MicroStrategy | 2021

Services Architect

Standardizing and Managing Custom Applications

CONTENTS

3. Building a Productive Development Environment

4. Managing the Application Life Cycle

1

PRE-COURSE SETUP

MicroStrategy Cloud environment

This course begins with the setup of the virtual MicroStrategy Cloud environment. The environment is composed of two servers: a Linux server and a Windows server. This environment is used to complete exercises throughout the course.

These steps can also be used to reset the environment to its original state, if you elect to redo the exercises.

Exercise 1.1: Access the environment

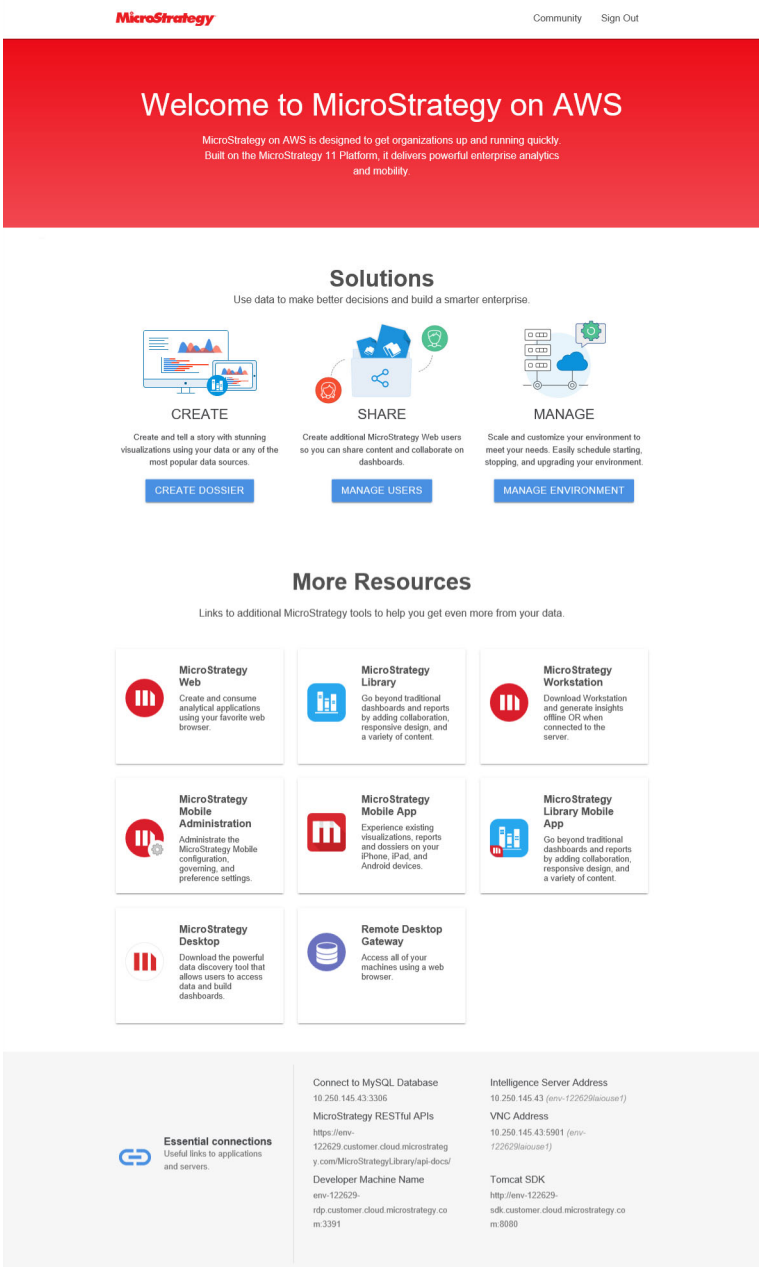
To prepare for the course, access your MicroStrategy Cloud environment. The login credentials and other information you need to access your environment are included in the MicroStrategy Cloud email.

Access the MicroStrategy platform

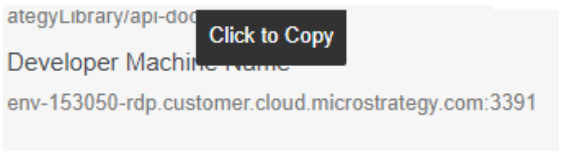
1 On your local machine, in the MicroStrategy Cloud email, click **Access MicroStrategy Platform**.

2 In the **User name** and **Password** boxes, enter the login credentials provided in the MicroStrategy Cloud email. Click **Login**.

The MicroStrategy Cloud landing page is displayed.



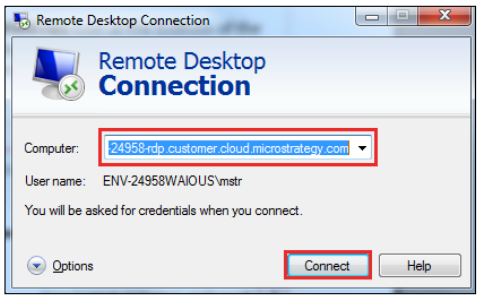
3 In the Essential Connections area, hover over **Developer Machine Name** and click **Copy** to copy the address to your clipboard. For example, the address is similar to: **env-64958-rdp.customer.cloud.microstrategy.com:3391**.



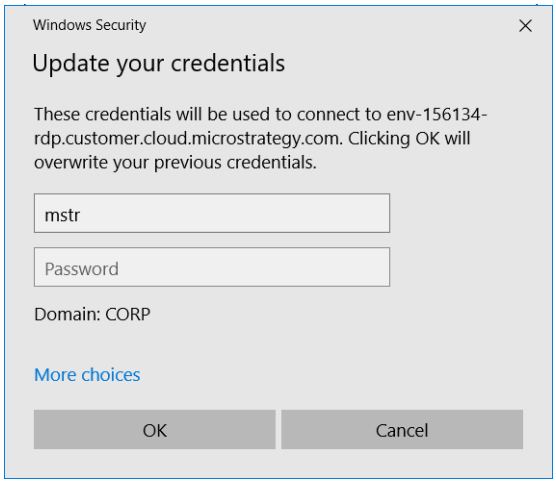
4 On your local Windows machine, from the taskbar, search for **Remote Desktop**. From the search results, select **Remote Desktop Connection**.

This step may differ, depending on the version of Windows operating system on your computer.

5 In the Remote Desktop Connection window, in the **Computer** box, paste the machine name, and click **Connect**.



6 In the Windows Security window, type the username and password from the Welcome to MicroStrategy on Cloud email, and click **OK**.



If an identity verification message is displayed, click **Yes**.

You are now connected to the Windows machine in your MicroStrategy Cloud environment.

Prepare the environment

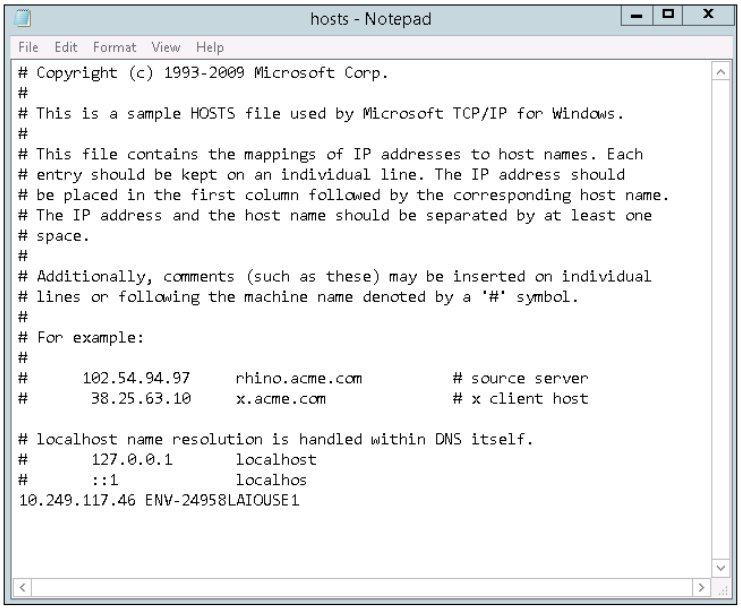
Now that you have configured the Remote Desktop application and have successfully connected to your Windows machine, you are ready to prepare the environment.

Reviewing the Windows hosts file

The hosts file cross-references specific IP addresses with user-friendly names, just like a Domain Name Server (DNS) that translates IP addresses to human-readable website names as you browse the Internet. DNS translation enable you to navigate to www.microstrategy.com instead of a forgettable series of numbers like 104.121.85.249.

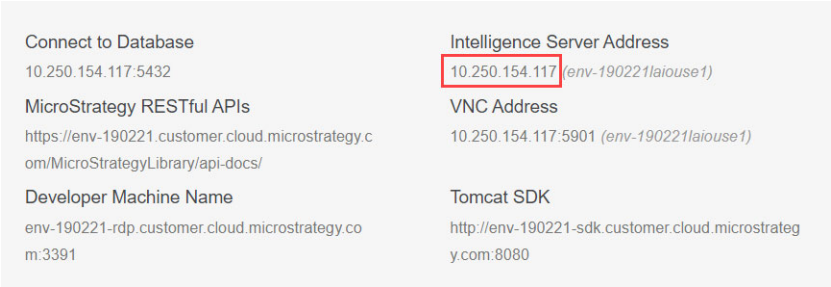
The hosts file serves a similar purpose, but it operates locally on a specific computer. We use the hosts file to map the Intelligence Server IP address to the readable machine name, as in the following example:

10.249.117.46 ENV-24958LAIOUSE1



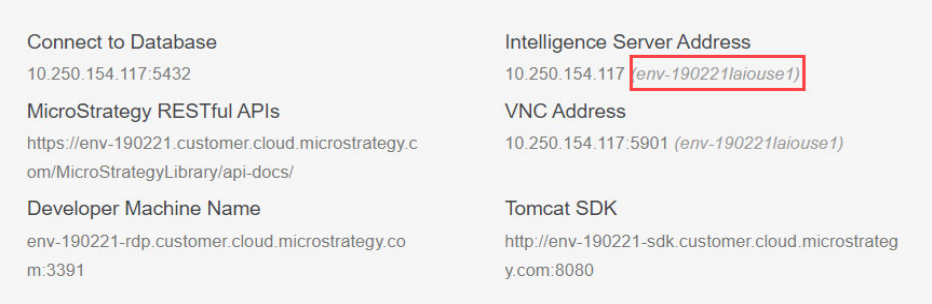
In the hosts file, the IP address is mapped to the machine name using the following information:

- **XXX.XXX.XX.XXX**: The IP address of the Intelligence Server located in the Essential Connections area of the MicroStrategy Cloud landing page, as shown in the image below. To copy the address, hover over it and click **Copy**.



- **TAB**: A tabulation character must separate the first and third parts of the line. A space will not work.

- **ENV-XXXXLAIOUSE**: The Intelligence Server machine name, where XXXX is your environment's unique ID number located in the Essential Connections area of the MicroStrategy Cloud landing page, as shown in the image below.



Exercise 1.2: Close Eclipse and stop your Tomcat web server

To complete the software installation for this class, you must first close the Eclipse IDE and stop the Tomcat web server on the Windows machine. To ensure proper performance on the Tomcat server, verify that an appropriate amount of memory is allocated to the Tomcat server's Java Virtual Machine.

Close Eclipse and stop Tomcat

1 On the Windows desktop, right-click **Tomcat_Stop.bat** and select **Run as Administrator.**

2 If the Eclipse environment is open, save your work and close it.

Exercise 1.3: Run the final installation script

The exercises in this class require additional software installation and configuration. In this section, you will run an automated script that performs the following tasks:

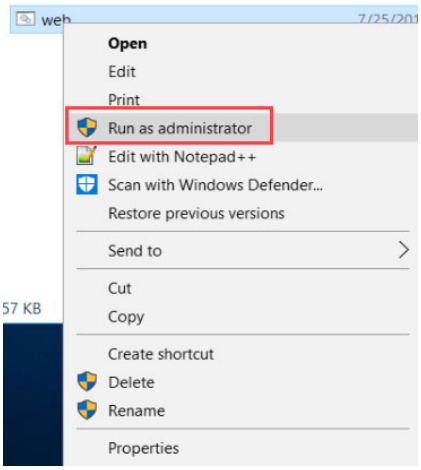
- Remove the current version of Eclipse. This is why you closed Eclipse earlier.

- Install MicroStrategy Web and MicroStrategy Library applications. This is why you shut down Tomcat earlier.
- Configure Tomcat to locate the MicroStrategy applications in c:\sdk_workshop\war.
- Install a recent version of Eclipse with a MicroStrategy plug-in already configured.
- Adjust Eclipse settings for MicroStrategy applications.

Run the installation script

1 In File Explorer, navigate to the **C:\sdk_automation\scripts** folder.

2 Right-click **web.bat** and select **Run As administrator**. In the User Account window, click **Yes**.



Run the script only once, as the script deletes all of your progress if it is executed twice. If you would like to revert the environment to a brand new state, you can run the script again.

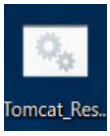
Once the script ends, the command line window closes automatically. This might take a few minutes.

Exercise 1.4: Verify the Eclipse installation

To confirm Eclipse was installed successfully, launch it and make sure it is properly configured.

Verify the Eclipse installation

1 From the Windows desktop, right-click **Tomcat_Restart**, and select **Run as Administrator**. The Tomcat web server restarts after a minute or so.

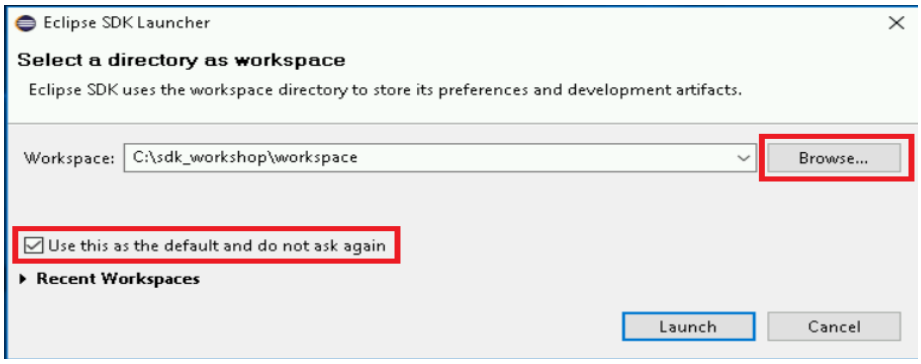


2 Right-click **Eclipse** and select **Run as Administrator**. In the User Account window, click **Yes**.

Select the project workspace

In Eclipse, a workspace is a folder that contains your Eclipse settings and projects. A workspace enables you to focus on a single project without impacting others. You can think of a workspace as a hierarchy of project files.

3 In the Eclipse SDK Launcher window, click **Browse**.



4 Click **This PC** on the left, then navigate through the following folders: **C:/sdk_workshop/workspace**.

5 Click **Select Folder**.

6 Select the **Use this as the default and do not ask again** check box, then click **Launch**.

7 In the Older Workspace Version window, click **Continue**.

8 In the upper right, select **MicroStrategy Web Customization Editor**.

Configure the MicroStrategy project

9 In the Package Explorer tab, right-click **MicroStrategy**, point to **New**, and select **MicroStrategy Project**.

10 In the Project Title box, type **MicroStrategy**.

11 Click the **Browse** next to **Project Root Location**, and navigate to **C:\sdk_workshop\war**, select **MicroStrategy**, and click **Select Folder**.

12 Click **Finish**.

Configure the Library project

13 Right-click **MicroStrategyLibrary**, point to **New**, and select **MicroStrategy Project**.

14 In the Project Title box, type **MicroStrategyLibrary**.

15 Click the **Browse** next to **Project Root Location**, and navigate to **C:\sdk_workshop\war**, select **MicroStrategyLibrary**, and click **Select Folder**.

16 Click **Finish**.

17 **Minimize** Eclipse.

Exercise 1.5: Connect Web to the Intelligence Server

To complete the MicroStrategy Web application setup, connect the application to the Intelligence Server located on the Linux machine.

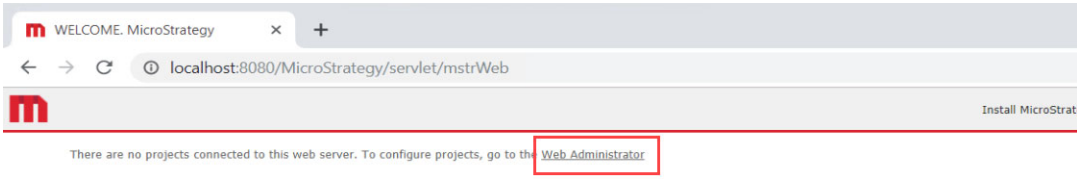
Complete the MicroStrategy Web setup

1 From the desktop, open a **Chrome browser**.

2 Navigate to the following URL to validate that Tomcat and MicroStrategy Web are running correctly:

http://localhost:8080/MicroStrategy/servlet/mstrWeb

3 Click the **Web Administrator** link.



4 In the Sign in window, type the following username and password:

Username = **admin**

Password =**sdkws**

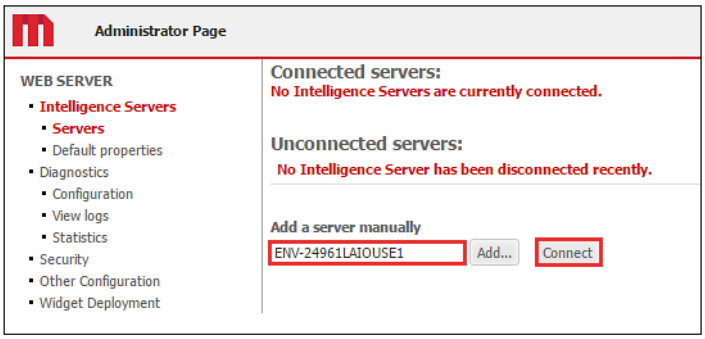
5 In the Save Password window, click **Yes**.

Connect your local instance of MicroStrategy Web to the Intelligence Server

6 From the desktop of the Windows machine, open **hosts.txt** in **Notepad**.

7 At the bottom of the file, copy the Intelligence Server machine name. For example, env-123456laiouse1.

8 In the Administrator Page in Chrome, in the **Add a server manually** box, paste the copied Intelligence Server machine name and click **Connect**.



9 Click the **MicroStrategy Web Home** link.

A list of connected projects is displayed. For all subsequent exercises in this course, access MicroStrategy Web using the Chrome browser on the Windows Developer Machine. This ensures that you are able to access custom files used in this course.

10 **Minimize** the MicroStrategy Web window.

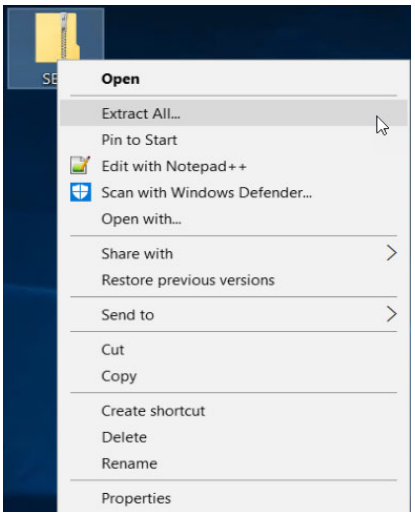
Exercise 1.6: Extract the class files

The exercise files for this class are in an archive that you must copy to the Windows machine in your Cloud environment.

Extract the class files

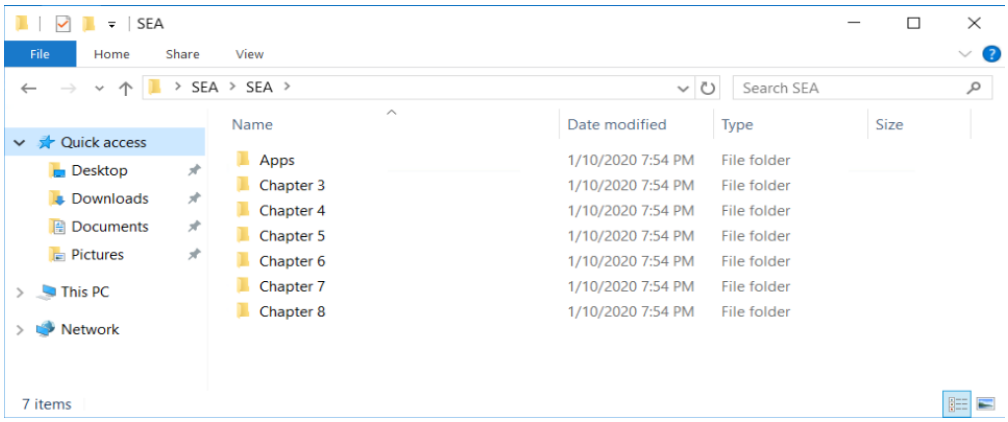
1 Copy the **SEA.zip** file to the desktop of the Windows machine.

2 Right-click the .zip file and select **Extract All**, as in the following image:



3 In the Select a Destination window, click **Extract**.

4 From the desktop, open the **SEA** folder.



Your environment is now ready for the class. Let's begin.

2

The Intelligent Enterprise

Analytics from the start

LUNAR WATER

Lunar Water is a startup gaming studio that leverages data analytics to expand all aspects of its business. The business insights gleaned from its MicroStrategy solution have positioned this small company to grow quickly.

At Lunar Water, predictive analytics is used to improve corporate development, supply chain optimizations, marketing trend analysis, mobile applications that enable analysis in the field, and for personnel and asset management. Lunar Water's main offering is a customizable card game that became an overnight success, allowing the company to grow its workforce exponentially.

As is often the case in startups, the company was not prepared for its overnight success. The large user base has led to personnel acquisitions around the world, with each office implementing its own processes and solutions.

Lunar Water executives have analyzed business process and discovered a fragmented landscape:

- Many users have their own dossiers and data.
- Each department owns information that is not shared with the rest of the organization.
- Each department has developed applications that are used by only a few people. Some applications are redundant, while others could benefit other segments of the company.
- Fragmented analytics have produced incomplete and conflicting goals for each department.

To create analytics applications that the entire company can leverage, and to limit analysis in silos, an enterprise approach must be adopted to:

- Establish corporate standards for consistency across applications.
- Ensure consistent work processes to achieve continuity and reliability.
- Utilize best practices to increase efficiency.

To ensure that Lunar Water stays at the forefront of their market, they want to implement the MicroStrategy Intelligent Enterprise to achieve their goals.

Introducing the Intelligent Enterprise

An Intelligent Enterprise is a data-driven organization that effectively designs and implements Business Intelligence (BI) solutions while promoting effective use of data across the enterprise. This fosters intelligent, efficient growth and development, with a focus on data governance and alignment of strategic business goals to technology investments.

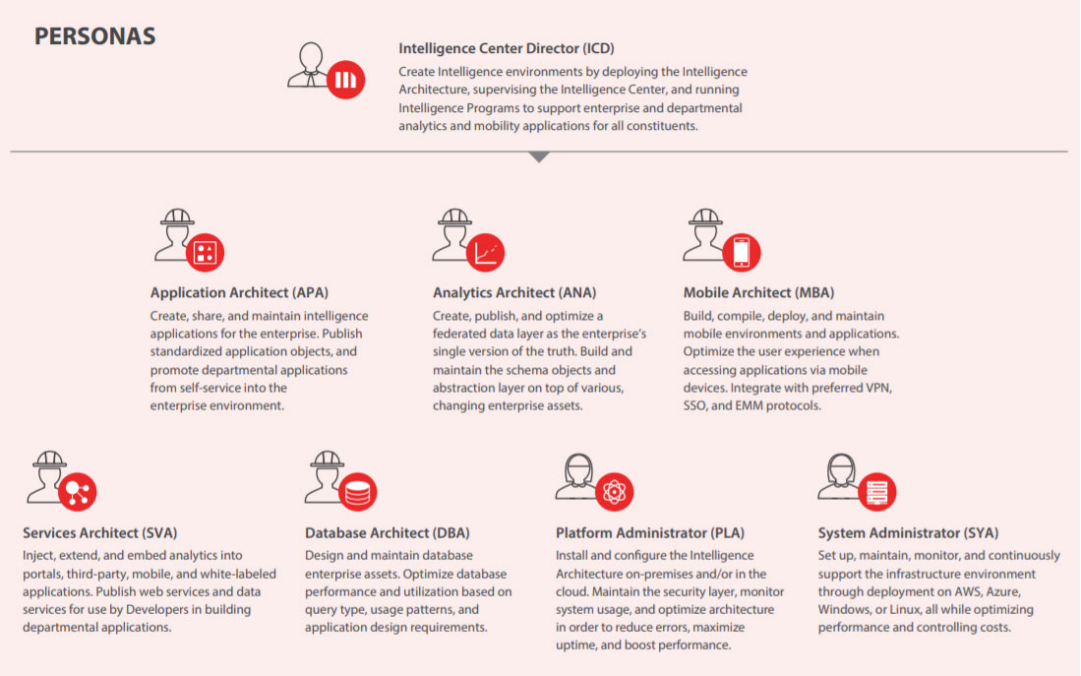
Getting there requires the right tools and structure to balance traditionally counteractive forces—agility and governance, convenience and security, ease of use and enterprise functionality—all critical capabilities that the MicroStrategy platform is positioned to support with its unique intelligence architecture. A successful Intelligent Enterprise:

- Drives adoption and success of enterprise BI.
- Coordinates BI implementations.
- Maintains sound data governance and a single version of the truth.

- Provides a formal approach to documenting processes, creating content, and ongoing maintenance.
- Ensures that BI is aligned with enterprise strategy.

Along with quick and easy ad-hoc departmental solutions, MicroStrategy has the robust, proven ability to support high-scale deployments and establish a single source of truth. MicroStrategy's tools include data-governance features, administrative controls, and management capabilities within the enterprise.

The people with the right knowledge to create, grow, and implement these programs and practices are key to an Intelligent Enterprise. The Intelligence Center personas develop the standards that bring the Intelligent Enterprise to life. Their role is to implement and foster best practices to make the most of the MicroStrategy platform. Their expertise influences the ideal implementation of tools, applications, and devices.

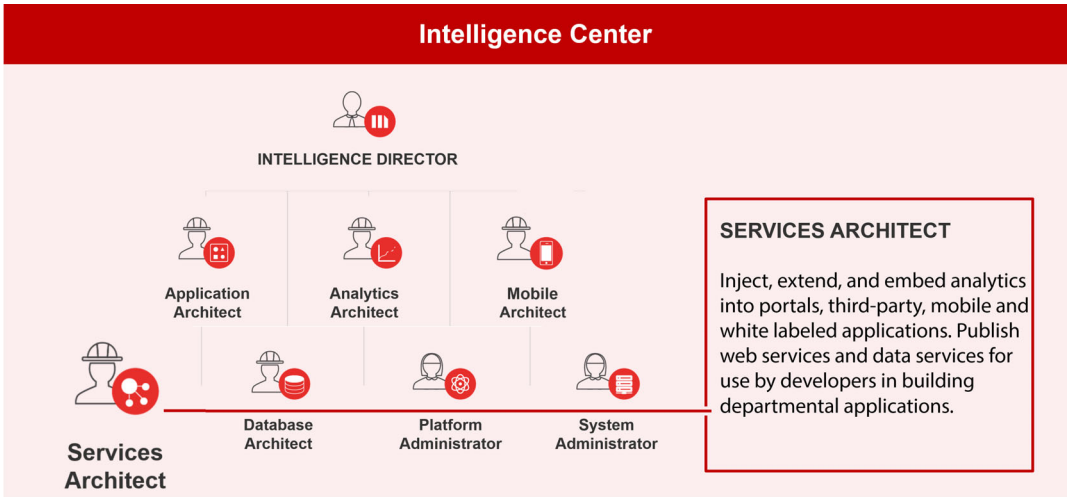


Your role: Services Architect

Lunar Water has hired you to take on the Services Architect role, responsible for software development standards and practices in the enterprise. You can prepare for this role through a combination of experience and the following MicroStrategy classes:

- SDK for iOS Applications

- SDK for Android Applications
- Advanced SDK for Customizing Visualizations
- Advanced SDK for Customizing Branding
- Advanced SDK for Integration and Single Sign-On
- Services Architect: Standardizing and Managing Custom Applications
- Services Architect (SVA) Certification



On a high level, your role consists of:

- Establish software development strategies to help developers consistently embed analytics in portals, third-party, and white-label applications.
- Establish an architecture that uses open standards to enable developers to efficiently and reliably publish web services.

This course helps you approach the challenges you will face as a Services Architect, and guides in you in making the right choices as you build an Intelligent Enterprise. As you develop your standards, work with other Intelligence Center personas to communicate your guidelines and avoid redundant or counter-productive efforts.

Standardizing software development

As you build a development stack and establish coding standards that help developers leverage MicroStrategy to create powerful applications for the organization, consider the following aspects of your software development efforts.

Establishing standards to develop line-of-business applications

A line-of-business application are business applications that influence the bottom line. To ensure that the business can conduct its operations without interruptions, these applications must be built and maintained to maximize efficiency and reliability.

For example, a retail organization might have an inventory system that must contain current and reliable data to serve its customers. As the Services Architect, your focus would be to develop best practices to build and maintain this application. You might establish coding standards that enable developers to quickly read and understand the work of their colleagues, or create repository guidelines that enable efficient revision management.

At Lunar Water, the line-of-business application embeds MicroStrategy functionality to analyze player history to build lootbox content. To ensure that the analytics solution creates a valuable experience for your customers, you must create standards that help developers embed and maintain MicroStrategy functionality in your own applications.

Cultivating supporting tools

Organizations often build applications to achieve supporting administrative tasks. For example, an airline company may use an application to record maintenance history for each plane, or track the number of flight hours for pilots. Some tools may inject analytics into a database or retrieve the latest data from MicroStrategy. For an airline company, telemetry metrics from a plane retrieved in real-time can help assign planes to flights and increase efficiency.

To ensure that supporting tools are efficiently and reliably developed, they must be built and maintained using the same standards as business applications. For Lunar Water, this means that the 3D engine, level editor, card studio, player profile management, and tournament manager supporting tools must meet the same standards as the company's user-facing application.

Supporting development activities with analytics

Data analytics solutions help your organizations leverage collected information to garner valuable insights and make decisions that drive the business forward. Custom analytics can include a plug-in to apply branding to MicroStrategy Web, a custom mobile app, data retrieval tools, or dossiers embedded on the company portal.

As the Services Architect, you establish the standards and methodologies used by developers to efficiently and predictably create reliable custom analytics solutions using MicroStrategy SDKs and REST APIs.

At Lunar Water, custom analytics projects include a mobile sales application and embedded dossiers on the leader board website. The best practices you implement should include rules for employing MicroStrategy SDKS and libraries to develop apps for data scientists, analysts, and business managers.

Exercise 2.1: Your own standards

Imagine yourself as the Services Architect in your actual workplace and think about your corporate standards and guidelines to answer the following questions. If a concept does not currently apply to your organization, ask yourself "How can we apply this in our enterprise and make it work?"

Answer the following questions, and then return here at the end of the class to see how your answers change.

1 List the software products installed in your organization's developer environments.

_____

_____

_____

_____

2 What are the naming conventions for your objects, classes, and variables in code.

_____

_____

3 Where are MicroStrategy custom plug-ins and visualizations stored?

_____

4 How are code snippets handled?

_____

_____

5 What is your source code management system?

_____

6 How do you handle a developer issue with a MicroStrategy REST API call?

_____

7 Describe your code review process.

Summary

In this chapter, you learned about the Intelligent Enterprise and its impact on developing custom analytics applications. We focused on the Services Architect persona, who standardizes all development efforts. You discovered the responsibilities of a Services Architect at Lunar Water, a gaming studio that leverages analytics improve its user experience.

In the next chapter, you will explore your role in curating the development environment.

3

Building a Productive Development Environment

In an Intelligent Enterprise the development environment consists of people, hardware, and best practices. As a Services Architect, your goal is to create a development environment that cultivates an efficient and reliable software engineering workflow.

Building a team

You are responsible for building and training a development team that creates and maintains applications for your organization. As you build your team, consider the following requirements:

- The team must have the skills to develop a project from scratch and provide maintenance in the form of bug fixes and enhancements
- The team is guided by process and standards documentation
- The team provides support for its products

Identifying key team roles

As you build or add to your team, keep in mind that members with diverse talents and levels of experience can help you form a well-rounded team that can handle a wide array of challenges. A typical development team might consist of the following members in your organization.

- **Manager**: People management and coding experience can help drive projects forward.
- **Lead Developer**: Senior members that can guide and train junior developers. Takes on project ownership to meet deadlines and quality expectations.
- **Developer**: This group makes up the bulk of a development team. Developers might have a limited amount of experienced, but can be groomed by the Lead Developers to conform to organizational standards and guidelines.
- **Technical Writer**: Maintains and organizes documentation that establishes standards and specifies project goals. To produce and maintain technical documents, processes, and guides, Technical Writers require technical knowledge, requirements gathering, and logical thinking skills.
- **Technical Support**: Support users with technical issues in your teams deliverables. The support staff investigates and solves bugs and enhancement requests submitted by the team or application users. This role depends on product maintenance and enhancement standards and rules you establish.

Foster MicroStrategy knowledge

MicroStrategy experience helps your team design and develop customized analytics applications with an end-user perspective. If your team members need to expand their MicroStrategy experience, they can take MicroStrategy Education classes and become certified as analysts, architects, and developers.

Hiring based on technical competency

As you hire developers and other roles for your team, look for technical experience, as well as other general skills that are required in a development

environment. The following table contains several examples of desired technical and general skills for your team members.

Technical skills		Other skills
Web development, including HTML, CSS, and JavaScript	MicroStrategy SDKs and APIs	Problem solving
Java	Swift	Detail oriented
XML	R	Sense of ownership
JSON	Python	Team player
D3	Security	Management
SVG	git and git Hub	Leadership
SOA	Build management	Learning appetite
REST services	Source code management	Out of the box thinker
Database management	SQL	Analytical thinking skills
IOT	Hardware	Critical thinking

Build your own list to prioritize the required skills in your organization and identify areas of required training for current team members and new hires. Use your compiled list of skills to build a diverse and experienced team that is suited to build created solutions using a broad set of technologies.

Tracking core competencies

A list of your team's core competencies can help you quickly identify whether you have the talent to address a new development project. As the Services Architect, the list becomes your catalog of assets. You can build and maintain your list by using a simple team survey. o ensure that your list accurately reflects your team's abilities, ask your employees to identify skills in which they are proficient.

For example, Lunar Water uses a dossier to analyze employee skills and other competencies. When a new project is proposed, you can filter the dossier based on the required skills, and identify candidates for the project.

As you build your list, categorize competencies into categories, as in the examples in the following table.

Categories	Categories
Features	Skills
Assets	Regulations
Products	Programs
Services	Geography
Solutions	

Once you establish your categories, identify the list of competencies within each of them. For example, the following programming languages are in the Skills category.

Skill	Skill	Skill
AJAX	HTML5	R language
ajax js	Java	Rake
angular js	JavaScript	react js
Bash	jquery	Ruby
Batch/Shell	JSP	Scala
bokeh	MDX	Servlets
C#	MSBuild	SQL
C++	Node.js	Swift
CSS3	Objective C	Tag libraries
D3	Perl	Typescript
Flash	php	VB6
Gherkin	PMML	XML/JSON
Gradle	Power Shell	XQuery
Groovy	Python	

The following is a list of developer tools that you might expect your team members to be able to use.

Skill	Skill	Skill
ANT	Git	Open Cpp Coverage
Adobe Creative Suite	grunt	PageSpeed
apktool	Gtest/gmock	PowerPoint
Appium	IBM App Scan	Postman
Browser debug tools (Ex. Chrome Developer tool)	Jacoco	rspec
Burp Suite	JMeter	Selenium
Calabash	JMockit	Surevy Gizmo
Code analysis tools (FindBug, Lint)	JUnit	UNIX Editor (emacs or VIM)
CSPAT	Kibana	Veracode
Cycript	Maven	Visual Studio
Debug tools (GDB)	Microsoft Office	webpack
Eclipse IDE	minitest	WebStorm IDE
Fiddler	Nagios Plugins	windbg
FrameMaker	Nessus	Xcode

This following table contains a list of application servers that you might expect your developers to have experience with.

Skill	Skill	Skill
Apache	IIS	Oracle iPlanet Web Server
Apache HttpD	JBoss	Oracle Web Application Server
Apache Tomcat	JBoss Enterprise Application Platform	SAP NetWeaver Application Server
Apache TomEE	Jetty	SAP NetWeaver Portal
Django	Liferay Portal	Tomcat
DotNetNuke web content management system	Microsoft Internet Information Services	WebLogic

Skill	Skill	Skill
Drupal	Microsoft SharePoint Server	WebSphere
IBM HTTP Server	Netty	WildFly
IBM WebSphere	Oracle GlassFish Server	
IBM WebSphere Portal	Oracle HTTP Server	

The examples in the above table can be used to build an employee survey in your organization. You can then use the survey response to create a dossier that tracks your team's abilities. For example, you can create a simple web page with the tables above, and position check boxes beside every item. Save the survey data to a database and create an Intelligent Cube in MicroStrategy to hold and update the data. Then create a dossiers based on the cube to find appropriate resources for projects in your organization.

Best Practice

Extend your skills survey to the entire enterprise to locate talent outside of your team for ad-hoc collaboration.

On-boarding new hires

When you hire a new employee, a skills inventory must be part of the on-boarding process. This becomes a baseline that you can use to:

- Create a customized initial training plan for each employee.
- Identify opportunities for employee growth and development.

Establishing a skills gathering process

Best Practice

Repeat the skills assessment survey on a regular schedule to reflect newly learned or developed skills.

With formal and informal training and self taught skills that are acquired throughout the year, the skills inventory must also be updated to provide insights into hiring decisions and training needs.

Exercise 3.1: Create a skills inventory sheet

In this exercise, prepare a skill inventory sheet to help Lunar Water identify the pool of skills your team members possess. You can build the sheet in Notepad++.

Follow these steps to build your sheet and compare with others once completed.

Create the skills sheet

1. On the Windows desktop, create a folder on named **Lunar Water**. This folder will server as the Lunar Water company portal throughout this class.
2. In the **Lunar Water** folder, create a folder named **Skills Inventory**.
3. From the desktop, double-click **Notepad++**. An empty file is displayed.
4. Save the new file as **Skills Sheet.txt**.

Add identification data

1. Add a title to the page, something along the lines of **Lunar Water Skills Inventory**.
2. Add an identification section that contains the following elements:
   - Name
   - Employee ID
   - Current position
   - Supervisor
   - Hire Date
   - Changes since last assessment (Y/N)
   - One more element of your choice

Core competencies

Lunar Water develops applications for the Web and uses Java as a back-end language. The company also develop mobile apps for iOS and Android. Keep that in mind for the next sections of this exercise.

1. Define a scale to indicate a level of mastery for each skill. Remember, the survey results will be used as source data for a dossier.

2. Using the skills in *Tracking core competencies, page 30*, add the desired technologies for your team members. Add two technologies not directly related to web development. Limit your list to 7-10 elements.

3. Add another section for applications and tools experience. Limit your selection to 10-15, including two not immediately related to development.

4. Add a section for Other Competencies and list three elements appropriate for your development team.

When you complete this part of the exercise, your sheet might look similar to the following example:

```
Lunar Water
Skills Assessment sheet

Name:_____

Employee ID:_____

Current position:_____

Supervisor:_____

Hire Date: ___/___/___

Changes since last assessment: Yes___   No____

_____
```

```
Core Competencies
Level:  0-Never heard
        1-Know the name
        2-Can explain the topic
        3-I use this a little
        4-I use contantly in my work
        5-Expert, I can teach it

Technologies
        AJAX                                                    0__ 1__ 2__ 3__ 4__ 5__
        ajax js                                                 0__ 1__ 2__ 3__ 4__ 5__
        angular js                                              0__ 1__ 2__ 3__ 4__ 5__
        C#                                                      0__ 1__ 2__ 3__ 4__ 5__
        CSS3                                                    0__ 1__ 2__ 3__ 4__ 5__
        HTML5                                                   0__ 1__ 2__ 3__ 4__ 5__
        Java                                                    0__ 1__ 2__ 3__ 4__ 5__
        JavaScript                                              0__ 1__ 2__ 3__ 4__ 5__
        jquery                                                  0__ 1__ 2__ 3__ 4__ 5__
        JSP                                                     0__ 1__ 2__ 3__ 4__ 5__
        react js                                                0__ 1__ 2__ 3__ 4__ 5__
        Servlets                                                0__ 1__ 2__ 3__ 4__ 5__
        Swift                                                   0__ 1__ 2__ 3__ 4__ 5__
        XML/JSON                                                0__ 1__ 2__ 3__ 4__ 5__
        XQuery                                                  0__ 1__ 2__ 3__ 4__ 5__

Applications/Tools
        Adobe Creative Suite                                    0__ 1__ 2__ 3__ 4__ 5__
        Android Studio                                          0__ 1__ 2__ 3__ 4__ 5__
        apktool                                                 0__ 1__ 2__ 3__ 4__ 5__
        Browser debug tools (Ex. Chrome Developer tool)         0__ 1__ 2__ 3__ 4__ 5__
        Code analysis tools (FindBug, Lint)                     0__ 1__ 2__ 3__ 4__ 5__
        Eclipse IDE                                             0__ 1__ 2__ 3__ 4__ 5__
        Fiddler                                                 0__ 1__ 2__ 3__ 4__ 5__
        Git                                                     0__ 1__ 2__ 3__ 4__ 5__
        JMeter                                                  0__ 1__ 2__ 3__ 4__ 5__
        JUnit                                                   0__ 1__ 2__ 3__ 4__ 5__
        Maven                                                   0__ 1__ 2__ 3__ 4__ 5__
        Microsoft Office                                        0__ 1__ 2__ 3__ 4__ 5__
        PageSpeed                                               0__ 1__ 2__ 3__ 4__ 5__
        Veracode                                                0__ 1__ 2__ 3__ 4__ 5__
        Visio                                                   0__ 1__ 2__ 3__ 4__ 5__
        Visual Studio                                           0__ 1__ 2__ 3__ 4__ 5__
        Xcode                                                   0__ 1__ 2__ 3__ 4__ 5__

Other
        Graphics Design                                         0__ 1__ 2__ 3__ 4__ 5__
        Python                                                  0__ 1__ 2__ 3__ 4__ 5__
        UML                                                     0__ 1__ 2__ 3__ 4__ 5__
        Other Programming Languages                             _____________________________
                                                                _____________________________
                                                                _____________________________
        Spoken/written languages (fluent)                       _____________________________
                                                                _____________________________
```

5 How would your skills profile differ for a junior developer? For a team lead? Compare your desired list of skills for these roles with your classmates' answers.

Exercise 3.2: Assess skills for a project

Lunar Water has two new development project proposals. As the Services Architect, create a list of skills that are required for employees working on each project.

Create a skills list for **Project 1**

The Sales team needs a web app to manage the availability of the 20 office spaces at Lunar Waters headquarters. The application must pull information about the employees from the MicroStrategy HR personnel Intelligent Cube and use it along with a local database to create, delete, and update the reservations for each office.

1 Below is a list of skills you might want your team to posses for this project. Pick the **top 10 skills** necessary for your project.

Skill	Skill	Skill
AJAX	HTML5	GitHub
Microsoft Word	Java	C#
Adobe Flash	JavaScript	Adobe Photoshop
Bash	jquery	Scala
Batch/Shell	Adobe Lightroom	Servlets
Unreal Engine	MicroStrategy	SQL
Tomcat	Active Directory	Swift
C++	Objective C	MicroSoft Excel
CSS3	Perl	Lenell hardware
Adobe XD	Visual Studio	VB6
MySQL	F Sharp	XML/JSON
Adobe InDesign	Adobe Illustrator	XQuery
Gradle	SourceSafe	Unity
ggplot	R language	Microsoft Visio

You might find that 6 or 7 skills are easy to identify. The rest of your selections may differ based on your personal experience with similar projects. You might feel

inclined to select a large set of skills, but try to narrow your choices to the essentials to avoid an unattainable requirement.

See *Appendix A, Answers to Exercises* for possible answers. Your choices may vary.

Create a skills list for Project 2

The development team wants to modernize the transaction server used for lootbox transactions. The current application contains components written in VB6 and C++, while others were developed in Objective-C.

A legacy database is in place to hold the transactions ledger. The team wants to use the latest programming languages for both Windows and MacOS. The new database will be hosted locally, while the source code will be hosted in the common repository used by Lunar Water.

To communicate with the application, a web page retrieves JSON files generated by the transaction server and displays the information to departments that do not have direct access to the server.

To staff this project, identify a list of required skills.

1 Pick 10-12 skills required for team members to complete the project.

Skill	Skill	Skill
AJAX	HTML5	GitHub
Microsoft Word	Java	C#
Adobe Flash	JavaScript	Adobe Photoshop
Bash	jquery	Scala
Batch/Shell	Adobe Lightroom	Servlets
Unreal Engine	MicroStrategy	SQL
Tomcat	Active Directory	Swift
C++	Objective C	MicroSoft Excel
CSS3	Perl	Lenell hardware
Adobe XD	Visual Studio	VB6
MySQL	F Sharp	XML/JSON
Adobe InDesign	Adobe Illustrator	XQuery

Skill	Skill	Skill
Gradle	SourceSafe	Unity
ggplot	R language	Microsoft Visio

Prioritizing skills

Depending on your project, you may have more skill requirements than your employees possess. To create a feasible list that can be fulfilled by your employees, rank the skills, prioritizing those that are necessary to complete the project. Less common skills can be acquired through training.

To prioritize your list, create a points system that captures the level of mastery for each skill. For example, a game developer might need extended experience in Unity and Unreal Engine, but require only general knowledge of Tomcat.

Prioritization scale example

Take, for example, a search for a developer with MicroStrategy and Java experience. To screen possible candidates, you might create the following formula to rank the skills for this project:

$$\text{Score} = (\text{JavaYears} \times \text{JavaWeight}) + (\text{MSTRYears} \times \text{MSTRWeight})$$

This formula enables you to easily associate a value to each candidate, and predict their ability to contribute to the project. Using this formula, you can look for well-rounded team members, or you can identify specialists that can be taught ancillary skills.

On-boarding team members

To ease the transition to your team, new employees and transfers from other departments must complete an on-boarding process that includes training on best practices, workflows, tools, technologies, and so on.

Work with Human Resources, the System Administrator, and your team to develop a plan to on-board new team members to ensure that core knowledge is distributed at an early stage.

Avoiding baptism by fire

Best Practice

To properly on-board employees, create a strict training schedule that must be completed before team members can contribute to projects. A required training schedule ensures that team members learn your established standards and guidelines prior to jumping into their work.

Establishing a standard on-boarding process

Best Practice

The on-boarding process should include a list of tasks and training that is reviewed by a mentor or manager. To make the process transparent to all development teams, post the list of tasks on an internal site or as a dossier on the corporate portal.

The tasks in your organization might include:

- Physical space allocation
- Security authorization
- Software installation and training
- Recurring team meeting schedule
- Initial project assignment

Scheduling regular one-on-one meeting

Communication is a key component of team success. To facilitate honest communication in a safe space, establish a scheduled time for one-on-one feedback,

An open dialogue among teammates keeps people focused on the right objectives, and illuminates problems before they derail a project. In your organization, establish a standard for managers to schedule one-on-one meetings with each team member on a monthly or bi-weekly basis.

Training the development team

Your Human Resources department might offer soft skills training, but a development team needs technical training to remain competitive and grow with the industry.Tools and technology training empowers team members with new skills and fosters growth by helping them move beyond foundational knowledge.

Training provides opportunities to contribute to a broader range of projects, and positions the organization to take on new and advanced challenges.

In your organization, development teams might require periodic training on:

- Refresh their knowledge related to existing skills
- Build on their foundational knowledge for an existing skills
- Learn new technologies to contribute to research and development and future projects

Best Practice

Work with the System Administrator to establish a training portal that can be used on demand. For example, you can create a SharePoint page with links to various free training (Khan Academy, edX, etc.) or purchase licensed training (Safari Online, Lynda, etc.). You can also invest in a formal Learning Management System (LMS) and hire a resource to maintain the system and analyze whether training goals are met.

Exercise 3.3: On-board team members

As you begin your role as the Services Architect at Lunar Water, you want to launch a site for your team's on-boarding process. In this exercise, you prepare an on-boarding document and publish it in your staging environment.

At a high level, perform the following tasks to complete the on-boarding site:

- Install a new instance of Tomcat server located in **c:\tomcat**, and run it manually from a prompt on port **9090**. You can find the install file in **Desktop\SEA_ExerciseFiles\Apps**.
- Adjust the content of the on-boarding web page found in **Desktop\SEA_ExerciseFiles\Chapter 3** to include additional training based on your recommendations.
- Create a Lunar Water site on the new Tomcat instance and place the on-boarding site in the **onboarding** subfolder.

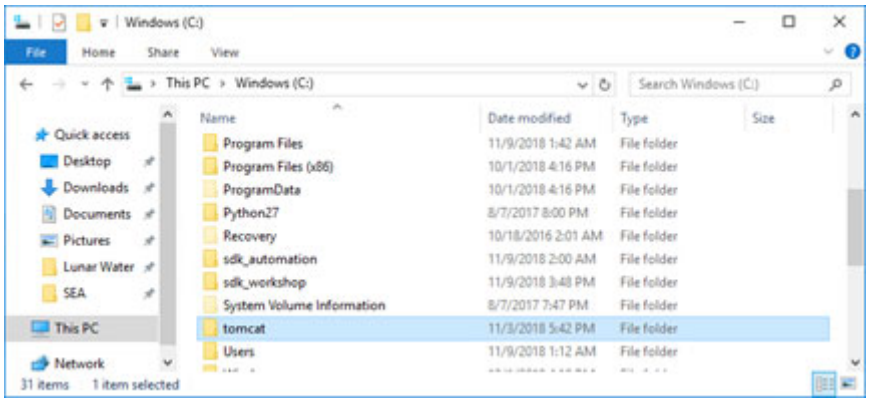
Use the following detailed steps to accomplish these tasks.

Install the staging environment

1. In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Apps.**
2. Right-click **apache-tomcat-8.5.35-windows-x64.zip** and select **Extract All**.
3. Click **Extract**. A new window opens and displays the apache-tomcat-8.5.35 folder.
4. Right-click the **apache-tomcat-8.5.35** folder, click **Rename**, and type **tomcat**.

   Make sure the tomcat folder does not contain a recursive apache-tomcat-8.5.35 folder.

5 Move the folder to the root of the **C:** drive as shown below.



Modify the Tomcat port

By default, Tomcat uses port 8080. If we keep that port, we lose our ability to access MicroStrategy Web locally. To avoid that, alter the configuration file for the Lunar Water Tomcat instance to use port 9090 in the following steps.

6 In File Explorer, navigate to **C:\tomcat\conf**.

7 Right-click **server.xml** and click **Edit with Notepad++**.

8 For each **port** and **redirectPort** value, replace each **8** with a **9**. For example, update port="8080" to port="9090".

Do not replace the "8" in "UTF-8".

```
<Server port="9005" shutdown="SHUTDOWN">
  <!-- Security listener. Documentation at /docs/config/listeners.html
  <Listener className="org.apache.catalina.security.SecurityListener" />


    <Connector port="9090" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="9443" URIEncoding="UTF-8" />


    <!-- Define an AJP 1.3 Connector on port 8009 -->
    <Connector port="9009" protocol="AJP/1.3" redirectPort="9443" URIEncoding="UTF-8" />
```

9 **Save** the file.

Set the JRE_HOME variable

To connect the Tomcat server to the Java Runtime Environment, you must configure the JRE location variable.

10 From the task bar, search for **environment variables**.

11 Click **Edit the System Environment Variables**. The System Properties window opens.

12 Click **Environment Variables**. The Environment Variables window opens.

13 In the System Variables area, click **JRE_HOME** and click **Edit**.

14 In the **Variable Value** box, type **C:\Program Files\Java\jdk1.8.0_172** and click **OK**.

15 Click **OK**, and then click **OK**.

Start the Tomcat server

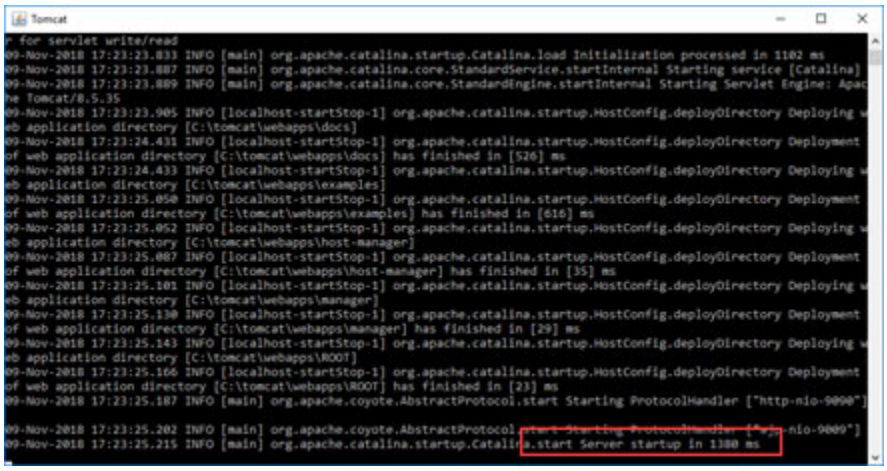
16 From the taskbar, search for **cmd**.

17 Right-click **Command Prompt** and select **Run as Administrator**.

18 Type **cd C:\tomcat\bin** and press **Enter**.

19 Type **startup.bat** and press **Enter**.

A tomcat window opens and displays **Server startup in xxxx ms** to confirm that the Tomcat server is running.



20 Minimize both windows.

21 Open a browser session and navigate to:

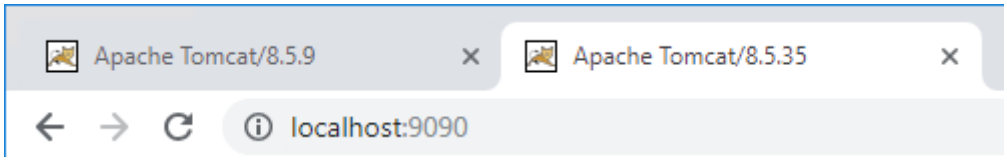
http://localhost:8080

Your main instance of Tomcat that runs MicroStrategy Web is displayed. This instance is referred to as the production Tomcat server.

22 Open a new tab in the browser and navigate to:

http://localhost:9090

This time, the Tomcat instance running from C:\tomcat is displayed.



Congratulations! Your staging environment for Lunar Water is up and running.

Modify the on-boarding document

1 In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 3**.

2 Right-click **index.html** and click **Edit with Notepad++**.

3 Browse the web for 3 training sites that would be valuable to developers.

4 Edit the three **Architect Selection** links and replace them with links to your desired training resources.

```
<ol>
    <li><a href="#" target="_blank">Architect selection 1</a> </li>
    <li><a href="#" target="_blank">Architect selection 2</a> </li>
    <li><a href="#" target="_blank">Architect selection 3</a> </li>
</ol>
```

5 **Save** the file.

6 Which sites did you add? Why?

Deploy the site in the staging environment

1 Copy **index.html** and **LunarWater3.png**.

2 Navigate to **C:\tomcat\webapps\ROOT**.

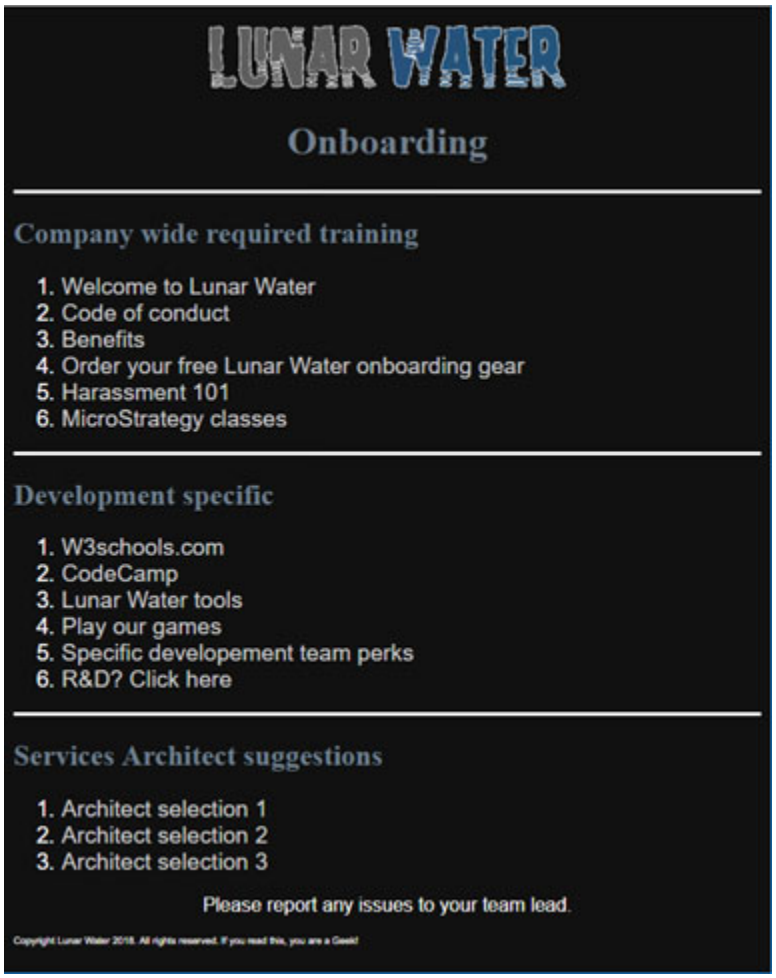
3 Create a new folder named **LunarWater**.

4 Paste the two files in **LunarWater**.

5 Open a browser and navigate to:

http://localhost:9090/LunarWater/

The site opens displays your personalized links



Procuring tools to manage projects

Development tools streamline common software engineering processes and reduce the amount of time required to perform administrative tasks. To ensure that processes are repeatable across teams and knowledge can be easily transferred across organization, procure development tools and create a mandate to implement them across all teams.

Tracking assigned tasks

An issue tracking tool helps managers and leads map their projects, set goals, and track completed and outstanding work. This enables project managers to analyze work performance and understand the path to project completion. In your organization, you might mandate that all teams use JIRA, Mantis, Team Foundation Server, or some other tool to track software development projects.

Managing projects

Project management tools help you create project timelines, designate milestones, manage resources, maintain timesheets, and son on. You can use these tools to analyze performance for a single team and compare project completion status across the organization. To create continuity and analyze project performance across the organization, specify a single tool like Microsoft Project, Trello, or LiquidPlanner for project management.

Object modeling and diagramming use cases

Object modeling tools help developers define the object model, diagram a use case, or abstract a requirement into a usable document, structure, or image. Standardize object modeling and use case diagramming tools across the enterprise to ensure reliable and consistent software products that are easily maintainable. Common tools in this category include Microsoft Visio, Visual Studio, and so on.

Establishing development tools

To accomplish their development tasks, your team members need software development tools. As the Services Architect, your goal is to curate a set of development tools that are employed across the organization, and ensure that employees are properly trained to use them. Create an approval process for tools to ensure compatibility across projects. Common tools for various tasks are outlined in the following sections.

Best Practice

Establish and maintain a list of approved tool versions to ensure that development practices are consistent across teams.

Designating web development tools

Developers require web application development tools to customize the following MicroStrategy applications:

- MicroStrategy Web
- MicroStrategy Library
- MicroStrategy REST server

Testing for scalability

Most web applications must be scalable to handle multiple concurrent users. To ensure that a web app can handle its predicted load, developers use a stress testing suit to push the application beyond its limits. Common tools for stress testing include JMeter, Selenium, and others.

Monitoring the MicroStrategy platform

MicroStrategy Platform Analytics includes a series of dossiers and reporting objects to help you monitor and improve your MicroStrategy environment.

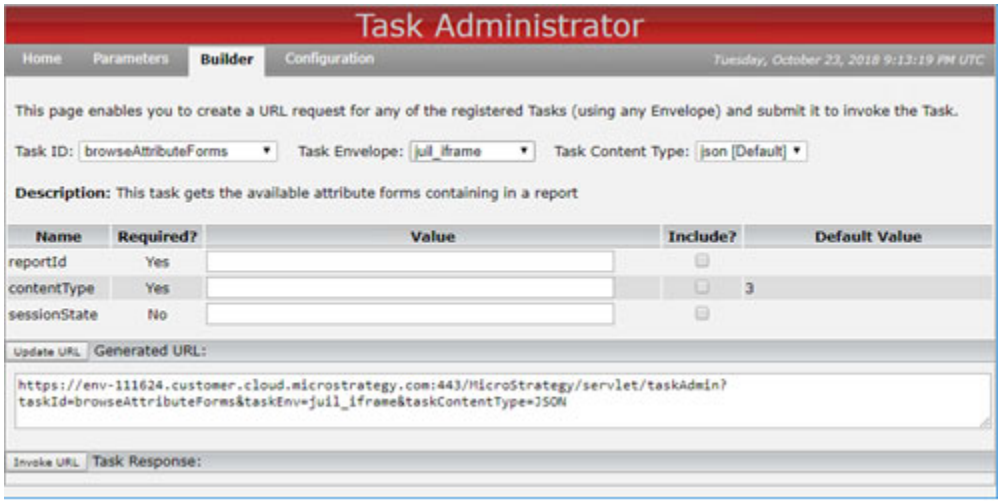
Best Practice

From a development perspective, Platform Analytics can be leveraged as a data source to monitor the performance of your analytics applications. For example, you can use Platform Analytics to monitor traffic in MicroStrategy Web or understand mobile app utilization patterns.

Invoking MicroStrategy functionality through URLs

Your organization might create custom applications that invoke MicroStrategy functionality. One way to do this is to leverage the MicroStrategy Web URL API, for example, when embedding a dossier or a report. Developers basic administrative

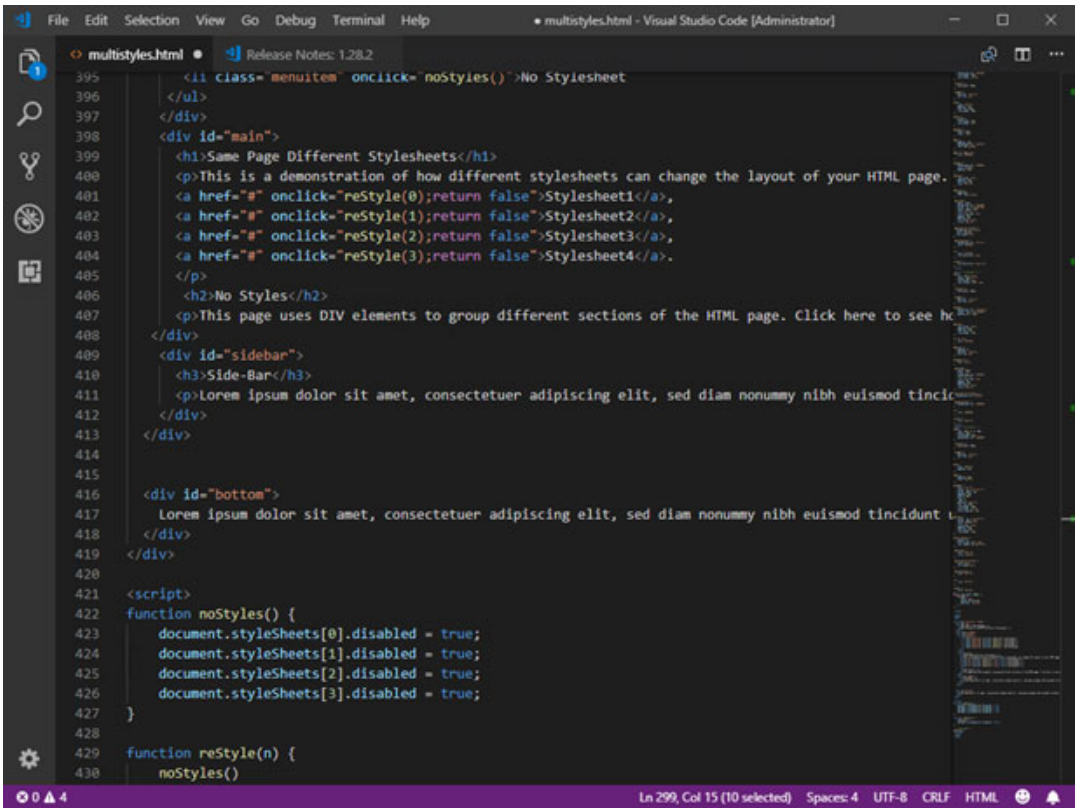
privileges to the platform to use the Task Administrator page to generate and test URLs.



Coding for the web in an IDE

A good Integrated Development Environment (IDE) simplifies the process of developing for the web through a collection of features such as debugging tools, version control, code completion, and syntax highlighting. The following are additional capabilities you should look for in an IDE:

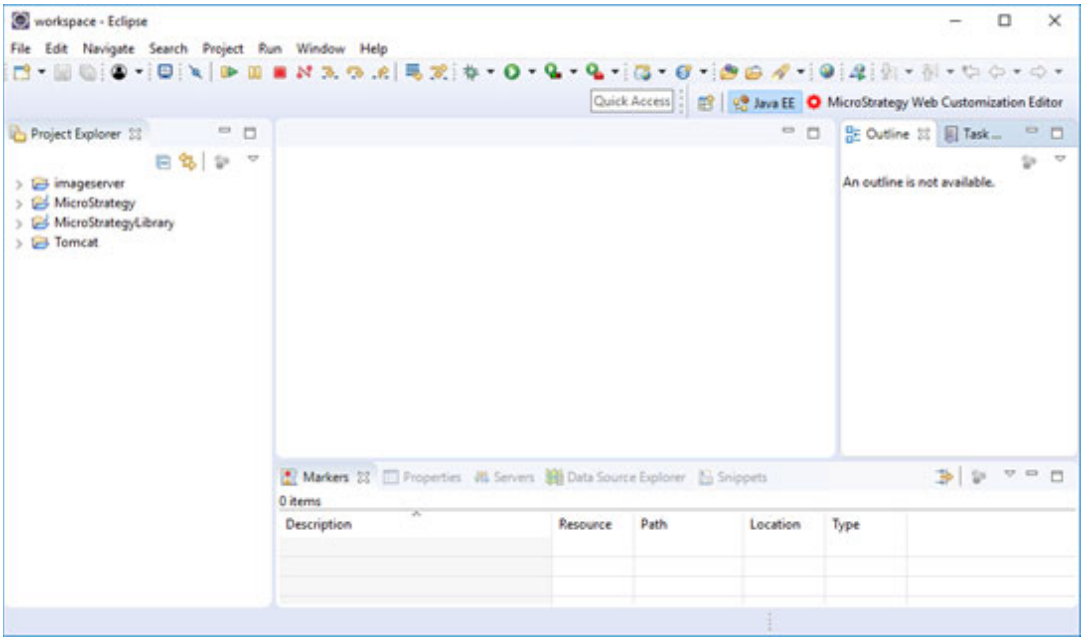
- Extensibility, flexibility, and ease of use
- Shortcuts for productivity
- Integrated debugging tools
- HTML, CSS, and JavaScript support
- Auto completion and Intellisense



Common tools used to develop JavaScript code include Komodo Edit, Notepad++, Eclipse, Visual Studio, Sublime Text, and others.

Developing applications using Java

To develop applications in Java, a robust IDE with debugging capabilities is required.



Options for an IDE include Eclipse, Intellij IDEA, Net Beans, and so on.

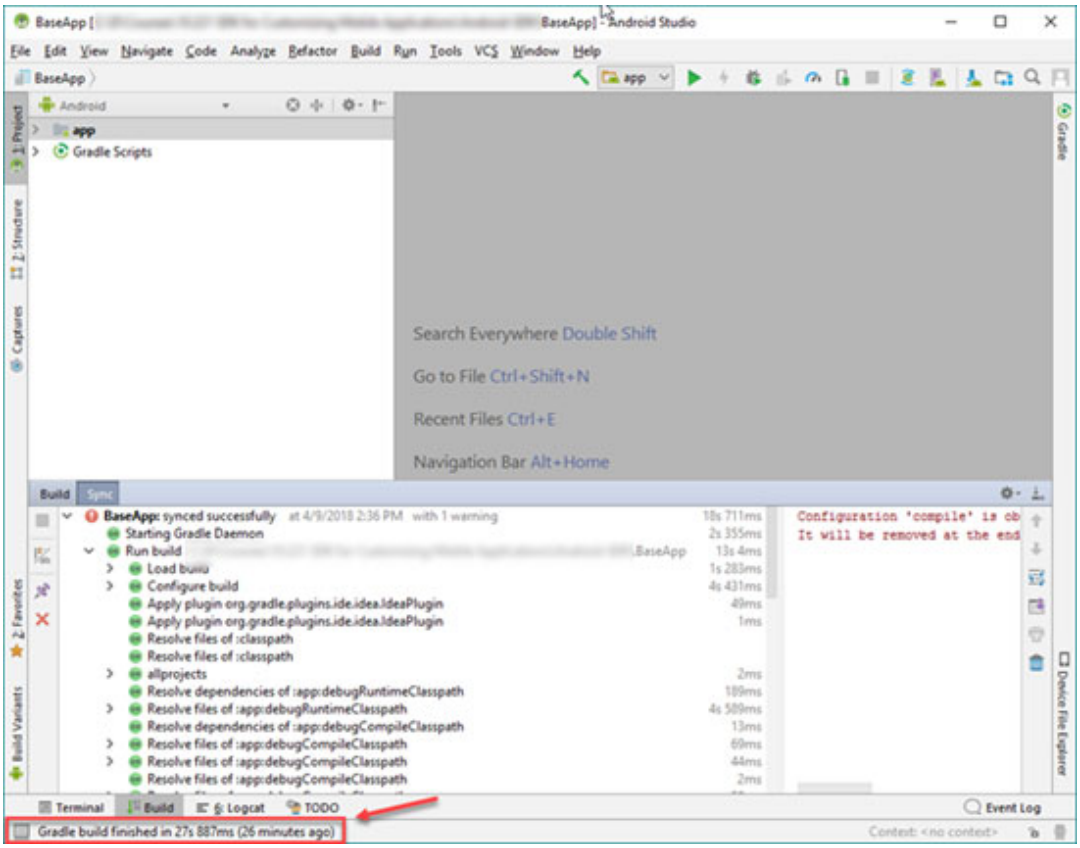
Best Practice

Java development involving MicroStrategy plugins should be performed in Eclipse using the Web Customization Editor.

Developing mobile Android applications

Developing for mobile devices brings its own set of challenges. The proliferation of operating systems, devices, and form factors makes it difficult to write an app

for all audiences. You can use software emulators to test on multiple devices and OS versions using Android Studio.



Testing on physical hardware

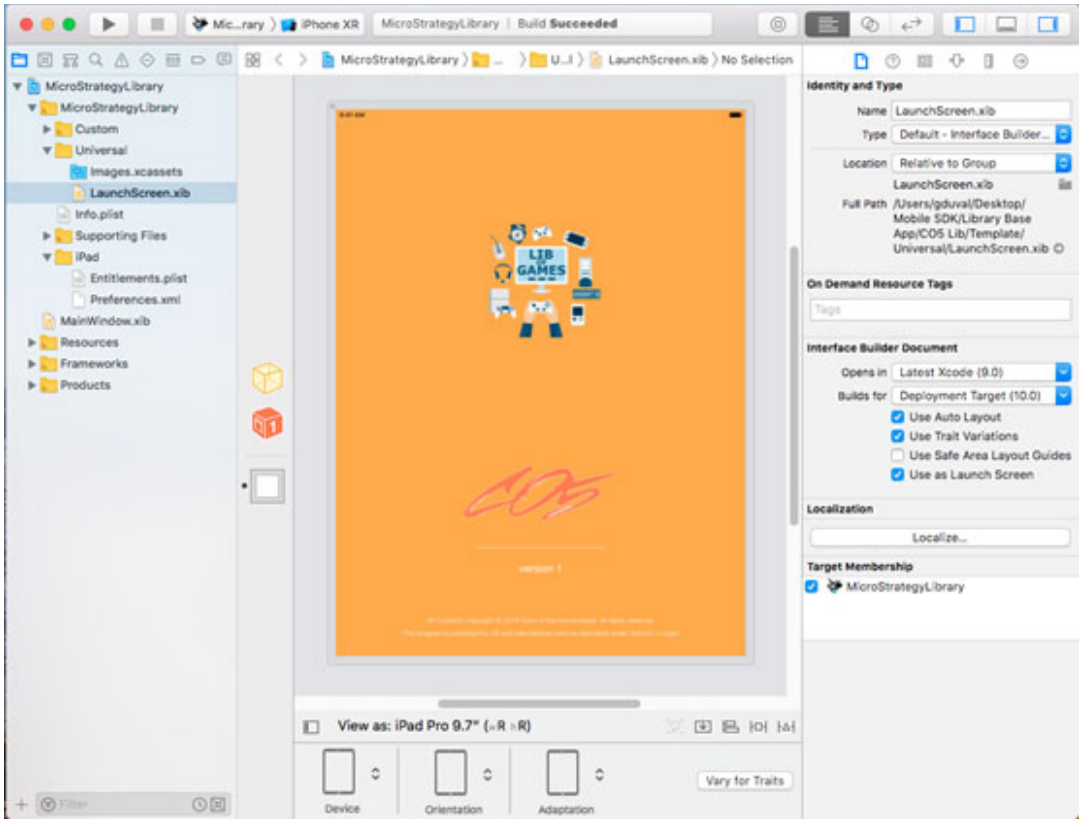
Software development requires testing on actual devices to confirm responsive design. Using physical devices helps you design your applications for all form factors. Ensure that testing is performed on a variety of devices and assign a variety of devices across the organization.

Best Practice

To encourage testing on a variety of devices, you can establish a broad device pool that allows developers to sign out devices from the IT department or a manager.

Developing mobile applications for Apple devices

Xcode is developed and maintained by Apple to help developers create native applications for iOS devices. Designate a specific version of Xcode to use throughout the enterprise to ensure compatibility.



Developing MicroStrategy applications

To develop customized MicroStrategy applications, your team might require a certain number of MicroStrategy licenses for products like MicroStrategy Workstation, MicroStrategy Developer, the Web Customization Editor, MicroStrategy Web, MicroStrategy Library, and so on.

Best Practice

For temporary MicroStrategy assignments, you can quickly deploy a MicroStrategy on Cloud environment that can be terminated when access is no longer required.

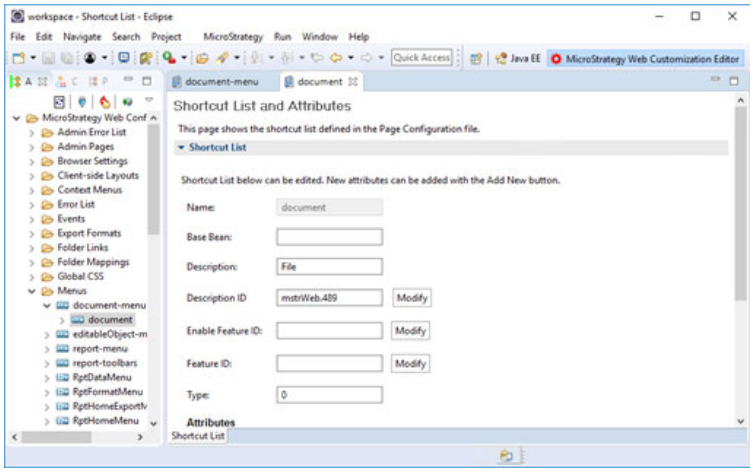
Distributing supporting tools

Every developer's toolbox contains various supporting tools that simplify code creation and management. As the Services Architect, you must be familiar with industry trends so that you can curate and distribute tools that foster an efficient and reliable development process.

Developing applications in Eclipse

Eclipse is a versatile IDE with myriad plugins and extensions that can support various development tasks. Distribute Eclipse as part of the MicroStrategy customization toolkit.

Customizing MicroStrategy applications



Best Practice

The MicroStrategy Web Customization Editor (WCE) is an Eclipse plugin that simplifies the task of creating plugins for MicroStrategy Web, MicroStrategy Mobile, and MicroStrategy Library. To safely compartmentalize your MicroStrategy customizations, establish a standard to create plugins in the WCE.

Select and distribute plugins

Eclipse can be updated with other plugins to simplify various development tasks. In your organization, designate specific plugins for installation by developers. You can mandate that certain plugins are installed during the software installation phase of the on-boarding process.

To obtain input from developers with hands-on experience create a verification process to test and deploy new or updated plugins on developer machines. By creating a standardized process, you can exhibit control over plugin utilization and ensure that development workflows are executed efficiently.

The following table provides a sample of useful Eclipse plugins.

Product name	Product name
CheckStyle	Scala IDE
CodeMix	SonarLint
Darkest Dark Theme	SpotBugs
DevStyle	Spring Tool
Eclipse Color Theme	Subclipse
JRebel	Subversive
JSweet	TestNG
Kotlin	UML Designer
M2Eclipse	Unnecessary Code Detector
Nodeclipse	Webclipse
PyDev	YEdit

Comparing file contents

Developers often need to quickly identify differences in file versions. For example, a Tomcat configuration file, a plugin definition, or an HTML page might contain minute changes between versions. Manually comparing files is a tedious and error-prone task that can be alleviated with a file comparison tool.

To establish an efficient and error-proof file comparison process for your development teams, identify a stand-alone or a plugin file comparison application that can be installed as part of the on-boarding process.

Editing code and configuration files

Developers regularly need to edit code, configuration data, and log files. Based on the coding languages and platforms used in your organization, identify a text editor such as Brackets, Notepad++, and Sublime Text. Limit the text editors used in the organization so that formatting and file manipulation capabilities are shared among all development teams.

Analyzing and improving code quality

Code profilers scan your code to improve performance, determine function call frequency, and identify complex problems like memory leaks. There are three types of code profilers:

- **Method/Line level**: High overhead profiler that scans individual lines of code to identify complex issues.
- **Transaction tracing**: Hybrid profile that tracks transactions like statement and error logging, web services, and SQL queries.
- **Application Performance Management (APM)**: Low overhead profilers usually installed on servers to find performance issues.

Best Practice

Common tools in this category include App Insights, NProfiler, Visual Studio Profiler, and so on. You might offer this tool to a selection of your team members that focus on performance and complex application analysis. It may not be necessary to secure licenses for every team member.

Identifying clients for server applications

Applications on corporate servers can be accessed through various clients. For example, your source code management software might have a dedicated desktop application, an IDE plugin, or a command line option that can be used as clients.

For every server-side application that needs a desktop client, specify an approved client application, and make sure that developers install it. Work with the System Administrator in your organization to establish requirements for client applications, explore potential conflicts, and define an effective solution.

Distributing to app stores

If your enterprise publishes mobile applications, developers likely interact with the following application stores:

- App Store (iOS)
- Google Play (Android)
- Samsung App store (Android)

If your business also publishes Windows or Linux applications, you might need to interact with other types of stores:

- Microsoft Store (Windows)
- Steam (Windows, Linux)
- GOG (Windows)
- Epic Store (Windows)

To interact with these stores, establish a process to conform to app store requirements and publish your applications to the public. For example, your process should include administrative tasks like preparing screen shots and videos, managing app store accounts, leveraging app analytics, and testing standards.

Publishing on the Apple App store

Best Practice

- To avoid conflicts and keep ownership of all development efforts, allocate a corporate ID that can be used to create an AppleID for the store.
- To deploy internal applications to your employees use the Apple Developer Enterprise Program.
- Allocate additional individual Apple development licenses for research and development purposes to delineate these efforts from your production applications.
- Use Test Flight to enroll beta testers for your applications.

Publishing on the Google Play store

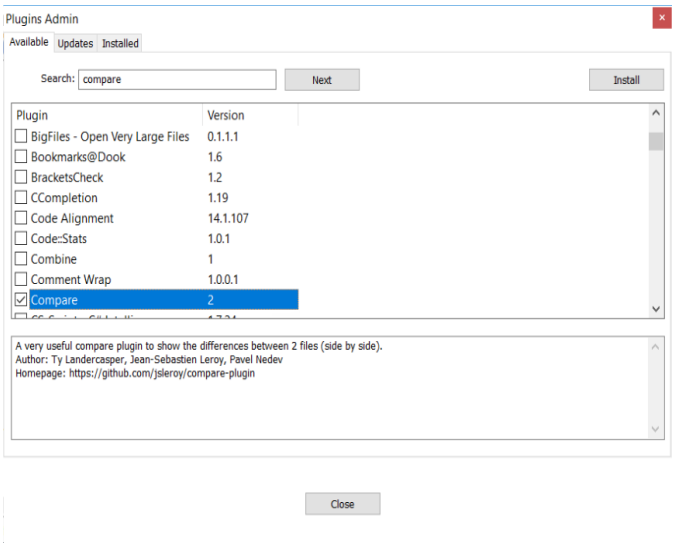
- To avoid conflicts and maintain ownership of all development efforts, allocate a Gmail account for the Play store.
- Create additional Gmail accounts to keep research applications separate from production work.
- Enroll people in limited beta testing when needed.

Exercise 3.4: Update the developer environment

You have established a standard developer environment at Lunar Water and you want to update it with new plugins and configuration settings. In this exercise, add plugins for Notepad++ and configure the Web Customization Editor.

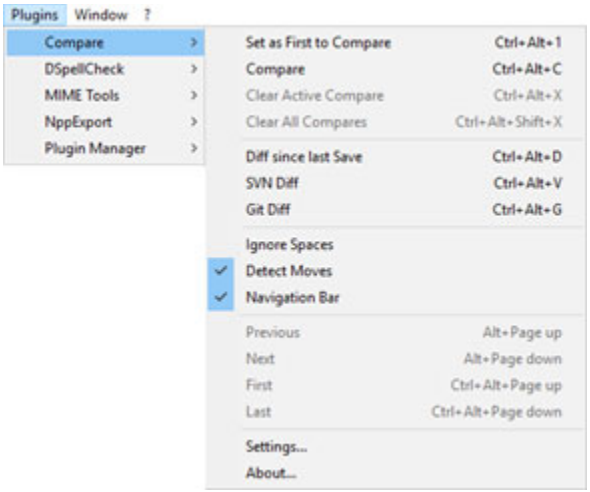
Add plugins to Notepad++

1. Close **Notepad++**, and then re-launch it.
2. Do one of the following:
   - If an upgrade prompt is displayed, click **Yes** and follow the instructions to complete the upgrade.
   - If you are not prompted, open the Help menu (**?**), and select **Update Notepad++** and follow the instructions to complete the upgrade.
   - If you already have already upgraded, skip to the next step.
3. From the **Plugins** menu, click **Plugins Admin**.
4. Click the **Updates** tab. If there are any plugins to install, click **Update**.
5. Click the **Available** tab.
6. In the **Search** box, type **compare**.
7. Click the **Compare** check box, as displayed in the following image.



8 Click **Install** and follow the instructions to install the plugin. A new toolbar section appears as shown below.

A new Compare menu is also added to the Plugins menu.

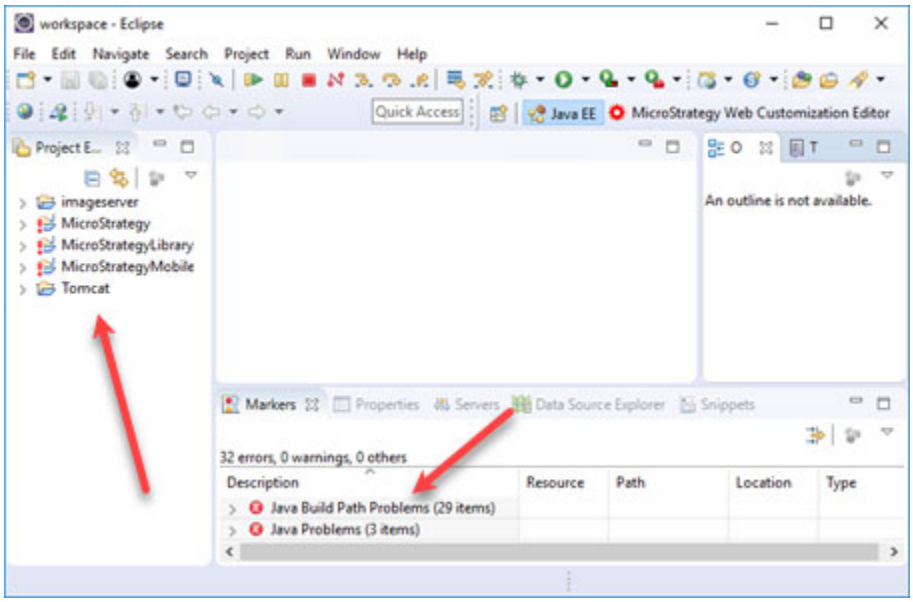


Verify the Web Customization Editor configuration settings

The Web Customization Editor is already installed on your MicroStrategy on Cloud environment. In this part of the exercise, review the configuration settings to ensure they point to the correct environment and plugin folder.

1 Right-click **Eclipse** and select **Run as Administrator**.

Eclipse opens.

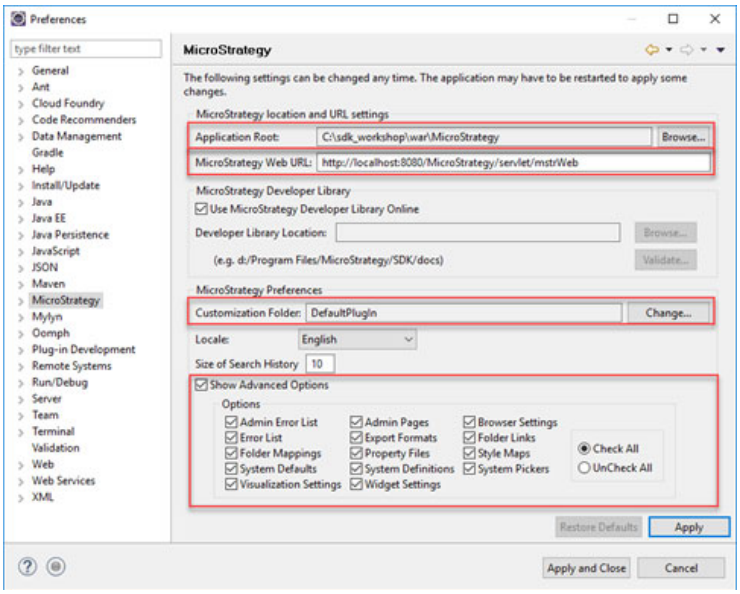


2 If you see errors in the Project Explorer pane, right-click each project and click **Refresh**.

3 Click **MicroStrategy Web Customization Editor**. The plugin takes a moment to open.

4 From the **Window** menu, click **Preferences**

5 In the left pane, click **MicroStrategy**.



6 Make sure the following settings are applied:

   a In the **Application Root** box, enter **C:\sdk_workshop\war\MicroStrategy**.

   b In the **MicroStrategy Web URL** box, type **http://localhost:8080/MicroStrategy/servlet/mstrWeb**.

   c In the Customization Folder box, enter **DefaultPlugin**

   d Click **Show Advanced Options**, and then click **Check All**.

7 Click **Apply and Close**, and then close Eclipse.

Congratulations! You updated the developer environment configuration. In your own organization, you would likely use an imaging system to create an image of this environment that can be used to create an environment for each new employee.

Creating system images to deploy identical machines

System images can save time and enforce software compliance rules by duplicating a standard environment for all developers. As you update your software and add new applications or plugins, update your image to ensure that new team members have access to the same tools as your established teams.

Best Practice

Collaborate with your organization's System Administrator to create and maintain an image of a developer machine. You can also create variations of the images for specialized roles on the team, such as a Technical Writer image that contains document management software.

Summary

An Intelligent Enterprise requires optimized teams who employ functional development environment with the required tools to complete their work. In this chapter, you learned about the skills, experience, and hierarchy required to establish successful development teams.

As you hire developers, create an inventory of their skills to efficiently manage your resources and project assignments. You can also leverage project management tools to track and prioritize your projects and meet your goals.

Each organization requires its own set of tools that enable developers to efficiently and predictably produce and maintain applications. An imaging system can help you create uniform software installations for developers.

4

Managing the Application Life Cycle

So far, we've considered the development environment, including the people, their tools, and best practices to achieve optimal team performance. In this chapter, we turn to the development process. We look at the life cycle of applications and how to ensure all enterprise development work remains efficient, consistent, repeatable, and remains usable across the organization.

Standardizing the application development life cycle

The Services Architect in an Intelligent Enterprise standardizes and organizes all development efforts across the enterprise. Effective oversight requires a formal process that acknowledges and documents the existence of applications, components, and coding projects in the enterprise. This approach establishes consistency and reliable repeatability across the enterprise for all software development projects, reducing errors, inefficiencies, user problems, poor user perception, and a host of other avoidable issues.

This process also brings stability and support to the development projects themselves. Providing consistency and clear communication to the development of applications, components, plug-ins, and other development tasks ensures coding efforts turn into effective results that benefit the whole company. When

the developers' time and energy is used effectively, their skills are focused on common, well understood goals.

An enterprise that has not achieved Intelligent Enterprise status, is an enterprise where the status quo is uncommunicative development teams doing their own ad hoc thing without input from colleagues. This is exactly what happened to Lunar Water when they started. But now they have you as their Services Architect, resolute in addressing the situation.

When the Services Architect establishes the components of an Intelligent Enterprise, repeatable, professional standards for all development efforts exist and are leveraged by all.

Work within the development methodology (waterfall, agile, etc.) that your organization has chosen, and establish your quality processes within that larger context.

Documenting software development processes

Development documentation is typically a collection of project-specific guidelines, corporate rules (such as for security or localization), and best practices related to a specific domain. For example:

- Requirements document
- Design specification
- Coding Style Guide
- Code review guidelines
- Build management document
- UI design guidelines
- Branding guidelines
- Release notes

The Intelligent Enterprise is easiest to achieve and maintain when templates are created for each of the required documents. Each template should include standards that apply to all development projects; for example, security policies, localization support, re-use of code, and other common design considerations should be considered for every development project design, and thus should be part of a design specification template.

Having all project documentation based on up-to-date, maintained templates ensures that the required development components are not forgotten, are addressed consistently across the enterprise, and include best practices that all teams are be able to reliably follow through time and across development teams.

On a high level, development documentation can be organized into enterprise-wide and project-specific purposes:

- Enterprise-wide, administrative management content in which the Services Architect establishes standards, guidelines, process, and best practices that are to be followed to achieve a Services team that is efficient, effective, follows corporate standards, and adheres to corporate security policies.
- Project-specific internal team development content meant to be shared among a development team as projects grow.

The enterprise-wide type of documentation content, is primarily what concerns a System Architect establishing an Intelligent Enterprise. This level of documentation is covered in detail here, and resurfaces throughout this class, because maintained documentation is one of the keys to achieve consistency, maintainability, and repeatability in software development.

We discuss the project-specific type of documentation briefly at the end of this section, because communication consistency within a development team on a specific project is a vital component of an enterprise operating intelligently.

Best Practice

Any standards documentation must identify what version of the software it covers. For example, in a Coding Style Guide, for mobile development, you would require coding with the latest MicroStrategy Mobile SDK app.

Best Practice

Document production should be done in parallel with the development time line. When a product is shipped, the accompanying documentation and other related documents (like release notes, training material, and tech briefings) should be fully up to date and able to also be shipped or made available.

Template items may include:

- List of initial conditions for tasks, for example a list of servers to back up
- Resources needed to complete the task, for example, a backup program, the script, and the destination for the backup
- Order of “to do” to complete the task, including each subtask, where appropriate
- In-progress checkpoints (milestones) to see whether the development process is on track

- Potential pitfalls and solutions, typically added to or updated at the end of each project's release
- Description of final conditions to consider tasks complete

Best Practice

Ensure that your documentation storage location primarily offers the latest version of information. Older documentation is often still valuable and it has its place, but should be kept in an archive structure you can reference if people need details on previous content.

Exercise 4.1: Integrate development team documents with Documentation team work

At Lunar Water, the Documentation team was previously isolated from the Development department. As a result, it is difficult to find product documentation on the internal portal.

As the Services Architect, you want to ensure Lunar Water's Documentation team has full access to the development team project documentation, and vice-versa. In this exercise, create a transparent workspace in the Lunar Water portal and web site to include documentation.

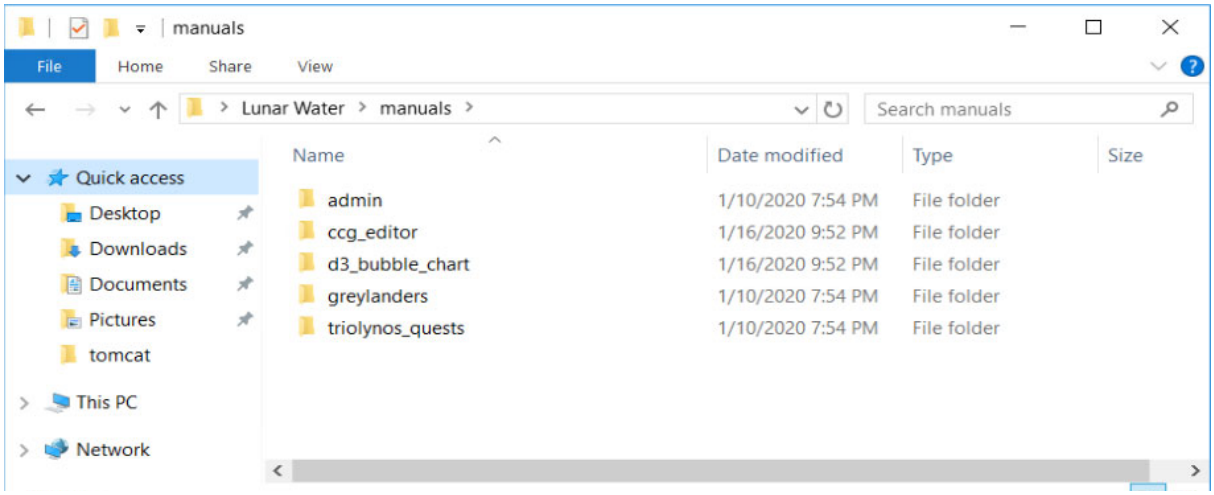
Create a documentation repository

1 In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 4**.

2 Copy the **manuals** folder.

3 Navigate to **Desktop\Lunar Water**. Paste the folder.

4 Double-click **manuals**. The following folders are displayed.

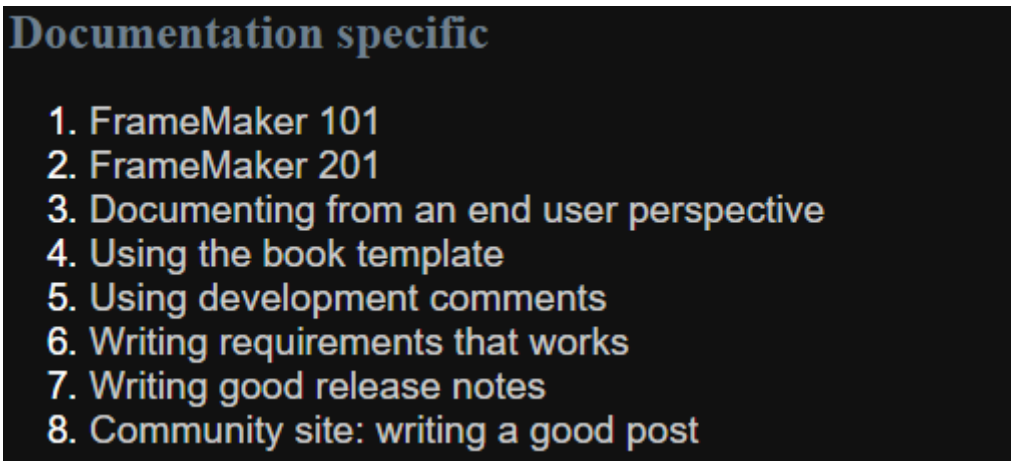


This is the folder structure for the Life Water documentation. Review the contents of the folders. Each version has its own folder, a final pdf, and a source folder where the FrameMaker files for the manual are stored. There is also a graphics folder for the images used in each manual.

The Documentation team now has a repository to save their work as they work alongside your development to create application help documents.

Adjust on-boarding

1. In File Explorer, open **C:\tomcat\webapps\ROOT\Lunar Water**.
2. Right-click **index.html** and select **Edit in Notepad++**.
3. Add a new section for the Documentation team. Add two entries in the section to include Documentation training, as in the following example.



4. **Save** the file. Documentation materials are now integrated with the rest of your site.

Creating project-specific development documentation

Development team members typically create these process-focused types of documents on how things are done. These documents should be stored with other documents for a given project, so that others can later refer to these processes for any maintenance or troubleshooting. Examples of process documents can include deploying a patch, upgrading a server, using a code library, merging two branches of code, or anything that needs a series of steps in the daily life of your team.

Project-specific documentation typically includes the following:

- **Tools and libraries**: Whether built for internal or external use, all tools built should be documented, with attention to the ultimate audience. Documentation is an intrinsic part of any SDK, and should describe the libraries, tools, and utilities that are part of the SDK.

- **Products**: Document all related products that a user can access. For example, for Lunar Water, this means the games released to the public. Even an internal product is considered a company product and receives some documentation, like the lootbox generator used at Lunar Water.
- **APIs and plug-ins**: APIs and plug-ins used within your organization or publicly should be documented to ensure the potential in those libraries and plug-ins is fully leveraged.
- **Tech notes**: Ensure that the organization's support team gets the information they need from development teams, to communicate accurate and appropriate information for users seeking tech note support.

To create and maintain vital development documentation, you need a software platform that lets your teams share content and knowledge. The best software development documentation platforms behave a little like a wiki, a collaborative site where users work together over time to build content. In this type of platform, a design document, for example, can evolve between members of a team as a project progresses. Common solutions you can use for effective project collaborative documentation include DocuWiki, IBM Connections, MediaWiki, and others.

Promoting collaborative communication

Best Practice

Communication is a key element of successful teams and an Intelligent Enterprise. You should put in place a communication platform that allow users to talk to each other, have group conversations around specific topics, or forums. The goal is to select a platform and implement it for all team members. But be selective; having too many ways to communicate becomes an obstacle rather than a facilitator. Common communication products include Yammer, Slack, Microsoft Teams, and others.

One of the most effective ways to communicate standards, progress on development projects, and everything in between across teams, is to develop and use a communication platform that contains all the pertinent documentation related to all your enterprise development projects. We create a portal later as a beginning to documenting and communicating our Intelligent Enterprise standards and best practices. We'll return to the vital importance of communication and this communication portal, in a later chapter.

Exercise 4.2: Prepare an internal portal

Lunar Water has multiple development teams that do not regularly communicate, and often duplicate work. This redundancy wastes valuable company resources and inhibits growth.

To establish transparent and effective communication across development teams and with the stakeholder community at Lunar Water, prepare a page for the internal portal to store team projects and document progress.

Create the page

1 In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 4**.

2 Copy **currentprojects.html.**

3 Paste the file in **C:\tomcat\webapps\ROOT\LunarWater**.

4 Open a browser and navigate to:

http://localhost:9090/LunarWater/currentprojects.html

Add new contact to currentProjects.html

1 Right-click **currentprojects.html** and select **Edit in Notepad++**.

2. Add a row to the **CCG** table and add the following information:
   - Name: **Lootbox Generator**
   - Published version: **1.1a**
   - Version **1.1c** is in design
   - Version **1.1b** is in dev
   - Version **1.1b** is in test
   - Assign **Dan B** as the project lead.
3. Assign Design, Dev, and Test teams to the new project, while considering the following:
   - A team should not have more than 2 design projects
   - A team should not have more than 1 dev project
   - A team should not have more than 2 QA projects
   - You have 4 dev teams and three QA teams
4. **Save** your changes and preview the page.

Standardizing the application life cycle

There are core processes your development teams perform for every development project. The length and form of the core process may vary from one methodology to the next, but the life cycle is present, always. We'll review these core development processes from the perspective of making them consistent, repeatably efficient, and making sure they all include corporate standards to ensure an Intelligent Enterprise.

Typical application life cycle processes include tasks in the following areas:

- Requirements-gathering
- Designing the product or component
- Coding the required features
- Testing processes
- Deploying the product
- Monitoring and maintaining the deployed product

We'll briefly address these project development areas in this chapter, with more emphasis on the coding phase in the next chapter, all with an Intelligent Enterprise in mind.

Before we discuss these phases of development, we touch briefly on risk management, which must be included in decisions and reflected in documentation and communication throughout the project development process.

Managing risk

As part of an Intelligent Enterprise, the Services Architect must perform risk management on every project. Every development project comes with inherent risks, and additional risks arise during the lifetime of the project. You must assess these risks, and then mitigate and manage them to alleviate the burden they bring to your team and to users who depend on delivery time lines, etc.

All of the following elements that apply to your organization belong in any development process template you create. For example, if industry regulatory requirements are part of your organization's agenda, that line item belongs in any development template where this consideration impacts development.

Delivery deadlines

The biggest risk is time. Will you have enough of it to complete the requested objectives of a coding effort? Make certain to add to your planning a buffer for inevitable slips and challenges to coding features that impact time lines. For a Services Architect new to an organization, the most accurate way to assess this type of risk is to capture time slips from the most recent release of a similar type and size of product, and add the same numbers as estimates of extra time that is needed to the project you are currently planning.

Best Practice

Over time, after enough completed projects, your captured data can be analyzed in a dossier to determine increasingly accurate estimated risks to deadlines based on project type, size, length of time, number of resources, and other data points.

Prioritizing feature work

Deciding what should go in a product or component can be considered a risk. It is important to decide to include those features or components that lead to the most user success and satisfaction. Both short-term and long-term user success must be considered. There must be an appropriate corporate balance between

reacting to existing product enhancements and remaining innovative based on corporate vision.

This type of decision-making is beyond the scope of this training, but a well-maintained product road map and regular communication with CXXs represent the primary contributors to mitigating risks due to mistaken priorities.

Regulatory and contract compliance

External entities may by law, by contract, or by other agreement impose a need to comply to regulations, to a process, or some other requirements. If known in advance, any risks can be managed much more effectively. In another common example, a newly designed mobile application that you customized and want to submit to the App Store may be delayed due to a new requirement for app submission, forcing a redesign of a part of the application.

Ensure that your team processes include appropriate research at an early date concerning compliance.

Stakeholder expectations

Stakeholders may sometimes request or require changes far into the development life cycle. Safeguards to this type of risk can include using a development methodology that allows short development cycles and stakeholder review can occur earlier while there is less time wasted redesigning; or including stakeholders in design review.

However, it is also important to have stakeholders agree to the project time line so that everyone is invested in completing the project and redesign does not become the norm.

Best Practice

Best practices for risk management

- Plan for the unknown: This includes risks in all aspects of a project. For example, if your team is designing a visualization plug-in, allow time for adjusting the behavior based on updates for a new version of the MicroStrategy platform, and new versions of the supporting libraries like D3.js.
- Respond to risk with honesty: Acknowledge any risk and put in place corrective measures, even if they can only be benefited from in the next project. For example, you might document enhancements to your testing process and design template if a security flaw is found after deployment, or an extra round of testing if your component is incompatible with a component installed on a client's machine and you could not anticipate this as a factor.

Gathering requirements

All development projects start with defining the requirements for a product, whether a full application, a plug-in, or a library. On a simple level, requirements list what the software should do.

The requirements list comes from the end users and stakeholders when launching new project ideas. Requirements can also come from explicit R&D activities within the enterprise. We explore R&D team standards briefly at the end of this section, as this source of new requirements is vital to the company moving forward, not just fixing existing products.

Additional requirements can come from enhancement issues in the teams' bug tracking system, whether from end users or from within the development team itself.

Establish requirements-gathering standards for all development teams, to be used for all development projects, to ensure that they are user focused, answer the stated need, are maintainable, are documented, etc.

Use the following high-level tasks as starting points for your Requirements template, to focus on establishing a standard approach that results in a consistent and repeatable requirements-gathering process across the enterprise:

- **Categorize requests**: To ensure solutions meet user needs, organize requirements into:
  - Business requirements (the what)
  - Technical requirements (the how)
  - Road map requirements (the what and how)

  For example, a plug-in you are requested to build for a department may have the following needs:
  - Locate the least performing store or agent within a branch or location (business requirement)
  - Display key performance indicators using a specific format (technical requirement)
  - The data used to calculate the KPIs are read in real time with a REST API call (technical requirement)
- **Rank requested features**: Prioritize requirements, in communication with the stakeholders. A variety of techniques exist, for example, the MoSCoW method:

  - Must have
  - Should have
  - Could have
  - Won't have

- **Validate the scope of requirements**: Team leads typically scope each requirement. Ensure that your requirements document template gathers scope decisions and requires validating the estimates to allocate the proper resources and set appropriate time estimates for projects.

  Make sure the template also calls for these common Requirements template items:

  - Value of feature and number of user impacted
  - Value of feature and number of key users impacted
  - Estimated time to implement
  - Impact of feature on other features and existing embedded solutions
  - Risk and opportunity costs

Best Practice

Deciphering client wants into real needs

When confronted to requirements gathering, your client can be another department withing the organization or an external client. Either way, position yourself as a consultant visiting a client in their organization. This way, you do not bring baggage to the table and can have a fresh eye on the client's situation and needs.

To be successful at gathering the proper requirements, you have sometimes to decipher what is intended from what is said. Finding the root of the pain points and understand fully the end goal is key. So you can recommend a proper solution, often not envisioned by the client, that brings to the client the end goal, even if the path to it differs from their initial request.

Here are some good practices you should put in practice when gathering requirements with a client.

Practice active listening

Listening here is the key word. Your objective is to capture the essence of the situation the client is in, and to make certain you grasp and see the end goal they

pursue. By using active listening techniques, you confirm the facts and sentiments the client send your way. It is up to you to make certain your understanding level is as close to 100% as possible; like you were describing the situation for yourself.

For example, a client might say:

> *I hate to have to enter passwords for every company's systems all day long. This is nonsense. I want to this once in the morning and be done with it.*

To this, you can reply:

> *So, you want to be able to authenticate once in MicroStrategy (single sign-on) and carry your credentials through other systems within your organization and save time while maintaining proper security protocols.*

The reformulation focuses on an end goal while addressing the emotional pain points current practices bring.

See the end goal

Sometimes, the client may bring you a solution they think solves their situation. Although it may sometimes be the right solution, dig deeper to find what the client really wants to accomplish.

For example, if a client says:

> *I need a plug-in to export dossiers to a folder on a schedule to attach them in our portal for the executives every morning.*

You can see the end goal is to put dossiers in front of the executives for their morning meeting. This can be accomplished as they described. However, you can accomplish the same end goal by embedding real time dossiers in the portal and provide the decision makers the appropriate information when needed. The solution is more flexible, easier to maintain, and more forward thinking.

Trust but verify

A claim made by a client about an attempted solution, or a failed implementation of a process does not mean to reject the solution without a second look. There could be many reasons the solution did not pan out:

- Wrongful implementation
- Failed good practices

- Lack of technical expertise
- Adjustments needed in environment not performed

Take a habit to ask follow up questions when an assertion is made about a failed solution that you think is appropriate.

Listen to the pain and frictions

Not every sentences containing emotions should be taken lightly when hearing clients. Such statements communicate a friction with the current situation and represent a request to make that pain point go away.

See in these statements clues where you can be successful in bringing added value to their current implementation.

Ask open-ended questions

In a TV series, the lawyer at a trial plays with the guilty on the stand by asking questions he already know the answers to waiting to catch a mistake or a lie. Most of the time, these are closed questions requiring a simple yes or no answer.

When with clients, you do not have the answers with you, so you want the details and long answers so you can extract information in your investigation of their need and wants. This is why open ended questions are more appropriate. They allow the client to express freely details you might have missed with a series of closed questions. When people express themselves freely, they tend to include unconsciously more information, even if they would find it irrelevant. However, this additional intake of information provides context and sometimes clues to the real end goal they seek.

Assuming is the enemy

Do not be afraid to confirm anything that seems obvious to you or the client. Assumption leads away from facts and this is where efforts are lost.

For example, when discussing MicroStrategy implementation with a client, ask if everyone is using the same version, or how many Intelligence Server are deployed. You can be surprised by the answer and may influence your whole approach to the solution.

See beyond the short term patch

It is sometimes easy to define a quick fix for a client. The client is happy, but your solution may conflict a future release your software, or become a risk later. Also, if you are circumventing a problem that is actually a defect, are helping the company by patching here and there instead of rooting the problem and assist fixing for everyone?

Use functionality first

Before thinking custom solution for every problem, make certain you cannot achieve the same end goal with current or soon to be released functionality.

Remember our example of request for a plug-in earlier? The response using embedded analytics was proper as an existing technology was available without digging into a custom solution that would need much more maintenance and headaches down the road.

Exercise 4.3: Gather information from clients

The Gamer Community department at Lunar Water sent you a request for a new development project. You send a team member to gather their requirements.

In this exercise, analyze the information provided by the client to clarify their statements and create open-ended questions.

Here is an excerpt from the conversation:

> *We need to change the look and feel of the MicroStrategy dossiers embedded in our gamer portal to reflect the game's current branding. The current visual design is not working for us. We cannot present information to our people looking like this! They see the page as a patch job, and it's tarnishing our brand, We need our logo, our fonts, our colors, and no white space anywhere so the dossiers are seamless with the rest of the web page. There are multiple dossiers in the page. We do not want to adjust every dossier manually, as a theme change renders them obsolete instantly.*

1 How would you rephrase the statement without the emotional charge and keep the facts straight?

_____

_____

_____

_____

_____

2 What follow up questions can you think of to remove assumptions or clarify their end goal?

_____

_____

_____

3 Try to convert these closed questions into open ended questions?

a Are all the games using the same theme on the Community portal?

_____

b Do you have plug-ins deployed currently?

c Do you have an existing .css file in each game's page?

R&D teams

Your organization may channel some of the development teams' creative force into research and development. Innovation results from special projects designed to explore new technologies and understand how to leverage them. R&D teams support:

- Investment in the organization's future
- Incorporating new technologies
- Fostering a general atmosphere of discovery and innovation

Best Practice

Best practices for R&D teams

Ensure R&D teams have the following in place to help support the R&D team as part of an Intelligent Enterprise:

- Dedicated team members so they have sufficient time to not just learn about new technologies but also understand implications of any new technology
  - Rotate other development team members into R&D periodically so everyone can contribute to product growth with new ideas, and everyone gets a chance to understand the drivers behind product initiatives
- A dedicated space so tools and hardware are available when needed
- A clear mandate supported by the executive team
- Periodic challenges that are explicitly defined, so the R&D team can be sent down new paths and identify new possibilities for development

Exercise 4.4: Communicate R&D standards

In this exercise, implement Lunar Water corporate standards to make them easily available for anyone who needs to keep up with R&D efforts or is rotated into the team and needs to get up to speed quickly.

Name the team

1. Create a name for the Lunar Water R&D team. For example, you may use something straightforward or something that provides a little inspiration:
   - mstR&D (MicroStrategy)
   - Coding Unchained
2. In **Desktop\Lunar Water**, create the folder for your R&D team.
3. Inside the folder, create three folders with the following names: **Codename1**, **Codename2**, **Codename42**.

Create a new page

1. In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 3**.
2. Duplicate **index.html** and rename it after your R&D team. For example, **CodingUnchained.html** or **CU-team.html**.
3. Copy the page and paste it in the Lunar Water website folder.
4. Edit your new page in Notepad++.
5. List the three code names used for the folders you created.
6. Remove the extra lists.
7. Above the list, insert an inspirational team mission statement.

Guiding the design process

After the requirements are defined, and prioritized and you have decided which features to be worked on, design documents are created. Establish a design

document template to ensure corporate and development standards are included in every organization development project. Development-specific items to add to a design document template can include:

- Template structure reflects an object-oriented approach
- Packaging (SDK for example, using correct namespaces, defining libraries' content)
- Protection against reverse engineering

Best Practice

- Integrated security from the start, to adhere to corporate security standards, and to ensure that security is not retroactively applied after the design phase is completed
- Design is optimized for efficiency
- Code is test-ready
- Correct modeling techniques are used

We expand on design considerations in the next chapter. Here, we focus on the important initial steps of any software design, establishing corporate standards and ensuring clear, maintained documentation exists and is readily available.

Exercise 4.5: Implement design standards

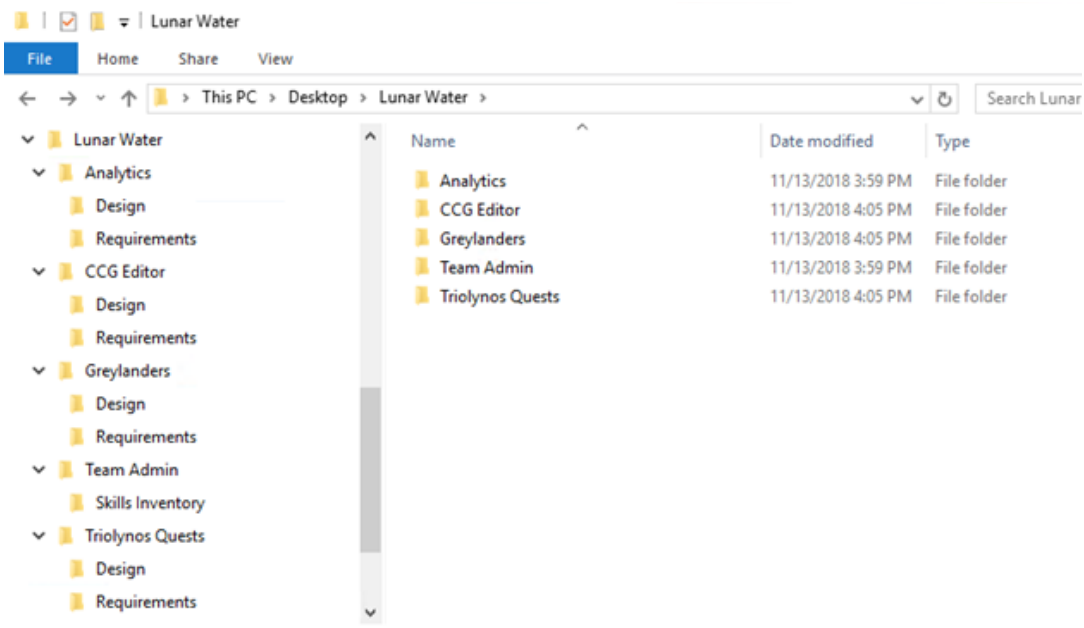
A documentation portal helps you communicate your standards and guidelines to teams across your organization. The portal can be built with a Document Management System like SharePoint, Liferay, or any other DMS installed and supported by the System Administrator.

In this exercise, build a communication and documentation portal structure in the MicroStrategy on Cloud environment.

Create the folder structure

1. In File Explorer navigate to **Desktop\Lunar Water**. This is your portal's base folder.
2. Create three of the following folders:
   - Analytics
   - CCG Editor
   - Greylanders
   - Team Admin
   - Triolynos quests
3. Move **Skills Inventory** inside **Team Admin**.
4. For Analytics, CCG Editor, Greylanders, and Triolynos quests folders, create the following sub-folders:
   - Requirements
   - Design

Once completed, your folder hierarchy should look like the following:



Integrate existing documents

Integrate your existing documents into the folder structure you built.

1. In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 4\Documents to integrate.**
2. Select two projects of your choice in the folder.
3. Move each file related to your selected projects to the appropriate Lunar Water folder.

   For example, **CCG Editor-Export format-design.txt** should be placed in the **Desktop\Lunar Water\CCG Editor\Design**. As another example, user stories have been collected as requirements for the software. Place the user stories documents in **Requirements**.

Build a cross reference page

Many employees work on multiple projects at the same time. To improve transparency and collaboration opportunities, build a cross reference page in

your site that contains status information for all development projects across the enterprise.

1 In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 4**.

2 Copy **documentsbytype.html**.

3 Past the file in **C:\tomcat\webapps\ROOT\LunarWater**.

4 To preview the file, open your browser and navigate to:

http://localhost:9090/LunarWater/documentsbytype.html

LUNAR WATER

Development Documents

Requirements

Project name	Feature name	Targeted version	Team Lead	Last Revised	Status	Approved by	Approval date	Link to document
Triolynos quests	New Tournament Mode	1.4	Jim K.	10/27/2018	In Design	Jim K.	10/29/2018	Triolynos-New Tournament Mode-req.txt
Triolynos quests	New Tournament Mode-userstory	1.4	Jim K.	10/24/2018	In Design	Jim K.	10/26/2018	Triolynos-New Tournament Mode-userstory

Design

Project name	Feature name	Targeted version	Team Lead	Last Revised	Status	Approved by	Approval date	Link to document
Triolynos quests	New Tournament Mode	1.4	Jim K.	11/04/2018	In Design			Triolynos-New Tournament Mode-design.txt

5 Right-click **documentsbytype.html** and select **Edit in Notepad++**.

6 To enable stakeholders to quickly view project status details, complete the information for the three documents you placed in the folder hierarchy using the following guidelines:

- Use the Project Lead name from the **currentprojects.html** page.
- Use the in-design version for the Targeted version.
- Add your own dates to establish a normal document creation and approval flow.

- For status, select from the following values:
  - Not Started
  - Writing
  - In Design
  - In Dev
  - In QA
  - Released

7 The Analytics project has no entry in currentptojects.html. Use the following information to add the Analytics project:

- Project Name: **Analytics**
- Feature name: **UserAnalysisPlugin**
- Published version: **1.5**
- Version **1.6** is in design
- Version **1.6** is in dev
- Version **1.5** is in test
- Project Lead is **Tim H**.

8 Save the file.

Requirements

Project name	Feature name	Targeted version	Team Lead	Last Revised	Status	Approved by	Approval date	Link to document
Triolynos quests	New Tournament Mode	1.4	Jim K.	10/27/2018	In Design	Jim K.	10/29/2018	Triolynos-New Tournament Mode-req.txt
Triolynos quests	New Tournament Mode-userstory	1.4	Jim K.	10/24/2018	In Design	Jim K.	10/26/2018	Triolynos-New Tournament Mode-userstory
Greylanders	Character Editor	1.0	Hans Q.		Not Started			Greylanders-CharacterEditor-req.txt
Greylanders	Character Editor-userstory	1.0	Hans Q.	10/24/2018	Writing			Greylanders-CharacterEditor-userstory.txt
CCG Editor	Export format	1.7	Chen T.	10/17/2018	In Dev	Chen T.	10/19/2018	CCG Editor-Export format-req.txt
CCG Editor	Export format	1.7	Chen T.	10/21/2018	In Dev	Chen T.	10/21/2018	CCG Editor-Export format-userstory.txt
Analytics	User Analysis Plugin	1.6	Tim H.	10/27/2018	In Dev	Chen T.	10/28/2018	Analytics-UserAnalysisPlugin-req.txt
Analytics	User Analysis Plugin	1.6	Tim H.	10/22/2018	In Dev	Chen T.	10/25/2018	Analytics-UserAnalysisPlugin-userstory.txt

Design

Project name	Feature name	Targeted version	Team Lead	Last Revised	Status	Approved by	Approval date	Link to document
Triolynos quests	New Tournament Mode	1.4	Jim K.	11/04/2018	In Design			Triolynos-New Tournament Mode-design.txt
Greylanders	Character Editor	1.0	Hans Q.		Not Started			Greylanders-CharacterEditor-design.txt
CCG Editor	Export format	1.7	Chen T.	10/21/2018	In Dev	Chen T.	10/21/2018	CCG Editor-Export format-design.txt
Analytics	User Analysis Plugin	1.6	Tim H.	10/22/2018	In Dev	Chen T.	10/25/2018	Analytics-UserAnalysisPlugin-design.txt

9 Use the Analytics project information above to update **currentprojects.html** with a new entry in the CCG table.

Current Projects

CCG

Project name	Published version	In design	Team	In dev	Team	In Test	Team	Project lead	Project site
Triolynos quests	1.1	1.4	D4	1.3	D3	1.2	QA2	Jim K.	Triolynos quests
CCG Editor	1.6	1.7	D1	1.6	D2	1.6	QA1	Chen T.	CCG Editor
Lootbox Generator	1.1a	1.1c	D2	1.1b	D4	1.1b	QA1	Dan B.	Lootbox Generator
Analytics	1.5	1.6	D3	1.6	D3	1.5	QA2	Tim H.	Lootbox Generator

RPG

Project name	Published version	In design	Team	In dev	Team	In Test	Team	Project lead	Project site
Greylanders	None	1.0	D1	1.0	D1	None	QA3	Hans Q.	Greylanders
DLC1	None	0.9	D4	None	None	None	None	Stacey P.	Greylanders

You portal (file system) and site reflect the current status of development projects.

10 Since Lunar Water's folder hierarchy is by project, we created a page to cross-reference all the same types of documents together. With the opposite approach, storing by document type, we would have made a page to cross-reference all documents by project. Both approaches have merits. Discuss your preference for organizing files in your enterprise.

_____

_____

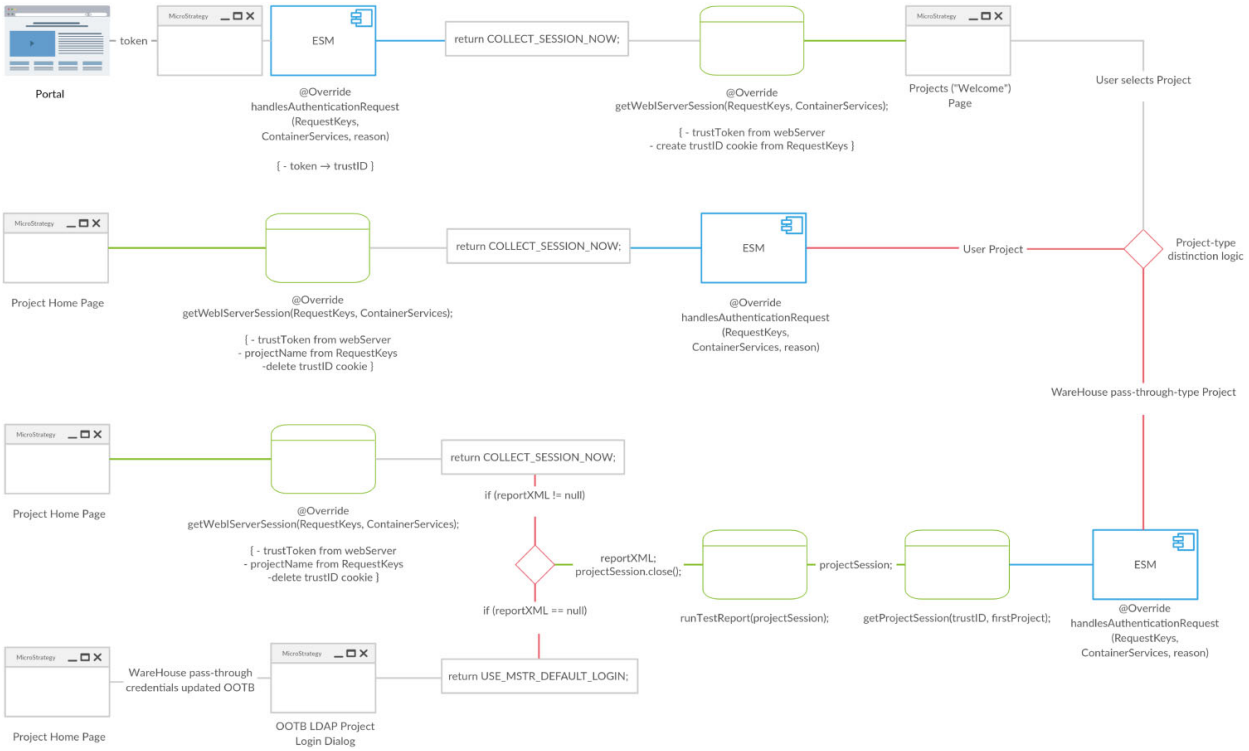
Building diagrams to design and communicate solutions

In an Intelligent Enterprise, communication is key across all facets of the organization. When working to solve a client problem, you and your team can enhance communication across departments with a visual representation of the proposed solution. A flowchart provides an easily digestible picture that helps clients better understand the process and accompanying narrative.

Additionally, from a development perspective, the diagram creation process provides the initial vision and starting point of a project, along with a solution implementation guide. Creating a flowchart also allows your team to fully engage

with the task at hand as they think through and draw each piece of the solution puzzle.

For example, a client came to the Services Architect's team requesting SSO throughout the enterprise using a third party technology. However, the enterprise's MicroStrategy implementation linked to an active directory source. Therefore, the usage of an External Security Module within MicroStrategy would provide the single sign-on feature without relying on another application layer for authentication. Once in place, the ESM provides the credentials to other systems in the organization, delivering a true SSO. To convey this clearly, the Services Architect created the diagram below displaying the MicroStrategy External Security Module.



As the Services Architect, you are responsible for ensuring you and your team create best-in-class diagrams and flowcharts for client solutions. You should also standardize the creation process, what symbols are used in the flowchart, and ensure that every new solution for a client is diagrammed. The basic symbols are discussed below and can be used as a guide as you work to standardize the diagramming process.

Basic flowchart symbols

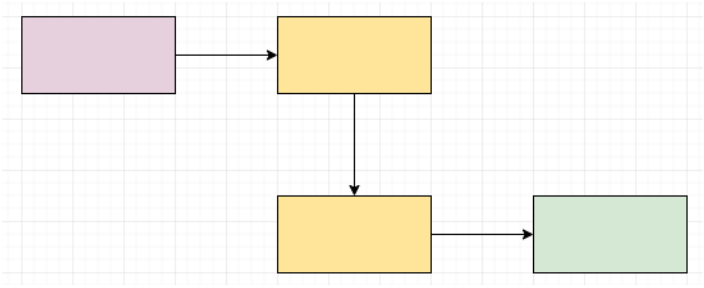
To build a successful flowchart, use a standard symbology defined by the American National Standards Institute (ANSI). The base symbols are most useful

to describe algorithms an computer programs. For visual impact, you can adapt the concepts presented below. For example, a rectangle representing a process could be substituted with a server. This implies both a process and a location at the same time.

Below is a quick overview of shapes and elements found in flowcharts.

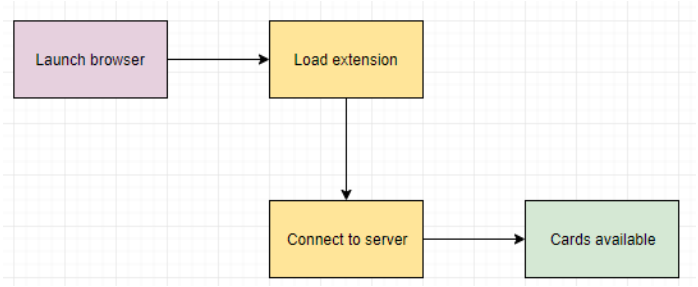
Arrows and lines

Arrows represent the flow itself within the diagram. Solid line arrows are the standard flow. If you have alternative flows or an arrow going from one diagram's element to another, color, thickness, and dash type can be effective modifications. If you do use different arrows and lines, make certain to include a legend.



Rectangles

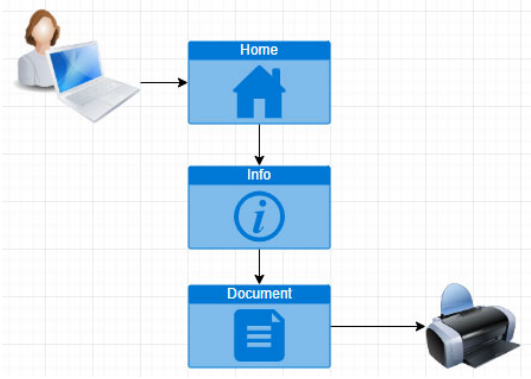
Any process that changes the current status can be described in a rectangle. The caption describes the process performed.



Start-stop

The ANSI symbols for a beginning or end of a flowchart use a rectangle with a half-circle on the left and right. You can use a different way to indicate the start by putting the start at the top left of the diagram (to be adjusted in other cultures)

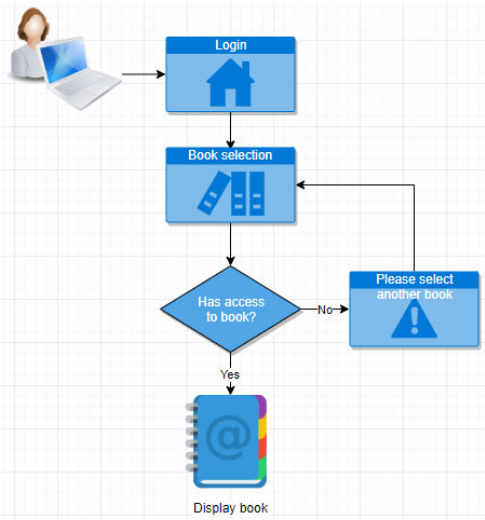
and having a single arrow starting from this element. For the end, the bottom of the page is preferred and no lines come out of the final element of the diagram.



As you can see above, there was no need to use special symbols. The start and end of this diagram are obvious.

Decision

The symbol universally recognized for decision is a rhombus or diamond. Usually, two arrows come out of the shape to represent a binary choice defined by the caption of the shape. You can add more choices coming from a decision. Avoid cluttering the element with too many possible outputs.



Other considerations

When designing a guide for your team to create diagrams, there are other factors you must keep in mind if you want your diagram to be an asset during presentations and not a distraction.

Size: Limit your diagrams to a single page or slide. If you need more than a page, make this a conscious choice and present it with the proper medium.

Density: Do not put your elements too close each other in your diagram. Leave room for the arrows and captions to do their job and let the user absorb the flow.

Color: Unless presenting a concept that requires neon colors and a disconnected palette, try to use simple colors that are compatible.

Template compliant: Use your organization's diagram template. Work with your graphics department to standardize your team's diagrams.

Exercise 4.6: Diagram applications and workflows

In this exercise, use draw.io to visualize various projects. In some organizations, other tools may be used to create diagrams, like Microsoft Visio or Lucidchart.

Scenario 1

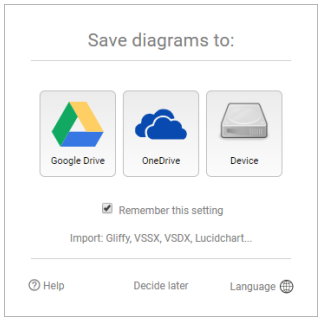
The mobile applications team is preparing a presentation to a client who wants their mobile app to automatically connect to an Intelligence Server depending on the current sales zone location.

Your team already knows that the MicroStrategy Mobile app can be reconfigured on the fly using a mobile configuration URL. To convey this concept in image rather than words, to build a flow diagram.

Launch draw.io

1 Open a browser session, and navigate to **https://www.draw.io**.

The Save window opens:



You can save your documents online or on a local device. You can also import documents from other diagramming products.

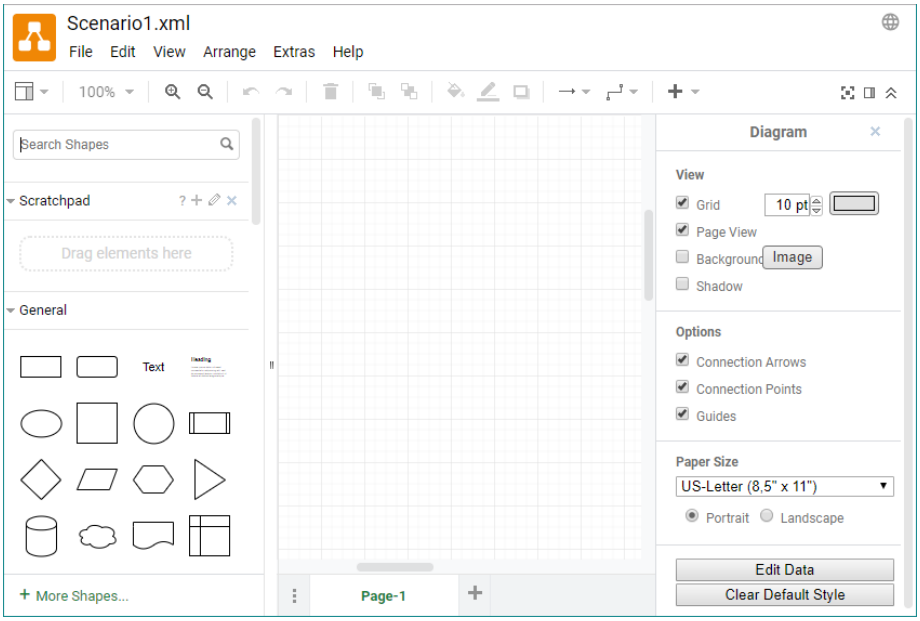
2 Click **Device**.

3 Click **Create New Diagram**.

4 Click **Blank Diagram**.

5 Click **Create**.

The draw.io basic diagram page opens as shown below.



Build a basic diagram

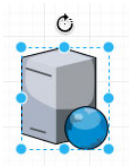
1 Take a moment or two to examine the panels.

You can drag shapes from the left panel to the canvas in the center. The right panel and top edit bar contain formatting options.

2 In the **Search Shapes** box, type **Server** and press **Enter**.

3 Click **More Results**.

4 Drag **Web Server** to the canvas.



You can drag the blue dots to modify the dimensions of the shape, or to anchor arrows and lines. Hover over the shape, to display four pale arrows that can be used to attach another identical shape with an arrow between them.

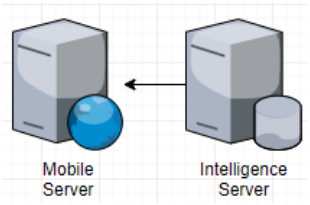
5 While the shape is selected, type **Mobile Server** in two lines.

The caption is displayed under the shape

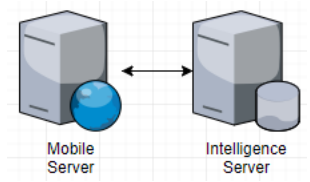
6 Drag one **Database Server** to the right of your Web server. Use the blue line that appears to align the shapes.

7 Type **Intelligence Server** to create a caption.

8 Using the blue arrows, connect the Intelligence Server to the Mobile Server.

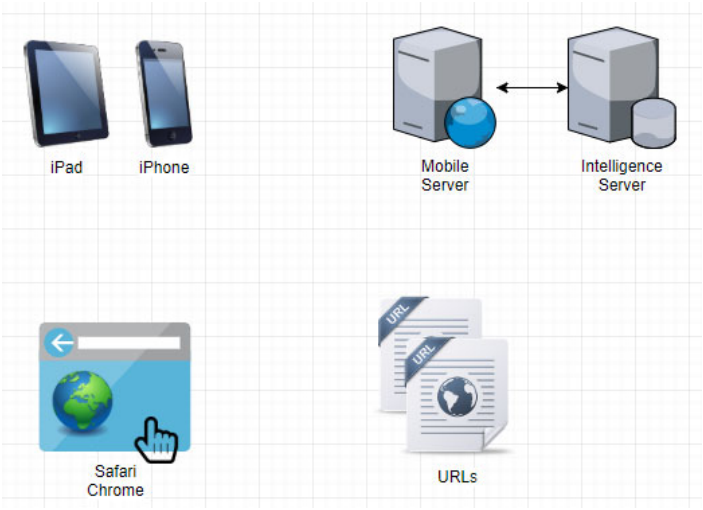


9 Select the **Mobile Server** and press **Shift** while clicking the blue arrow to force the arrow to link the servers, as shown below.

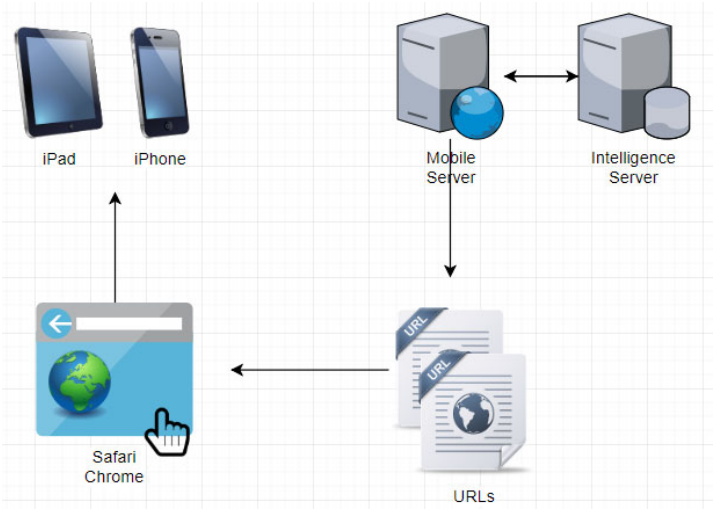


10 Use the Search box to add the following shapes and captions to the canvas.

Shape	Caption and details
URL	Caption: URLs Place 2 of them, you can use CTRL+D to duplicate on the canvas
Browser	Caption: Safari Chrome Place in two lines
Pointer (hand)	Caption: None Place as if clicking the browser
Earth	Caption: None Place inside browser
iPad	Caption: iPad
iPhone	Caption: iPhone



11 Add arrows to connect the shapes, as in the following image.



12 Add a **bidirectional arrow** (from the General section of Shapes).

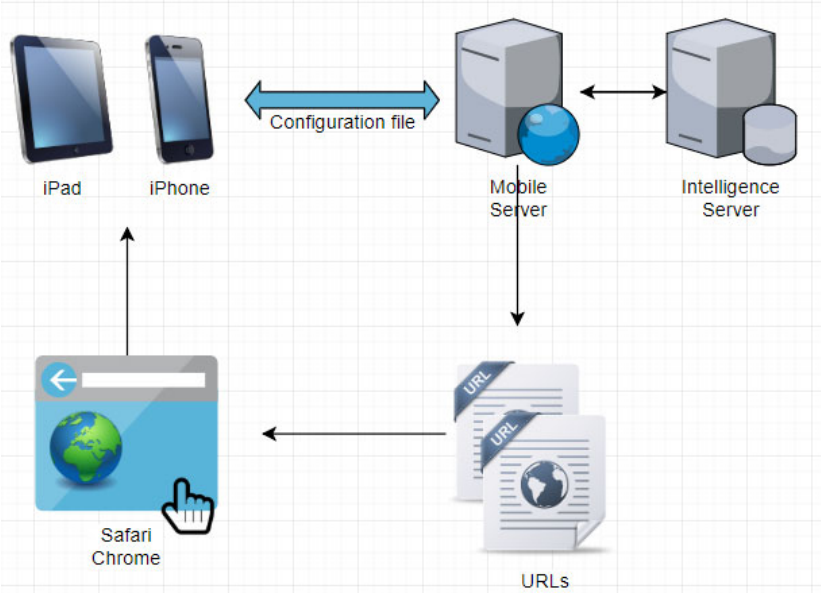
13 Add a **Text** element from the Plus (+) menu. Enter **Configuration File**. Place the text under the bidirectional arrow.

14 Click **File**, then **Save As**.

15 In the Filename box, type **Scenario1.xml** and click **Download**.

The file is generated and downloaded in the Downloads folder.

Your finished diagram should resemble the picture below.



Congratulations! Your flowchart in draw.io is complete.

Create a flowchart

The card editor application at Lunar Water includes a feature to check for application updates. When the update is selected, the software updates and relaunches. You have been tasked to create a flowchart that illustrates the following process:

> *When the application's Check for Update menu option is clicked, the application grabs the version number from the card editor. Then, it makes a web service call to check the latest available editor version. If the version number is higher, the upgrade begins, otherwise a message displays that the latest version is installed. If an update is available, a small executable upgrade manager is launched that closes the card editor. Then, it downloads the latest card editor version from the FTP site.*
>
> *The upgrade manager launches the installer. The installer uninstalls the previous version and installs the new version. If the installer is successful, the upgrade manager launches the new card editor and terminates. If the upgrade fails, the upgrade manager informs the user before terminating.*
>
> *One special case is if the card editor has the developer flag set. When the developer flag is set, the small executable also stops the local Tomcat server on the developer's machine and retrieves the MicroStrategy plug-ins from the same*

folder as the card editor release. It then installs the plug-ins in the appropriate folder and restarts the local Tomcat server.

1. Armed with this summary, create a flowchart to visualize the process. Start with a classic software set of shapes.

2. Compare your solution to one possible solution in *Exercise 4.6: Creating diagrams, page 201*.

Establishing best practices for code

In an Intelligent Enterprise, you put in place best practices that move an organization's coding activities into a repeatable, reliable, well-structured deliverable that becomes a solid foundation for future software development work. This also means standardizing the principles that developers follow throughout the enterprise for all coding efforts, big or small.

- **Ensures uniformity**: When coding standards are in place, the organization can be assured of a level of uniformity in how things are done, and what is produced, from one coding project to the next. For example, this means that in any project folder, you easily find the source code in the same subfolder; a thorough and accurate readme.txt file exists; and developers can easily identify a variable from an object just by looking at its name.

  This professional and efficient reliability is created by a specific set of behaviors, and it results in a predictable set of source files that any employee can rely on when looking at any project. The uniformity and consistency assists in team members locating information and knowing where and how to add content.

- **Improves readability**: Opening a code file badly laid out for human consumption is a challenge.

```
1  package com.microstrategy.sdk.samples.transforms;
2  import com.microstrategy.web.app.transforms.ReportPathTransform;
3  import com.microstrategy.web.beans.MarkupOutput;
4  import com.microstrategy.web.beans.ObjectBean;
5  import com.microstrategy.web.beans.ReportBean;
6  import com.microstrategy.web.beans.WebBeanException;
7  public class CustomReportPathTransform extends ReportPathTransform {
8  public String getDescription() {
9  return "This transform displays custom text after the report name.";
10 }
11 public void renderCurrentItem(MarkupOutput paramMarkupOutput) {
12 ObjectBean localObjectBean = getObjectBean();
13 ReportBean localReportBean = (ReportBean)localObjectBean;
14 int i = localReportBean.getViewMode();
15 int info=0;
16 try {
17 info = localReportBean.getReportData().getGridTotalRows();
18 } catch (WebBeanException e) {
19 e.printStackTrace();
20 }
21 String str = ""+localReportBean.getName();
22 if (i == 1) {
23 str = "(Grid)";
24 } else if (i == 2) {
25 str = "(Graph)";
26 } else if (i == 3) {
27 str = "(Grid & Graph)";
28 }
29 super.renderCurrentItem(paramMarkupOutput);
30 paramMarkupOutput.append("<SPAN CLASS='mstrPathLast1'> " + str + ": "
   + Integer.toString(info) + " rows" + "</SPAN>");
31 }
32 }
```

Opening a code file where indentation is in place, comments abound, and you can discern objects from variables and from methods, you can read the code and understand what it is doing quickly and accurately.

- **Easier use of unfamiliar code**: If you ask a developer to look at another project's code to assist in a situation when all hands are required, having standards in place makes the situation much more effective. The unfamiliar code can be presented in a known format, allowing the reader to locate important content quickly and focus on the work needed, rather than finding the code buried under bad formatting, with one-letter variable names and one line of comment at the top of a file containing a thousand lines of code.
- **Allow interchangeability**: Standards support easy rotation of team members into different projects. Standards help avoid a loss of productivity due to the time needed to adapt to different styles in a different projects.
- **Improve productivity**: Teams that use defined design patterns, code snippets, and a predictable way of doing things in the same manner improves productivity. Standards reduce the need to think about how to do certain tasks; for example, when logging an exception, having a piece of code you can reuse and adapt quickly helps doing it right the first time, along with keeping the end result more predictable and easier to maintain later.

We cover design and coding in depth in the next chapter, due to the many phases of these more complex efforts where corporate standards must be established and referenced throughout the design and coding work.

Testing application code

Testing involves verifying that the code written responds to the requirements without any issues, and that it passes the level of quality expected from the enterprise. When a product has passed QA, this means it meets all the elements defined in the QA plan.

Ensure the following core testing processes are part of the standards you establish for effective enterprise quality.

- **Types of testing**

  There are different levels of testing in a professional development chain. These may include:

  - Unit testing
  - Black box testing
  - White box testing

Best Practice

As best practices:

- Ensure that all types of testing are evaluated for best practices, consistency, and are clearly documented and maintained.
- Include unit testing in the design specifications, to ensure these tests are properly fit into any development plan time line.
- Ensure that the QA plan includes the black box and white box testing, for the same reason.
- As with all Services Architect efforts to make all processes intelligent, establish test plan templates for each type of testing, so that all test plans for all projects benefit form the best ideas and from corporate standards and expectations.

- **QA plan**

  The QA plan covers testing the requirements and expected functionality. A QA plan is a key component of coordinating efforts with the development team, so the QA plan is part of good communication. Coders must know how the application is tested and testers should have details of the features they are testing to be able to determine a pass/fail level.

Best Practice

Ensure that a QA plan is available for every development project and is based on a quality template.

- **Functional testing plan**

  This type of black box testing tests the software features based on their written specifications. Coding structure is almost never part of functional testing. The plan should include what portions of functionality to test, and standards for performing that testing effectively.

- **Nonfunctional testing plan**

  Some components of an application are tested using a different approach than functional testing and include:

  - User interface testing, for example, with focus groups
  - Performance
  - Reliability

Best Practice

Ensure standards for non-functional testing match functional testing. The testing template is an excellent example of how to achieve this important best practice.

- **Automated testing plan**

  A portion of the testing for an application or component can be automated using specialized tools. Some testing has to be manually performed.

  Ensure standards are written and readily available for automated testing, and that the processes are regularly updated every time automated tests are performed.

Best Practice

Make the final testing step for an automated test be about updating the automated testing process with anything uncovered during this automated test that has changed from what is documented. Update the test plan template for any improvements that can be standardized across development projects.

- **Blitz/stress testing plan**

  Web applications should undergo a stress testing phase. This can be automated or you can also involve the whole company in the case of a company-wide application.

  For example, at Lunar Water, you can set a specific hour when everyone in the company accesses the application and tries to break it by using it to its maximum. You instruct users to log defects. You might offer a bug bounty score by teams and turn this into a motivating activity.

  Ensure stress testing plans include clear outcomes; this can be achieved reliably by stating this requirement clearly in the stress test plan template.

- **Performance testing plan**

  An application that can perform well under load is important, but an application that demonstrates reliable performance across features is just as important. Ensure tools are in place to measure how the application performs, how it compares to the previous build, and the previous version. Common tools to consider in this category include Apache JMeter, Httperf, LoadRunner, and others.

  As with all other types of testing, ensure a testing plan template exists to structure performance tests consistently across development projects.

- **Security testing/penetration testing plan**

  You ensured that security was injected into all software design from the start. You must also make certain your application and its components resist penetration, by exposing any vulnerabilities while developing. The goal is to reduce security risks to users, as well as reduce the number of after-deployment security patches.

In addition to establishing a security test plan template to make this type of testing consistent enterprise-wide, a Systems Architect should set up regular reviews of standard reports on testing results from QA leads and others.

At Lunar Water, you institute static and dynamic analysis tools. This provides your QA leads with a mix of manual and automated tools to track any software vulnerabilities.

- **SDK-specific testing plan**

  For SDK-specific testing, utilize the test types described above, and especially the following types of testing that are pertinent to SDK testing:

  - Unit test code
  - Back end testing
  - Front end testing
  - Automated testing
  - If you are developing a library or component aimed for developers that uses the same languages, protocols, and conditions used by potential users

    For example, if you create a library in Python that extracts data from an external system and pushes the data to the MicroStrategy Intelligence Server, make certain to use the library with a Python application. If you know the library is to be used over a VPN, arrange testing in similar conditions. Your test plan can reflect this need so the best practice can be repeated wherever the same design decision occurs in a development project.

Exercise 4.7: Create QA plans

The Quality Assurance team is rapidly growing and need to standardize their processes to ensure they are working as a unified team. They have a series of plans for the upcoming QA rounds. As is the standard for all development documents, you want to centralize the QA documents to easily monitor progress and content.

Create the QA plans

1. In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 4\Base documents**.
2. Duplicate the **QAPlanBase.txt** file twice and rename each file using the suggested names in the list below.
   - Analytics-QAPlan-1.5.txt
   - CCG Editor-QAPlan-1.6.txt
   - Greylanders-QAPlan-1.0.txt
   - Triolynos-QAPlan-1.2.txt

Think about why the QA plan filenames contain numbers. The answer is presented at the end of the exercise, when we discuss the value of naming conventions in the Intelligent Enterprise.

Move the plans

1. For two project folders of your choice in **Desktop\Lunar Water**, add a new subfolder named **QA Plans**, then move each QA file to the appropriate subfolder.

Optional: Create the QA plans page

To ensure consistency across your Intelligent Enterprise, create a page on the Lunar Water site listing the projects with QA plans and name it **qaplans.html**. Each QA plan should contain the following details:

- Project name
- Team Lead

- Targeted version
- QA Lead
- Last Revised
- Status
- Approved by
- Approval date
- Link to document

Naming conventions for test plans: Testing for a version

Best Practice

The files of QA plans you used in the previous exercise had no feature in the filename, but had a version number. This is an important naming decision that you must consider, for the following reasons:

- A test plan is designed for testing against a specific build version of the software. Even if the plan had no change between version 1 and 2, you are still testing against version 1 or 2. The plan used against a specific version is numbered with that version.
- When a feature is being developed, until the release is declared, you cannot assume the feature is included in that release. So test plans for a specific feature must reflect the specific software build version that the feature is tested in, not the release version the feature is assumed or expected to ship with.

Standardizing software deployment

Moving applications or components into the final destinations for production is the responsibility of the Systems Architect working with the System Administrator. For example, you consult with the System Administrator for release phases and dates. Any deployment of an application or components should have required steps to ensure repeatable best practices. Common goals in this phase include:

- Least friction for the end user
- Done during down time if possible
- Have a fall-back plan

Managing a deployment

Deployment can happen along different paths. These commonly include:

- From code repository to dev environment
- From dev environment to QA environment
- From QA environment to staging environment
- From staging environment to production environment

Each of these paths are under the shared responsibility of the Services Architect, responsible for the status of the code, and the System Administrator, responsible for transport, delivery, and set up of the software in these environments.

Write a deployment plan that explain the what, when, where, and who for each of these paths. Include sanity checks within each path to monitor progress of the process. For any plan involving production, the plan must often be reviewed and approved by one or more CXXs.

Deployment scenarios

Common deployment scenarios you may encounter are:

- A standard, regular deployment
- More complex scenarios, such as private data centers, private clouds, public clouds
- Manual vs automated deployment

Ensure a written, documented plan is in place for those scenarios that apply to your organization.

Tracking progress (KPIs)

It is important to be able to measure the progress and current status of each deployment path.

Using MicroStrategy, put in place a dossier with a page for each component or application being deployed, and displaying the following key performance indicators:

- Name of component or application
- Current version number in production

- Date of release
- Number of implementations

Then, for each deployment path:

- Name of the path
- Current version deployed
- Date of install in path
- Next scheduled deployment
- Spot checks during last deployment

Update all SDK libraries to the latest/supported version

When deploying SDK components, make certain your standards documents ensure that development team members deploy the version that is compatible with the current platform version installed in the destination environment.

For example, if you deploy the MicroStrategy Mobile SDK for developers in a department where the MicroStrategy platform is on version 2021, you provide them with version 2021. You also deploy the mobile applications using the MicroStrategy Mobile application compiled with that same SDK version.

Deploying components to the correct server

Deployment of a component to the right server(s) has to be planned in coordination with the following Intelligence Center roles:

- System Administrator
- Platform Architect
- Analytics Architect

Best Practice

If the components are part of a larger update, make certain your processes require that your team members test the component against the version of the product being deployed at the same time. This has to be done in a similar environment.

If using MicroStrategy on Cloud as a deployment model, it is easy for your team members to provision an environment to test the component in parallel conditions.

Exercise 4.8: Deploy plug-ins

The finance manager has asked for two plug-ins for departmental versions of MicroStrategy Web. The CTO has approved your team to deploy the latest plug-ins in the production environment. After consulting with the System Administrator, you established a process and are ready to proceed with deployment.

In this exercise, do the following:

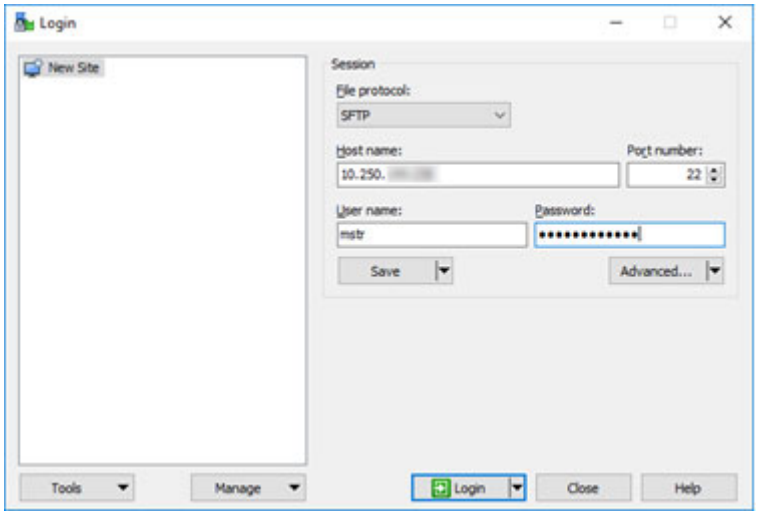
- Install two plug-ins in the Linux MicroStrategy Web site, which is considered the production environment. You can find the plug-ins to install in **Desktop\SEA_ExerciseFiles\Chapter 4\Visualizations**.
- Install the same two plug-ins in the local instance of MicroStrategy Web, which is considered the departmental environment.
- After both installations, verify the plug-ins are functioning.

You can use the bullets above to try this on your own. If you need details, use the steps below.

Install the plug-ins in the production environment

1. Navigate to **Desktop\SEA_ExerciseFiles\Chapter 4\Visualizations**.
2. For each zip file, do the following:
   a. Right-click the file and select **Extract All**.
   b. Click **Extract**.
   c. Remove the duplicate folder. To do this, rename the top level folder and add a **1** to the end of the name. For example, D3BubbleChart1.
   d. Cut the second level folder and paste it in the **Visualizations** folder.
   e. Delete the empty D3BubbleChart1 folder.
3. From the Windows desktop, open the **hosts** file in Notepad.
4. Copy the Intelligence Server IP address at the bottom of the file.
5. From the desktop, double-click **WinSCP**.

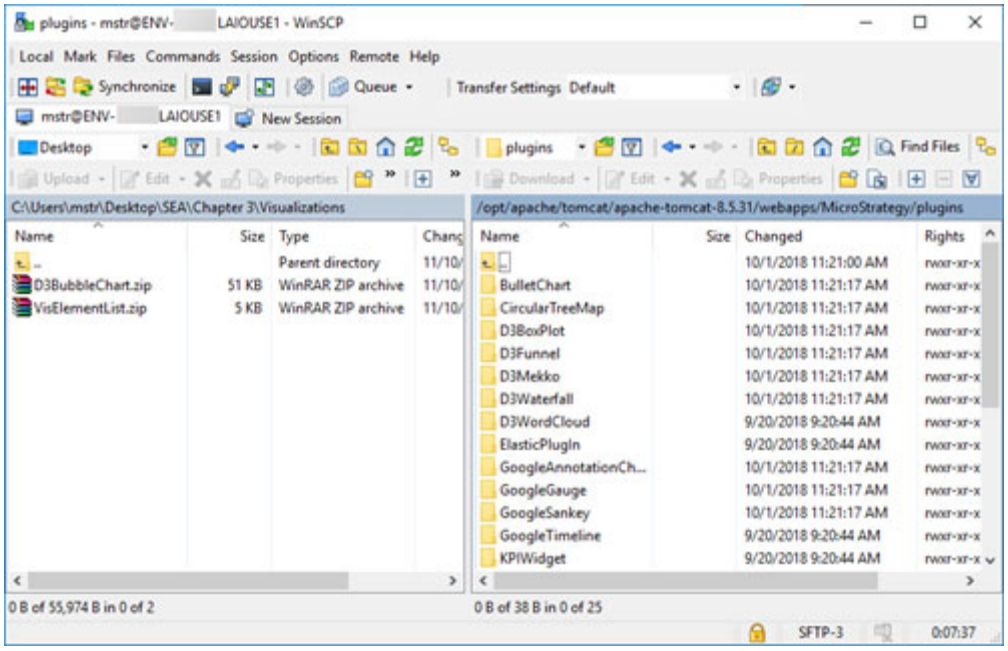
6 Using your Intelligence Server IP address and the credentials from the MicroStrategy Cloud email, establish a connection for file transfer.



7 If prompted, click **Yes** to continue connecting to an unknown server.

8 On the left side of the WinSCP window (the Windows server), navigate to **Desktop\SEA_ExerciseFiles\Chapter 4\Visualizations**.

9 On the right side of the WinSCP window (the Linux server), navigate to **/opt/apache/tomcat/apache-tomcat-9.0.30/webapps/MicroStrategy/plugins**, as shown below.



We navigated to **/opt/apache/tomcat/apache-tomcat-9.0.30/webapps/MicroStrategy/plugins** to deploy a plug-in for MicroStrategy Web. If we wanted to deploy a mobile plug-in, we would have used the **/opt/apache/tomcat/apache-tomcat-9.0.30/webapps/MicroStrategyMobile/plugins** folder instead.

10 Select the **D3BubbleChart** and **VisElementList** folders in the left pane.

11 Click **Upload** above the pane.

12 Click **OK** to confirm the transfer. Check that the two folders appear in the right pane.

13 Close the WinSCP session.

Start the application server

You have placed the plug-ins in the Web server folder structure, but Tomcat is not yet aware of the changes. You need to restart Tomcat so it can detect the new plug-ins.

1 From the Windows desktop, double-click **VNC - Viewer**.

2 From the **File** menu, select **New Connection**.

3 In the **VNC Server** box, type **XX.XX.XX.XX:5901**, replacing the Xs with the Intelligence Server IP address in the hosts file. (5901 is the port that VNC is assigned to in this environment.)

4 In the **Name** box, type an identifier for this connection.

5 Click **OK**.

6 Double-click the new connection and log in using the password from the MicroStrategy on Cloud email.

7 From the **Applications** menu, point to **System Tools**, and click **Terminal**.

8 Enter the following command: **service mstr stop**.

Wait for the command to complete.

9 Enter the following command: **service mstr start**

After the prompt returns, wait about a minute to allow the Intelligence Server to restart.

10 Enter the command: **service mstr status**

Three status messages are displayed in green, indicating the servers restarted.

11 Close the VNC session.

Install plug-ins in the departmental environment

Now install the same plugins on the departmental instance of MicroStrategy Web on the Windows server.

1 In File Explorer navigate to **Desktop\SEA_ExerciseFiles\Chapter 4\Visualizations**.

2 Copy the **D3BubbleChart** and **VisElementList** folders.

3 In File Explorer, navigate to **C:\sdk_workshop\war\MicroStrategy\plugins**.

4 Paste both folders.

5 Using the shortcuts on the desktop, stop and restart Tomcat.

Remember that you have another instance of Tomcat running our Lunar Water site. These shortcuts do not affect it.

Test the plug-ins

1 Open a browser with two tabs and navigate to these addresses:

- https://env-XXXXX.customer.cloud.microstrategy.com/MicroStrategy/servlet/mstrWeb

  where XXXXX represents your environment number.

- http://localhost:8080/MicroStrategy/servlet/mstrWeb?evt=3002

2 Perform the following steps in each browser tab:

a On the localhost tab, reconnect Web to Intelligence Server.

b Log in using your MicroStrategy on Cloud credentials.

c Click **MicroStrategy Tutorial**.

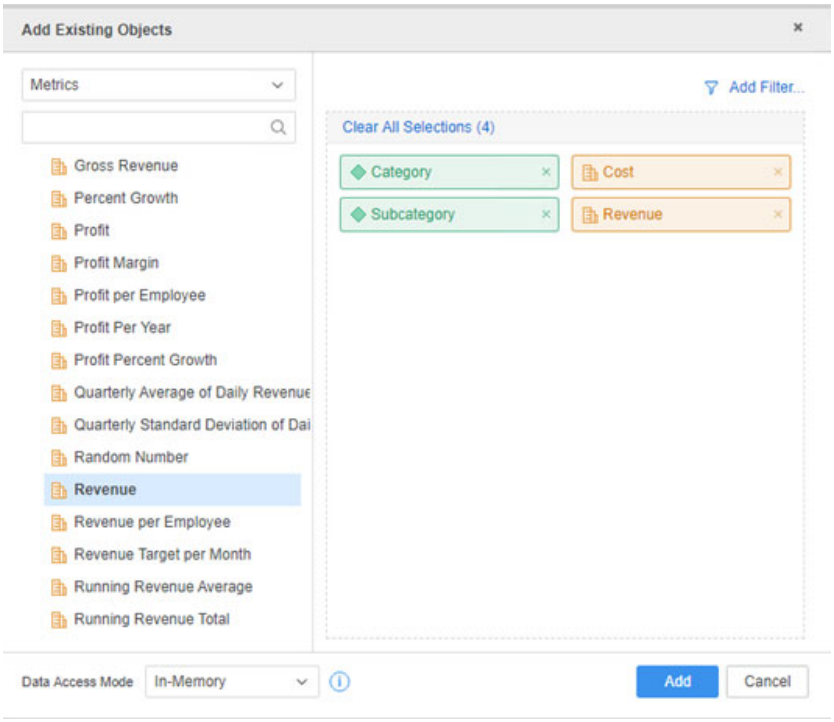
d Click **Go to MicroStrategy Web**.

e Click **Create** and select **New Dossier**.

f In the Datasets pane, click **Existing Objects.**

g Select **Attributes** from the top drop-down list. From **Products**, double-click **Category** and **Subcategory**.

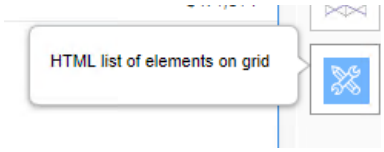
h Select **Metrics** from the drop-down list. From **Sales Metrics**, double-click **Cost** and **Revenue**. Then click **Add**.



i Drag **Category** and **Subcategory** to the **Rows** drop zone.

j Drag **Cost** and **Revenue** to the **Metrics** drop zone.

k In the Gallery panel, click **HTML List of Elements on Grid** custom visualization.



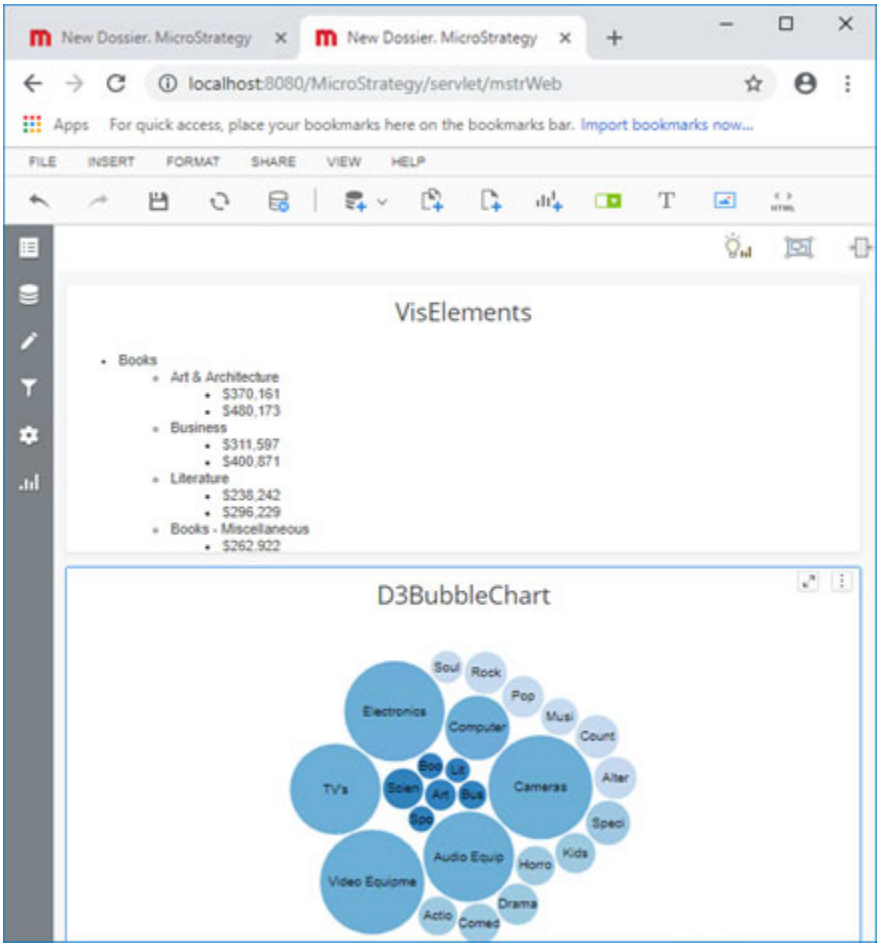
l Name the visualization **VisElements**.

m Duplicate the visualization, and place the new visualization below VisElements.

n Name the second visualization **D3BubbleChart**.

o In the Gallery panel, click **D3 Bubble Chart**.

Your dossier contains two custom visualizations.



In this set of exercises, you deployed custom plugins to two distinct environments.

Optional: Adjusting the plug-ins registry

To monitor plug-ins deployed in different environments at Lunar Water, you want to create a plug-in tracking web page in collaboration with the System Administrator.

1. Locate **pluginsdeployed.html** page located in **Desktop\SEA_ExerciseFiles\Chapter 4**.

2. Install it in your Lunar Water site.

Monitoring and maintaining applications

When an application or component is successfully deployed in production, you begin the process of application monitoring and maintenance.

Monitoring deployed products

To support effective and efficient maintenance efforts, you must have pertinent application monitoring plans in place. Monitoring helps your team identify bottlenecks and issues early. Keeping an eye on the performance and stability of deployed applications or components is an important process because you can often solve end user problems before they begin to negatively impact adoption of an application.

Monitor and observe application performance in its deployed environments. Work with clients and other user groups to obtain error logs whenever issues arise, no matter where they are reported from. This gives you a wider view into an issue when the feature or component is used in a variety of ways.

You should also monitor and observe the rate of errors resulting from all embedded analytics solutions. Those errors may not show easily as the warnings and exceptions may be broadcast in the browser console alone. This is where logs are effective monitoring tools.

Exception and performance logs

If you have put in place logging inside your application or components to monitor exceptions and performance, use those to gather actionable intelligence you can fold back into development. The analysis of application or component performance may influence priority in the development schedule of items that are observed as bottlenecks or are identified as candidates for optimization.

If your application has logging capabilities, use those logs in conjunction with third party tools to gather the performance and behavior metrics you need.

Third party tools

There are application performance management (APM), real user monitoring system (RUM), and search engine optimization tools (SEO) that can assist in monitoring performance and availability of applications, and provide analytics about your application or components. Common choices include AppDynamics, JMeter (Apache), GitPrime, and so on.

Maintaining deployed products

Maintenance includes fixing bugs and solving defects, but also includes defining and tracking the evolution of the application or component. As the Services Architect, you are responsible for ensuring that the evolution of your organization's deployed products is defined and tracked (documented).

Sources of issues that require maintenance effort include logs and other tools used to monitor application issues, direct user feedback, and R&D which can uncover issues with existing code when researching new features.

User feedback

You can get user feedback from multiple sources: an email address dedicated to suggestions, in-person meetings at conferences or networking events, or through a community site you have established where people can provide information about their challenges and how your software could help if it did this or that.

Product road map

Use all this information to fuel and prioritize where the next development efforts should be for a product or component. Integrate that list with your product road map and work with executives to determine priorities and vision.

Change management

As soon as a development team is finished coding an application or component, there is typically already a list of changes to make. Change can originate from a variety of sources, and it is important to establish a process of communication with these and other sources of changes so that new requirements can be

captured early and information can be shared easily both directions (user to development team and vice versa).

Common sources of changes include:

- Within the development team, as improvements team members want to add to the software
- Defects reported internally and by users
- Wish lists from a Community site
- Enhancements from clients and executives
- Longer-term road map

Change requests are commonly grouped into the following high-level categories which determine how they are then handled as a maintenance task:

- A change may be deferred to the next scheduled release
- A hot fix (or patch) may be created, often specific to a client and often based on a documented support case
- A broader software update may be created to include this and other changes

Patching

Creating a patch (or a hot fix) becomes a mini project on its own and should be managed with the same development standards you have established for all software development projects. If applicable, include the changes in the root code set. If a specific branch is needed, work closely with a Branch or Release Manager.

Work with the support team to coordinate the deployment of the patch to a specific client. If patching production environments internally, coordinate with the System Administrator.

Logs reporting issues

As discussed above, logs can reveal issues in the software. By resolving issues generated by logs from custom classes integrated in all environments, within the enterprise, and from clients if the logs are available, you proactively handle potential risks of future problems.

Establish a regular review process of the logs among development team leads to identify any exceptions that must be remedied.

Best Practice

Review all deprecated API references

Log and web traffic monitoring tools can assist development teams to identify API calls made by your libraries or third part libraries (like MicroStrategy REST APIs).

For example, it is important to identify any applications or components that are using a deprecated reference (an end point that has been replaced by a new one and is announced to be removed in a future version).

The Services Architect should ensure that development teams monitor and use this list of deprecated calls to raise awareness of any deprecated calls and educate developers and clients to modify their code to adopt the new end point to future-proof their code.

Best Practice

Best practices for your Change Management documentation

Additional change management best practices to include in your change management standards:

- Before coding, ensure that what is about to be coded is in scope
- Design the fix, no matter how small the change, so that unintentional consequences can be avoided
- Write small commits
- Write a meaningful commit message for other, later team members to be able to successfully update or change the work
- Never code in production
- Test all changes using established processes
- Document all changes
- Assess performance, security, and user experience per standard, established processes
- Ensure the change management documentation is also updated with any new process steps or updates required due to a new tool or better idea

Exercise 4.9: Risk management decisions for deployment and maintenance

Risk management is an important aspect of the software development process that enables you to keep your project on track and avoid pitfalls. In this exercise, explain how you would handle each scenario, then discuss your answers with your colleagues.

1 A client requests changes that are not part of the future road map and are specific to that client. How would you handle this request? Why?

_____

_____

2 When deciding on a solution for a client issue, do you prioritize a patch that fixes the problem for all customers; or do you prioritize a hotfix for this client alone with the plan to roll the fix into the next release? What considerations (risks) go into this decision?

_____

_____

3 A patch has to be delivered to a client. You thoroughly tested it in a test environment, but the client wants to install it directly into the production environment because their staging environment is compromised. What risks must be considered to make this decision, and why?

_____

_____

Standardizing software deprecation

Applications, components, and libraries all reach a point when they are replaced with a newer alternative. As the Services Architect, you need to create a general plan for how to handle this last stage of an application or component's life. Then, using a template, apply that general plan, with appropriate adjustments, to the application or component that is being deprecated.

Any template for deprecation should include the following sections, with best practices and guidance included for later use when deprecating a specific application:

- **Planning**: Add deprecation details to the application's road map; if a component is being deprecated, add this information to the larger product's road map.
- **Time line**: The deprecation template should include the following sections to be filled in for the specific application or component being deprecated:
  - Date when the application/component was launched
  - Deprecation reason, specifically whether the application/component code is being transitioned to something new or is being eliminated entirely

    This reason is useful for other teams to know, because Sales and Marketing may have an easier time selling an upgrade to a new feature if it can be directly tied as an improved replacement of an old feature.
  - Date when updates end
  - Date when technical support for the general public ends
  - Date when extended technical support ends
- **Communication**: Plan when to announce the end of life for the application/component, including the following actions:
  - Add the fact of deprecation to any internal training so all internal stakeholders know they need to update their corporate contributions, such as user documentation, marketing brochures, event materials, sales lists, and so on.

Best Practice

  - Have the development team create a support document that walks users through any uninstall steps, any changes required to adjust related software that remains in use, any install and configuration for a replacement feature or component.
  - Post all supporting documentation to internal locations (Intranet or portal site) and external locations (Community site, etc.)
- **Technical support**: The tech support team is the primary driver of assisting customers away from deprecated products. Ensure this team receives all deprecation planning documentation and access to any updates on the deprecation, as well as information on replacement products or feature sets.

Summary

A foundation for application life cycle standards is the most important structure the Services Architect puts in place after building development teams. By having a structure of intelligent templates that reflect best practices and lessons learned

over releases, the details of development projects start moving in the same direction and contain far fewer conflicts and issues to resolve after deployment.

To support the application life cycle, we looked at tools in different categories to support development. Among these tools, a collaborative communication tool is the most important.

We looked at core processes of the application life cycle: requirements, design, coding, testing, deployment, and maintenance. We highlighted important guidelines to put in place and good practices your documented processes and standards should include to make the Intelligent Enterprise successful. We also touched on change management and managing requests, defects, road map items, and risk management.

For two phases of the life cycle, we saved details for the next chapter, so that we can explore these phases in an Intelligent Enterprise at a deeper level: design and coding.

5

Establishing Design and Coding Standards

As the Services Architect, you establish the processes needed to support efficient, problem-free development teams and you ensure these processes and standards are easily available so they can be followed and kept up to date and relevant. This is key to constructing an Intelligent Enterprise that adheres to corporate standards, creates development products users actually use, and supports long-term company vision.

Specifically, core standard documents you must establish include a Coding Style Guide and naming standards for all artifacts of the coding process.

Among the development phases involved in producing pertinent, popular, and performant features and products, design and coding phases tend to have more complexities and are closely bound up in each other. We'll look at setting up intelligent design specifications and establishing coding standards that support your teams effectively while delivering on promised projects.

Creating software design guidelines

Design specifications are obvious documentation that must be in place for any development effort. This class focuses on additional, intelligent processes and standards you can have development teams include during any design process to

ensure greater product consistency and development efficiency, while ensuring that security is baked into every product from the start.

Establishing documentation standards

Design documents are the basis of all development efforts. Design specifications typically evolve throughout the process of a development project, to reflect better understanding and unexpected road blocks.

However, it is important the design documentation starts from a template that contains standardized, corporate-required information (such as adherence to a corporate security policy) for all development projects. This standardization ensures that important pieces aren't forgotten, or slapped on at the end without enough planning, for example, localization support or security features. By requiring all design documents to be developed from a common template, you can also ensure that efficiency measures and other best practices are always included in every design

For the greatest efficiency of development teams, the Services Architect should use the design template to establish rules for re-use of code and design patterns. Additionally, security and localization decisions must be made during the design phase because they are much easier to establish early than to try to retrofit into completed code that wasn't designed to support them.

Best Practice

For any external software referenced in a design document, make certain the applicable version is documented and that it reflects the version used throughout the enterprise or used in the target systems.

Producing reusable code

Small, bite-sized pieces of code can often be re-used with a new coding project. Re-using code provides increased reliability, reduces errors, and keeps products and features more consistent and thus easier to troubleshoot and maintain. Several methods to save and share code for re-use are discussed below.

Best Practice

These are included as part of the design planning because it is during design decisions that code re-use opportunities should be identified; this task is made much simpler when a formal code repository is established as part of an Intelligent Enterprise.

Using snippets within the IDEs is another way to improve productivity and uniformity within your team.

A snippet of code typically contains a response to a specific function, and they are typically grouped into the following purposes:

- Time saver for recurring structures
- User-defined code chunk with a single purpose
- Short code sample demonstrating a technique or algorithm

Snippet repository

Establish a snippet repository to store code snippets to be made available to the organization. Ensure that the following best practices are followed:

- Require each snippet to have a detailed description added to it.
- Organize a repository into groups of snippets, preferably around the snippet's usefulness by future development teams. For example:
  - Exception handling
  - File operations
  - Arrays manipulations
  - External Security Module
  - Embedding dossiers
  - Sorting techniques
  - CSS tricks

3rd party libraries

There are many 3rd party libraries available to code web applications. It is important to establish guidelines and recommendations that can be referred to, to control the libraries your teams use in app development and components.

Incorporate the following requirements into your standards for establishing an approved libraries list for developers to use:

- When, during the design of an application, a developer realizes that a library will be needed, the developer should be required to submit a request to add a library to the authorized tools available to developers. Establish a submission mechanism such as a standardized form asking for the information listed here.

For example, if a data scientist wanted to use MicroStrategy's mstrio library to work on predictive machine learning, this need must go through the library request process.

- After approval, the library (name, language, version number, and download link) should be added to an official list of supported libraries. Once a library is on the list, it can be used in any coding project.
- Approved libraries must be reviewed periodically for upgrades. An upgrade determination should be accompanied by the following work:

  1. Determination that there is a need for the new feature offered by the new version of the library.
  2. A regression test is done on the application by the team asking for the upgrade.
  3. An impact study is completed for related applications.
  4. If the upgrade is approved, change management is initiated by all the teams impacted by a new version of the library.
  5. If one or more products cannot adapt to the new version, decide if you authorize having more than one version of that library in the authorized list.

Exercise 5.1: Manage code libraries

Coding teams are currently working with different sets of support libraries, due to the lack of standards and communication. As the Services Architect, you want to publish the list of authorized libraries so that authorized versions and new libraries are consistently adopted across the organization.

Add libraries to the files portal

1. In File Explorer, navigate to **Desktop\Lunar Water**.
2. Create a folder called **libraries**.
3. Open the **libraries** folder.
4. Create the following folders:
   - mstrio
   - mstrio-py
   - d3js
   - embedding.js

Create a libraries inventory page on your site

1. Navigate to **Desktop\SEA_ExerciseFiles\Chapter 4** and duplicate **currentprojects.html**.
2. Rename the file **libraries.html**. Copy the file.
3. Navigate to **C:\tomcat\webapps\ROOT\LunarWater**. Paste **libraries.html**.
4. Edit the file in Notepad++.
5. In a single section, create a table with the following columns:
   - Library name
   - Version
   - Language

- Product(s) using it
- Server/client
- Download link
- Official site
- Tutorial

6 Optionally, fill in the table with your own data.

7 Save the file.

Discussion: Using external libraries

1 Name a few challenges you have faced implementing third party libraries.

_____

_____

_____

2 Would you rather use a third party library that brings 90% of the functionality you want, or set aside two of your developers to take a month to deliver what you need? Keep in mind the cost vs effort.

Exercise 5.2: Snippets management

Code snippets promote code reuse and foster an efficient development process. In this exercise, create a snippets repository.

Add snippets to the files portal

1. In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 5**.
2. Copy **snippets**.
3. Paste in **Desktop\Lunar Water**.
4. Open **snippets**.
5. Create the following folders:
   - javascript
   - nodejs
6. Open **java**. Inspect the folder's content. These are actual sample snippets.

Create a snippets inventory page on your site

You want to promote the re-use of good code, and this repository is an important start.

1. Navigate to **Desktop\SEA_ExerciseFiles\Chapter 4.** Duplicate **currentprojects.html**.
2. Rename the file **snippets.html**. Copy the file.
3. Navigate to **C:\tomcat\webapps\ROOT\LunarWater**. Paste **snippets.html**.
4. Edit the file in Notepad++.
5. Create a section for the categories:
   - java
   - javascript

- d3js
- nodejs

6 Use the table below to adjust the html code. Fill in the table using the names of the snippets file as title and description. Your tables should have the following columns:

- Snippet name
- Description
- Keywords
- Libraries needed
- Link

Snippet Name	Description	Keywords	Libs needed
Append text file	Append to a text file	Append Text	--
Array resize	Resize an array	Array Resize	--
Array to map convert	Put array values on a map	Array Map	java.util.Map
capture screen shot	Capture current screen	Capture Screen	java.awt.Dimension java.awt.Toolkit;
connect oracle odbc	Connect to Oracle with ODBC	Oracle ODBC	--
convert int string	Convert an int into a string	Convert	--
convert string date	Convert a string to a date	Convert	--
convert util date sql date	Convert util date into a SQL date	Date SQL	--
email send	Send email	Email	--
file copy	Copy a file	Copy File	--
files directory listing	List files in a directory	File Directory	--

Snippet Name	Description	Keywords	Libs needed
get name current methods	Get the name of the method executed	Method	--
http proxy settings	Get http proxy settings	HTTP Proxy	--
http request send fetch	Manage the http send/fetch	HTTP Send Fetch	--
image thumbnail	Get an image thumbnail	Image Thumbnail	--
pdf create	Create a PDF	PDF	java.io.FileOutputStream;
xml reading parsing	Parse an XML file	XML Parse	--
zip jar create	Create a .zip file	ZIP	--

7 Save the file.

Establishing design patterns

A design pattern is a re-usable solution to a common programming problem. Using a design pattern is a way of keeping development teams efficient, since a solution to a specific coding challenge already exists. Most patterns offer an object-oriented solution.

Patterns are identified during the design phase of a project. Some examples of patterns are:

- n-tier
- Model-View-Controller (MVC)
- Model-View-viewmodel

MicroStrategy uses the MVC model in its MicroStrategy for Mobile app, for example.

Best Practice

Best practice to take advantage of design patterns

- Train developers: Add to your development teams' training requirements some initial formal training on patterns. This information should also be part of the on-boarding process for new hires.

Creating standards for plug-ins and components

Like all development projects in the company, plug-ins and libraries are valuable assets. The Services Architect must define specific standards for designing and coding components like plug-ins and libraries.

There are also additional elements to consider when these plug-ins and libraries involve MicroStrategy components. You must establish documented processes and standards to support this important requirement, and make them available in well-maintained, clearly labeled documentation libraries.

Development standards

Component development should be handled with the same processes and standards as regular applications. This includes:

- Project management practices
- General life cycle development
- Source code management
- Security
- Implementation guidelines

Best Practice

Component-specific best practices

When developing components, include the following best practices in any design-specific template:

- **Design with a specific version in mind**

  A plug-in integrates with another piece of software. For this reason, your team must build the component aimed at a target version of the software. For example, a plug-in for the MicroStrategy Web product is built for the specific platform version currently used by departments or enterprise users. Backwards compatibility may not be mandatory for every component your team produces.

Using Lunar Water as an example, the company has a library handling lootboxes in the main game of GameNameHere. The library your team writes for the next expansion is built specifically for this version in mind and does not have to be retro-fitted for an earlier version.

- **Include components in base image**

  You have to decide if the components created becomes part of the computer base image used by the System Administrator for preparing development team images. For example, a specific plug-in for MicroStrategy might need to be deployed to everyone using MicroStrategy Web in the enterprise, or a library used in the line of business application must be present in every developer local code library repository.

- **Use latest approved versions with new development**

  Unless a conflict is identified, new development of a component or plug-in should start by using the latest version of the underlying technologies they rely on.

- **Test interaction**

  For a component, testing must go beyond basic functionality testing, regression testing, and integration testing. This way your team can certify a component against different versions of a software.

- **Design with vision**

  Just like an application, product design principles must be applied when designing a component. Your design templates should include the following considerations for reusable components like a library or a plug-in.

  - **Security**

    Components must have an extra level of scrutiny when it come to security. For example, your policies must protect intellectual property. You also want to make certain your libraries do not open any vulnerabilities in the applications using them.

  - **Logging**

    To facilitate component debugging, put logging capabilities into the requirements of every component your team creates. It offers the developers using the components a way to do their own debugging. You should have a logging engine inside the component that can be turned on and off.

    You can also include a deeper logging mechanism your team can use when debugging a component with a client. This added level of detail may be beneficial to your support team as they can assist customers with

additional information on how the component is behaving in their applications.

- **Maintainability**

  Structure component architecture with expansion in mind. Without over-designing the code, require that a level of abstraction is designed, and define a mini road map for the component, even if never used. This is a is a better approach than returning to code and patching something together, adding features, and leading to unnecessarily complex, bloated, and inefficient code.

- **The next version**

  Have your team consider the next evolution in mind when designing or putting down specifications for a component. You can increase the efficiency of design and the maintainability of the architecture.

- **Black box**

  The APIs you define for a library or an application must be consistent from release to release. There is nothing more frustrating for a developer than to discover that a call using your library has changed and must be modified in all the places where the component is used.

- **Document early**

  In this chapter, we discussed documenting the design, development, and progression of a product as an investment. This is also true for components. As you develop components, development teams must also prepare examples and sample code to offer developers an easy introduction to the component. The same is true for plug-ins. A sample dossier with a basic dataset is appreciated by end users.

  The Services Architect needs to ensure any component design document includes not only the component's API, but also the samples to be provided with the component.

Designing standards for REST API services

Lunar Water wants developers to be able to create applications that use the information the company collects about game players for different purposes:

- Targeted publicity
- Custom applications using a character's data
- Build a league of friends to play locally

- Analyze player statistics
- Provide a reference database of lore elements

As Lunar Water's Services Architect, you want to establish REST Web services that end users find easy and powerful. At the same time, you want to set in place a development process and practices that keep the company's data secure and the development cycle as efficient as regular applications.

To achieve these goals, you and your team need to provide the software infrastructure. After analysis, you conclude that the best way to approach these challenges is to have an open REST API in the game engine combined with some hooks in the MicroStrategy platform to retrieve data related to each game.

Advantages of REST

Developers appreciate applications that can be accessed by external entities. Developers can take advantage of your application, while integrating the functionality within their own application. This integration can be done for new applications or retrofitted inside existing applications.

REST is an easy to implement and easy to consume type of web service. It offers a quick and simple mechanism to summon the functionalities of your application from their applications.

Internally, you can have your developers build on a REST architecture for new products.

At Lunar Water, having a game engine that uses some REST services is important as they prepare a new game that uses the same engine, and adds new REST services specific to the new game.

Now that you and Lunar Water have taken the step to launch web services, you must define the services you want to expose from your application. For web services, either data is exposed to the caller, or a far away server performs some computing work and you consume the result or answer. All services offered can be categorized within a data provider or remote processor.

Below are the considerations that a development team should review before implementing a REST web. These considerations can become part of any Services Architect's guidelines or standards for design.

Categorize APIs

Best Practice

Of all the information and processing you want to expose, developers should categorize the services (called end points. Group together end points of a similar

goal. For example, authentication end points should be in an Auth group, while data about a product may be part of end points under a Products category. Think of the different scenarios you want to respond to. Once you have end points all identified, developers should define end points and place them in categories.

For Lunar Water, for example, the end point categories might be:

- Cards
- Players stats
- Players profiles
- Authentication
- Lore
- Lootboxes

Offer just enough

Having an open web services architecture is not a complete open door. The services must be defined to offer the consumers of these services the right information. For example, in a banking system, this might mean an end user endpoint that returns financial details about this specific user, while a manager can access data about multiple clients of the bank.

Best Practice

Keep proprietary data and operational data secret. Authentication should define the types of services and the possible data available to each user.

Best Practice

Work with the Platform Architect to define the security roles to expose the right data from the MicroStrategy platform.

The agreement with the service consumer should define the extent of available information.

API Business model

Whether you offer internal data within your enterprise or offer your data to any user accessing it through web services, that puts you in the data business. A well-thought-out data business model is an important piece of the Intelligent Enterprise.

For Lunar Water, the data business model was laid out in the following way:

- Each player can freely access anything related to them.
- Some public data can be accessed by all, like cards and lore.

- Non-profit organizations wishing to use data can get an account with limited access to player data.
- For-profit organizations can subscribe to and access the data that was released by the user agreement.
- No operational and sensitive data can ever be available to anyone, other than as directed by the Legal department.

By putting in place a business model, you have a clear definition on how to approach data access and define the appropriate access levels.

Best Practice

REST API-specific best practices

To put in place a rewarding set of web services, there are specific practices you want to put in place.

- **Treat as an application**

  Use the same principles as for plug-ins and libraries presented in *Development standards, page 131*

- **Back end technology to use**

  Select a back end technology to support the web services that your team members excel at. You want a proven methodology and tools to support this infrastructure. Examples include C#, Java, PHP, Python, Ruby, and many others.

- **Design adjustment**

  The design of URI templates must reflect the transition into objects providing services and not objects acting on themselves.

- **Security**

  Guided by corporate security policies enterprise-wide, your software security policies must define how secure to make your APIs, as well as the method by which APIs/data is secured (either by a token or key.)

  - Token based
  - Key based
    - Public key
    - User key
  - Paid-for access key

- **Penetration testing**

Ensure QA test plans include a requirement to attempt penetration testing against REST services. Have them try to get data by circumventing the security in place.

- **Public API**

  Publish the public API with a public web site. Announce the API portal through your community site.

  Most dedicated tools that assist in building REST APIs also manage the generation of a portal, such as IBM API Connect, Spring REST Docs, Swagger, and so on.

  Ensure that technical information about the API is included in any Documentation team efforts. Use your established communication mechanisms to share development information and review Documentation team efforts.

- **Mandatory black box**

  Apply the same diligence to the REST services you apply to components. It is vital to have a stable set of end points. End points can be added, but do not allow disabling of existing end points. End points can be transitioned to something newer by adding an information message to the response.

  If your developers optimize an end point, ensure they keep the same end point signatures. For new functionality, add a parameter in the signature to make it consistent.

  Use the community site to make your clients aware of additional of end points. To deprecate an end point, use the same practices mentioned for an end of life product support.

- **Backwards compatibility**

  REST API end points can be integrated into custom applications and be present in the code for a long time. As you evolve your product, clients expect their old applications to continue to function as before.

- **Communication to end users**

  - Use the community site to publicize changes. Consider establishing a specific area for REST API.
  - Host a GitHub repository presenting the API and links to documentation and samples.

- **Documentation and samples**

Although many of the tools used to build a formal REST API structure assist in generating a documentation site for the end points, that does not replace the need for formal documentation to supplement automatically generated material.

Also ensure some samples using the REST API are available, with details on how the REST API was used in the sample, and best practices for end points.

Promoting standards for integrating software

When a developer uses your application as a back-end support technology, whether to embed information on a web page, or to provide functionality within the wall of their applications, your application is being integrated.

Another example is to integrate MicroStrategy technology into your custom application.

As the Services Architect, both types of integrations occur in the enterprise. You must define how you want integration to happen in your company.

Standards for integration

The best integration ensures third party apps play nicely with everyone else. This means simple, expected communication in any dialogue between your application and other applications. You must make sure developers comply with industry standards.

Formats

If you provide data to a third party application, your developers must use JSON or XML as this is what a REST API call is expected to provide as a response. Developers can set up the REST end points to provide this by detecting the format desired from the request itself.

Exceptions

Follow the standard http response codes with appropriate responses. For example, a response code in the 200 range demonstrates success, a code in the 400s means a missing resource or something that could not be accessed, while a code in the 500s represents a server error.

As the Services Architect, have your development teams define the responses for end points in line with current industry standards.

Best Practice

Best practices when you integrate

Below are some protocols you should to put in place for a successful integration with other products, such as MicroStrategy.

- **Do a pilot**

  Any integration should start with a simple pilot program to confirm basic connectivity and authentications, followed with a simple request. For example, when using MicroStrategy, put in place a simple call to the auth/login end point followed by a simple request to retrieve a list of projects.

  The idea behind such a project is to build the skills of team members, and create a code base your team can then re-use in other projects.

- **Use samples**

  If a third party vendor provides samples of integration or a tutorial, ensure development teams start there to learn the specifics of the new web services your team wants to integrate into an application.

  If this integration becomes a frequent task, include this training as part of the on-boarding process of a new hire.

- **Put logging in place**

  A standard for development teams should be to add code into the integration that logs the important integration steps. For example, when connectivity is achieved, development teams should make available a copy of a request, the response code, and connectivity details with a time stamp. Those logs can be studied for optimization and/or debugging purposes.

- **Use a config file**

  To facilitate the portability of your application, connectivity variables should be placed in a file so that moving the application is simply a question of changing a few parameters in that config file. If security of the details in the file is a concern, teams should encrypt the file and add code to decrypt the data from within the application.

  By doing this, the transition from a developer's machine to a staging area, to a QA server, and then to production becomes much smoother.

Best Practice

Best practices when you get integrated

The Services Architect should recommend and publish protocols on your community site for people seeking to integrate your technology into their custom applications.

For Lunar Water, you decide to recommend the following principles to people wishing to integrate with one of Lunar Water's games.

- **Secure appropriate key**

  All users should pick up a public key or request a personal key for people creating apps that retrieve public data or information about a specific user. You can post in the community site to relay this message.

  As the Services Architect, you coordinate with Lunar Water's finance team to confirm services agreement access rights so that keys can be generated with appropriate rights.

- **Make the demo work in your environment**

  Remind anyone trying to access your application that you provided a sample app to show developers the ropes on how to achieve connectivity and operate simple transactions. Once they make this sample work within their environment, they have the blueprint to build a custom interface to your systems.

- **Add logs**

  To help monitor integration and assist in debugging, advise your users to include a logging mechanism to follow the integration. If the integration is developed using a library your team built, communicate about the integrated logging capabilities the library has (which is in place because your development teams followed your best practices to design logging for libraries and components.)

- **Use latest version of using a library**

  Any integration using a library should be accomplished using the latest version of the library.

- **Publish creative and successful integrations**

  Invite integrators to share the integration with your product they put in place, if the integration can be publicized from their end.

  Your marketing team may use these clients as an example, a case study, a white paper, or a speaker at an event, to showcase your company's development efforts in new and interesting ways.

Exercise 5.3: Consider Plug-ins, REST APIs, and integration

Discuss the development plug-ins, REST end points, and integrations, and how the implementation varies by enterprise.

Share your answers with your colleagues in class.

Plug-ins

1 Do you use the same methodology for developing plug-ins and libraries as for general applications? If not, which one do you use specifically for components?

2 What are the main plug-ins used in your development operations? Are they made in-house or by a third party?

REST APIs

1 Do you simply consume REST services, or do you develop REST end points?

2 Is your API public or private? Why? Are there any services you could expose to the outside world?

Integration

1 What is your ratio of integrator to integratee? What challenges do you deal with when integrating other companies' REST APIs?

2 What special measures do you have in place when a third party wants to integrate your technology?

Mandating secure coding

Security policies are established by an organization and should consist of written, clear policies in regard to what security features and standards any software produced must include.

The earlier you incorporate the appropriate security requirements into an application design, the easier it is to include security in a software product rather than having to try to force fit security into a product that was not designed for it.

Best Practice

Best practices to include security in the design process

- Training: Adding security to design and development requires a thoughtful and deliberate approach:
  - Consider establishing training from 3rd party experts on the appropriate topics.
  - Make security training part of the on-boarding process.
  - Work with the other Intelligence Center roles to plan for a quarterly security bulletin.

- Find and fix security vulnerabilities while writing code. Developers and testers should have design directives to consider during coding and testing, for example:
  - What is exposed with this code?
  - How can I circumvent this code and do malicious things with it?
  - Is there a way to make the code do something else?
  - Will I expose the server if someone injects malicious code in this text box and the validation function breaks?
- Use open source in a secure way: Open source code must be reviewed with an open source license to be used in any internal and external coding projects, for example, a free JavaScript library to assist in making a new visualization plug-in, or a dll to generate a PDF automatically.

Establishing a global organization: Localization

If your organization supports multiple languages, this added challenge for your development team requires consideration during the design phase of any software development project. As the Services Architect, having your applications and components reach your global users requires a localization plan that is deeply incorporated into your software design.

The best localization design principles allow localization in the company's products without adding too much complexity to the regular coding activity.

Localization design principles center around resource files containing strings for every piece of text used by the application or component. You must decide how to implement resource files with processes that can be repeated for each project so there is consistency across products and all team members know how to inject localization support into any code they write.

For mobile development, localization is natively supported by iOS and Android, so the process is embedded into the development of apps for these platforms.

It is also important that any design document includes localization requirements for related systems, such as documentation, help systems, and the community sites.

Best Practice

Due to the potential for complexity with some features, training for development teams may be necessary to establish your methodology and ensure best practices are followed by everyone.

Designate a localization lead to work with a translation coordinator

Establish the team role requirement of a localization contact within each development team. Then require this member of the team to work closely, early, and often with your organization's translation coordinator. The development team member must be a point of reference for all things related to string and functionality translation.

Among many tasks, this role must ensure that any string freeze dates are established early and communicated frequently so development teams can ensure they can adhere to these dates.

Translation tools

Translation efforts across multiple products spanning multiple languages must be organized and the tasks baked into the design process of any software development project. Much of this work is made easier if you establish a translation software system, such as SDL Trados or a similar enterprise-quality translation tool.

Best Practice

Even when you outsource translation work, your company owns the translation memories (translation databases, often called TMs), so a translation tool allows you to update and maintain those company assets over time and across translation vendors.

Best Practice

Design best practice to support localization

Include the following best practice in your software design documentation template so that these design decisions are always be considered as part of localization decisions for any product development.

- Never hard code anything text related: Design standards for localization should include the directive to never add text directly into applications for captions or other text content. All developers should use the localization structure in place for all text.

Translation of MicroStrategy platform plug-ins

The MicroStrategy platform uses a localization structure where strings used in the application are defined in files called MessagesBundle or Localization Descriptor files. In these files, each string has a bundle ID. For example, mstrWeb is the prefix used for all strings related to the MicroStrategy Web application.

Best Practice

- When you create new IDs for strings, it is a good practice to use your own prefix instead of mstrWeb in the MessagesBundle files. For example, myApp.1="My New String", myApp.2="My Second New String".
- If you change the default strings in the MessagesBundle files, keep a log of all changes to help you with future product upgrades. Replace the changed elements after each major upgrade.
- When you add new strings to MessagesBundle files, add them to MessagesBundle files for all supported languages to maintain consistency. If you do not have a translation for a specific string, the default should always be to use an English entry so that when users switch interface languages, there is never the danger of an empty string in the GUI.

Guiding the coding process

When Lunar Water began, as with many startups, individual developers made their code work, without reference to how they got there or the ability to replicate or build later features on the work they had done. There were no other developers to consider as they made their daily coding decisions.

In an Intelligent Enterprise, you put in place best practices that move Lunar Water's coding activities into a repeatable, reliable, well-structured deliverable that has become a solid foundation for future software development work. This also means standardizing the principles that developers follow throughout the enterprise for all coding efforts, big or small.

Defining a Coding Style Guide

To establish an Intelligent Enterprise, the Systems Architect must create a Coding Style Guide, based on existing corporate best practices and spelling out standards for specific aspects of software development. This section provides an extensive starting point for creating an effective Coding Style Guide for any organization.

On a high level, as part of an Intelligent Enterprise, coding standards should always include:

- **Naming conventions**, to make development work logical and efficient for team members coming and going on development teams, to enable fluid maintenance down the road, and a host of other benefits from a consistent naming scheme.
- **Standards for commenting code**, to support effective and efficient revisiting and maintenance of older code, often by people who had nothing to do with

the code when it was new, as well as to keep track of fixes and defects, assist documentation, and many other benefits.

- **Additional coding best practices**, specific to MicroStrategy plug-ins, code samples, and other specific coding goals and products.

Promoting naming conventions

A vital, but often overlooked, part of coding standards is a naming convention for elements in the code. Naming conventions are usually related to the programming language used. However, you can define a standard within the programming language's usual parameters.

Style guides for programming languages are freely available and references to these (as applicable) should become part of your style guide, as the basis on which to build your organization's own standards. A few examples include:

https://google.github.io/styleguide/javaguide.html

https://www.javaranch.com/style.jsp

https://www.w3schools.com/js/js_conventions.asp

http://www.cs.cornell.edu/courses/JavaAndDS/JavaStyle.html

http://cr.openjdk.java.net/~alundblad/styleguide/index-v6.html

http://cs.brown.edu/courses/csci0180/content/docs-spec/java-style.pdf

https://github.com/airbnb/javascript

https://petroware.no/javastyle.html

On top of the programming languages' style guides, you should add corporate standards for the following elements (provided with a sample recommended style):

- **Packages**: For a package name, use a reverse domain structure in all lowercase. For example:

```
com.microstrategy.sdk.auth

com.lunarwater.cards
```

- **Classes and interfaces**: Use full camel casing for classes and interface names. For example:

```
class Shape;
class Card;
```

- **Methods**: Use verbs for method names and use lower camel casing. For example:

```
shuffleCards();
createDossier();
```

- **Variables**: Use meaningful yet short names for variables. Use lower camel casing. For example:

```
int myBasket = 0;
char lstName = "";
```

For throw-away variables, like an index in a loop or a temporary variable, use a one letter name. For integers, use i, j, k, and m. For characters, use c, d, or e. For example:

```
for (i=1, i<10, i++) {
    console.log ("Number:" + i);
}
```

- **Constants**: Use full upper case letters and separate words with an underscore (_) for constant names. For example:

```
static final int MIN_SIZE_WINDOW = 640;
static final char PATH_TO_INTEL_SERVER =
"ENV-XXXXXLAIOUSE1";
```

- **Filenames and folders**: For code files and asset files, always use lowercase without spaces. Underscore is allowed but no hyphen. Use the appropriate extension for the file type. For example:

```
logo.png
styles.css
index.html
config.json
cards.java
cards.class
```

For folder names, use lowercase without spaces. Underscore is allowed, but no hyphen. Keep the name significant and short. For specific usage folders in code, use the following names:

Folder type	Name to use
Library	lib
WEB_INF	WEB_INF
Configuration	config
Styles	styles
Scripts	scripts
Images	images
Source	src

Exercise 5.4: Create a coding style guide

Before your arrival as the Services Architect, it was every developer for himself when it came down to coding conventions. Reading someone else's code was a tedious task. You decided to implement coding standards to promote continuity across teams and projects. In this exercise, establish a set of coding requirements for Lunar Water. For possible answers, see *Answers to Exercises, page 198*.

The following are the style rules you need to enforce for this exercise:

1. Packages use a reverse domain name style in lowercase.
2. Classes and interfaces use full camel casing.
3. Methods use verbs for names and use lower camel casing.
4. Variables use meaningful yet short names with lower camel casing.
5. Constants use full upper case letters and separate words with an underscore (_).
6. A statement with a block of code opens the brackets on the same line and ends with a bracket. For example:

```
if (a > 5) {
    object.doSomething();
}
```

7. Indentation is required for all opening brackets to a new level. Indentation can be two or four spaces (as in the code example above). No tabs allowed.
8. File names, folders, and asset files always use lowercase without spaces, but underscores are allowed.
9. Exceptions to folder names are standards specific to languages. Always use lower case without spaces, and underscores are allowed but no hyphens.

Based on the standards above, review the following code.

1. Do the lines of code follow the standards above? If not, what rule number applies? Write the code line number and the rule broken, for example: Line 1, rule 2.

a Look for three infractions:

```
1   package lunarwater.com.greylanders.character;
2
3   public class Character {
4
5   private int NUMBER_LIVES = 3;
6   public string characterName;
7
8   public static void StartCombat() {
9   if (currentLife >= 100) {
10      lifeForce += 200;
11  }
12  }
13  }
```

_____

_____

_____

b Look for five infractions:

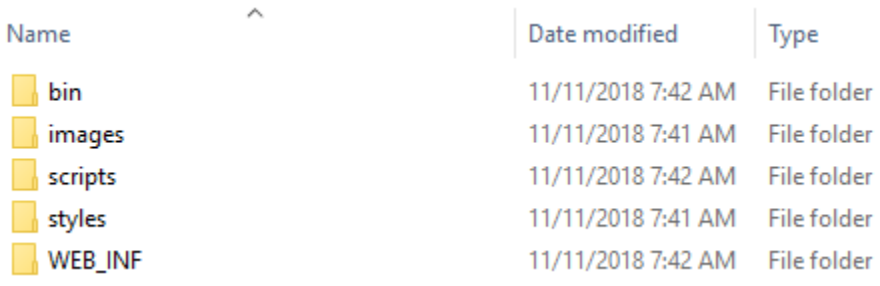
```
1   package com.lunarwater.greylanders.character;
2
3   public class landscape {
4
5   private string PATH_TO_DATA = 'C:\lunarwater\config\';
6
7   public static int calculateLifeForce() {
8       public int XP = 0;
9       if (currentLife >= 100) {
10          lifeForce += 200;
11      }
12
13  }
14
15  public static void Level() {
16
17  private int thisVariableNameIsOkButWayTooLongYouKnow = 1
18      if (currentLife >= 100) {
19          lifeForce += 200;
20          if (levelTime <=0) {
21              game.gameover();
22          }
23      }
24  }
25
26  }
```

c Look for four infractions:

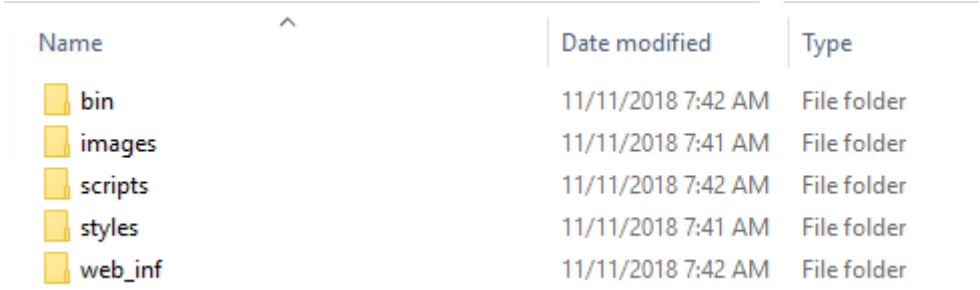
```
1   package com.lunarwater.greylanders.Character;
2
3   public class Guildmember {
4
5   public static void loadlevel(){
6
7       private string PATHTODATA = 'C:\lunarwater\config\';
8       if (currentLife >= 100) {
9           lifeForce += 200;
10      }
11  }
12
13  }
```

2 In the following images, find the errors in the folder and file naming conventions.

a

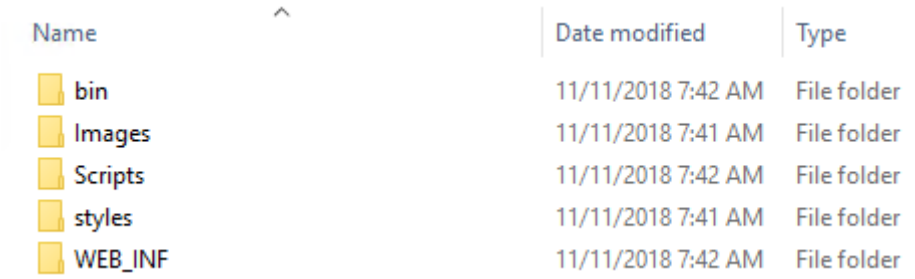


b



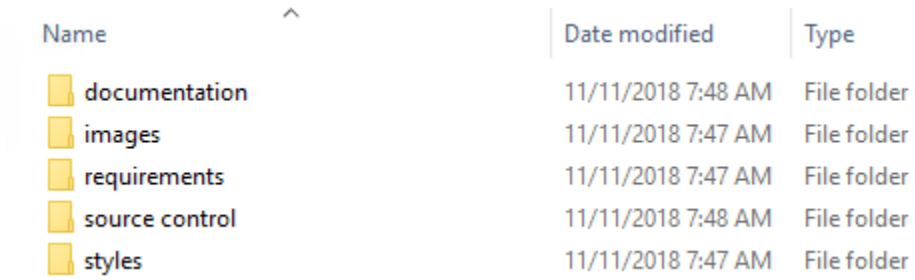
Name	Date modified	Type
bin	11/11/2018 7:42 AM	File folder
images	11/11/2018 7:41 AM	File folder
scripts	11/11/2018 7:42 AM	File folder
styles	11/11/2018 7:41 AM	File folder
web_inf	11/11/2018 7:42 AM	File folder

c



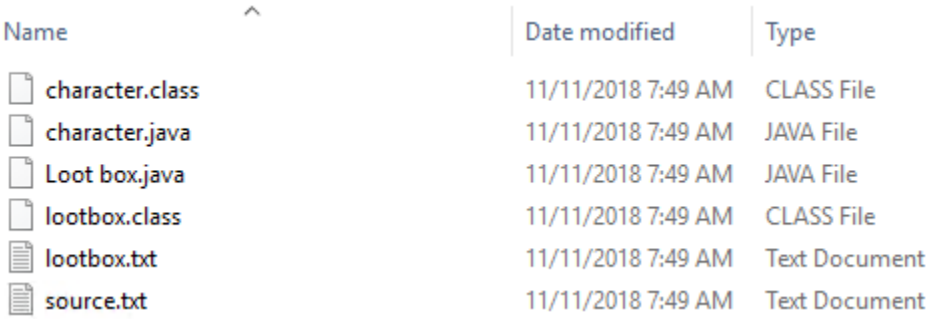
Name	Date modified	Type
bin	11/11/2018 7:42 AM	File folder
Images	11/11/2018 7:41 AM	File folder
Scripts	11/11/2018 7:42 AM	File folder
styles	11/11/2018 7:41 AM	File folder
WEB_INF	11/11/2018 7:42 AM	File folder

d



Name	Date modified	Type
documentation	11/11/2018 7:48 AM	File folder
images	11/11/2018 7:47 AM	File folder
requirements	11/11/2018 7:47 AM	File folder
source control	11/11/2018 7:48 AM	File folder
styles	11/11/2018 7:47 AM	File folder

e



_____

_____

Publish the Coding Style Guide

You have created the Coding Style Guide, and you want to make it public to all team members by putting it on your portal. Follow these steps.

1 In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 5**.

2 Copy **coding_style_guide.txt**.

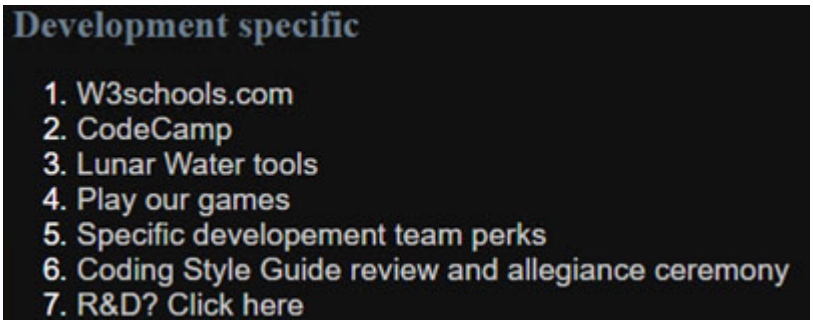
3 In your portal, paste the Coding Style Guide document in **Team Admin**.

Update the on-boarding site

You need to add the Coding Style Guide as part of Lunar Water's new hire on-boarding. Follow these steps.

1 Return to **index.html** in your Lunar Water web site.

2 Add one entry for **Coding Style Guide** in **Development specific**.



Discuss style choices

1 A thorough Coding Style Guide can easily contain a hundred rules or more. What is your implementation approach: do you train team members in one go, or do you break up the standards and establish them one logical section at a time?

2 If you arrive in a team that is set in its ways, what is your strategy to ease the transition between what exists and the standards that are needed?

3 If you had to concede a rule in your own Coding Style Guide, which one could you let go? Which rule would you fight to keep?

Optional: Apply established coding standards

Up until now, the files and folders you created in the portal do not all follow the conventions described here. On behalf of your responsibilities to Lunar Water, go back to files and folders we've created so far for this class and rename them appropriately based on our established standards.

Commenting code

Most developers discover the benefits of commenting code when they have to return to their code after some time has passed and they lose valuable time re-discovering what they meant when they originally coded it.

The Services Architect must establish standards for commenting code to support effective and efficient revisiting and maintenance of older code, often by people

who had nothing to do with the code when it was new. Code commenting standards have to manage coding behavior for many people.

Standards for code commenting support the following corporate goals for corporate ownership and maintenance of code assets:

- **Better understanding**: By adding comments at the beginning of a section of code, the developer can explain in context what the code is about and how it works, for those who come after.
- **Justification**: A code comment can help others understand why this was done that way. Comments allow the developer to provide the why and detail the specific reasons this code was added or modified.
- **Keep track of fixes**: When a fix is added to the code of an application, the developer can use comments to justify the code change and leave details like the number for the defect that was fixed, the developer name, and the date the fix was implemented.
- **Assist in documentation**: Some comments, when placed in the right format, can be automatically picked up by specialized applications built to scour the code and extract comments and turn them into documentation. Additionally, leaving extra details to explain how a method works in the context of end users assists the documentation team to better relay the intent of the code by rewriting the comments for an end user point of view.

Commenting standards: Placement

The following sections provide recommendations that should have a place in any organization's Coding Style Guide.

There are two places in a code file where comments should be located:

- Beginning of the file
- Before the targeted code

Beginning of the file

Place comments at the beginning of a code file to immediately identify the file and its content. This information might be required from comments containing the following:

- Name of company
- Date file created

- Description

```
//-------------------------------
//                               |
//                               |
//         Lunar Water           |
//                               |
//        January 13 2017        |
//                               |
//                               |
//          Card class           |
//       Definition of card      |
//                               |
//         Sam. K Bauer          |
//-------------------------------
```

Best Practice

To make it simple, Lunar Water included this bit of code in their snippet repository so developers simply copy, paste, and modify the block for each new file.

Before the targeted code

A comment should be inserted before any key element of the code, to explain the how and why of the subsequent code. For example, below is an extract of code from a Lunar Water code file.

```
var cards = []; //Array containing all available cards
var oneCard;


//Load the latest definition of cards from central
repository
cards = loadCards();
//Display all cards
for (oneCard in cards) {
    //Skip the cards marked as seasonal or events
```

```
        //Defect 1132 03/11/2018
        //JD - Reviewed: ST 3/12/2018
        if (oneCard.Seasonal === true) {
        continue;
        }
        //Display the card in UI
        displayCard(oneCard);
    }
```

In the code extract above, the special condition on seasonal cards is explained and you see a reference to a logged defect and a trace of the code review.

Best Practice

All comments in source files can be removed for performance and file size using a minifier for code. This process is done just before deployment and before a final round of QA.

Best Practice

Best practices for commenting code

The following are additional recommendations that should have a place in any organization's Coding Style Guide.

- Do not comment each line. Always comment the why. Comment the how if it is a tricky piece of code.

  Use judicious comments that explain the purpose of one line or a block of lines. For example, this block of code needs no comment:

```
    for (i=0, i < cards.length, i++) {
        cards.displayCard(i)
    }
```

  However, this one does:

```
    //Loops through all cards
    //If a card is in the player's deck
    //and in the opponent's deck
```

```
//and player's chance dice better than opponent's
//Opponent loses that card
for (i=0, i < cards.length, i++) {
    if (currentDeck.includes(cards.findCard(i)) ||
        opponentDeck.includes(cards.findCard(i)) ||
        playerDice(chance) > opponentDice(chance) ){
            opponentDeck.removeCard(cards.findCard(i));
    }
}
```

- Specify when code represents a temporary fix.

  If a piece of code is to counter a flaw in the browser, for example, or a patch for a specific client situation, make note of that fact. These pieces of code may often need to be removed later so a comment can make that task far easier and more likely to be done accurately.

- Add any related task number or defect number for easy tracking.

  Adding the work item number or defect number to a piece of code change helps development teams track work-related information and report on it. It also helps later code maintenance where the impact of a single defect can be identified easily.

- Add pending tasks in To Do comments, so that these code related tasks are easy to report on and keep track of.

  Most IDEs allow special types of comments that can be easily searched, for example, **//TODO**. These types of comments reflect project progress and can be helpful for managers.

  Ensure your coding standards require developers to not leave that type of comment in a completed file.

- Consider using automated tools that comb through source code and extract code comments, for easy reviewing. Use this type of tool to spot-check developer work. Popular tools include Doxygen, Natural Docs, Javadoc, and so on.

Exercise 5.5: Standardize comments in code

To help you prepare to create a set of standards for code comments, practice commenting code to think about useful comment content. Weed out comments you consider unnecessary.

Define useful comments

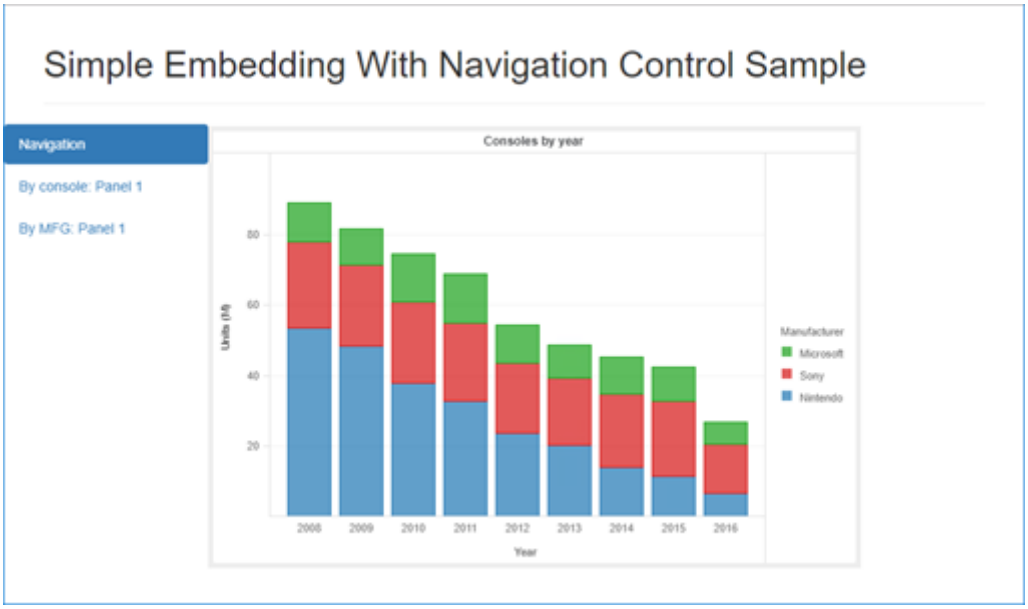
1 In Notepad++, open the file **Desktop\SEA_ExerciseFiles\Chapter 5\create_chapters.txt**.

Its content is provided below for reference.

```
function createChapterPageNav(dossier, div) {
var navData = dossier.getTableContent();
var pill = document.createElement("ul");
pill.className = "nav nav-pills nav-stacked";
var child = document.createElement("li");
child.className = "active";
var href = document.createElement("a");
href.href = "#";
href.innerHTML = "Navigation";
child.appendChild(href);
pill.appendChild(child);
for (var i = 0; i < navData.chapters.length; i++) {
    var chapter = navData.chapters[i];
    for (var j = 0; j < chapter.pages.length; j++) {
        var page = chapter.pages[j];
        var newChild = null;
        var newHREF = null;
        newChild = document.createElement("li");
        newHREF = document.createElement("a");
        newHREF.href = "#";
```

```
            newHREF.innerHTML = chapter.name + ": " +
page.name;

            newHREF.id = page.nodeKey;

            console.log(newHREF.id);

            newHREF.onclick = function() {

                console.log("trying to navigate to: " +
this.id)


dossier.navigateToPage(dossier.getPageByNodeKey(this.id
))

            };

            newChild.appendChild(newHREF);

            pill.appendChild(newChild);

        }

}

div.appendChild(pill);

}
```

This code is used to create the Table of Contents for a dossier in a second (in addition to the containing the embedded dossier). Below is a picture to help you visualize the result.



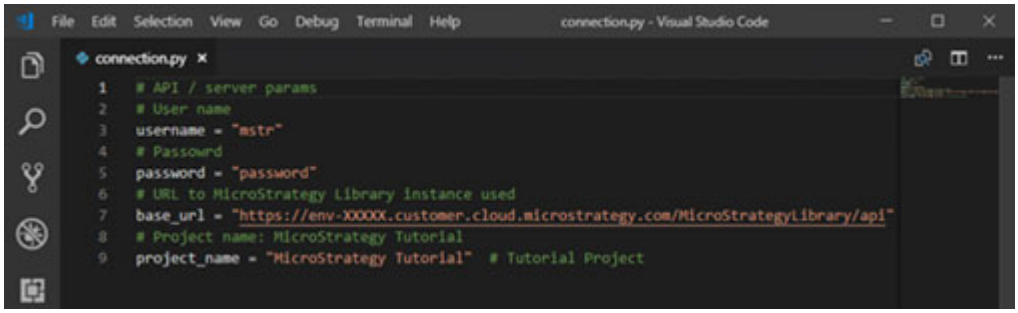
2 Add a header comment to the file.

If an automated documentation tool were in place, the header format would be modified with a tool that crawls the code for information. Assume for this exercise that there is no such tool.

3 Add useful comments to the function, per the recommended standards above, and be sure to leave information on how the various parts work. Compare your comments with solutions in *Exercise 5.5: Comments in code, page 203*.

4 In **Desktop\SEA_ExerciseFiles\Chapter 5** right-click **connection.py** and click **Open with Visual Studio Code**.

Visual Studio Code opens as show below:



The content is listed below for reference.

```
# API / server params

# User name

username = "mstr"

# Passowrd

password = "password"

# URL to MicroStrategy Library instance used

base_url = "https://
env-XXXXX.customer.cloud.microstrategy.com/
MicroStrategyLibrary/api"

# Project name: MicroStrategy Tutorial

project_name = "MicroStrategy Tutorial"  # Tutorial Project
```

5 Remove the comments you consider superfluous and compare your answer to the proposed solution in the appendix. Did you leave more or fewer comments than the proposed solution?

Update the Coding Style Guide

As Lunar Water's Services Architect, you now have an idea of the level of commenting you want in code. Update the Coding Style Guide with the new standards.

1 In your Lunar Water Portal, open **Desktop\Lunar Water\Team Admin\ coding_style_guide.txt.**

2 Add a section at the end of the document about your standards for commenting code. Include rules for headers of code files and for comments inside the code.

3 Save the file.

Group discussion: Taking comments further

1 Comments can be specially formatted for external products, like Javadoc or Swagger. How do you balance the effort versus the benefits for your team in taking the time to add and format additional content for these types of tools?

2 Commenting tends to get tossed to the side when time is short. What strategy can you put in place to instill in your developers that, even under a time crunch, commenting is vital?

Best Practice

Best practices for a Coding Style Guide

The following are best practices implemented at Lunar Water that you can add to any organization's Coding Style Guide. This list of best practices goes beyond naming conventions and commenting code, to cover a variety of coding goals and coding product types.

General

- Practice multi-tier development. Use the following layers to separate functionality:
  - Presentation layer
  - Business layer
  - Data layer

- No hard coding. Use constants/configuration values or parameters.
- Use enumerations to group similar values.
- Avoid multiple if/else blocks. Use a Switch statement where applicable.
- Use an existing framework or existing code wherever possible instead of writing custom code. Check the Snippet repository.
- Find and fix code while writing code. Never commit broken code.
- Single Responsibility Principle (SRS): Do not place more than one responsibility into a single class or function. If necessary, refactor into separate classes and functions.
- Use the Open Closed principle: While adding new functionality, existing code should not be modified. New functionality should be written in new classes and functions.
- Use the Liskov substitutability principle: The child class should not change the behavior (meaning) of the parent class. The child class can be used as a substitute for a base class.
- Interface segregation: Do not create lengthy interfaces; instead split them into smaller interfaces based on the functionality. The interface should not contain any dependencies (parameters) that are not required for the expected functionality.
- Dependency Injection: Do not hard-code dependencies; instead, inject them.

Coding practices for MicroStrategy plug-ins

- Use Best Practices Fundamentals outlines in the MicroStrategy Community site at:

  https://lw.microstrategy.com/msdz/MSDL/GARelease_Current/docs/projects/WebSDK/Content/topics/bestpract/BP_Use_best_practices_fundamentals.htm

- Use privileges and MicroStrategy Web features: Intelligence Server privileges and permissions govern how much functionality a user has in a project.
- Be aware of options for portal integration: Use the MicroStrategy portal integration kit to integrate Web with any enterprise portal. There are two approaches to portal integration.
  - Leverage MicroStrategy Web's presentation features using URLs
  - Use the MicroStrategy platform as a data provider

- Know when and how to use the External Security Module: To customize authentication and authorization, the best practice is to use the External Security Module.
- To minimize the impact of upgrades and to avoid losing your customizations, follow these practices before changing any files (configuration, JSP or ASP.NET, images and so on):
  - Use the Web Customization Editor provided by MicroStrategy SDK for all your customization needs. This editor creates customization plug-ins that reduce the need for maintenance and facilitates easy upgrades.
  - Do not modify the page template to create new page sections. Reuse existing page sections to add or modify content rather than creating new page sections. For example, if you want to add content to the footer of a MicroStrategy Web page, reuse the page section called Footer used by that page or the default page, rather than create a new page section called footer2.
  - When upgrading, check whether there are newly deprecated API entities that you have used in your customizations to older MicroStrategy versions.
  - Transforms should be used for rendering data. All logic-related changes should be done using add-ons and events.
  - Create or duplicate new pages by inheriting from existing pages rather than making copies of existing pages. The Web Customization Editor allows you to create new pages by inheriting from existing MicroStrategy Web pages.
  - Do not modify out-of-the-box files JavaScript files. To add custom JavaScript methods, make sure that these custom files are placed together in a common but separate customization folder and referenced from the *links* page section of the page in which you wish to use them.

    Unlike Java classes, JavaScript methods are not backwards compatible. Any customizations performed on out-of-the-box JavaScript files have to be manually re-applied after upgrading your custom Web application.

Code optimization

- Maintainability of embedded solution: The application should require the least amount of effort to support in the near future. It should be easy to identify and fix a defect. Maintainability is determined by factors such as:
  - Readability: Code should be self-explanatory

- Testability: Solution should be easy to test
- Debuggability: Solution should provide exception details to find the root cause easily
- Configurability: Solution should not change code when values are modified
- Reusability: No repetitive code in solution, eliminate any redundant solutions
- Reliability: Solution should have exception handling and clean up resources
- Extensibility: Solution should be easy to add enhancements to with minimal changes to code
- Security: Solution should be up to date with latest security requirements and vulnerabilities
- Performance: Code should contain optimizations to enhance return time on all solutions
- Scalability: Solution should support a growing infrastructure of users and data
- Usability: User interface/API should be easy to understand and use

Code samples and demos

- Write with documentation and training in mind, not just for a Proof Of Concept (POC).

Maintaining code

Post-code best practices in an Intelligent Enterprise include reviewing code to ensure all the standards put in place are being followed; and build management following best practices.

Managing builds

The operation of compiling code into an application produces a build of the application. After a build is made, as soon as new code is checked into the code repository, the process of preparing a new build has started.

An application is a living entity whose progress must be assessed and features reviewed and tested as early as possible. In a prototype scenario, you need to gauge which build is a work in progress or worthy to be the prototype you are ready to show stakeholders.

To achieve a successful and useful progress assessment, a tracking process must be established that is used for all builds.

The Services Architect establishes guidelines to control the builds. Which one goes to QA, which one do you want to present to a client for feedback, and which one becomes the official release version? These decisions must be made consistent across the enterprise and must be based on solid, functional foundations that are documented for easy reference and regular use by all.

Tied to source code management

Source code management is covered throughout this chapter. When code is committed into the source code repository, the possibility exists of creating a build with all the files present in the current branch. During a coding period, building with incomplete features may not be useful depending on the purpose of the build. The build may be successful, but you may not have the behavior expected.

For example, build management documentation may require developers to commit only functioning code to the main branch, even if the feature is incomplete.

Automated build management

The build process can be triggered manually by a Code Manager or Build Manager, or can be automated and scheduled. Tools can provide this functionality.

Define a build schedule with a mix of manual and automated builds that are appropriate for corporate goals and your development methodology. Some build management products also provide continuous integration (CI) support, if you plan to use CI.

Continuous integration (CI)

When a developer gets approval to check code into a version of the branch, the longer that person waits until committing changes, the more commits from other developers are also occurring. The high number of changes being integrated into the code branch at the same time can cause “merge chaos”. To avoid this, Build

Management documentation can include a development policy of continuous integration (CI) by requiring a daily commit schedule or multiple times a day.

This type of policy can include:

- Build merges support by a build server.
- A unit test completed on the developer's environment before committing.
- Feature toggles applied to ensure the build contains stable features. These can include decisions on what to expose, or to withdraw a feature without jeopardizing the rest of the application or component.

If CI is part of your organization's development methodology, work with the System Administrator and Code Manager or Build Manager to configure a CI system.

Best Practice

Best practices for build management

The following list is a common starting point for requirements in your Build Management documentation:

- Set a build schedule
- Use a daily build to test
- Review all build reports with appropriate stakeholders
- Enforce unit testing before code is committed (checked in)
- Implement a code commit schedule

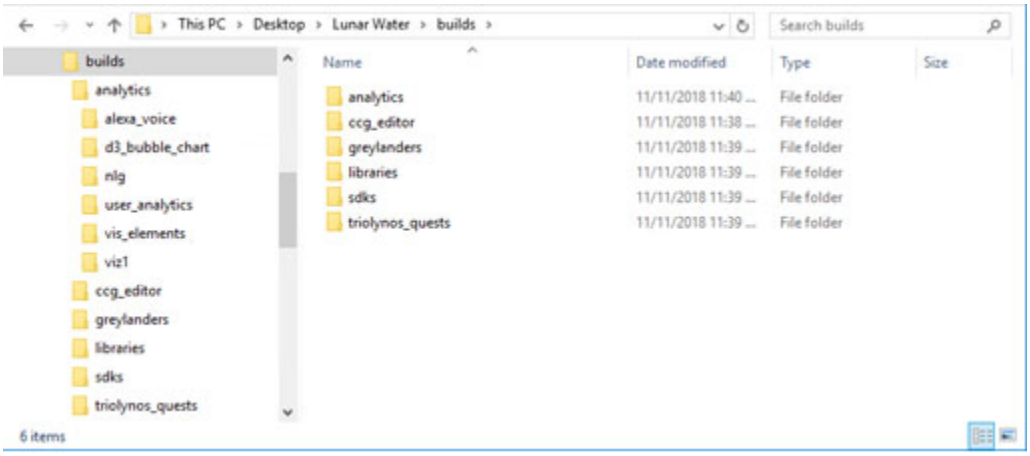
Exercise 5.6: Practice build management

In this exercise, you define a portal structure to include the builds for the different Lunar Water products.

Define a high-level structure

1. In File Explorer, navigate to **Desktop\Lunar Water**.
2. Create a folder named **builds**.
3. In this folder, create the following hierarchy:
   - triolynos_quests
   - ccg_editor
   - greylanders
4. Optionally, add the following folders to the hierarchy.
   - analytics
     - viz1
     - user_analytics
     - vis_elements
     - d3_bubble_chart
     - nlg
     - alexa_voice
   - libraries
   - sdks

Once completed, your structure should look like the picture below:

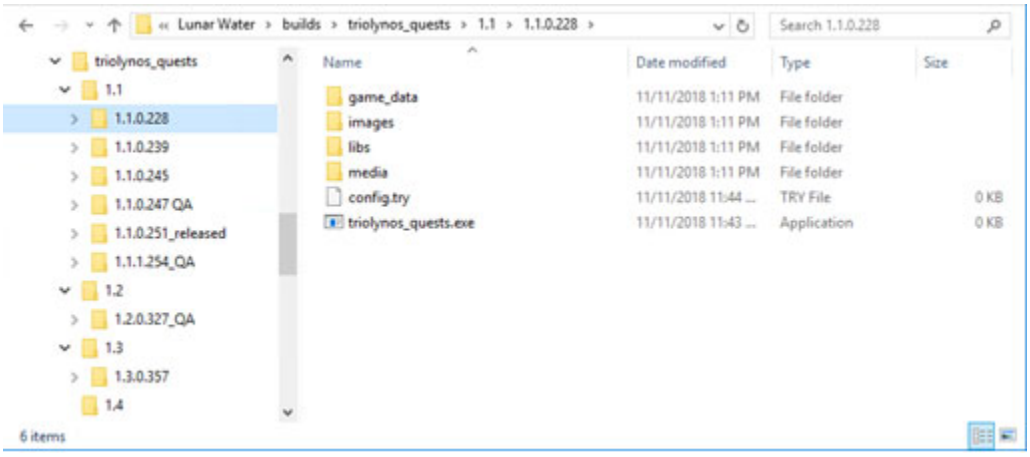


Define builds for products

Working with Lunar Water's Build Manager, you discover a trove of build folders with no significant names. After identification (and eliminating duplicates), you name the folders based on your new standards, and separate them by build version.

1. In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 5\builds\triolynos_quests**.
2. Copy the four folders.
3. Paste them into **Desktop\Lunar Water\builds\triolynos_quests**.
4. Navigate to **Desktop\Lunar Water\builds\triolynos_quests\1.1\1.1.0.228**.

If you expand the left pane to see all versions, your picture should look like this:



Take a moment to ensure you understand your naming and organization conventions in practice:

- For Triolynos Quests, you have four versions of the software. Remember the currentprojects.html page? Consult it if necessary.
- Version 1.1 was released. In the 1.1 folder, you can find a history of the series of builds. You see build 1.1.0.247 made its way to QA. For now, there is no way to tell how it went from here. We discuss that later.
- Version 1.1.0.251 was released to the public. The official version you and the Code Manager used is 1.1. Then you have version 1.1.1.254, a potential patch that made it to QA. Again, you can't gather further details from reviewing folders alone.
- Each build folder contains a complete set of files to run the software. This is not the place for source files, as they are managed in the source code repository.
- Two other versions (1.2 and 1.3) reflect the status in the current products page. 1.2 is in QA and 1.3 is still in development and does not have a build for QA yet.
- Version 1.4 is in design, so it is no surprise to see there is no build available. Per your established standards, the 1.4 folder was created as soon as the project debuted.

Create a Builds page on your site

As the above exercise makes clear, looking at your portal for build information is only half the story. To manage all the projects and versions, and have a complete portrait of their statuses, you need your Lunar Water site.

If you take advantage of embedded analytics, you could have an automated system that pulls data from the automated testing system, or you can read a data source maintained manually and queried on a schedule. With that data, you can create a dossier that you embed in the HTML page and use analytics to manage builds and project statuses.

1. Navigate to **Desktop\SEA_ExerciseFiles\Chapter 4.** Duplicate **currentprojects.html**.
2. Rename the file **buildhistory.html**. Copy the file.
3. Navigate to **C:\tomcat\webapps\ROOT\LunarWater**. Paste **buildhistory.html**.
4. Edit the file in Notepad++.
5. Create a section for each of the products:
   - CCG Editor
   - Greylanders
   - Triolynos Quests
6. Optionally, create an **Analytics** section.
7. Use the table below to adjust the HTML code and fill the tables in the page.

Version	Build Date	Dev	QA	Date Tested	Defects List	RC	GA	GA Date
Triolynos Quest								
1.1.0.228	09/27/18	X						
1.1.0.239	10/03/18	X						
1.1.0.245	10/07/18	X						
1.1.0.247	10/08/18		X	10/09/18	DE1024, DE1027			

Version	Build Date	Dev	QA	Date Tested	Defects List	RC	GA	GA Date
1.1.0.251	10/11/18		X	10/11/18		X	X	10/12/18
1.2.0.327	10/04/18		X	10/04/18		X		
1.3.0.357	10/10/18	X						

8 Save the file.

Discuss build management

1 The build structure presented above for Lunar Water is one approach for managing builds. What would you change, and why?

Defining a code review process

Establish procedures for development teams to follow a repeatable code review process to evaluate all new code against established standards, and require a spot check for code that might be identified as not needing a full review. Automatic style checkers can be part of this solution as they refuse to incorporate into the code repository any code that does not match verification rules.

Standards documented for required code review should include the following requirements:

- Review should be completed by a developer who was not involved in writing the code being reviewed.
- At least two people should review code against the established standards.
- If an automated code review tool is used, it should always be understood to be performing a partial code review, and manual review must follow for standards an automated tool cannot review. Common areas for manual review include design specifications that were not fully implemented and security breaches.
- Code reviews must be completed before any code is integrated into the main branch of the source code repository.

Best Practice

- A successful Intelligent Enterprise requires that all issues caught during reviews are documented, so that ongoing team training can be focused on specific improvements in developer knowledge and practices. Additionally, these saved data points can be analyzed in a dossier to determine larger

trends among development teams, may support requests for deeper training or new tools, and can provide proof of success in improvement programs and trainings.

Common code review areas include, broadly:

- Inefficient code that can be improved
- Code segments that pose a potential security risk
- Code that does not follow standards in the Coding Style Guide

Automated code reviews

An automated code review is always considered a partial code review, as automation tools cannot completely perform every review you need. A complete code review demands a human presence. For example, finding an oversight in a design element, a missing user requirement, or a situation not present at design time, requires a manual review to identify and know how to address the situation.

Automated reviews are good at verifying coding rules programmed into the verification system. This is known as static code analysis. Such a system can also identify classic patterns of bad coding habits.

Automated code review tools should be added to your processes in communication with the System Administrator, as well as a Code or Build Manager, because automated tools can perform automated verifications pre-commit and post commit. A Code or Build Manager may need to adjust settings on the source code management system to work well with the code review system. Common tools include Codeplex, Git Lab, Rational TeamConcert Code Review, Visual Studio Code Analysis, and others.

Adding to the snippet repository

Establish a requirement for development team leads to gather all code snippets the teams can identify and decide if they can be useful to, and should be made available to, the organization. It is often useful to have a scoring system to identify the most useful code snippets.

Work with the System Administrator to post the snippets into a repository in an internal portal, or to a database for a snippets manager solution. Tools can include desktop applications and web sites, such as {CodeStore}, Code Bank, GitLabs Snippets, and many others.

Remember that some code snippets might qualify as being useful to customers and can thus be posted to a customer-facing location such as a Community site.

Best Practice

Best practice to promote snippet creation

Promote snippet identification by rewarding people that take the time to develop a snippet useful to all development teams. Teams can nominate best contributors per release.

Managing assets

Assets represent common files programmers use inside a project that are not code. For example these files could be:

- Corporate graphics
- Corporate logos
- Sound
- Global copyright .txt file
- Reference documents for development
- Published applications and components for internal consumption

Often these assets are owned by other departments in the organization, such as Marketing, Graphics, or Legal. Developers are a consumer of these assets. These assets are different than specific graphics used in an application that belong in source code.

The Services Architect should include the periodic, probably release-based, review of these assets within any Coding Standards documentation. Reviews should include the following actions:

- Communicate with the owning department on whether a new version is available or other updates have occurred.
- Communicate across development teams to update their asset repositories accordingly.
- Update assets with new versions, in an easily accessed assets repository, within a folder hierarchy that makes assets easy to find and identify their purpose.

- Ensure that the files in an assets repository are the original file and cannot be referenced in code. Source code must use a copy located in an appropriate application folder for the asset.

Writing release notes

A release notes document operates as a series of headlines from developers to users. Release notes provide enough information for a user to determine whether a new feature, a change, or a fix is something the user needs more details about.

Best Practice

Make sure your release notes include a link to details for each topic.

The readme files included in libraries your teams build must be managed closely for each version of the library. A readme file must include technical resources for the library.

Summary

One area that directly influences the deliverable (code) of your team members is coding standards. As the Service Architect, you want to establish a series of standards everyone must follow. We discussed the benefits of coding standards, like naming conventions and commenting code. All these standards are part of the Code Styling Guide, a living document that must be maintained to capture new best practices that surface from each release.

We discussed build management and how to make the best of protocols around building your application frequently, including continuous integration. To help development team members, we talked about design patterns they should recognize and implement. Another important challenge in coding is security and we saw that the earlier you put security on the table when you design a product, the easier it is to integrate.

Having a snippet repository promotes re-use of code and establish better code produced by the team. Another resource you need to establish in your internal network is a repository of libraries and assets.

Code review is a commitment you ask from your development team to adhere to the coding standards.

In the next chapter, we turn our focus to the source code and how to manage it efficiently.

6

Maintaining Code with Source Control

This chapter briefly covers responsibility for the code written in your enterprise and the measures you put in place to protect it. We discuss management of source code, from establishing an intelligent management process to necessary skills for team members. We include best practices to successfully protect the code your team members produce.

Maintaining a code repository

A source code management (SCM) solution is a required component of an Intelligent Enterprise. A primary objective is to preserve every effort development teams have made when coding. Source code is considered a corporate asset that must be managed and protected.

Source code management is a set of processes designed to achieve the following objectives:

- Archive every line of code created by development team members
- Manage change in the code for revisions
- Manage conflict resolution when multiple versions of a file exist

- Provide an environment where code can be assembled into specific versions of an application

Implementing a source code management solution

On a high level, SCMs operate within, and support, development efforts in the following ways:

- Your SCM has one repository per project. Its root (or origin) contains the current code.
- A developer pulls a copy on a local machine to work with. They work off a branch of the root code. Changes are local to this user.
- When ready, changes are committed to the SCM, parked in an area away from the root.
- When appropriate, the branch is merged to the root. If multiple branches conflict, the merge process must be handled manually.

SCMs offer:

- A safe home for code

  A Source Code Management (SCM) system is a central location that stores the code and handles changes of files for every developer. In a sense, there are versions of the code for everyone and at the same time, but a single one copy for a complete version at a time.

- A source of intelligence

  An SCM also serves as a source of intelligence on development operations. Your company can monitor the progress of projects, people, and initiatives using the SCM system logs and monitoring tools. Combined with the good practice of making your developers commit changes often, the progress of any project can remain transparent.

  You can connect your SCM data to a BI platform, to display essential KPIs in a dossier, as well as perform predictive analytics on specific projects.

  A good SCM should be:

  - Scalable, leaving your company room to grow.
  - Supportable, with an eye to the level of skill needed to run the system.

- Key support for code review

The SCM is an important part of code review, which we discussed in a previous chapter. Developers commit code to an SCM. At this point, the code becomes available for review following the code review workflow you have defined for developers.

To implement an SCM system, coordinate with the System Administrator to ensure installation and configuration supports the standards you establish for your development organization's goals.

Standardizing code management

In an Intelligent Enterprise, the Services Architect must establish standards for important processes related to source code management. Examples include:

- Defining a recovery solution for source code
- Providing SCM training to development teams
- Defining and enforcing SCM best practices
- Establishing when each project should deliver versions of products when ready to publish

Best Practice

To help you define SCM best practices, start with the following list as part of a written, stored, and easily available document of source control rules:

- Keep source code files, not the compiled version
- Always get the latest file before working on it
- Only check out what you work on
- Check in as soon as you are done
- Only commit code that works
- Commit often, it offers more rollback options
- Review your changes before committing
- Add a meaningful comment when committing code
- Commit only your own changes
- Add dependencies to source control
- If it is not in source control, it does not exist
- Versionize the database

- Do not commit personal settings to source control
- Do not use comments to save deleted code

Training for SCM solutions

All development team members must have training related to source code management on the following:

- Principles of highly functioning SCMs
- Detailed processes like get, commit, branches, and merge
- Common scenarios using an SCM
- Principles the development teams will use

Establishing source control roles and responsibilities

Common areas of responsibilities include:

- A Code Manager. The Code Manager alone should be managing the code repository for important operations like branches, complex merging, and publishing. A Code Manager may spend a significant amount of time working in the SCM.
- Developer, Technical Support, and Documentation team members: Anyone who contributes to the source content must adhere to all required practices put in place. This includes:
  - What to include in source control
  - Time line of committing changes
  - Practices to avoid unnecessary merge conflicts
  - Report to appropriate team levels any errors or problems with the SCM system to avoid loss of content and efforts.

Best Practice

- Team leads: Every person in your team responsible for other people must ensure they follow the practices in place. On a regular basis, require an audit of the commit schedule.

Exercise 6.1: Create a source code management solution

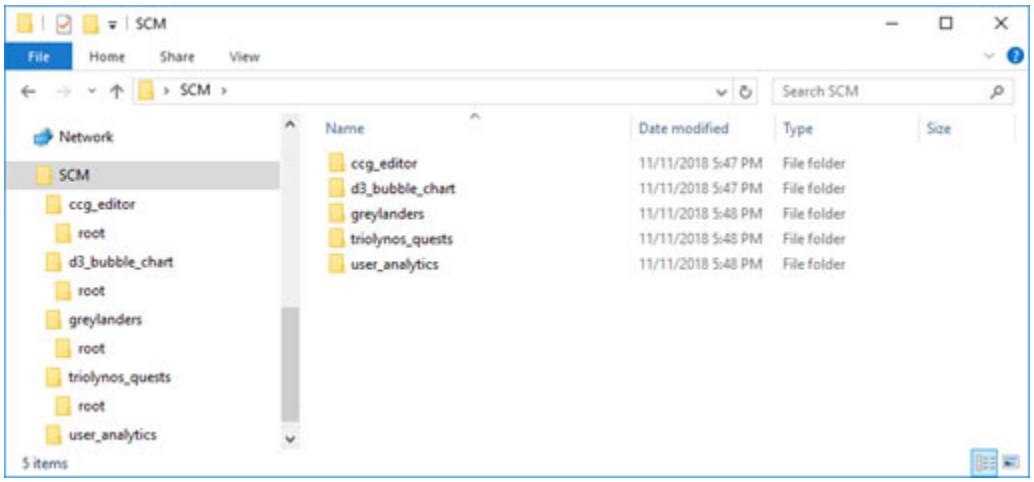
When you started at Lunar Water, source code could be found everywhere, even on employee laptops. Implementing an enterprise-quality SCM takes time and is complex. In this class, we use the metaphor of folders to practice some best practices you can enforce as the Services Architect. We start by creating the repository and the root folders for our main development projects.

Create the repository

1. In File Explorer, navigate to **Desktop**.
2. Create a new folder name **SCM**.
3. Open **SCM**.
4. Create the following folders to reflect current projects:
   - triolynos_quests
   - greylanders
   - ccg_editor
5. Optionally, create these additional folders:
   - d3_bubble_chart
   - user_analytics

   The book shows as if all folders are created throughout the class.
6. Navigate to **Desktop\SEA_ExerciseFiles\Chapter 6**.
7. Copy **code_files**.
8. Paste a copy into each of the folders in **SCM**, then rename each of them **root**.

After this step, your repository should look like the picture below.



Congratulations! You have a sample structure for an SCM in place.

Adjust on-boarding

1 Navigate to your Lunar Water site.

2 Open **index.html** in Notepad++.

3 Add two entries in the Development specific section to include SCM training. Here is a picture to inspire you.

4 Save the file. Your on-boarding process is now up to date.

Discussion: SCM in the enterprise

1 Where does your source code reside: in-house, on public or private cloud? What solution makes sense to you?

_____

_____

_____

2 Aside from technical skills, what are you looking for in a Code Manager?

_____

_____

_____

Versioning your code

Versioning within an SCM is the process of assigning software that is in specific state, a number or name to identify it. When your piece of software, whether an application, component, library, and plug-in, has achieved a level of functionality that is defined by its specifications, the code can be tagged in this state with a version number and be published.

Version numbers are your way to track evolution of the software, and this is reflected in your SCM system.

Establishing versioning standards

You must establish a logical numbering identification system within your development standards and ensure it is consistently used. The following are some common identification practices recognized by the developer community:

- Basic scheme. Many software are numbered using the following schema:

  major.minor.revision.build

  You might encounter schema using revision and build numbers in slightly different ways. What is important is that the scheme used has a specific meaning and is used consistently.

- Semantic versioning. Another scheme is published online under a Creative Commons license called Semantic Versioning. It uses in its simple form three numbers as follow:

  major.minor.patch

  For more details on how to use this scheme, consult https://semver.org/

Your versioning standards may include any or all of the following additional considerations. These are provided to give you a starting point on intelligent versioning guidelines for numbering your products:

- Degree of compatibility
- Development stage
- Position of the decimal number
- Separating or joining numbers
- Using the date
- Adding lowercase letters for hot fixes
- Corporate political and cultural meaning of numbers
- Marketing strategy to jump to a specific, meaningful number
- Alignment with another product version number

Prepare a document detailing the versioning scheme and how it reflects in your current projects within the company and projects intended for customer use. An important requirement for the versioning of each product, should be that the versioning scheme should be tied closely to the product's road map in terms of timing of incremental number changes. Train development team members about the scheme and ensure it is used.

Broaden versioning training to the wider organization, as versioning impacts a multitude of business operations, from Tech Support as they prepare for a product launch, to the finance office, to marketing, to Human Resources as they make plans for timing internal trainings or track hiring patterns.

Handling versioning between releases

At Lunar Water, your responsibilities include keeping the root code set pristine and evolving toward the next version. Let's look at some scenarios and think about what standard you would establish for each situation. Some of these are

relatively simple but they are common and thus need established standards for handling.

Scenario 1

You shipped version 2.3 sometime in the past and you just recently shipped 2.4. A client requests a defect fix for 2.3.

Common solution:

- If the defect is already fixed in 2.4: You need to create version 2.3.1 that offers the client a version with the fix for the defect, but allow the client to stay on version 2.3.
- If the defect is not fixed in the root code set: You still need to create 2.3.1, but you also need to put the same fix into your root code set and ship a 2.4.1 release.

Scenario 2

A client of your application wants specific changes to match their workflow in their enterprise. These changes are partially aligned with the road map you have established for this application.

Common solution:

- You can integrate the changes to matching your road map but you need to find a way to deliver a version specific to their specifications without compromising your road map or bloating the application with ultimately conflicting requests from clients.

  Including everything and disabling functionality for other workflows is a complex path that elevates the level of complexity of your software and impacts developers, who must always keep in mind scenarios with disabled features not available mainstream.

The answer to all these challenges is a branch in your repository in the SCM system.

Branching code

A branch is a specific snapshot of the code that is set aside and tagged with an identifier in your SCM system. You can branch for multiple reasons like the scenarios above. They all end up with a branch in the code set.

Best Practice

Branch decisions must consider implications of a branch and whether there are other options to address the specific situation. Establish standards for branching that include the following best practices and considerations:

- Always use the SCM to manage branches
- A branch should never evolve on its own away from the root code set
- Merge often
- Locking files can minimize merges

Exercise 6.2: Create a code branch and a new version

The development of the next version of Triolynos Quest is launching and you are about to manage the SCM implications.

Create a branch

1 Navigate to **Desktop\SCM\triolynos_quests.**

2 Duplicate **root** and rename the folder 1.3.

Your SCM now has a branch the developers are working against for this version. Work is on its way.

A defect is found

The support team reports a defect in version 1.1 that was released to the public. The bug causes the player to be able to see the opponent's deck of cards when the user is of the necromancer class. You assign your team to find a solution. This forces you to make a special branch from root with the 1.1 release files. Your developers find the defect and have a fix for it.

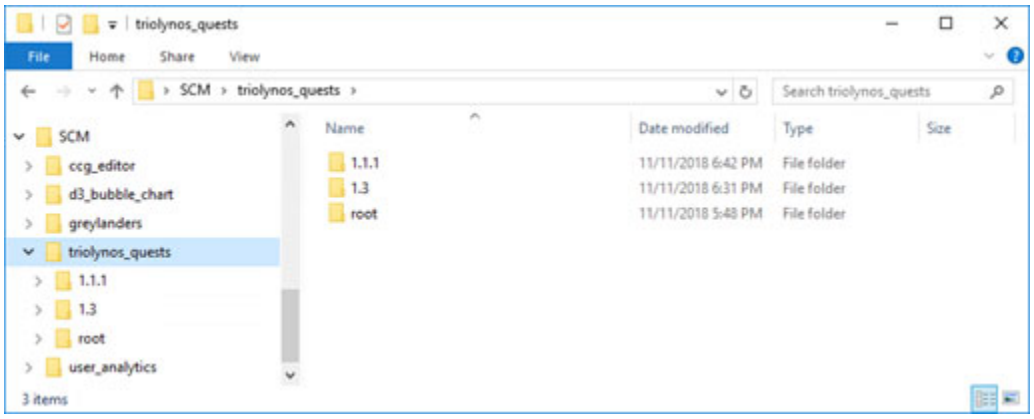
1 Navigate to **Desktop\SEA_ExerciseFiles\Chapter 6**.

2 Copy **1.1.1**.

This represents the branch that you created and the changes your development team have made.

3 Paste the branch in **Desktop\SCM\triolynos_quests**.

At this point, your repository looks like the picture below.



You branched the code and your team found a way to correct the defect. You release 1.1.1 to the general public. Everybody is happy.

Merge the changes

The changes must find their way into the root code to correct the defect in the root of the product. Here are your options:

1. Merge changes between 1.1.1 and the root. Then, later, merge 1.3 to the root to ultimately release 1.3.

2. Merge 1.1.1 with 1.3 now. Then, merge 1.3 with the root, knowing you added to the complexity of merging 1.3 with root with this additional change.

3. Migrate the change to both branches and eliminate the second merging later as the code has already been migrated.

Option 3 represents the best option if the change is not affected in the new development. This is another variable to consider. In our case, the necromancer class is getting a big update in 1.3, so option 2 is more appropriate as the other changes in this version might influence the fix from version 1.1.1. If there is no influence in the new code in version 1.3, then the final merge is not affected in that specific area. Let's merge 1.1.1 and 1.3.

The SCM would offer a merging tool and UI at this point of the process. We use Notepad++ with the Compare plug-in instead.

1. In Notepad++ Open the following files:

- Desktop\SCM\triolynos_quests\1.1.1\bin\ CustomReportsPathTansform.java.
- Desktop\SCM\triolynos_quests\1.3\bin\ CustomReportsPathTansform.java.

Best Practice

As a good practice, open the source document first and the destination second. This places the documents in order from left to right in the Notepad++ tabs.

2 Select the first document, the source file.

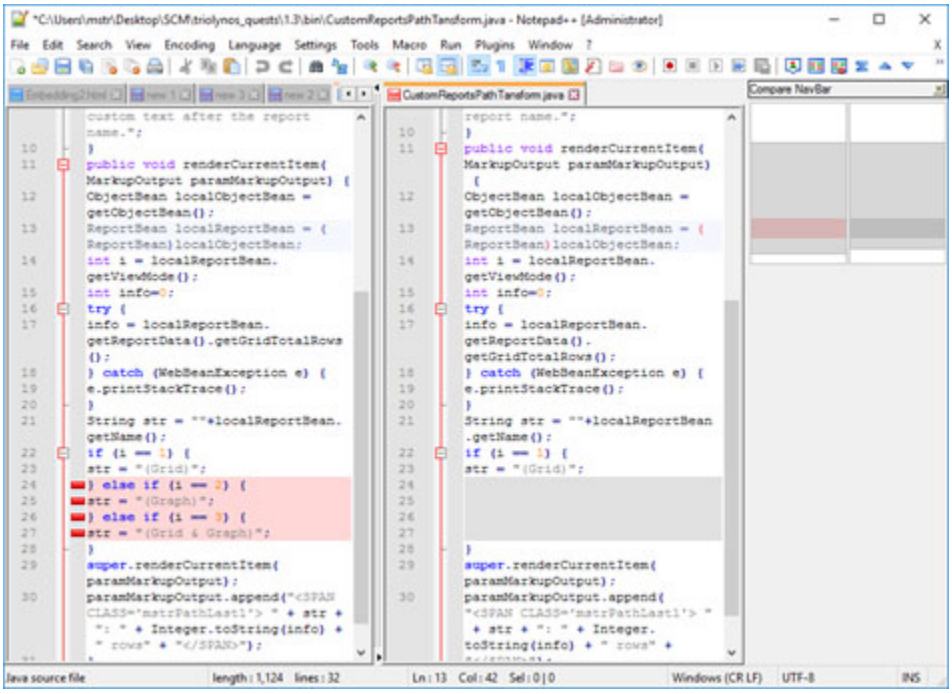
3 In the **Compare plug-in toolbar** (shown below), click the first icon to mark this file as the first document to compare.

The tab for the file adds ****Old** to compare to its title.

4 Select the second document.

5 In the **Compare plug-in toolbar**, select the second icon to launch the compare process.

The plug-in identifies the differences between both files, as shown below.

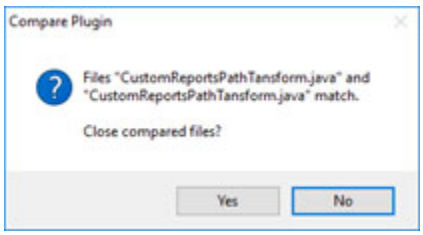


A single change needs to be migrated, and there are no conflicts.

6 Copy the missing lines to the second tab.

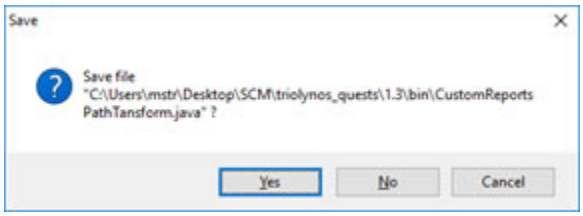
This does cause the tab to lose sync, as shown below.

7 Remove the extra space until both files are identical. Click **Compare** and adjust if necessary until you see the following message:



8 Click **Yes**.

Another dialog appears, as you did change the second file.



9 Click **Yes**. Both tabs close and your file is saved.

Congratulations! You just managed a software change under our sample SCM system. Now version 1.3 has the change brought by version 1.1.1, and when development is completed, 1.3 joins the root, ready for the next version.

Summary

We started this chapter by defining what source code management is. To have a successful SCM system in place, every team member has a role to play, including a Code Manager who oversees the operation of the SCM. As an added bonus, an SCM provides intelligence on your development team and their efforts and can help with operational decisions.

We noticed that an SCM assists in the process of code review. We then turned our attention to versioning, a key element in source control. We looked at a few schemas to number the version of a software or component. This also brought up the concept of branches.

Communicating information about new versions is important as people can prepare internally for a new version, while a new version announcement to the public must be orchestrated.

Branches and merge are two operations to be taken seriously within a SCM system and so must be accompanied by established best practices.

7

Publishing for Internal and External Audiences

It may seem obvious that development information should be documented and the important details are shared with users. However, these vital components of an Intelligent Enterprise are often overlooked or treated as an afterthought.

The Services Architect creating an Intelligent Enterprise establishes a communication system and reliable, repeatable processes to share information with people outside of the enterprise. To do this, you must deploy a communication plan whose end goal is to transparently share the important information users and customers need at the right time, while reflecting corporate policies, branding, and other product guidelines.

Intelligent Enterprises also create and maintain robust internal communication mechanisms to encourage internal information sharing about the development process, product experiences, and so on.

Communicating outside the team

Communication with the public requires a permanent online presence. Consider any community site, blogs, and similar public communication as a type of documentation that reflects enterprise branding and messaging standards, with customers as the audience, along with all of the helpful support and details that users need to be successful.

The Services Architect works closely with Marketing on these types of efforts, to coordinate the types of content that are contributed by the development teams, and to find efficient ways to re-use existing documentation content for this somewhat different audience and purpose.

Two-way communication with the public is also an excellent method to exchange information directly with end users. The direct, immediate feedback from users on your work products can be vital to ongoing development designs and related decisions.

Publishing development work

Talking to the public about your work and the products your development teams produce is vital to making users successful with your products, and to developing the support you need for future projects. Looking at these important goals, it becomes clear that "the public" includes both external customers as well as internal personnel who often have a lot of influence over the organization's projects.

Consider any efforts to publish the development teams' agendas, efforts, and deliverables as achieving not just what your users need, but also supporting your personal goals. The Services Architect should take the communication responsibility seriously as a key component of a successful Intelligent Enterprise, understanding the many levels at which intelligent, maintained communication efforts work on behalf of your goals.

Engaging customers through a community site

A community website can offer an excellent combination of forums, blogs, technical and non-technical content, sample code, sample plug-ins, and many other customer-facing offerings, all while keeping this array of information in a single place for easy monitoring and control of the message.

Best Practice

Best practices for publishing development teams' work include:

- **Tone**: Appropriate level of formality and tone for the publishing method
- **Etiquette**: Appropriate etiquette from everyone contributing information
- **Moderators**: Identify key team members that can be active in the community as moderators. Moderators monitor, curate, and organize community topics to stay relevant.
- **Organization**: Areas might include product features, product issues, sample code, and enhancement wish lists.

- **Kick-off**: Moderators should prepare a large number of starter posts in each area, on a variety of topics, to help make it clear the type of content to be found in each area, as well as jump-start the usefulness of the site in the eyes of users. Moderators should also comment on each others' posts, again to encourage the behavior by users and to provide expert-level additions or related thoughts to what gets posted.
- **Tracking**: Make the site simple to track internally. When defining the structure of the site, make sure to involve other Intelligence Center architects to help define an internal method to retrieve analytics from the site to evaluate its performance.
- **Existing solutions**: Always verify whether a similar problem has already been solved in your Tech Support team's knowledge base. That information is fast and easy to share and keeps customer responses consistent.

  The reverse is also true. Whenever a community site solution is posted to solve a user's problem, bring that information back to the knowledge base for everyone to use next time.
- **Keep the site compelling**: Encourage development team members to post innovative content. Add an incentive for posts.
- **Engage top customer contributors:** In the same way you have a system for the development team to contribute, establish a system for the top 10% contributors from the user side.

Publishing community content

Content destined for end users include but is not limited to:

- Product manuals
- Release notes
- Technical supplements posted in community site
- Public side of knowledge base
- Online help within product
- Code samples
- Sample plug-ins
- Sample graphs, charts, visualizations

An Intelligent Enterprise defines processes to provide these deliverables on the company's site and manages an archive of older versions.

Fostering internal communication

A communication structure must exist so developers can maintain a dialogue with each other for collaboration; so developers and other departments can share product experiences and possible issues; so important dates can be shared, updated, and referenced for company-wide needs; and so on.

Tools for these types of communication can include Slack, Atlassian, and other similar solutions.

Commenting is communicating

Code comments serve as an important method of communication between developers. By commenting the code based on established standards, developers initiate a dialog with the unknown developer next to revise or look at the code.

Best Practice

Best practices

Here is a summary of elements you should consider putting place for a successful communication platform for your team members.

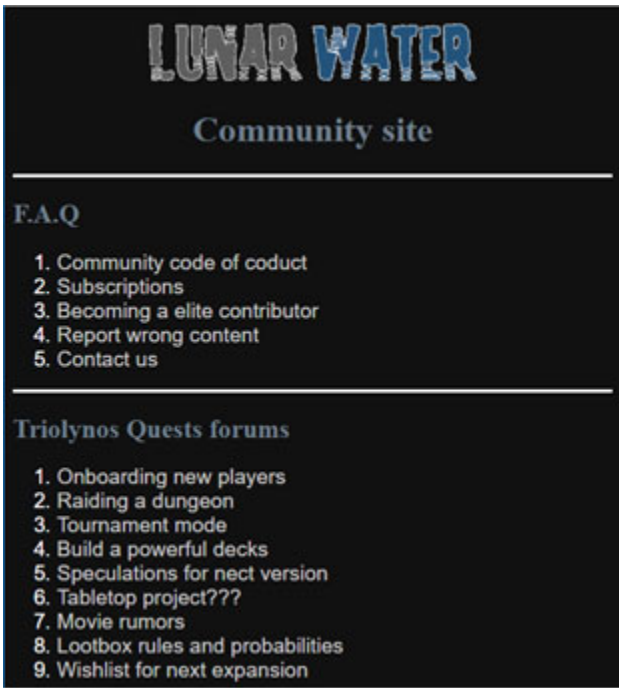
- Limit the number of platforms.
- Organize communication into different channels for operational efficiency.
- Allow read-only guests in channels for people that need to know but not to participate. For example an external contractor doing translations, accessing developer channels.
- Have moderators for channels, like your external community site.
- Moderators should publish cross-channel important information.
- Encourage use of multiple versions of the communication platform (mobile, web, desktop). This makes contributing easier.
- Have a developer use the platform API to pull analytics and push data in a MicroStrategy cube.
- Coordinate with the Application Architect to do telemetry on platform usage.

Exercise 7.1: Create a community site

In this exercise, create the front page for a starter community site for Lunar Water. This page enables your team to respond to messages.

Install the community page

1. In File Explorer, navigate to **Desktop\SEA_ExerciseFiles\Chapter 7**.
2. Copy **community.html**.
3. Navigate to **C:\tomcat\webapps\ROOT\LunarWater.** Paste the file.
4. Open community.html in Notepad++.
5. Adjust the Services Architect suggestions section with the following instructions:
   - Adjust the section title to **Moderators**.
   - Change the three entries to topics dedicated to moderators.
6. Save the file.
7. Navigate to http://localhost:9090/LunarWater/community.html.



Distributing operational data

Best Practice

Part of the communication goals your development teams must meet is to provide operational data internally. This includes the progress of internal initiatives in development, along with related information such as is listed below:

- Current build efforts for products
- Status of testing on latest build
- Stage of development of road map items
- Number of customer issues by level opened
- Performance of support close rate and speed
- Utilization and adoption reports

Work with the Application Architect and the Analytics Architect to build the back end and dossier to showcase these KPIs. Then embed the dossier in a corporate portal page all employees can have access to.

The solution can be as simple as the number of registration of products, or number of hits on your server per day. The goal is to have actionable data about your products to fuel the design of the next steps and initiatives your enterprise wants to invest time and effort in.

For example, have your team code in an API to provide analytics if queried by REST API from a custom connector. Have this response fuel a data cube and work with the Analytics Architect to build a dossier showcasing the retrieved data.

Summary

To communicate to the outside world, a community site is an excellent medium. Internally, you must encourage developer communication so they know what is going on within and across product development.

You also want operational data made available to everyone to help communication and efficiency.

A

Answers to Exercises

Exercise 3.2: Assess skills for a project

Project 1

Skill	Skill	Skill
AJAX	**HTML5**	**GitHub**
Microsoft Word	Java	C#
Adobe Flash	**JavaScript**	Adobe Photoshop
Bash	jquery	Scala
Batch/Shell	Adobe Lightroom	Servlets
Unreal Engine	**MicroStrategy**	**SQL**
Tomcat	**Active Directory**	Swift
C++	Objective C	MicroSoft Excel
CSS3	Perl	Lenell hardware
Adobe XD	Visual Studio	VB6
MySQL	F Sharp	XML/JSON

Skill	Skill	Skill
Adobe InDesign	Adobe Illustrator	XQuery
Gradle	SourceSafe	Unity
ggplot	R language	Microsoft Visio

The application to be built is a web app, so the skills related to web development and code management were chosen. The application is using MySQL as a database, hence the selection of SQL and MySQL as preferred skills. MicroStrategy is being used in the app, so its skill is also selected. Since the application is used by employees, the authentication goes through Active Directory.

Project 2

Skill	Skill	Skill
AJAX	**HTML5**	**GitHub**
Microsoft Word	Java	**C#**
Adobe Flash	**JavaScript**	Adobe Photoshop
Bash	**jquery**	Scala
Batch/Shell	Adobe Lightroom	Servlets
Unreal Engine	MicroStrategy	**SQL**
Tomcat	Active Directory	**Swift**
C++	**Objective C**	MicroSoft Excel
CSS3	Perl	Lenell hardware
Adobe XD	**Visual Studio**	**VB6**
MySQL	F Sharp	**XML/JSON**
Adobe InDesign	Adobe Illustrator	XQuery
Gradle	SourceSafe	Unity
ggplot	R language	Microsoft Visio

The key elements from the project description are:

- Legacy languages
- New Languages

- Transactions ledger
- Web Page
- Standard file formats

These key elements determined why the skills bolded above were chosen. VB6, C++, and Objective C cover the legacy languages bullet. Swift and C# cover the new languages bullet. To manage code, Visual Studio is used along with GitHub. SQL is needed as a new transactions ledger is expected. For standard file formats, XML/JSON becomes another skill to look for.

A web page demands HTML5, JavaScript, and jQuery to be added as desired skills in this team.

Exercise 4.3: Gathering information from clients

1. Here is an example of summary of the conversation:

   To summarize, you want to adjust the look and feel of the dossier coming out of the MicroStrategy Library so they match your brand's corporate assets (logo, color, fonts), while seamlessly adapting to the current game's portal theme. The solution should change the design of all dossiers without having to retouch dossiers manually.

2. Here are examples of follow up questions:

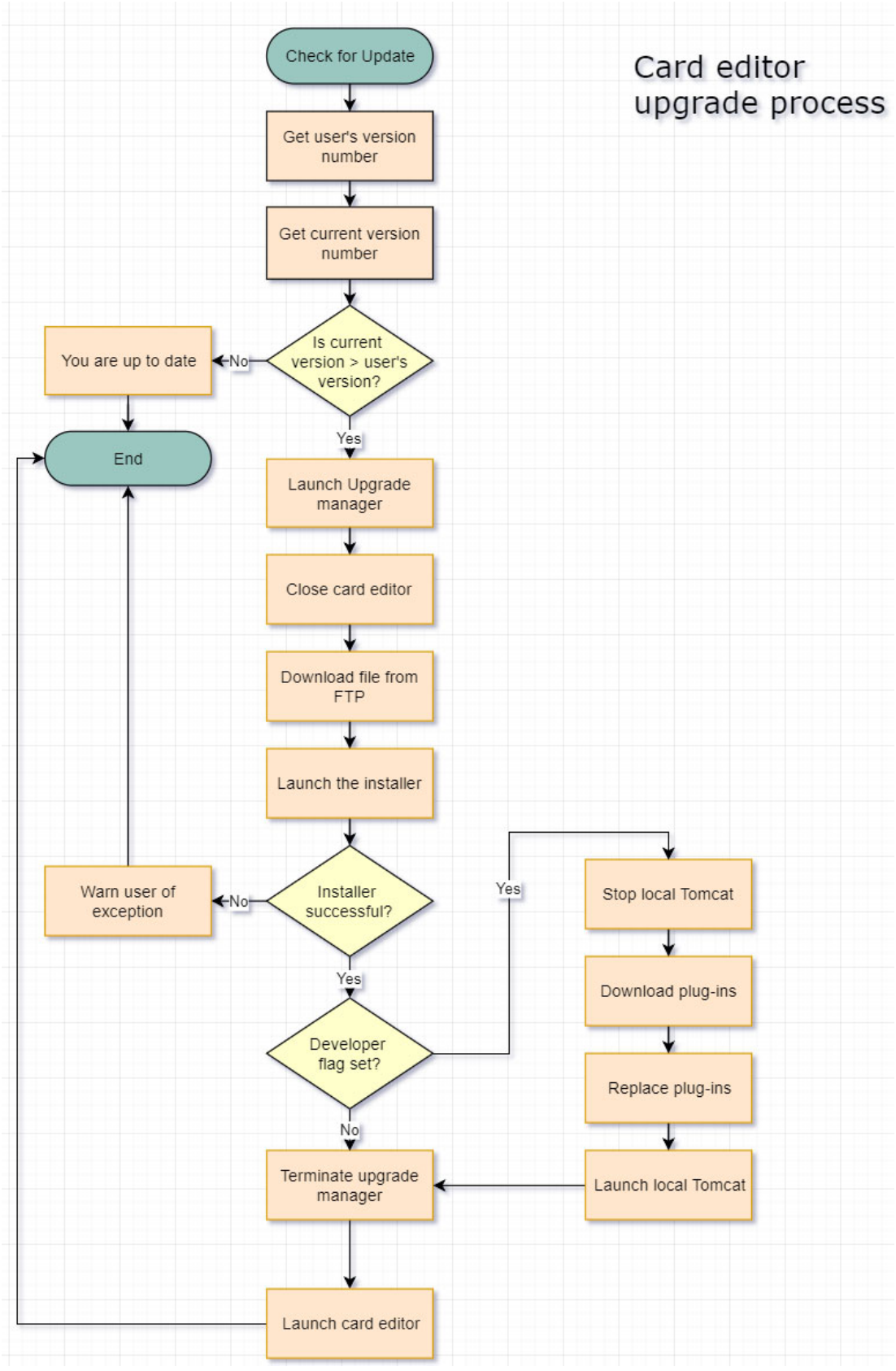
   1. What games are exposed in the Community Portal?
   2. Tell me more about the brand's assets you want to use.
   3. Do you have multiple instances of MicroStrategy Library installed?
   4. How many Intelligence Servers are providing dossiers to the Gamer Community Portal?

3. Here are possible open ended question you might have come up with. Your answers may vary.

   a. Tell me about the theme used for each game in the Community portal.
   b. What plug-ins are currently deployed in your environment(s)?
   c. How do you currently handle .css style sheets in the Gamer Community Portal?

Exercise 4.6: Creating diagrams

Here is one possible diagram for the scenario:


Card editor upgrade process
Check for Update
Get user's version number
Get current version number
Is current version > user's version?
No
You are up to date
End
Yes
Launch Upgrade manager
Close card editor
Download file from FTP
Launch the installer
Installer successful?
No
Warn user of exception
Yes
Developer flag set?
Yes
Stop local Tomcat
Download plug-ins
Replace plug-ins
Launch local Tomcat
No
Terminate upgrade manager
Launch card editor

Exercise 5.4: Coding Style Guide

1 Here are the line numbers and infraction referring to the numbered rule.

   a Line 1, rule 1. Line 8, rule 3. Line 9, rule 7.

   b Line 3, rule 2. Line 8, rule 4. Line 15, rule 3. Line 17, rule 4. Line 21, rule 3.

   c Line 1, rule 1. Line 3, rule 2. Line 5, rule 3. Line 7, rule 5.

2 Here are the infractions for the folders and files.

   a No infractions

   b WEB_INF must be uppercase.

   c Scripts and Images contain uppercase letters.

   d source control has a space.

   e Loot box.java has a space and an uppercase.

Exercise 5.5: Comments in code

1 File provided in **Desktop\SEA_ExerciseFiles\Chapter 5**.

2 Here is a sample header:

```
/*
|            createChapterPageNav
|   Input
|       dossier : Dossier pulled from MicroStrategy
|       div     : div in the HTML page to embed TOC
|
|   Output
|       --      : Indirect output pushed inside the div parameter
|
*/
function createChapterPageNav(dossier, div) {
```

3 Here is a commented version of the file. Your answer may vary. Here we commented sections of code rather each line.

```
function createChapterPageNav(dossier, div) {
//Get the TOC from the dossier
var navData = dossier.getTableContent();

//pill will be the unordered list for the TOC
//Sets the style of TOC
var pill = document.createElement("ul");
pill.className = "nav nav-pills nav-stacked";

/*  Creates the header of the TOC
Adds a link to nowhere
Styles makes this one colored
Add as the first child of pill
*/
var child = document.createElement("li");
child.className = "active";
var href = document.createElement("a");
href.href = "#";
href.innerHTML = "Navigation";
child.appendChild(href);
pill.appendChild(child);

//Outer loop for every chapter
for (var i = 0; i < navData.chapters.length; i++) {
    //Grab the chapter
    var chapter = navData.chapters[i];

    //Inner loop for the pages in the chapter
    for (var j = 0; j < chapter.pages.length; j++) {
        //Grab the page, and prep for the list item and link
        var page = chapter.pages[j];
        var newChild = null;
```

```
            var newHREF = null;

            /*Create a li item for the page
            Create the link to nowhere for the page
            Put the link id in console
            Add an event on the link to use
            MicroStrategy API to navigate the dossier to that page
            Write that navigation to the console
            Add the li item to the pill variable (list)
            */
            newChild = document.createElement("li");
            newHREF = document.createElement("a");
            newHREF.href = "#";
            newHREF.innerHTML = chapter.name + ": " + page.name;
            newHREF.id = page.nodeKey;
            console.log(newHREF.id);
            newHREF.onclick = function() {
                console.log("trying to navigate to: " + this.id)
dossier.navigateToPage(dossier.getPageByNodeKey(this.id))
            };
            newChild.appendChild(newHREF);
            pill.appendChild(newChild);

        }
    }
    //Add the unordered list completed to the div
    div.appendChild(pill);

    }
```

4 File is provided in **Desktop\SEA_ExerciseFiles\Chapter 5**.

5 Here is the file with fewer comments.

```
# API / server params

username = "mstr"

password = "password"

# URL to MicroStrategy Library instance used

base_url = "https://env-XXXXX.customer.cloud.microstrategy.com/
MicroStrategyLibrary/api"

project_name = "MicroStrategy Tutorial"  # Tutorial Project
```

Exercise 8.2

1 Here are the levels of severity for the issues. Your answer may vary slightly, but by no more than one level. Discuss with your instructor if you have a significantly different answer.

a 5

b 1

c 3

d 1

e 3

f 3

g 3

h 5

i 4

j 5

k 4

l 2

m 1

n 4

o 1

p 4

B

Appendix: Services Architect Checklist

Services Architect description

This position leads injecting, extending, and embedding analytics into third party, mobile, and white-labeled applications as well as portals in an Intelligent Enterprise. Furthermore, the Services Architect oversees publishing web and data services for developers to use when building departmental applications. The goal is to optimize development efforts within the company and create a structure that allows team members to complete their own tasks quickly and efficiently using the newly projected standards.

Check list overview

Assess

1. Average runtime compared to out-of-the-box
2. Number of executions from application
3. % error rate (from custom applications)
4. Adoption rate of embedded application

Plan

1. Custom application life cycle management
2. Enterprise application integration model
3. SDK development process guidelines
4. SDK code review policy
5. SDK library life cycle management
6. SDK code quality guidelines
7. Security integrations

Create

1. Shared integration and extension repository
2. Source code control
3. Code libraries
4. Plug-ins

Publish

1. Enterprise developer network report

2 Utilization and adoption reports

3 Comments, snippets, and documentation

4 Techniques documentation

5 Common source code libraries

6 Continuous integration pipeline

7 Developer community posts

Operate

1 Monitor embedded analytics report

2 Support developers

3 Troubleshoot integration issues

4 Coordinate with Intelligence Center team

5 Handle embedded analytics cases

Optimize

1 SDK code base

2 Mobile SDK plug-ins

3 SDK libraries

4 Server SDK

5 Web SDK plug-ins

Assets and tooling

Assess

- Platform analytics: test the percentage rate of error during a stress test against a custom application

- File comparison tools (e.g. Code Compare): assess modified and deprecated files

Plan

- Text editor tools (e.g. Sublime, Brackets): create documentation of custom applications
- Agile development tool (e.g. CA Agile Central, Atlassian Jira): plan, track and release custom applications

Create

- XCode IDE: develop an iOS mobile application archive file to be deployed
- Android Studio IDE: develop an Android mobile application package file be deployed
- MicroStrategy Web: develop embedded applications (e.g. URL API)
- Eclipse IDE: develop embedded applications with MicroStrategy's Web Customization Editor
- Java IDE (e.g. IntelliJ): develop custom Java applications with the MicroStrategy Web API
- MicroStrategy Runtime: develop COM API custom applications
- JavaScript IDE: develop custom visualizations
- Microsoft Office Developer: develop embedded applications with MicroStrategy Office
- Text editor tools (e.g. Sublime, Brackets): develop with the Data Connector SDK
- Standard profilers (e.g. Prefix, XRebel): provide traces of code and assist in response time analysis
- Apple App Store: publish iOS mobile applications for users outside the enterprise
- Google Play Store: publish Android mobile applications for users outside the enterprise

Publish

- MicroStrategy Community: publish developer posts, ask and answer questions, and provide code samples
- Javadoc tool (e.g. Doclava): generate API documentation in HTML from comments in source code
- Snippet tool (e.g. Cacher): organize and curate a library of useful code snippets
- Text editor tool (e.g. Notepad++, Sublime): manage code and write notes or documentation

Operate

- Application performance management (e.g. New Relic, AppDynamics): debug production live, and track highly used requests for development time optimization
- Real user monitoring (e.g. Stackify Retrace): provide insight on web request use and response time

Optimize

- JVM profilers (e.g. Apache JMeter): track and find CPU and memory consumption hotspots
- Website optimization tools (e.g. Google PageSpeed): provide suggestions and auto-optimize websites

Detailed checklist

Assess

Maintainability

Calculate a maintainability index value that represents the relative ease of maintaining the code based on a scale of 0-100. A higher value means better

maintainability. Color-coded ratings are used to easily distinguish problematic sections in the evaluation.

Color Scale	0-9: red
	10-19: yellow
	20-100: green

The following are the point values subtracted for each item found to increase the difficulty of maintaining a project. These point values are for each associated file, method, etc. found in the project.

Custom Plug-ins	-0.25: CSS selector method
	-1.00: delta File
	-1.00: modified property file
	-1.50: custom Java file
	-2.00: modified layout XML file
	-3.00: custom JavaScript file
URL API	-0.25: embedded application URL
	-2.00: absolute URL used in object definition
Custom Compiled Code	-5.00: deprecated class used

Continuous delivery: Maturity assessment

Calculate the level and maturity at which a business is providing continuous delivery in the custom application life cycle. The level is calculated on a scale of 0-4 with 0 being baseline and 4 being expert, while the maturity is calculated on a scale of 0-20 from adequate to master.

Level	0: Baseline
	1: Novice
	2: Intermediate
	3: Advanced
	4: Expert

Maturity	0-7: Adequate
	8-11: Average
	12-14: Skilled
	15-17: Adept
	18-20: Master

The following are sections of the custom application life cycle with descriptions for expectations at each level of maturity. Assign a 0-4 rating for each section, then sum all the sections to get the maturity level.

Source Control	**Baseline**	Early branching
		Branches tend to remain apart
	Novice	Branches used in isolated work
		Merges are common
	Intermediate	Pre-tested commits
		Integration branch is pristine
	Advanced	All commits are tied to tasks
		History used to rewrite features before pushing to central repository
		Version control database schema changes
	Advanced	All commits are tied to tasks
		History used to rewrite features before pushing to central repository
		Version control database schema changes
	Expert	Traceability analysis and release notes auto-generated
		Commits are clean enough for the master branch/trunk

Build Process	**Baseline**	Official builds are not performed on developers' machines
		Self-service or nightly build
	Novice	System polls source control and builds on commit
		Build artifacts are managed, some manual scripts still used
	Intermediate	Build artifacts are managed by purpose-built tools, no manual scripts
		Dependencies are managed in a repository
	Advanced	Distributed builds on build cluster, can be done in a sequence
		Source control tells system when to build, no polling
	Expert	Build environments based on VMs
		Streams are never broken
		Gated commits
Testing & QA	**Baseline**	Automatic unit testing with every build
		Code coverage is measured
	Novice	Peer-reviews
		Mockups and proxies used
	Intermediate	Periodic static code analysis
		Automated functional testing
	Advanced	Integrated management and maintenance of test data
		Automated performance and security test in target environments
	Expert	Ability to implement continuous deployment

Visibility	**Baseline**	Build status notification is sent to committer
	Novice	Latest build status is available to all team members
	Intermediate	Trend reports are automatically generated from build server events
		People outside the team can subscribe to build status
	Advanced	Stakeholders have dashboard with real-time product and dependency stats
	Expert	Cross-team data mining and analysis

Plan

Plan the custom application life cycle management

Define a vision:

Define a vision for embedded solutions with key stakeholders. The goal is to bridge the gap between internal and external stakeholders by focusing questions on clearly defining the three key areas below:

- **Business requirements**: the overall value for customers, future objectives of the business, and the financial measures of success for the projects.
    - Are we going to offer this package as part of our core offering or as an add-on?
    - How will the additional insights provided affect our customer's perceived value?
- **Technical requirements**: potential deployment models, necessary hardware and software, and expectations for how the product will be embedded.
    - Will the reporting and analytics technology deploy on-premises or in the cloud?
    - What development resources can we allocate to deploying third-party analytics solutions?

- **Roadmap requirements**: how future features will rely on an embedded product and any future dashboard and reporting needs.
  - How will additional insights evolve the customer use of our product?
  - Which future roadmap items will use the additional insights created?

Determine best methodology for business needs and requirements

Make sure the methodology works for both internal and external stakeholders and is committed to fully to avoid any issues further into the product's development.

- Waterfall: phase-based (complete phase, move to next)
- Agile: adaptive planning, highly rapid and flexible response to external factors, collaborative
- Lean: make decisions as late as possible, view development as broadly as possible
- DevOps: build on collaboration and communication, automate processes, environment-focused
- Iterative: incremental approach with shorter and repeated development cycles, typically used for large projects
- Spiral: risk-based approach to identify the best choice for a given situation
- V-Model: extension of waterfall that tests methods, uses a V-shaped model for validation

Work with System Administrator to:

- Develop the release phases and dates
- Deploy the MicroStrategy plug-ins to the Web Application Servers that will host the MicroStrategy Web customizations, visualizations and data connectors

Plan the enterprise application integration model

Create an enterprise application integration model plan using modeling tools. The following tools can be used to help model embedded analytics solution requirements:

- Context: defines system boundaries, surrounding environment and interacting entities
- Functional: top-down view of a business process or system's major functions
- Use case: defines interaction between the system and its users
- Sequence: defines interactions between objects over time
- AS-IS Process Model: defines current business process flow
- TO-BE Process Model: defines the business process when adjusted to the system in place
- User stories: define major functions of a system

Plan the SDK development process guidelines

Develop a plan to release a product onto the production servers. Make sure to keep in mind packaging, managing, and deploying multiple complex releases across various environments (private data centers vs. public cloud resources). Keep optimization in mind by moving away from manual release processes and towards automation, and make sure to test the product's security as part of its final quality checks.

Process to patch products

1. Check scope
2. Create patch
3. Provide documentation
4. Test changes
5. Add reviewers and external testers
6. Respond to test failures and feedback
7. Push release

While going through the process, make sure to completely test every piece of the new patch and have multiple external testers review it before submitting for final approval. To help with timely feedback and keep everything on track for the releases, provide code review office hours and meetings.

Plan the SDK code review policy

Determine the number of required reviewers, successful builds, and performance tests before merging. Make sure said reviewers provide ample feedback on code changes and nominate a gatekeeper to oversee major changes to the code. This will ensure systems stay maintainable and scalable and consistently perform.

Plan the SDK library life cycle management

Make sure to develop a performance, QA, functional, and non-functional (UX) testing plan. The following methodologies are good baselines for some forms of testing that are critical to measuring a product's success.

Testing Methodologies

- Unit testing: all code will have a corresponding test suit to verify every outcome works properly
- Black box (Dynamic Analysis Security Testing): looks for vulnerabilities an attacker could exploit
- White box: static analysis to spot common flaws without executing software

Plan the SDK code quality guidelines

Choose tools that find design, usage, maintainability, and style issues in code. Many of these issues can lead to bugs that are hard to reproduce in a standard testing environment, so it is always optimal to get ahead of them. Furthermore, find duplicate code by using tools such as Code Clone Detection (Microsoft), and aim for feedback-driven development to fix problems before they appear. Make sure to include front end, back end, and automatic testing.

Plan the security integrations

Make sure to train developers on secure coding and develop a plan to test the application against security policy. Static, dynamic, software composition analysis, and manual penetration testing are recommended to fully test the security of a product.

Helpful tools:

- CA Test Manager: creates numerous virtual copies of large test data sets within seconds, reducing a costly storage footprint and providing testers with instant access to all the test data they need, on demand.

- CA Blazemeter: provides massive scalability with all the collaborative benefits of performance testing in the cloud and the security benefits of an on-premise solution.

Create

Create the shared integration and extension repository

Use a source control repository, such as Git, for team development. Make sure to split the repositories into multiple layers and tiers as per the business requirements. Code should be in sync with existing code patterns and technologies

Create the source code control

Enforce merge checks with tools for code review by approvers prior to merging code. Make sure to set up code review software and best practices for the project.

Code review software recommendations:

- Static code analysis tools: SonarQube, NDepend, FxCop, TFS
- Standardize review process: Collaborator
- Track comments: Crucible, Bitbucket, TFS

Create the code libraries

Find and fix and defects or security vulnerabilities while writing code instead of waiting until after the code is written. Make sure to use open-source components (if any) in a secure way and ensure they do not add any vulnerabilities to a project. Conduct a comprehensive array of performance, functional, unit, and integration testing using the same language and protocols of the systems being tested.

Principles:

- Single responsibility: do not place more than one responsibility into a single class or function
- Open-closed: while adding functionality, existing code should not be modified
- Liskov substitutability: child class should not change the behavior of the parent class

- Interface segregation: split interfaces into smaller pieces based on functionality
- Dependency injection: inject dependencies rather than hardcode them

Create the plug-ins

When creating the plug-ins, make sure to stick to the best practice fundamentals located at developer.microstrategy.com, and know when to use specific options not found out-of-the-box instead of what is provided as a standard.

Publish

Publish the Enterprise developer network report

Make sure to track core competencies, build securely, and manage risk. The report should fully outline everything necessary for others to understand the current status of the network.

Publish the utilization and adoption reports

Make sure to show fully the platform's analytics on report usage and offer key solutions that might help where there is trouble.

Publish the code comments, snippets, and documentation

Use tools such as Doxygen or Doclava to derive code comments directly from code and provide references to interfaces, methods, and properties. Make sure to document code snippets to demonstrate specific functionality using a snippet tool such as Cacher to organize and curate a library of useful code snippets.

Publish techniques documentation

Document programming techniques, adopted to ensure the efficiency and integrity of the framework used, with any text editor tool such as Notepad++.

Publish the common source code libraries

Document version control systems (VCS) to monitor changes to source code over time using tools such as Git or Jira. Make sure to also document README with

general information about all embedded analytic solutions, including all libraries used, with any text editor tool such as Notepad++.

Publish the continuous integration pipeline

Automate build testing and integration using a continuous integration tool such as Jenkins.

Publish the developer community posts

For example, create MicroStrategy SDK community posts including but not limited to best practices, examples, and ideas. Furthermore, respond to inbound MicroStrategy SDK Community Discussions including SDK Community Pages related to Data Connector SDK, Embedding API, Intelligence Server API, Library SDK, Mobile SDK-Android, Mobile SDK-iOS, Office API, REST API, Usher SDK, Visualization SDK and Web SDK. Monitor any community tags including Data Connector SDK, Embedding API, Intelligence Server API, Library SDK, Mobile SDK-Android, Mobile SDK-iOS, Office API, REST API, Usher SDK, Visualization SDK and Web SDK to ensure all questions are answered. The top 10% of MicroStrategy SDK community members should be found and engaged with.

Operate

Monitor embedded analytics report

Monitor and observe the application performance as well as the rate of error resulting from all embedded analytics solutions. Any issues should be handled appropriately.

Troubleshoot integration issues

Resolve any issues reported by error messages logged by custom classes integrated in all environments. Check Community forums to ensure up-to-date best practices are followed.

Handle embedded analytics cases

Meet with both internal and external stakeholders to ensure embedded solutions meet their needs and expectations categorized by business, technical, and roadmap requirements. Feedback and requirements should be documented with what needs to be added, removed, or updated in an agile software development

tool. The backlog should be scoped through value or feature, by number of users impacted and top key users impacted, estimated time to implement, impact of feature on other features, and risk. Each requirement should be reviewed using MoSCoW, which is "must have, should have, could have, and won't have."

Support developers

Establish a feedback loop to ensure operational data is made available to developers and testers. Continue to test and monitor applications in products and re-assess applications for performance, security, and user experience as they are updated or changed. Reduce unproductive time that developers spend waiting for test results.

Coordinate with the Intelligence Center team

Establish a feedback loop with the following architects to gather feedback on embedded analytics solutions that address the business, technical, and roadmap requirements: Application, Analytics, Database, Mobile, Platform, and System.

Optimize

Optimize SDK code base

The application should require the least amount of effort to support in the near future, and any defects should be easy to identify and fix. The following are standards for how the code and solution should be set up:

- Readability: code should be self-explanatory
- Testability: solution should be easy to test
- Debuggability: solution should provide exception details to find the root cause easily
- Configurability: solution should not change code when values are modified
- Reusability: no repetitive code in solution; eliminate any redundant solutions
- Reliability: solution should have exception handling and cleanup resources
- Extensibility: solution should be easy to add enhancements to with minimal changes to code
- Security: solution should be up to date with latest security requirements and vulnerabilities

- Performance: optimizations to enhance return time on all solutions
- Scalability: solution should support a growing infrastructure of users and data
- Usability: user interface and API should be easy to understand and use

Optimize SDK libraries

Review all deprecated API References and update all SDK libraries to the latest supported version.

Optimize Web SDK plug-ins

When preparing for the upgrade, determine the need for existing customizations including their business, technical, and roadmap requirements. Determine whether the new version of MicroStrategy Web offers a replacement to existing customizations so out-of-the-box functionality can be leveraged.

Optimize Mobile SDK plug-ins

Identical to optimizing Web SDK plug-ins: when preparing for the upgrade, determine the need for existing customizations including their business, technical, and roadmap requirements. Determine whether the new version of MicroStrategy Web offers a replacement to existing customizations so out-of-the-box functionality can be leveraged.

Make sure to use the latest version of MicroStrategy Mobile, which has a different versioning than the web solution, and convert EMM integration MicroStrategy solutions to AppConfig MicroStrategy Mobile solutions.

Optimize Server SDK

Identical to optimizing Web SDK plug-ins: when preparing for the upgrade, determine the need for existing customizations including their business, technical, and roadmap requirements. Determine whether the new version of MicroStrategy Command Manager provides existing out of the box scripts to replace custom COM API solutions.

Copyright Information

Trademark Information