can cause the software product to get wrong answers or abort execution. Test cases would attempt to execute the software:

- with missing data files, missing data records in databases, or scrambled data.
- with misspelled links, or undefined, wrong, or missing configuration parameters.
- on platforms it was not intended to execute on.
- with missing communication lines, or bad incoming or outgoing data.
- powered-off peripherals, like printers, scanners, external CD or CD-RW drives, external hard drives, etc.

Operations Acceptance Testing
This phase of testing allows IT Operations staff to ensure that the developed system is capable of running in 'live' conditions. (Also called Job Stream Testing)

Parallel Testing
Parallel Testing is managed by the Users and is usually the last test before a system goes live. If the new system is replacing an existing one, both systems are operated side by side for a pre-determined period of time to ensure that the output from both systems is the same, or that any differences are expected.

Performance Testing
Involves inputting a large number of transactions into a computer application to see if all of the components (application, hardware, telecommunications, people, etc.) can accommodate the peak volume within acceptable time frame. (see Load and Stress Testing)

Pilot Testing
One or more sites are selected to perform a final component of User Acceptance Testing. Usually the Pilot sites reflect a representative set of user types, sophistication levels, software and hardware platforms. Pilot testing may also be considered part of the Implementation phase.

Platform Configuration Testing
Testing the software on representative customer software and hardware platforms. It is not possible to test software on all combinations of drivers, operating systems, software configurations on personal computers. The tester should apply test cases to the most prevalent user base combinations.

Recovery Testing
Tests performed to see if Application restart, back-up, and restore facilities operate as designed.

Regression Testing
The product has changed in one area and you want to be sure that ***it still passes all the tests it did before the change***. Testing to make sure the software hasn't taken a step backwards, or "regressed", is called "regression testing". If you run the different tests after each change, you have no way of knowing for sure that no new defects were introduced. Consequently, regression testing must run the same tests each time. Sometimes ***new tests are added as the product matures, but the old tests are kept too***.

- From the start of the software project, every new capability is accompanied by a short test battery
- Correct results are garnered for all the tests, and stored as files (text, data, screen images, etc.).
- Anytime a new capability is added, with its new test battery, all previous, validated tests are run, and the results compared with the standard results already stored on file.
- The same full regression test is run whenever the implementation is changed, even if no new capability is introduced.

Reliability Testing

This kind of testing is based on how well a product handles failures, data integrity, and safety and security.

Risk-based Testing
There are some places where defects are less tolerable-places where data, software, hardware, etc could be damaged or lost. These areas can be considered high-risk, and they are the areas the risk-based technique seeks to validate. This technique focuses time and energy in the areas that, if they contain failures could cost the organization money, cause embarrassment or compromise the quality of services offered.

Risk can be defined as a combination of the *likelihood* of a problem occurring, and the *impact* it would have on a user. Risk-based Testing analysis focuses on three things:

1. What areas the user is most likely to experience a problem?
2. What the impact of a certain type of problem would be?
3. What is the testing priority of each potential problem?

Security Testing
Testing done to ensure that the application systems control and auditability features of the application are functional. Includes: Penetration Testing, Denial of Service Testing, Covert Testing, and Vulnerability Assessment.

Specification-based Testing
A specification is anything that comes with the product – the box, the instructions, and any readme or help file. The functionality of a product can be tested against these specifications.

Stress Testing
Subjecting a system to an unreasonable load while denying it the resources (e.g., RAM, disc, mips, interrupts, etc.) needed to process that load. The idea is to stress a system to the breaking point in order to find bugs that will make that break potentially harmful. The system is not expected to process the overload without adequate resources, but to behave (e.g., fail) in a decent manner (e.g., not corrupting or losing data). Bugs and failure modes discovered under stress testing may or may not be repaired depending on the application, the failure mode, consequences, etc. The load (incoming transaction stream) in stress testing is often deliberately distorted so as to force the system into resource depletion. (see Load and Performance Testing)

System Testing
System Testing mainly differs from Unit Testing in that the software is viewed as a coherent whole for the first time, rather than as a series of individual programs. The goal is to ensure that all of the code works as defined by the requirements and design documents before it is delivered to the Users. In simple words, system test is a three-step test:

- Know and understand what the system does as a whole, i.e. as an application
- Know and understand what are the needs of the customer/user
- Check if the results of point 1 and 2 matches.

Technical Testing
The goal of Technical Testing is to ensure that files are correctly established, that reports work properly with page breaks as well as other formatting issues. The conditions defined by Technical Testing are generally consistent across all applications and are not unique to the specific application.

Unit Testing
Unit Testing is the first step in the testing process and is carried out by developers on individual programs, or parts of a program. The developer generally creates their own test cases, inputs the data to the program and verifies the results.

The developers usually concentrate their efforts on proving that their programs perform correctly at a technical level. Each program has its own technical specification, and the developer constructs a Unit Test script which will ensure that the program can deal with each requirement as specified.

The tester should read and use the source code of the applications being tested, through which, applying test cases and various inputs in order to test branches, conditions, loops and the logical sequence of statements being executed.

The tester should make sure that each unit produces the appropriate output for the input given it, including sensible error trapping.

Unit Testing may also include memory leakage testing (for specific languages) and run-time error checking.

User Acceptance Testing (UAT)
User Acceptance Testing is a very critical stage of the testing process since it is managed by the Users to determine if the application meets the terms of the requirements document.

This test consists of a series of predetermined test cases, with defined expected results, that will validate the functionality of the system and ensure that the Users can work with the system as it has been designed.

This testing phase differs from System Testing in the following ways:
- It exists to give confidence to business staff that the system is ready to be put 'live'
- It is planned and carried out by business staff, usually with support from IT staff
- It focuses purely on proving that Business Requirements have been met

Volume Testing
See Load, Performance, and Stress Testing.

Dynamic Testing Techniques

Boundary Value Analysis
A test design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, boundary value analysis selects elements at the "edges" of the class. It has been found empirically that bugs tend to cluster at the boundaries of the input domain rather than in the center.

An example would be in income limit for receiving a benefit is $1,000, then boundary value analysis testing would include test cases at $999, $1000, and $1001.

Boundary Value Analysis can be extended into including the affects of "rounding" of data on the program. Additional test cases of $999.49, $999.50 might be useful if the system "round" to the nearest dollar. The impact of "truncation" can be tested with test cases of $1000.00 and $1000.01.

Domain Testing
The idea that all possible values can be tested in all possible combinations for each of the input fields is not realistic-especially in a test cycle as short as the one we're working with for this project. Domain testing is the division of possible input ranges into a series of domains (i.e. 1-100, 101-1000, 1001-10,000, etc…for number input fields), and then testing these domains using a subset of data values.

An example would be: use 1, 25, 50, 75, and 100 for the 1-100 domain, and assume that the remaining 95 values will produce similar results.

Equivalence Partitioning
A test design technique for reducing the total number of tests required to validate a program's functionality. The basic idea is to divide the input domain of a program into classes of data. By designing tests for each class of data rather for each member of a class, the total number of tested needed is reduced.

As an example, zip code entry can be portioned into classes of valid and invalid inputs as follows:

- Valid inputs are all sets of five numeric characters that constitute an operational zip code.
- Invalid inputs include:
    - Sets of numeric characters with less than 5 characters
    - Sets of numeric characters with more than 5 characters
    - Sets of five numeric characters that do not constitute an operational zip code
    - Sets containing non-numeric characters

Setups and Cleanups (Testing from a known state)

One of the basic principles of testing is that the system under test should always be in a known state. If a bug is found but the tester does not know all the steps that led up top the failure, it may not be possible to reproduce the bug. Ideally, automated tools would be used to inventory the state of the system under test, log discrepancies, and allow the tester to change the system to a desired state.

There are two approaches to dealing with test initialization. One is to use setup routines to bring the system to a known state at the start of test, and use cleanup routines that “undo” the changes made during testing. Another approach is to do setups only, and forget cleanups. If for each test there is always a setup operation, thin it does not matter what happens at the end of a test. Generally it is a “best practice” to perform cleanups whenever a test is run that puts the systems into an undesirable state. For example, if a tests corrupts a database, puts the system into an error condition, fills a file to capacity, etc.

Sequence (Scenario) Testing
Testing an application through a series of little steps-steps that individually may not uncover any bugs-but steps that combined may generate any number of problems, and even failures. It is important to remember that testing a piece of software is much deeper than simply selecting menu options. One function-key may successfully take me ahead in the application, and another function-key may return me to my starting point. Doing this again is the step that might lead to a bug.

Static Testing Techniques

Audits
An independent examination of a work product or set of work products to assess compliance with specifications, standards, contractual agreements or other criteria.

Checklists
Connect to Review or technique section for Static Testing

Code Coverage Analysis
Code Coverage Analysis is the process of:
- finding areas of a program not exercised by a set of test cases,
- creating additional test cases to increase coverage, and
- determining a quantitative measure of code coverage, which is an indirect measure of quality.

An optional aspect of Code Coverage Analysis is:
- identifying redundant test cases that do not increase coverage.

A code “coverage analyzer” automates this process.

Coverage analysis is used to assure quality of test suites, not the quality of the actual product. The coverage analyzer is not generally used when running the test suite.

Coverage analysis requires access to test program source code and often requires recompiling it with a special command. Most beneficial for “rules-based” programs.

Code Inspections
Inspection’s “goal” is to improve the quality of the program by reviewing programmers’ work.

Manual or automated:

- **Programming Standards Verification.** Assesses whether the source code conforms to a set of user-configurable programming standards.
- **Structured Programming Verification.** Assesses whether the source code is properly structured.

Automated:

- **Full Variable Cross Reference.** Examines global and local variable usage within and across procedures.
- **Unreachable Code Reporting.** Searches for areas of redundant code.
- **Static Data Flow Analysis.** Follows variables through the source code and reports any anomalies in their use.
- **Information Flow Analysis.** Analyzes inter-dependencies of variables for all paths through the code.
- **Loop Analysis.** Reports the looping structure and depth of nesting within the code.
- **Procedure Interface Analysis.** The interface for each procedure is analyzed for defects and deficiencies.

Cyclomatic Complexity Analysis
A metric that measures logical complexity of a module. The value indicates the minimum number of independent paths (DD-path: decision-to-decision paths) that would need to be tested to ensure complete coverage of the program. If full coverage is required, there would be one test case for each path. (other examples: McCabe's metric and control flow knots)

Reading
A technique that is used individually in order to analyze a product or a set of products. Some concrete instructions are given to the reader on how to read or what to look for in a document. Reading is embedded in methods like inspections, audits or reviews. Therefore it is used as a technique for verification in the software development process.

Requirements Validation
Requirements are validated using the Requirement Traceability Matrix and interviews.

Reviews
A process or meeting during which a work product, or a set of work products is presented to project personnel, managers, users, customers, or other interested parties for comment or approval. Types include code reviews, design reviews, formal qualification reviews, requirements reviews, and test readiness reviews.

Version Comparators
Determines where changes have been made to source code or to data files between one version and the next of the program.

Walkthroughs
A static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a segment of documentation or code and the participants ask questions and make comments about possible errors, violation of development standards, and other problems.

The following table clarifies the differences between these terms:

Type	Scope	Purpose	Method
Reviews	Usually broad	Project progress, assessment of milestones	Ad hoc
Walkthroughs	Fairly narrow	Assess specific development products	Static analysis of products
Inspections	Narrow	Assess specific development products	Non-interactive fairly procedural
Audits	Range from narrow to broad	Check process and products of development	Formal, mechanical and procedural
Reading	Narrow	Analysis of products, prepare for reviews, inspections...	Not a method, a technique

Appendix C: Features and Benefits of the Virginia.gov Payment Portal

Virginia.gov receives calls weekly from government entities interested in using the Virginia.gov Payment Portal. Due to the scalability of Virginia.gov's solution, all Virginia government entities could have an online payment process that addresses their individual needs implemented quickly and easily.

Interoperability
Virginia.gov's Payment Portal consists of various modules. These modules are used by over 30 applications and services with diverse platforms and interfaces, including .NET, ASP, Perl, Oracle, XML, etc. The Virginia.gov Payment Portal has been incorporated into many of Virginia.gov's custom built online services. Additionally, it has been integrated with numerous stand alone agency or vendor developed services.

Technical Environment Overview
Virginia.gov's Payment Portal is hosted on a SUN Solaris server. All data is backed up daily, with monthly archival tapes kept at an offsite location. A backup server in a remote data center is updated daily, and can be brought online within minutes should the primary data center go down.

Security
Virginia.gov's Payment Portal modules and applications use 128 bit Secure Sockets Layer (SSL) certificate provided by Thawte, renewed annually. When necessary, Virginia.gov restricts IP access from an entity's application to its payment modules. Virginia.gov's communications with VeriSign for credit card payments also utilize SSL. VeriSign's servers, using a multi-threaded processing environment, receives the information from Virginia.gov and then transmits it over a secure private network to the appropriate financial processing network for real-time payment authorization. For ACH payments, Virginia.gov's communications with Wachovia use PGP (Pretty Good Privacy) FTP (File Transfer Protocol) for the exchange of transaction data via the scheduled batches.

Privacy
Government entities strive to meet strict privacy guidelines. To address privacy needs, Virginia.gov ensures that no full credit card numbers are stored within our systems. Virginia.gov's full Privacy Policy can be viewed at: http://www.vipnet.org/cmsportal/vipnet_987/policy_1112/index.html#privacy.

Availability
The server that hosts Virginia.gov's Payment Portal has been available 99.87% over the past year. NOVA's Via Klix gateway was available approximately 97.00% in 2004. VeriSign's gateway was available for 99.99% in 2004. The availability of the Payment Portal is impacted by these electronic processors' reliability.

The Virginia.gov Payment Portal offers Virginia government partners numerous other features and benefits as part of the Enterprise Solutions:

Customization
- Able to be integrated with diverse platforms and systems within days, including through an XML web service approach.
- Multiple payment options – credit card (auto settle or delayed capture) and/or ACH.
- Solutions to address special credit card settlement and ACH batch time needs.
- Pre-notification ACH process can be set-up to validate customer bank accounts.

Account Set-up and Testing
- Coordination of the set-up and testing of credit card merchant and VeriSign accounts.
- Significant discounts of VeriSign monthly fees and set-up fees due to the Virginia.gov's partnership with VeriSign.
- Integration with Virginia.gov's billing and credit card logging systems.

Reporting – Credit Card Payments
- Transaction level confirmations via an email or a batched daily file.
- Access to a password protected Virginia.gov credit card transaction log tool.

Reporting – ACH Payments
- Email confirmation of data transmissions.
- View ACH account activity, including actual settlement information, through a web-based administrative interface provided by Wachovia.

Monitoring
- Enforce (AVS) address collection to ensure lowest discount rate on credit card transactions.
- Internal monitoring for duplicate transactions and unavailability of credit card gateways.
- 24/7 monitoring of Virginia.gov servers and applications.

Customer Support
- Education and guidance about online payments.
- Research regarding transactions.
- Reconciliation invoicing provided to Payment Portal customers.
- Set-up of any necessary refunds or credits.
- Virginia.gov's partnership with VeriSign includes Premium Support.

Appendix D: References and Links

References:

State and Federal Sites:

The application domain team would like to publicly thank their counterparts in the many states and federal government agencies whose excellent work preceded this. We could not have completed this report as quickly as it was done without the tireless energies obviously expended to complete their Enterprise Architecture documents. We also hope that other states will find this document useful in the design and updating of their own Enterprise Architecture. Significant contributions, references, and insights were derived from the following documents and web sites.

E-Gov: Federal Enterprise Architecture (FEA)
http://www.whitehouse.gov/omb/egov/a-1-fea.html

FEA Consolidated Reference Model Document: May 2005
http://www.whitehouse.gov/omb/egov/documents/CRM.PDF

Department of Interior
Interior Enterprise Architecture: Conceptual Architecture Principles: January 4, 2002
http://www.doi.gov/ocio/architecture/conceptual/Conceptual_Architecture_Final.pdf

Housing and Urban Development:
Enterprise Architecture Practice
http://www.hud.gov/offices/cio/ea/newea/index.cfm

Enterprise Architecture Practice: Future State Technical Architecture Guidelines for the Application Development Practice: March 26, 2003
http://www.hud.gov/offices/cio/ea/newea/resources/devguide.doc

Enterprise Architecture Principles: 8/03/2005
http://www.hud.gov/offices/cio/ea/newea/resources/eaprin.pdf

Department of Education:
IT Architecture Principles Guidance: March 1999
https://www.ed.gov/offices/OCIO/archived_information/downloads/prin.doc

California:
Service-Oriented Architecture: Draft: December 8, 2005 (updated: April 21, 2006)

http://www.cio.ca.gov/ITCouncil/Committees/PDFs/SOA_Details_2006-04-21.pdf

Connecticut:
Application Development Domain Technical Architecture:05/08/2003
http://www.ct.gov/doit/lib/doit/Application_Architecture_5-8-2003_ver_2-5.pdf

Collaboration & Directory Services Domain Technical Architecture: 1/04/2001
http://www.ct.gov/doit/lib/doit/downloads/dirserv.pdf

Application Development Domain: Addendum A: Java Code Conventions
http://www.ct.gov/doit/lib/doit/downloads/Addendum_A.pdf

Application Development Domain: Appendix B: Java Coding Standards and Conventions: 2/06/2003
http://www.ct.gov/doit/lib/doit/downloads/Appendix_B.pdf

Application Development Domain: Appendix C: Microsoft .NET Migration Guidelines
http://www.ct.gov/doit/lib/doit/Appendix_C_Microsoft_dot_net_migration_strategy.pdf

Massachusetts:
Enterprise Technical Reference Model - Version 3.5
http://www.mass.gov/?pageID=itdsubtopic&L=5&L0=Home&L1=Policies%2c+Standards+%26+Legal&L2=Documents+by+Type&L3=Enterprise+Technical+Reference+Models&L4=Enterprise+Technical+Reference+Model+-+Version+3.5&sid=Aitd

ETRM Version 3.5 Application Domain:
http://www.mass.gov/Aitd/docs/policies_standards/etrm3dot5/etrmv3dot5applicationdomain.pdf

North Carolina:
Statewide Technical Architecture: Implementation Guidelines: Application Architecture
http://www.ncsta.gov/docs/Implementation%20Guidelines/domain/Application%20Domain%20Implementation%20Guidelines.pdf

Statewide Technical Architecture: Application Domain
http://www.ncsta.gov/docs/Principles%20Practices%20Standards/Application.pdf

Statewide Technical Architecture: Collaboration Domain
http://www.ncsta.gov/docs/Principles%20Practices%20Standards/Collaboration.pdf

Implementation Guideline: Applications Development Guidelines for JAVA 2 Platform, Enterprise Edition
http://www.ncsta.gov/docs/Implementation%20Guidelines/technology/J2EE%20Implementation%20Guidelines.pdf

Implementation Guideline: Enterprise Application Development Guidelines for Microsoft .Net Framework
http://www.ncsta.gov/docs/Implementation%20Guidelines/technology/Microsoft.Net%20Enterprise%20Development%20Guidelines.pdf

Pennsylvania:

Database Management Systems: Production and Operational Standards: February 23, 2005
http://www.oit.state.pa.us/oaoit/lib/oaoit/STD_INF001B.doc

Other Information References:

Gartner Group:
http://www.gartner.com/
The ICC and SOA Governance: Managing a Successful Integration Project. Paolo Malinverno

SOA Governance: Frameworks, Registries and Policy Enforcement. L. Frank Kenney and Daryl Plummer

SOA Governance: Frameworks, Registries and Policy Enforcement. Gartner. L. Frank Kenney and Daryl Plummer. 5-7 December 2005, JW Marriott Grande Lakes Orlando, Florida

Enterprise Architecture Executive Council (EAEC):
http://www.eaec.executiveboard.com

EA Framework and Governance Deliverables: The IBM Approach

John Hancock: Embedding Enterprise Architecture Across the Systems Life Cycle

Motorola: Development Asset Reusability

Business Process Reuse via Service-Oriented Architecture

Microsoft: Application Portfolio Management

Pfizer: Alternative Architectural Approaches for Information Integration Service-Oriented Architecture

Other Articles:
Selecting a Development Approach: February 17, 2005
http://www.cms.hhs.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf

System Development Methodologies for Web Enabled E-Business: A Customization Paradigm; Linda Night, Theresa Steinbach, and Vince Kellen; November 2001; http://www.kellen.net/SysDev.htm

N-Tier Application Development with Microsoft.NET by Karim
http://www.microsoft.com/belux/msdn/nl/community/columns/hyatt/ntier1.mspx

SOA Governance, WebLayers, Inc.
http://www.weblayers.com/gcn/whitepapers/Introduction_to_SOA_Governance.pdf

Other Website References:

The Open Group:
http://www.opengroup.org/architecture/togaf8-doc/arch/toc.htm

The Center for Open Source & Government
http://www.egovos.org/

The National Center for Open Source Policy and Research
http://www.ncospr.org/

Open Source Initiative (OSI)
http://opensource.org/

Microsoft:
http://www.microsoft.com/resources/sam/Implementing_Policy.mspx

IBM:
http://www.ibm.com/us/