# Architectural design

The architecture serves as the blueprint for the system. It defines a structured solution to meet all the technical and operational requirements, while optimizing the quality attributes.

Further, it involves a set of significant decisions related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product.

## Quality attributes

The system considers the following quality attributes, ranked from most important to less important.

|  |  |  |
| --- | --- | --- |
| **Quality type** | **Description** | **Importance** |
| Performance | The algorithm has to be able to predict missing skills of one profile in 1 second or less. Eventually, the algorithm has to make predictions for millions of profiles and the scraping of profile data will be a continuous process, meaning the algorithm needs to be able to keep up. Additionally, users can provide profile data themselves and the algorithm should be able to predict the missing skills in an instant. | 5 |
| Performance | The system is able to predict missing skills of 1 million profiles in 24 hours. | 5 |
| Adaptability/  scalability | The algorithm must be adaptable in different or evolving software products that support Python 2.7. Additionally, the system must support hardware scalability, meaning multiple servers could be introduced that are used to make predictions with the algorithm to surpass the 1 million profiles per day prediction goal. | 5 |
| Interoperability | The algorithm and 8vance's Analysis environment as well as the algorithm and analysis tool must be able to exchange information with each other. | 5 |
| Testability | The analysis tool will mainly be used to test the quality and performance of an algorithm as well as the performance of the product itself. Multiple test criteria can be established that should measure the quality and performance of an algorithm. The input and output data of an algorithm can also be tested on correctness. | 4 |
| Correctness | The algorithm must be able to calculate certainty scores for every prediction with great precision. | 4 |
| Modifiability | Both products must be easily modifiable to improve the product quality. | 4 |
| Reusability | The algorithm must be usable in both 8vance's Analysis environment as well as in the analysis tool. | 4 |
| Installability | The analysis tool must be installable and thus usable on any operating system. | 4 |
| Modularity | Both products are composed of discrete components. When one component is changed, it should have a minimal impact on the other components. This allows for more effective changes to the system. | 4 |
| Safety | In the analysis tool, whenever the user wants to select/specify a new algorithm, data source, dump target, trained algorithm, or start an analysis process, kill an analysis process or import an analysis result, the tool must query the user if he's certain he wants to continue to prevent the loss of previous data. | 2 |
| Safety | In the analysis tool, the user is able to kill an analysis process. When the user starts an analysis process, the user can't perform any actions in the application except from killing the analysis process. If there's a problem with the analysis resulting in the process to never end, that means the user could never perform any actions. By providing the user with the option to kill an analysis process, there's a safeguard to prevent this problem. | 2 |
| Learnability | The analysis tool must be easy to use. Apart from algorithm developers, users without any understanding of the inner workings of the algorithm and the tool should be able to use and understand the intended use of the tool. | 1 |

## Architectural design options

The absolutely most important quality attributes are performance, adaptability and interoperability. Followed by the other important quality attributes testability, correctness, modifiability, installability, reusability and modularity.

### Data flow architecture

In data flow architecture, the whole software system is seen as a series of transformations on consecutive pieces or set of input data, where data and operations are independent of each other. In this approach, the data enters into the system and then flows through the modules one at a time until they are assigned to some final destination (output or a data store).

The main objective of this approach is to achieve the qualities of reusability and modifiability. It is suitable for applications that involve a well-defined series of independent data transformations or computations

There's one module that peaks the most interest.

#### Pipe and Filter

This approach lays emphasis on the incremental transformation of data by successive component. The connections between modules are data stream which is first-in/first-out buffer that can be stream of bytes, characters, or any other type of such kind. The main feature of this architecture is its concurrent and incremented execution.

##### Quality attributes coverage

This architecture design ensures the performance quality attribute. The pipe and filter pattern makes use of parallelism which increases performance. However, if every filter uses a different data structure which requires a continuous transformation of data that could negatively impact the performance. (http://portal.ou.nl/documents/114964/2986739/IM0203\_03.pdf, page 53)

This architecture design ensures the adaptability and modifiability quality attributes. Filters can be added and replaced easily. This is because of the standard interface that's required for the filters to use. (http://portal.ou.nl/documents/114964/2986739/IM0203\_03.pdf, page 43)

This architecture design ensures the reusability quality attribute. It's possible to build different pipelines by recombining a given set of filters. Filters are stand-alone and can be treated as black boxes. This isolation of functionality ensures modifiability and reusability. (http://portal.ou.nl/documents/114964/2986739/IM0203\_03.pdf, page 43; http://www4.desales.edu/~dlm1/it533/class03/pipe.html)

This architecture design ensures the modularity quality attribute to a certain point. Filters do not know anything about other filters. To increase modularity, object-oriented frameworks can be developed that allow stages to be represented by objects or procedures that can easily be used by the pipeline. Such frameworks aren't difficult to construct using standard OOP techniques. (http://www.informit.com/articles/article.aspx?p=366887&seqNum=8)

This architecture design ensures the testability quality attribute. Analysis/testing of the pipe and filter system is easy, because it's a simple composition of the behaviours of the filters involved. When the input is called x, the behaviour of the first filter is described by function *g*, and the behaviour of the second filter is described by function *f*, the result of the pipeline can be described as: *f(g(x))*. Because of this composition, it's possible to analyze/test throughput as well (determined by slowest filter). http://portal.ou.nl/documents/114964/2986739/IM0203\_03.pdf, page 43)

##### Advantages

It has following advantages:

* Provides concurrency and high throughput for excessive data processing.
* Provides reusability and simplifies system maintenance.
* Provides modifiability and low coupling between filters.
* Provides simplicity by offering clear divisions between any two filters connected by pipe.
* Provides flexibility by supporting both sequential and parallel execution.

##### Disadvantages

It has some of the following disadvantages:

* Not suitable for dynamic interactions.
* A low common denominator is needed for transmission of data in ASCII formats.
* Overhead of data transformation between filters.
* Does not provide a way for filters to cooperatively interact to solve a problem.
* Difficult to configure this architecture dynamically.

##### Analysis

This architecture design is particularly useful for pre- and post-processing the data, and executing the algorithm. These steps can be linked together and executed in a particular order. However, having the flexibility to switch filters isn't particularly useful for this system because there simply won't be a need to switch them. The pipeline would always start with pre-processing the data and stop with post-processing the data. The filters in between would call the algorithm functionality to predict the missing data. This order of filters won't ever change, so the strengths of this architecture design like modifiability and reusability don't count. This means the strength of this design currently lies in the performance, testability, and to a certain point, modularity.

#### Layers architecture

Layered architecture focuses on the grouping of related functionality within an application into distinct layers that are stacked vertically on top of each other. Functionality within each layer is related by a common role or responsibility. Communication between layers is explicit and loosely coupled.

The layers of an application may reside on the same physical computer (the same tier) or may be distributed over separate computers (n-tier), and the components in each layer communicate with components in other layers through well-defined interfaces. For example, a typical layered architecture consists of a presentation layer (functionality related to the UI), a business layer (business rules processing), and a data layer (functionality related to data access, often almost entirely implemented using high-level data access frameworks).

##### Quality attributes coverage

This architecture design ensures the reusability quality attribute. For example, future presentation layers can reuse the business and data access layers because the lower layers have no dependencies on the higher layers.

This architecture design ensures the manageability quality attribute. The separation of core concerns helps to identify dependencies, and organizes the code into manageable sections.

This architecture design ensures the performance and adaptability quality attributes. Distributing the layers over multiple physical tiers can improve the scalability, fault tolerance and performance.

This architecture design ensures the testability quality attribute. Having well-defined layer interfaces increases the testability, as well as the ability to switch between different implementations of the layer interfaces. Separated Presentation patterns can be considered to improve the testability by building mock objects that mimic the behaviour of concrete objects such as the model, controller or view.

This architecture design ensures the modularity quality attribute. The layered architecture is implemented by using component-based technology which makes the system much easier to allow for plug-and-play of new components.

##### Advantages

It has following advantages:

* Design based on incremental levels of abstraction.
* Provides enhancement independence as changes to the function of one layer affects at most two other layers.
* Separation of the standard interface and its implementation.
* Implemented by using component-based technology which makes the system much easier to allow for plug-and-play of new components.
* Each layer can be an abstract machine deployed independently which support portability.
* Easy to decompose the system based on the definition of the tasks in a top-down refinement manner
* Different implementations (with identical interfaces) of the same layer can be used interchangeably

##### Disadvantages

It has the following disadvantages:

* Lower runtime performance since a client’s request or a response to client must go through potentially several layers.
* There are also performance concerns on overhead on the data marshaling and buffering by each layer.
* Opening of interlayer communication may cause deadlocks and “bridging” may cause tight coupling.
* Exceptions and error handling is an issue in the layered architecture, since faults in one layer must spread upwards to all calling layers

##### Analysis

The system involves distinct classes of services that can be organized hierarchically. The system also has clear divisions between core services, critical services and user interface services. Performance can be a concern if there're a lot of layers information must go through, but that won't be the case for this system.

So the strength of this design lies in the reusability, manageability, performance, adaptability, testability and modularity.

#### Blackboard

In Blackboard Architecture Style, the data store is active and its clients are passive. Therefore the logical flow is determined by the current data status in data store. It has a blackboard component, acting as a central data repository, and an internal representation is built and acted upon by different computational elements.

Further, a number of components that act independently on the common data structure are stored in the blackboard. In this style, the components interact only through the blackboard. The data-store alerts the clients whenever there is a data-store changes. The current state of the solution is stored in the blackboard and processing is triggered by the state of the blackboard.

A major difference with traditional database systems is that the invocation of computational elements in a blackboard architecture is triggered by the current state of the blackboard, and not by external inputs.

The blackboard model uses so-called Knowledge Sources (KS) to solve parts of a problem and aggregate partial results. Interaction among knowledge sources takes place uniquely through the blackboard.

##### Quality attribute coverage

This architecture design ensures the scalability quality attribute. Knowledge sources can work in parallel as they're independent of each other. More knowledge sources can be added or updated easily.

This architecture design ensures the reusability quality attribute. Knowledge sources can be reused as they do not have direct communication with each other.

##### Advantages

* Blackboard Model provides concurrency that allows all knowledge sources to work in parallel as they are independent of each other.
* Its scalability feature facilitates easy steps to add or update knowledge source.
* It supports experimentation for hypotheses and reusability of knowledge source agents.

##### Disadvantages

* The structural change of blackboard may have a significant impact on all of its agents, as close dependency exists between blackboard and knowledge source.
* Blackboard model is expected to produce approximate solution; however, sometimes, it becomes difficult to decide when to terminate the reasoning.
* This model suffers some problems in synchronization of multiple agents, therefore, it faces challenge in designing and testing of the system.

##### Analysis

This architectural design could be used to execute the algorithm functionality to create the predictions. However, the blackboard design is mostly effective when the knowledge sources are truly independent of each other, which isn't the case for this system. At least one knowledge source has to do the pre-processing part and the other knowledge sources that create predictions or post-process the data need to have the pre-processed data. So there's already a dependency here between those knowledge sources. This also means this design loses some scalability as not every knowledge source can work in parallel.

Performance is not a strength of this architecture design because of the mentioned problem with not being able to run all the knowledge sources in parallel. Knowledge sources that require pre-processed data need to wait for the knowledge sources that deliver that data. And knowledge sources that want to post-process the data need to wait for the knowledge sources that deliver the predicted data. The knowledge sources that deliver the predicted data can be run in parallel, but that also means that the blackboard needs the synchronize the delivery of this data and the execution of the post-processing knowledge sources. This means the blackboard would be complex to build and manage, and can be the bottleneck of the system.

So the main strength of this design lies in its reusability.