Aditya Khanna

12 December 2019

Software Architecture – Deployment Blueprints (DB)

<u>Deployment Blueprint Topic:</u> Climate Change Data Aggregation Web Application

<u>Table of Contents:</u>

Deployment Blueprint #1 (DB1) – Data-Centered Architecture Solution

Part 1: Deriving Deployment Blueprints (DB)..2

Part 2: Evaluating Solution Blueprint Compliance..19

Part 3: Planning Evaluation of DB Using an ATAM Quality Attribute Utility Tree........22

Deployment Blueprint #2 (DB2) – Client/Server Architecture Solution

Part 1: Deriving Deployment Blueprints (DB)..29

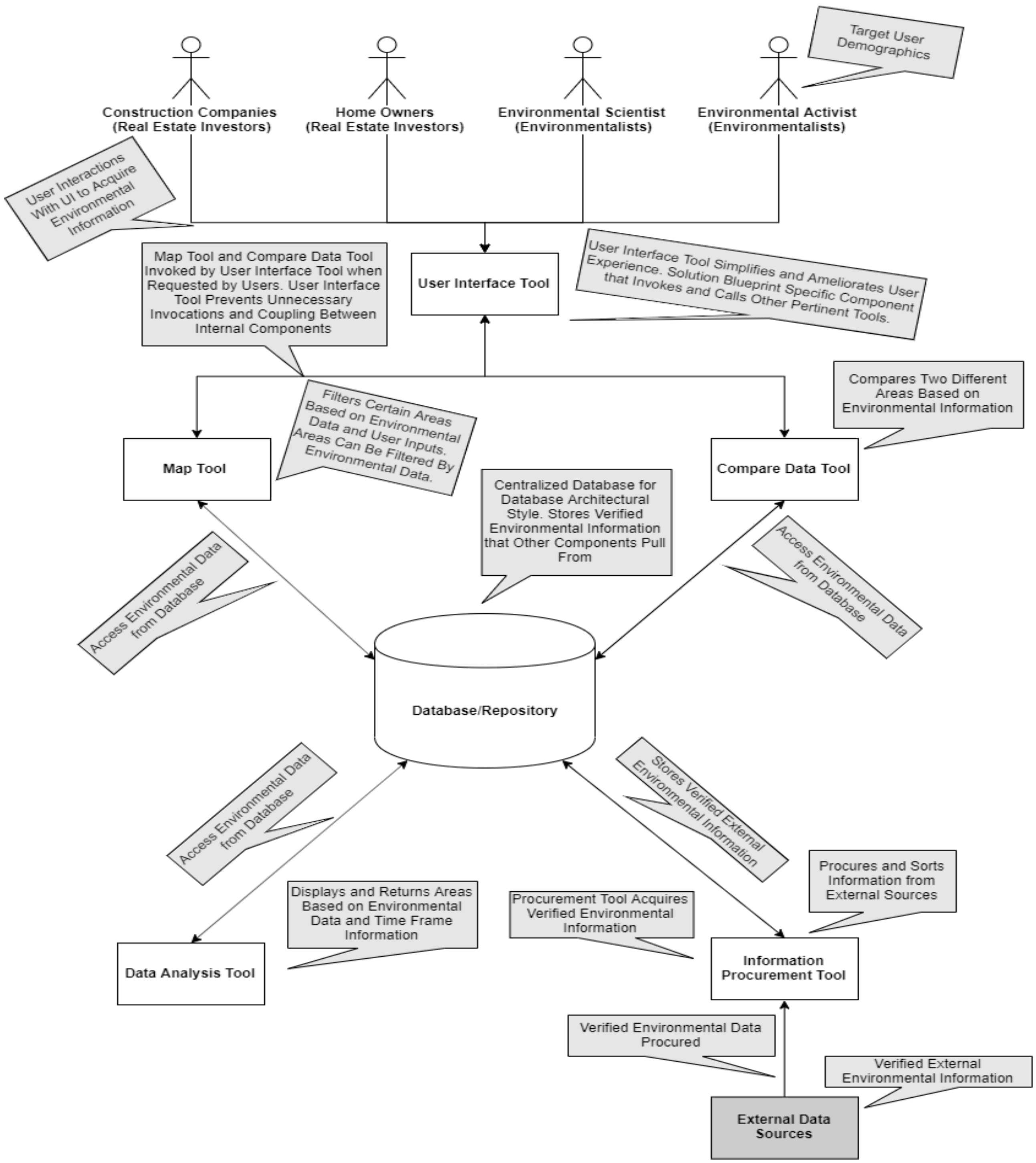
Part 2: Evaluating Solution Blueprint Compliance..47

Part 3: Planning Evaluation of DB Using an ATAM Quality Attribute Utility Tree........50

Deployment Blueprint #1 (DB1) – Data-Centered Architecture Solution

Part 1: Deriving Deployment Blueprints (DB)

Figure 1. Graphical Depiction of DB1



Reference with Example of Repository Architecture Style:

TutorialsPoint. “Data-Centered Architecture.” https://www.tutorialspoint.com/software_architecture_design/data_centered_architecture.htm. Accessed 9 December 2019.

Relevant Excerpt from Reference:

“In Repository Architecture Style, the data store is passive and the clients (software components or agents) of the data store are active, which control the logic flow. The participating components check the data-store for changes.”

Table 1. Description of Component Elements

Deployment Blueprint Component	Description
Database	The database component is a centralized repository that all of the other internal components pull verified environmental data from. In this deployment blueprint, we make the database centralized and emphasize its importance in order to model the Repository Architecture Style for Data-Centered Architecture Solutions. The other components are independent of one another in this deployment solution, and they can each access environmental information within the database. By modeling the database in this manner, we reduce the work done for data transfers between the other internal components and. Additionally, we ensure data integrity since the database is one centralized location. This also allows for scalability and reliability.
Map Tool	The map tool component filters certain areas based on environmental data and user inputs

	that are acquired from the user interface tool. The map tool is able to show, filter, and delineate regions that are associated with certain environmental filters as well as show the environmental information of a specific area (which it acquired from the centralized database).
Data Analysis Tool	The data analysis tool displays and returns areas based on environmental and time information. The data analysis is used by the other internal components to acquire environmental information from the centralized database – both current and for a specified time frame. In this deployment blueprint, the data analysis tool now updates and stores information in the centralized database.
Compare Data Tool	The compare data tool compares the environmental information of two specified areas. It acquires the environmental data and then returns special comparison metrics that are used to compare the different areas and portray the differences between them. In this deployment blueprint, the compare data tool now pulls information from the centralized database.
User Interface Tool	The user interface tool is specific to the solution and deployment blueprints; it was not specified in the business blueprint. The user interface tool invokes the other internal components so that the users are not required to access and modify the internal system. The user interface tool reduces coupling between components and adds a layer of abstraction to the system (because it invokes internal methods according to the user’s needs so that it can satisfy their requirements).
Information Procurement Tool	The information procurement tool is specific to the solution and deployment blueprints; it was not specified in the business blueprint. It

	procures environmental information from verified external sources through API calls and web-scraping methodologies. It then sorts the acquired environmental information and then stores the most accurate data in the centralized database. It is an extra layer of protection that ensures reliability by comparing pulled information from different sources and only storing verified information.

Table 2. Description of Connector Elements (I/O Dependencies)

Deployment Blueprint Connectors	Description
From: Database To: Map Tool	Sends environmental data to the map tool component according to input parameters that it received.
From: Database To: Data Analysis Tool	Sends environmental data to the data analysis tool component according to input parameters that it received.
From: Database To: Compare Data Tool	Sends environmental data to the compare data tool component according to input parameters that it received.
From: Database To: Information Procurement Tool	Sends a request for new or updated information periodically so that the information stored in the centralized database is up to date.
From: Map Tool To: Database	Sends an area or requirements to the database as input data. Expects to receive locations that fit the filtered criteria and/or environmental data about the specified region.
From: Data Analysis Tool To: Database	Sends an area as input data. Expects to receive environmental information from the specified region.
From: Compare Data Tool	Sends two areas as input data. Expects to receive environmental information from the specified regions.

To: Database	
From: Information Procurement Tool To: Database	Sends updated environmental data to the database periodically in order to keep the centralized database up to date.
From: User Interface Tool To: Map Tool	Sends user inputs in a formatted manner to invoke the map tool component and retrieve environmental information.
From: User Interface Tool To: Compare Data Tool	Sends user inputs in a formatted manner to invoke the compare data tool component and retrieve environmental information about two areas.
From: Map Tool To: User Interface Tool	Sends environmental information back to the user interface tool, which can be accessed by the users.
From: Compare Data Tool To: User Interface Tool	Sends environmental information from two locations back to the user interface tool, which can be accessed by the users.
From: Users To: User Interface Tool	Users send their inputs and requirements to the user information tool. These inputs will be circulated throughout the system in order to return the what the user requires.
From: External Data Sources To: Information Procurement Tool	Sends verified environmental data to the information procurement tool. There it will be sorted and checked before being stored in the centralized database.

Table 3. Allocation of Solution to Deployment Components

DB Component	**SB Components Allocated to DB Component**
Database	Database
Map Tool	Map Tool
Data Analysis Tool	Data Analysis Tool

Compare Data Tool	Compare Data Tool
User Interface Tool	User Interface Tool
Information Procurement Tool	N/A

Table 4. Satisfaction of Domain Functions by Solutions

SB Solution Component	**BB Functions Satisfied (39/39)**
Database	• pullSoilData() • pullWaterData() • pullAirData() • pullTemperatureData() • pullWindSpeedData() • pullHumidityData() • pullPrecipitationData()
Map Tool	• filterAreaBasedOnSoilData() • filterAreaBasedOnWaterData() • filterAreaBasedOnAirData() • filterAreaBasedOnTemperatureData() • filterAreaBasedOnWindSpeedData() • filterAreaBasedOnHumidityData() • filterAreaBasedOnPrecipitationData()
Data Analysis Tool	• displayCurrentSoilData() • displayCurrentWaterData() • displayCurrentAirData() • displayCurrentTemperatureData() • displayCurrentWindSpeedData() • displayCurrentHumidityData() • displayCurrentPrecipitationData() • displaySoilDataForTimeFrame() • displayWaterDataForTimeFrame() • displayAirDataForTimeFrame() • displayTemperatureDataForTimeFrame() • displayWindSpeedDataForTimeFrame() • displayHumidityDataForTimeFrame()

	• displayPrecipitationDataForTimeFrame()
Compare Data Tool	• compareSoilData() • compareWaterData() • compareAirData() • compareTemperatureData() • compareWindSpeedData() • compareHumidityData() • comparePrecipitationData()
User Interface Tool	• searchLocation() • clear() • zoomIn() • zoomOut()

Table 5. Satisfaction of Domain Data by Solutions

SB Solution Component	**BB Data Satisfied (24/24)**
Database	• Soil Data • Water Data • Air Data • Temperature Data • Wind Speed Data • Humidity Data • Precipitation Data
Map Tool	• Primary Location Data • Filtered Area Data – Soil • Filtered Area Data – Water • Filtered Area Data – Air • Filtered Area Data – Temperature • Filtered Area Data – Wind • Filtered Area Data – Humidity • Filtered Area Data – Precipitation

Data Analysis Tool	• Specified Time Frame
Compare Data Tool	• Secondary Location Data • Comparison Data – Soil • Comparison Data – Water • Comparison Data – Air • Comparison Data – Temperature • Comparison Data – Wind Speed • Comparison Data – Humidity • Comparison Data – Precipitation
User Interface Tool	N/A

Table 6. BB I/O Mapping to DB

DB Component	**I/O Dependency**
From: Map Tool To: Database	• *pullSoilData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullWaterData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullAirData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullTemperatureData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullWindSpeedData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullHumidityData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullPrecipitationData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component

From: Compare Data Tool To: Database	• *pullSoilData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullWaterData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullAirData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullTemperatureData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullWindSpeedData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullHumidityData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullPrecipitationData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component
From: Database To: Map Tool	• *filterAreaBasedOnSoilData()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *filterAreaBasedOnWaterData()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *filterAreaBasedOnAirData()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *filterAreaBasedOnTemperatureData()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *filterAreaBasedOnWindSpeedData()* requires *Wind Speed Data* from *pullWind SpeedData()*, which was allocated in the *Database* component • *filterAreaBasedOnHumidityData()* requires *Humidity Data* from *pullHumidityData(),*

	which was allocated in the *Database* component • *filterAreaBasedOnPrecipitationData()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component
From: Map Tool To: Data Analysis Tool	• *displayCurrentSoilData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentWaterData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentAirData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentTemperatureData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentWindSpeedData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentHumidityData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentPrecipitationData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displaySoilDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayWaterDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayAirDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayTemperatureDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayWindSpeedDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayHumidityDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component

	• *displayPrecipitationDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component
From: Database To: Data Analysis Tool	• *displayCurrentSoilData()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *displayCurrentWaterData()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *displayCurrentAirData()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *displayCurrentTemperatureData()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *displayCurrentWindSpeedData()* requires *Wind Speed Data* from *pullWindSpeedData()*, which was allocated in the *Database* component • *displayCurrentHumidityData()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component • *displayCurrentPrecipitationData()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component • *displaySoilDataForTimeFrame()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *displayWaterDataForTimeFrame()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *displayAirDataForTimeFrame()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *displayTemperatureDataForTimeFrame()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *displayWindSpeedDataForTimeFrame()* requires *Wind Speed Data* from *pullWindSpeedData()*, which was allocated in the *Database* component

	• *displayHumidityDataForTimeFrame()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component • *displayPrecipitationDataForTimeFrame()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component
From: Map Tool To: Compare Data Tool	• *compareSoilData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareWaterData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareAirData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareTemperatureData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareWindSpeedData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareHumidityData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *comparePrecipitationData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component
From: Database To: Compare Data Tool	• *compareSoilData()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *compareWaterData()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *cmopareAirData()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *compareTemperatureData()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *compareWindSpeedData()* requires *Wind Speed Data* from *pullWindSpeedData()*,

	which was allocated in the *Database* component • *compareHumidityData()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component • *comparePrecipitationData()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component

Derivation Rationale:

- **Architectural Styles and Stakeholder Needs:**

    1. Repository Architecture Style (Data-Centered Architecture): We modeled our system in this deployment blueprint (DB) primarily according to the Repository Architecture Style. This is clearly illustrated in our graphic, since we have a centralized database that every other component is centered around. The map tool, data analysis tool, compare data tool, and information procurement tool all pull data from and update the central database. These are known as the data accessors in this architectural style. This model ensures that we maintain a certain level of data integrity, for all of the data is located in one accessible and centralized location. This means that we satisfy our highest priority quality – reliability. We ensure that the data is reliable with the information procurement tool from verified external sources and then store that information in the centralized database where it is accessed (from the entire system). It also satisfies scalability for the resources that we invest in the system can be scaled through individual components. For example, if we want more repository space or want to increase the bandwidth of data transmission, we can just scale the database itself. Next, the

system satisfies compatibility, for the database can adapt and be modified to have endpoints that can be accessed on any platform. Finally, we can modify the data we store, what data is available, what sources we pull from, and how much information we transmit. This means that we satisfy flexibility as well.

Stakeholder Needs: Reliability, Scalability, Compatibility, Flexibility

2. Object-Oriented Architecture Style (Call-and-Return Architecture): Our system implements an Object-Oriented Architecture Style as well. This is also illustrated in our graphical representation, for the components are al modular. Our system uses an abstraction of layers between the user and the internal structure. Additionally, we instantiate objects and classes that we use as inputs and outputs between the components. This style ensures that we satisfy the condition of an on-time delivery, for our modular and abstracted code allows us to maintain our intent, avoid unnecessary code, and foresee risks/challenges. This means that we can avoid delays and deliver on-time. Additionally, this allows us satisfy the condition of reasonable costs. Having a shorter delivery time and avoiding mistakes allows us to spend less resources (time, effort, and money) fixing mistakes that we avoided. Additionally, Object-Oriented systems are easy to maintain and scale. Classes and objects can be modified easily, and those changes are able to be seen throughout the system. This means that our model is also maintainable. Finally, our application satisfies availability, for Object-Oriented systems allow the system to always be available and running. By maintaining modularity, we are able to create parallel streams that provide similar functionalities (service-drive architectures) that satisfy availability.

Stakeholder Needs: Time Delivery, Cost, Maintainability, Availability

3. Layered Architecture Style (Call-and-Return Architecture): We also somewhat model a Layered Architecture Style in this deployment blueprint. We create an outer layer that serves as the user interface for users and clients to interact with (that hides the internal structure of our system). This layered approach allows us to satisfy the stakeholder need of usability. We can modify and cater the user interface so that it maximizes usability and utility for users through this layered model.

   Stakeholder Needs: Usability

4. Pipe and Filter Architecture Style (Data Flow Architecture): We also have a structure that uses some fundamental ideas behind the Pipe and Filter Architecture Style. We use a unidirectional flow of data with the information procurement tool in order to update and store environmental data in the centralized database. This Pipe and Filter ideology allows us to procure, sort, and check information before using it in our application. This ensures reliability. Additionally, this allows us to satisfy the stakeholder need of extensibility. By having a data flow network for storing data in the database, we are able to expand our procurement tools and pull from more sources very easily. This means our system is extensible and has the potential to grow and meet the needs of stakeholders in the future.

   Stakeholder Needs: Extensibility

- **Architectural Style References:**

1. TutorialsPoint. "Data-Centered Architecture." https://www.tutorialspoint.com/software_architecture_design/data_centered_architecture.htm. Accessed 9 December 2019.
2. TutorialRide. "Object Oriented Architecture." https://www.tutorialride.com/software-architecture-and-design/object-oriented-architecture.htm. Accessed 9 December 2019.
3. Mallawaarachchi, Vijini. TowardsDataScience. "10 Common Software Architectural Patterns in a Nutshell." https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013. Accessed 9 December 2019.
4. TutorialRide. "Data Flow Architecture." https://www.tutorialride.com/software-architecture-and-design/data-flow-architecture.htm. Accessed 9 December 2019.

- **Potential Conflicts:**

1. One potential conflict that could arise when implementing the model enumerated within this deployment blueprint (DB) is an increase in coupling between components. By implementing a centralized database in accordance with the Repository Architecture Style (Data-Centered Architecture), we have the potential of increasing dependency between internal components and the centralized database component. This could interfere with how we satisfy the stakeholder need of scalability. By increasing coupling, we make it more difficult to engender change throughout the system by only modifying a single component. In order to

scale, we might have to modify multiple components based of the increase in coupling.

- **Trade-Offs:**

    1. By implementing the Repository Architecture Style (Data-Centered Architecture) in this deployment blueprint (DB), we are able to increase reliability – which is our highest priority according to our stakeholder needs. A centralized database allows us to maintain a certain level of data integrity since all of the environmental data is located in one accessible and centralized location. This ensures that only this data is accessed and distributed (which ensures reliability). The trade-off, however, is a slight decrease in scalability. By centralizing the data, we increase the reliance of other internal components on the database. This means that we could potentially increase coupling between our internal components. This trade-off is justified, since we are ensuring our highest priority stakeholder need. Additionally, we increase the security and value of our application by increasing reliability.

- **Refactorings:**

    1. <u>Created User Interface Tool:</u> For this deployment blueprint (DB), we created a user interface tool component that did not exist in the business blueprint (BB). This refactoring allows us to implement the fundamental ideas of a Layered Architecture Style, which allows us to ameliorate the user interface as well as the usability of our application through an abstraction of functionality. This refactoring could potentially sacrifice the BB's vision to reduce system complexity, for we are adding an additional component. This refactoring,

however, is justified because this component allows us to improve usability and directly help the users by making the application easy to navigate through an abstraction of functionality. Thus, we are able to reduce user complexity at the expense of potentially increasing system complexity – which is justified for our application.

2. Created Information Procurement Tool: For this deployment blueprint (DB), we created an information procurement tool component that did not exist in the business blueprint (BB). This refactoring allows us to improve the reliability of our application by checking the information that we pull from external sources. This refactoring could potentially sacrifice the BB's vision to reduce system complexity, for we are adding an additional component. This refactoring, however, is justified because this component allows us to ensure that the environmental data stored in our database is reliable – which is our most prioritized quality.

Part 2: Evaluating Solution Blueprint Compliance

$$\mathbf{CompFuncCoeff(c, t)} = \frac{|\boldsymbol{CompFunc_{regd}(c,t)}|}{|\boldsymbol{CompFunc(c)}|}$$

Table 7. CompFuncCoeff(c, t) Calculations

Component (BB)	Technology (SB)	CompFuncCoeff(c, t)
Map Tool	Map Tool	**7/11**
	Data Analysis Tool	**0/11**
	Compare Data Tool	**0/11**

	Database	**0/11**
	User Interface Tool	**4/11**
Data Analysis Tool	Map Tool	**0/14**
	Data Analysis Tool	**14/14**
	Compare Data Tool	**0/14**
	Database	**0/14**
	User Interface Tool	**0/14**
Compare Data Tool	Map Tool	**0/7**
	Data Analysis Tool	**0/7**
	Compare Data Tool	**7/7**
	Database	**0/7**
	User Interface Tool	**0/7**
Database	Map Tool	**0/7**
	Data Analysis Tool	**0/7**
	Compare Data Tool	**0/7**
	Database	**7/7**
	User Interface Tool	**0/7**

$$\textbf{CompDataCoeff(c, t)} = \frac{|\boldsymbol{CompData_{regd}(c,t)}|}{|\boldsymbol{CompFunc(c)}|}$$

Table 8. CompDataCoeff(c, t) Calculations

Component (BB)	Technology (SB)	CompDataCoeff(c, t)
Map Tool	Map Tool	**8/8**
	Data Analysis Tool	**0/8**
	Compare Data Tool	**0/8**
	Database	**0/8**
	User Interface Tool	**0/8**
Data Analysis Tool	Map Tool	**0/1**
	Data Analysis Tool	**1/1**
	Compare Data Tool	**0/1**
	Database	**0/1**
	User Interface Tool	**0/1**
Compare Data Tool	Map Tool	**0/8**
	Data Analysis Tool	**0/8**
	Compare Data Tool	**8/8**
	Database	**0/8**
	User Interface Tool	**0/8**
Database	Map Tool	**0/7**
	Data Analysis Tool	**0/7**
	Compare Data Tool	**0/7**
	Database	**7/7**

	User Interface Tool	**0/7**

Table 9. CompFuncBoundaryError(t) Calculations

Technology (SB)	CompFuncBoundaryError(t)
Map Tool	(1 - (7/11)) + (0/11) = **0.3636**
Data Analysis Tool	(1 - (14/14)) + (0/11) = **0**
Compare Data Tool	(1 - (7/7)) + (0/7) = **0**
Database	(1 - (7/7)) + (0/7) = **0**
User Interface Tool	(1 - (4/11)) + (0/11) = **0.6363**

Part 3: Planning Evaluation of DB Using an ATAM Quality Attribute Utility Tree

Utility Definition: Utility is the measurement of an element's value or worth. In a software architecture context, utility is the combination of a software system's usefulness and usability.

Aggregation Method for Quality Assessments: In order to obtain an overall "goodness" of the system by aggregating quality assessments, we must set objectives to satisfy each quality attribute and specify metrics on how to measure the success of meeting the aforementioned objectives. Additionally, we must keep in mind that the quality attributes are weighted by priority, so a system is better if it meets its objectives and satisfies the quality attributes higher in the Utility Tree.

- **UTILITY**
    - **Reliability**
        - Ensure Data Integrity

    - *Objective:* Ensure that the Database contains the most recent environmental information and is updated correctly from the external data sources (H, M)
        - *Metric:* Search for environmental data with the user interface tool component. Check what information is stored in the database for that area by viewing the expected output. Then check the information procurement tool for the selected area. Compare the information and check if it is identical. (H)
    - *Objective:* Increase the number if verified data sources to pull information from (H, M)
        - *Metric:* Add sources to the infrastructure of the information procurement tool. Search for environmental data with the user interface tool component. Then check the information procurement tool for the selected area. Check to see if the number of sources and comparison calculations has increased. (H)
- <u>Enable Source Confirmation Service</u>
    - *Objective:* Ensure that the information procurement tool analyzes environmental data from the sources and takes the most appropriate value (H, H)
        - *Metric:* Set the sources for the information procurement tool to a program that we create. Populate the information

procurement tool with information – specifically an array of many numbers. Check that the components chooses the reliable and weighted source, and check that it matches with the mode value (H)

- **Scalability**
    - Maximize Scalability
        - *Objective:* Display that the database component is able to scale as the environmental information stored becomes more diverse and the database becomes more populated (H, M)
            - *Metric:* Increase the number of elements in the database by a set number n for some bound. Then run a script after every addition of elements that checks how fast it takes to return environmental information. Check if the time increases linearly or faster with the increase in elements within the database (H)
- **Usability**
    - Ensure Services are Accessible
        - *Objective:* Navigate between services through the user interface tool in less than 1 second and less than an n number of clicks for an n number of services, where n > 0 (M, M)
            - *Metric:* Write a front-end Selenium testing script that starts at the main user page and navigates between an n number

of pages in all combinations of n-orderings. Check that each instance is less than 1 second (H)

- **Compatibility**
  - Maximize Platform Accessibility
    - *Objective:* Ensure that the web application is accessible on all major web browsers (Google Chrome, Firefox, Microsoft Edge, Safari, etc.), and check that all of the services function appropriately (H, H)
      - *Metric:* Write multiple scripts (one for every web browser) that automates navigation between services within our application. Then let the script check each functionality and check it against an expected output. Check if the results are appropriate (M)
- **Flexibility**
  - Maximize System Flexibility with Regard to Time
    - *Objective:* Ensure that the system is able to remain functional and swift in terms of time even with the addition of multiple new features and services (M, M)
      - *Metric:* Measure the time of the system before any additions of features and services. Create a load-heavy service (mock function) that execute 2^n searches. Add this service to the system and measure the difference in time.

Check to see if the increase in time is less than an increase of 2^n (M)

- **Time Delivery**
  - <u>Minimize Projected Timeline</u>
    - *Objective:* Minimize the projected timeline for the project and reduce the number of hours invested into development as well as testing (H, L)
      - *Metric:* Check the original schedule. Compare how many deliverables were expected to be completed and check the actual number of deliverables completed (M)
- **Cost**
  - <u>Minimize Cost of Project</u>
    - *Objective:* Minimize the cost of the project and reduce investment of resources (H, M)
      - *Metric:* Determine the initial cost of investment (hardware, server-space, database orientation) (H)
    - *Objective:* Minimize the number of hours that the development team works and invests into the project (H, M)
      - *Metric:* Project how many hours are required to maintain the application and how many hours are required to complete development (H)
- **Maintainability**
  - <u>Minimize Resource Consumption</u>

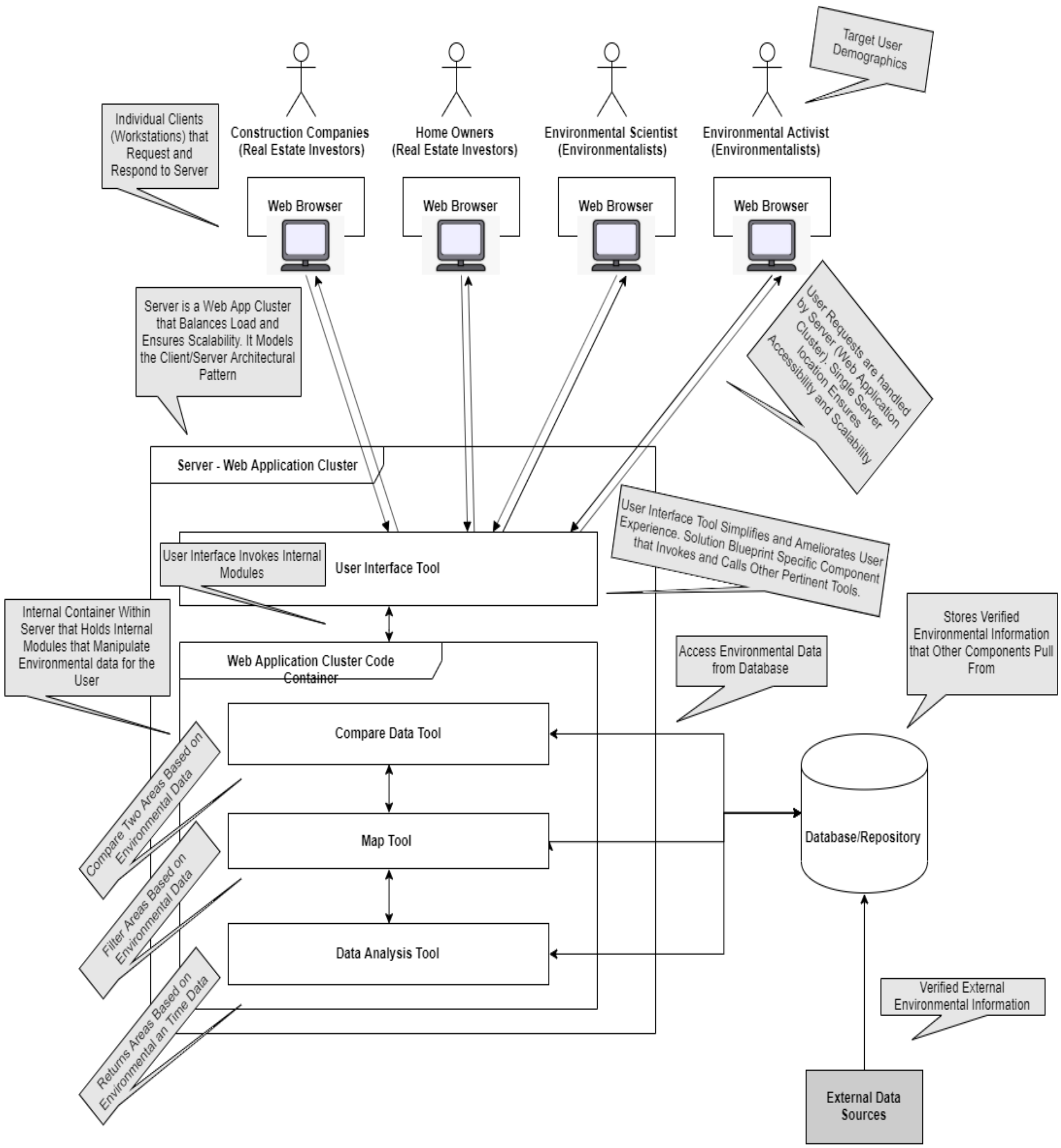
            - *Objective:* Ensure that the consumption of data, memory, and server (bandwidth) resources are reduced for both client-side and server-side interactions (M, M)
                - *Metric:* Instantiate instances of the application n multiple devices and machines. Exhaustively run every combination of services. Check the memory, space, and bandwidth consumed by the processes on both the client-side and server-side (M)
    - **Availability**
        - Maximize Active (Operation-Ready) Time
            - *Objective:* Ensure that the application is always active and is always readily accessible. Ensure there are no instances of the application going down (M, M)
                - *Metric:* Instantiate a live session of the application on a machine. Measure the connection between the browser and the application and record whether there are any instances of a loss of connection. Check these recordings after an n number of days (H)
    - **Extensibility**
        - Minimize Coupling Between Components
            - *Objective:* Minimize coupling between components and ensure that system components are highly cohesive as independent entities. Minimize dependencies between components (L, L)

- *Metric:* Measure coupling using the "Number of Dependencies Between Components" calculations from the BB (L)

Deployment Blueprint #2 (DB2) – Client/Server Architecture Solution

Part 1: Deriving Deployment Blueprints (DB)

Figure 1. Graphical Depiction of DB2



Reference with Example of Repository Architecture Style:

CCM. “Client/Server Environment.” https://ccm.net/contents/152-client-server-environment. Accessed 9 December 2019.

Relevant Excerpt from Reference:

“Numerous applications run in a client/server environment, this means that client computers (computers forming part of the network) contact a server, generally a very powerful computer in terms of input/output, which provides services to the client computers. These services are programs which provide data such as the time, files, a connection, etc.”

Table 1. Description of Component Elements

Deployment Blueprint Component	Description
Database	The database is accessed from components within the server component. This allows for scalability, reliability, and security because all of the internal structure is located within the centralized and accessible server component. The database component is a repository that all of the other internal components pull verified environmental data from. The other components are independent of one another in this deployment solution, and they can each access environmental information within the database. Additionally, we ensure data integrity since the database is one centralized location. This also allows for scalability and reliability.
Map Tool	The map tool component is now a component within the server component. The server component is accessible and centralized so that our system can ensure scalability and security. Additionally, the system is more

	reliable because the internal structure is hidden within the server component – away from unnecessary modification. The map tool component filters certain areas based on environmental data and user inputs that are acquired from the user interface tool. The map tool is able to show, filter, and delineate regions that are associated with certain environmental filters as well as show the environmental information of a specific area (which it acquired from the database).
Data Analysis Tool	The data analysis tool component is now a component within the server component. The server component is accessible and centralized so that our system can ensure scalability and security. Additionally, the system is more reliable because the internal structure is hidden within the server component – away from unnecessary modification. The data analysis tool displays and returns areas based on environmental and time information. The data analysis is used by the other internal components to acquire environmental information from the database – both current and for a specified time frame. In this deployment blueprint, the data analysis tool now updates and stores information in the database.
Compare Data Tool	The compare data tool component is now a component within the server component. The server component is accessible and centralized so that our system can ensure scalability and security. Additionally, the system is more reliable because the internal structure is hidden within the server component – away from unnecessary modification. The compare data tool compares the environmental information of two specified areas. It acquires the environmental data and then returns special

	comparison metrics that are used to compare the different areas and portray the differences between them. In this deployment blueprint, the compare data tool now pulls information from the database.
User Interface Tool	The user interface tool component is now a component within the server component. The server component is accessible and centralized so that our system can ensure scalability and security. Additionally, the system is more reliable because the internal structure is hidden within the server component – away from unnecessary modification. The user interface tool is specific to the solution and deployment blueprints; it was not specified in the business blueprint. The user interface tool invokes the other internal components so that the users are not required to access and modify the internal system. The user interface tool reduces coupling between components and adds a layer of abstraction to the system (because it invokes internal methods according to the user's needs so that it can satisfy their requirements).
Web Browser	This is the platform on which a user that is using a workstation (a hardware endpoint) can access the server component. The web browser can make requests to the server and respond to activities on the server. It cannot access the internal components of the server component.
Server – Web Application Cluster	The server component is a centralized and easily accessible component that can be accessed by users through web browser endpoints. It can receive requests and release responses from these endpoints. This increases security and reliability, for the internal components of the server are closed for modification and access. Additionally, the

	server is scalable because it is a centralized component that only needs to change in a single instance. It contains the map tool, data analysis, compare, and user interface components.

Table 2. Description of Connector Elements (I/O Dependencies)

Deployment Blueprint Connectors	Description
From: Database To: Server	Sends environmental data to the server component according to input parameters that it received. It returns information that satisfies the criteria of the contained components (map tool, data analysis tool, compare tool)
From: Server To: Database	Sends an input data to the database according to its inner components (map tool, data analysis tool, compare tool). Expects to receive locations that fit the filtered criteria and/or environmental data about the specified region.
From: User Interface Tool To: Map Tool	Sends user inputs in a formatted manner to invoke the map tool component and retrieve environmental information.
From: User Interface Tool To: Compare Data Tool	Sends user inputs in a formatted manner to invoke the compare data tool component and retrieve environmental information about two areas.
From: Map Tool To: User Interface Tool	Sends environmental information back to the user interface tool, which can be accessed by the users.
From: Compare Data Tool To: User Interface Tool	Sends environmental information from two locations back to the user interface tool, which can be accessed by the users.
From: Users To: Server	Users send their inputs and requirements to the server. These inputs will be circulated

	throughout the system in order to return the what the user requires.
From: Server To: User	The server sends an appropriate response to the users based on its internal components (map tool, data analysis tool, compare tool).

Table 3. Allocation of Solution to Deployment Components

DB Component	**SB Components Allocated to DB Component**
Database	Database
Server (Map Tool)	Map Tool
Server (Data Analysis Tool)	Data Analysis Tool
Server (Compare Data Tool)	Compare Data Tool
Server (User Interface Tool)	User Interface Tool
Web Browser	N/A

Table 4. Satisfaction of Domain Functions by Solutions

SB Solution Component	**BB Functions Satisfied (39/39)**
Database	• pullSoilData() • pullWaterData() • pullAirData() • pullTemperatureData() • pullWindSpeedData() • pullHumidityData() • pullPrecipitationData()
Server (Map Tool)	• zoomIn() • zoomOut() • filterAreaBasedOnSoilData()

	• filterAreaBasedOnWaterData() • filterAreaBasedOnAirData() • filterAreaBasedOnTemperatureData() • filterAreaBasedOnWindSpeedData() • filterAreaBasedOnHumidityData() • filterAreaBasedOnPrecipitationData()
Server (Data Analysis Tool)	• displayCurrentSoilData() • displayCurrentWaterData() • displayCurrentAirData() • displayCurrentTemperatureData() • displayCurrentWindSpeedData() • displayCurrentHumidityData() • displayCurrentPrecipitationData() • displaySoilDataForTimeFrame() • displayWaterDataForTimeFrame() • displayAirDataForTimeFrame() • displayTemperatureDataForTimeFrame() • displayWindSpeedDataForTimeFrame() • displayHumidityDataForTimeFrame() • displayPrecipitationDataForTimeFrame()
Server (Compare Data Tool)	• compareSoilData() • compareWaterData() • compareAirData() • compareTemperatureData() • compareWindSpeedData() • compareHumidityData() • comparePrecipitationData()
Server (User Interface Tool)	• searchLocation() • clear()

Table 5. Satisfaction of Domain Data by Solutions

SB Solution Component	**BB Data Satisfied (24/24)**
Database	• Soil Data • Water Data

	• Air Data • Temperature Data • Wind Speed Data • Humidity Data • Precipitation Data
Server (Map Tool)	• Filtered Area Data – Soil • Filtered Area Data – Water • Filtered Area Data – Air • Filtered Area Data – Temperature • Filtered Area Data – Wind • Filtered Area Data – Humidity • Filtered Area Data – Precipitation
Server (Data Analysis Tool)	• Specified Time Frame
Server (Compare Data Tool)	• Comparison Data – Soil • Comparison Data – Water • Comparison Data – Air • Comparison Data – Temperature • Comparison Data – Wind Speed • Comparison Data – Humidity • Comparison Data – Precipitation
Server (User Interface Tool)	• Primary Location Data • Secondary Location Data

Table 6. BB I/O Mapping to DB

DB Component	**I/O Dependency**
From: Server (Map Tool) To: Database	• *pullSoilData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullWaterData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component

	• *pullAirData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullTemperatureData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullWindSpeedData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullHumidityData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullPrecipitationData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component
From: Server (Compare Data Tool) To: Database	• *pullSoilData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullWaterData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullAirData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullTemperatureData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullWindSpeedData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullHumidityData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullPrecipitationData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component
From: Database To: Server (Map Tool)	• *filterAreaBasedOnSoilData()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component

	• *filterAreaBasedOnWaterData()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *filterAreaBasedOnAirData()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *filterAreaBasedOnTemperatureData()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *filterAreaBasedOnWindSpeedData()* requires *Wind Speed Data* from *pullWind SpeedData()*, which was allocated in the *Database* component • *filterAreaBasedOnHumidityData()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component • *filterAreaBasedOnPrecipitationData()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component
From: Server (Map Tool) To: Server (Data Analysis Tool)	• *displayCurrentSoilData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentWaterData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentAirData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentTemperatureData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentWindSpeedData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentHumidityData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component

	• *displayCurrentPrecipitationData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displaySoilDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayWaterDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayAirDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayTemperatureDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayWindSpeedDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayHumidityDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayPrecipitationDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component
From: Database To: Server (Data Analysis Tool)	• *displayCurrentSoilData()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *displayCurrentWaterData()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *displayCurrentAirData()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *displayCurrentTemperatureData()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *displayCurrentWindSpeedData()* requires *Wind Speed Data* from *pullWindSpeedData()*, which was allocated in the *Database* component • *displayCurrentHumidityData()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component

	• *displayCurrentPrecipitationData()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component • *displaySoilDataForTimeFrame()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *displayWaterDataForTimeFrame()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *displayAirDataForTimeFrame()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *displayTemperatureDataForTimeFrame()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *displayWindSpeedDataForTimeFrame()* requires *Wind Speed Data* from *pullWindSpeedData()*, which was allocated in the *Database* component • *displayHumidityDataForTimeFrame()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component • *displayPrecipitationDataForTimeFrame()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component
From: Server (Map Tool) To: Server (Compare Data Tool)	• *compareSoilData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareWaterData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareAirData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareTemperatureData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareWindSpeedData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component

	compareHumidityData() requires *Primary Location Data*, which was allocated in the *Map Tool* component *comparePrecipitationData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component
From: Database To: Server (Compare Data Tool)	*compareSoilData()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component *compareWaterData()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component *cmopareAirData()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component *compareTemperatureData()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component *compareWindSpeedData()* requires *Wind Speed Data* from *pullWindSpeedData()*, which was allocated in the *Database* component *compareHumidityData()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component *comparePrecipitationData()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component

Derivation Rationale:

- **Architectural Styles and Stakeholder Needs:**

    1. Client/Server Architecture Style: We modeled our system primarily after the Client/Server Architectural Style. This can be clearly illustrated in our graphical representation; we have a server component that contains the map tool, data analysis tool, compare data tool, and user interface tool. This server component is accessible through web browser endpoints that can request certain services. This

allows the internal structure of the system to be closed for modification but still accessible through the abstracted server component. This ensures that the environmental information that we use is secure, which increases the reliability of our web application (which is the most prioritized stakeholder need). Additionally, the server component is centralized and easily accessible. This means that we can ensure the condition of scalability, for we only need to modify the server in order to engender change throughout the system. Finally, we can also improve the usability of the system by only allowing to make simplified requests through the client/server model. This ensures that server-side data is not manipulated and client-side requirements are met easily.

Stakeholder Needs: Reliability, Scalability, Usability

2. Object-Oriented Architecture Style (Call-and-Return Architecture): Our system implements an Object-Oriented Architecture Style as well. This is also illustrated in our graphical representation, for the components are al modular. Our system uses an abstraction of layers between the user and the internal structure. Additionally, we instantiate objects and classes that we use as inputs and outputs between the components. This style ensures that we satisfy the condition of an on-time delivery, for our modular and abstracted code allows us to maintain our intent, avoid unnecessary code, and foresee risks/challenges. This means that we can avoid delays and deliver on-time. Additionally, this allows us satisfy the condition of reasonable costs. Having a shorter delivery time and avoiding mistakes allows us to spend less resources (time, effort, and money) fixing mistakes that we avoided. Additionally, Object-Oriented systems are easy to

maintain and scale. Classes and objects can be modified easily, and those changes are able to be seen throughout the system. This means that our model is also maintainable. Finally, our application satisfies availability, for Object-Oriented systems allow the system to always be available and running. By maintaining modularity, we are able to create parallel streams that provide similar functionalities (service-drive architectures) that satisfy availability.

Stakeholder Needs: Time Delivery, Cost, Maintainability, Availability

3. Repository Architecture Style (Data-Centered Architecture): We modeled our database in this deployment blueprint (DB) similar to the Repository Architecture Style. This model ensures that we maintain a certain level of data integrity, for all of the data is located in one accessible and centralized location. This means that we satisfy our highest priority quality – reliability. We ensure that the data is reliable with the information procurement tool from verified external sources and then store that information in the centralized database where it is accessed (from the entire system). It also satisfies scalability for the resources that we invest in the system can be scaled through individual components. For example, if we want more repository space or want to increase the bandwidth of data transmission, we can just scale the database itself. Next, the system satisfies compatibility, for the database can adapt and be modified to have endpoints that can be accessed on any platform. Finally, we can modify the data we store, what data is available, what sources we pull from, and how much information we transmit. This means that we satisfy flexibility as well.

   Stakeholder Needs: Reliability, Scalability, Compatibility, Flexibility

4. Layered Architecture Style (Call-and-Return Architecture): We also somewhat model a Layered Architecture Style in this deployment blueprint. We create an outer layer that serves as the user interface for users and clients to interact with (that hides the internal structure of our system). This layered approach allows us to satisfy the stakeholder need of usability. We can modify and cater the user interface so that it maximizes usability and utility for users through this layered model. We can also add additional features without interfering with the user experience. Thus, we can also increase the extensibility of the system.
   Stakeholder Needs: Usability, Extensibility

- **Architectural Style References:**

1. CCM. "Client/Server Environment." https://ccm.net/contents/152-client-server-environment. Accessed 9 December 2019.
2. TutorialRide. "Object Oriented Architecture." https://www.tutorialride.com/software-architecture-and-design/object-oriented-architecture.htm. Accessed 9 December 2019.
3. TutorialsPoint. "Data-Centered Architecture." https://www.tutorialspoint.com/software_architecture_design/data_centered_architecture.htm. Accessed 9 December 2019.
4. Mallawaarachchi, Vijini. TowardsDataScience. "10 Common Software Architectural Patterns in a Nutshell." https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013. Accessed 9 December 2019.

- **Potential Conflicts:**

1. One potential conflict that could arise when implementing the model enumerated within this deployment blueprint (DB) is an increase in complexity. By implementing a centralized server according to the Client/Server Architecture Style, we create an abstracted and accessible component that users can access through designated endpoints. This model, however, structures the system so that the server component contains other important components (the map tool, the data analysis tool, the compare data tool, and the user interface tool). This increase in complexity means that there could potentially be an increase in cost and extensibility when there are system-wide modifications that are required. Thus, in this deployment blueprint, we must consider increasing reliability at the potential cost of extensibility.

- **Trade-Offs:**

1. By implementing the Client/Server Architecture Style in this deployment blueprint (DB), we are able to increase reliability – which is our highest priority according to our stakeholder needs. A centralized and accessible server allows us to maintain a certain level of data integrity since all of the environmental data is located in one accessible and centralized location. This ensures that only this data is accessed and distributed (which ensures reliability). Additionally, we are able to ensure that the data is secure, for the server only allows users to access internal information through certain endpoints with requests. This means that the system is closed for modification and is accessed by the server itself. The trade-off, however, is a slight decrease in extensibility due to the increase in complexity. This complexity hinders the ease in which we can modify the entire system to add

new features. The benefits, however, far outweigh the costs, and this trade-off is justified. By increasing reliability, we ensure that we have satisfied our most important stakeholder need. We greatly increase the value of our app at the potential cost of extensibility on the developer-side of the application.

- **Refactorings:**
    1. <u>Created Server Component:</u> For this specific deployment blueprint (DB), we created a server component that was not enumerated within our business blueprint (BB). This refactoring allowed us to implement the Client/Server Architecture Style for this deployment blueprint (DB), which allowed us to create a centralized and accessible server component that abstracted information and protected the internals of the system structure. This refactoring could potentially sacrifice the BB's vision to reduce system complexity, for we are adding an additional component that encapsulates many other important components. This refactoring, however, is justified because this component allows us to improve reliability – which is our highest priority stakeholder need. Thus, we are able to protect the environmental information as well as ensure that the system is closed for modification. This increase in reliability increases the value of our web application and greatly helps users, which is why it is justified.
    2. <u>Created User Interface Tool:</u> For this deployment blueprint (DB), we created a user interface tool component that did not exist in the business blueprint (BB). This refactoring allows us to implement the fundamental ideas of a Layered Architecture Style, which allows us to ameliorate the user interface as well as the usability of our application through an abstraction of functionality. This

refactoring could potentially sacrifice the BB's vision to reduce system complexity, for we are adding an additional component. This refactoring, however, is justified because this component allows us to improve usability and directly help the users by making the application easy to navigate through an abstraction of functionality. Thus, we are able to reduce user complexity at the expense of potentially increasing system complexity – which is justified for our application.

Part 2: Evaluating Solution Blueprint Compliance

$$\textbf{CompFuncCoeff(c, t)} = \frac{|CompFunc_{regd}(c,t)|}{|CompFunc(c)|}$$

Table 7. CompFuncCoeff(c, t) Calculations

Component (BB)	Technology (SB)	CompFuncCoeff(c, t)
Map Tool	Map Tool	**9/11**
	Data Analysis Tool	**0/11**
	Compare Data Tool	**0/11**
	Database	**0/11**
	User Interface Tool	**2/11**
Data Analysis Tool	Map Tool	**0/14**
	Data Analysis Tool	**14/14**
	Compare Data Tool	**0/14**
	Database	**0/14**
	User Interface Tool	**0/14**

Compare Data Tool	Map Tool	**0/7**
	Data Analysis Tool	**0/7**
	Compare Data Tool	**7/7**
	Database	**0/7**
	User Interface Tool	**0/7**
Database	Map Tool	**0/7**
	Data Analysis Tool	**0/7**
	Compare Data Tool	**0/7**
	Database	**7/7**
	User Interface Tool	**0/7**

$$\textbf{CompDataCoeff(c, t)} = \frac{|CompData_{regd}(c,t)|}{|CompFunc(c)|}$$

Table 8. CompDataCoeff(c, t) Calculations

Component (BB)	Technology (SB)	CompDataCoeff(c, t)
Map Tool	Map Tool	**7/8**
	Data Analysis Tool	**0/8**
	Compare Data Tool	**0/8**
	Database	**0/8**
	User Interface Tool	**1/8**
Data Analysis Tool	Map Tool	**0/1**
	Data Analysis Tool	**1/1**

	Compare Data Tool	**0/1**
	Database	**0/1**
	User Interface Tool	**0/1**
Compare Data Tool	Map Tool	**0/8**
	Data Analysis Tool	**0/8**
	Compare Data Tool	**7/8**
	Database	**0/8**
	User Interface Tool	**1/8**
Database	Map Tool	**0/7**
	Data Analysis Tool	**0/7**
	Compare Data Tool	**0/7**
	Database	**7/7**
	User Interface Tool	**0/7**

Table 9. CompFuncBoundaryError(t) Calculations

Technology (SB)	**CompFuncBoundaryError(t)**
Map Tool	(1 - (9/11)) + (0/11) = **0.1818**
Data Analysis Tool	(1 - (14/14)) + (0/11) = **0**
Compare Data Tool	(1 - (7/7)) + (0/7) = **0**
Database	(1 - (7/7)) + (0/7) = **0**
User Interface Tool	(1 - (2/11)) + (0/11) = **0.8181**

<u>Part 3: Planning Evaluation of DB Using an ATAM Quality Attribute Utility Tree</u>

<u>Utility Definition:</u> Utility is the measurement of an element's value or worth. In a software architecture context, utility is the combination of a software system's usefulness and usability.

<u>Aggregation Method for Quality Assessments:</u> In order to obtain an overall "goodness" of the system by aggregating quality assessments, we must set objectives to satisfy each quality attribute and specify metrics on how to measure the success of meeting the aforementioned objectives. Additionally, we must keep in mind that the quality attributes are weighted by priority, so a system is better if it meets its objectives and satisfies the quality attributes higher in the Utility Tree.

- **<u>UTILITY</u>**
  - **Reliability**
    - <u>Ensure Server is Closed for Modification and Open for Requests</u>
      - *Objective:* Ensure that the server component is accessible by users through web browsers and designated endpoints. Restrict the server component so that users are not able to access or modify the internal structure of the system (H, M)
        - *Metric:* Set-up an n number of machines or devices and open live connections with the application's server component. In each of the n machines or devices, run every combination of service with boundary input parameters from the client-side. Check if all of the internal structures and information is still the same before the metric evaluation (H)

- Ensure Data Integrity
  - *Objective:* Ensure that the Database contains the most recent environmental information and is updated correctly from the external data sources (H, M)
    - *Metric:* Search for environmental data with the user interface tool component. Check what information is stored in the database for that area by viewing the expected output. Then check the information procurement tool for the selected area. Compare the information and check if it is identical. (H)
  - *Objective:* Increase the number if verified data sources to pull information from (H, M)
    - *Metric:* Add sources to the infrastructure of the information procurement tool. Search for environmental data with the user interface tool component. Then check the information procurement tool for the selected area. Check to see if the number of sources and comparison calculations has increased. (H)

- **Scalability**
  - Maximize Scalability with regard to Time
    - *Objective:* Display that the server component is able to scale as the number of web browsers connecting to the application increases

overall. Ensure that the server component is able to handle an increase in traffic while hosted (H, M)

- *Metric:* Increase the number of web browsers connected to the server component by a set number n for some bound. Then run a script after every addition of elements that checks how fast it takes to return environmental information. Check if the time increases linearly or faster with the increase in web browser connections to the applcation (H)

- **Usability**
  - Ensure Services are Accessible
    - *Objective:* Navigate between services through the user interface tool in less than 1 second and less than an n number of clicks for an n number of services, where n > 0 (M, M)
      - *Metric:* Write a front-end Selenium testing script that starts at the main user page and navigates between an n number of pages in all combinations of n-orderings. Check that each instance is less than 1 second (H)
- **Compatibility**
  - Maximize Platform Accessibility
    - *Objective:* Ensure that the web application is accessible on all major web browsers (Google Chrome, Firefox, Microsoft Edge,

Safari, etc.), and check that all of the services function appropriately (H, H)

- *Metric:* Write multiple scripts (one for every web browser) that automates navigation between services within our application. Then let the script check each functionality and check it against an expected output. Check if the results are appropriate (M)

- **Flexibility**
  - Maximize System Flexibility with Regard to Time
    - *Objective:* Ensure that the system is able to remain functional and swift in terms of time even with the addition of multiple new features and services (M, M)
      - *Metric:* Measure the time of the system before any additions of features and services. Create a load-heavy service (mock function) that execute 2^n searches. Add this service to the system and measure the difference in time. Check to see if the increase in time is less than an increase of 2^n (M)
- **Time Delivery**
  - Minimize Projected Timeline
    - *Objective:* Minimize the projected timeline for the project and reduce the number of hours invested into development as well as testing (H, L)

- *Metric:* Check the original schedule. Compare how many deliverables were expected to be completed and check the actual number of deliverables completed (M)

- **Cost**
  - Minimize Cost of Project
    - *Objective:* Minimize the cost of the project and reduce investment of resources (H, M)
      - *Metric:* Determine the initial cost of investment (hardware, server-space, database orientation) (H)
    - *Objective:* Minimize the number of hours that the development team works and invests into the project (H, M)
      - *Metric:* Project how many hours are required to maintain the application and how many hours are required to complete development (H)
- **Maintainability**
  - Minimize Resource Consumption
    - *Objective:* Ensure that the consumption of data, memory, and server (bandwidth) resources are reduced for both client-side and server-side interactions (M, M)
      - *Metric:* Instantiate instances of the application n multiple devices and machines. Exhaustively run every combination of services. Check the memory, space, and bandwidth

consumed by the processes on both the client-side and server-side (M)

- **Availability**
  - <u>Maximize Active (Operation-Ready) Time</u>
    - *Objective:* Ensure that the application is always active and is always readily accessible. Ensure there are no instances of the application going down (M, M)
      - *Metric:* Instantiate a live session of the application on a machine. Measure the connection between the browser and the application and record whether there are any instances of a loss of connection. Check these recordings after an n number of days (H)
- **Extensibility**
  - <u>Minimize Coupling Between Components</u>
    - *Objective:* Minimize coupling between components and ensure that system components are highly cohesive as independent entities. Minimize dependencies between components (L, L)
      - *Metric:* Measure coupling using the "Number of Dependencies Between Components" calculations from the BB (L)