**Time Series Analysis Support for Data Scientists  
Software Requirements Specification**

E. Petersen (ezekielp) - 1-23-2021 - v1.1

**Table of Contents**

1. SRS Revision History 2

2. The Concept of Operations (ConOps) 2

2.1. Current System or Situation 2

2.2. Justification for a New System 2

2.3. Operational Features of the Proposed System 3

2.4. User Classes 3

2.5. Modes of Operation 3

2.6. Operational Scenarios (Also Known as “Use Cases”) 4

3. Specific Requirements 5

3.1. External Interfaces (Inputs and Outputs) 6

3.2. Functions 7

3.3. Usability Requirements 7

3.4. Performance Requirements 7

3.5. Software System Attributes 7

4. References 8

5. Acknowledgements 8

# 1. SRS Revision History

This lists every modification to the document. Entries are ordered chronologically.

**Date Author Description**

**————— ——— ————————————————————————————**

1-22-2021 ezekielp Added initial document to repo, will modify template for project

1-23-2021 ezekielp Replaced template text with project specifics in ConOps 2 - 2.5

# 2. The Concept of Operations (ConOps)

The concept of operations (ConOps) “describes system characteristics for a proposed system from the users’ viewpoint.” (IEEE Std 1362-1998) The ConOps document communicates overall system characteristics to all stakeholders.

## 2.1. Current System or Situation

Data Scientists are uniquely situated to apply emerging scientific and machine learning processes and algorithms to analyze real world data and make important inferences about the future. One such category of inferences concerns Time Series forecasting, a task for predicting information about a wide range of human activity to improve real world decision making. For example, predicting solar irradiance as a function of time can help solar engineers make operational decisions at renewable energy plants. Predictions made by programs created by Data Scientists can be vital to maximizing energy capture and minimizing operating costs.

There are existing libraries, such as those provided by pandas or scikit-learn that Data Scientists currently use to analyze various data sets and generate “pipelines” to convert raw TS data into usable forecasting models for industry professionals and others. Data Scientists use these libraries to decide which pipeline most accurately predicts the future state of the TS. However, this is a non-linear search for effective pipelines that is currently performed in a largely “test and check” manner.

## 2.2. Justification for a New System

The current “system” is not as thorough or efficient as it could be because Data Scientists are forced to do non-linear searches for effective pipelines manually. In a literal sense, there is no existing system for achieving a comprehensive search for pipelines at low cost to the Data Scientist.

To illustrate the problem, we introduce the math involved for constructing a hypothetical pipeline. Even in a modest case where we apply 3 preprocessing directives out of 10 available to us, a single modeling strategy out of 6, and a single evaluation strategy out of 3, there are thousands of possible pipelines that could be formed (10 \* 9 \* 8 \* 6 \* 3 = 12,960), which would be difficult to track and compare without additional software support. Since we know that our list of basic transformations described later (which approximately reflects the above situation) is not exhaustive and we do not include the various plotting or graphical analysis steps available in the hypothetical pipeline, we can reasonably assume this is by no means the worst-case problem complexity. So, given the shortcomings of the current solutions, we argue that our system can improve the accuracy of these pipelines and the productivity of the Data Scientists to develop better forecasting models.

## 2.3. Operational Features of the Proposed System

Our new system aims to represent the challenge of conducting a non-linear search for an effective pipeline and model by treating the transformation steps taken by Data Scientists as nodes in a “Transformation Tree”. We aim to give Data Scientists a library of functions to modify this Transformation Tree and keep all possible pipeline steps in one place to be later executed and compared. These possible modifications will include creating a new tree, inserting new operator nodes, replacing existing operators with another, replicating subtrees or tree paths, and adding a subtree or path to an existing node. We also wish to allow Data Scientists to extract successful pipelines or trees and save/load them when necessary to put into production or update/maintain later.

By organizing the TS transformation steps as an n-ary tree, Data Scientists will be able to keep all design decisions in one place and compare the many variants of a pipeline. However, the system will limit types of operators to follow the logical progression from preprocessing to modeling and finally to evaluation to ensure data consistency as it moves through the tree. This will prevent nonsensical situations such as an attempt to evaluate a model prior to a model being created. In order to achieve this, nodes and operators may be classified according to their role in the pipeline process.

## 2.4. User Classes

For the proposed system, there is only one major user class: Data Scientists. We are defining a library of functions and tree transformations to be used by these Data Scientists to streamline their everyday processes. We are assuming a proficient level of programming knowledge such that users may call our transformation functions to process, analyze, and visualize a given Time Series in ways they see fit according to their broad domain knowledge. Data Scientists already use the libraries behind the functional modules of our system, so it is reasonable to assume Data Scientists will understand and be able to use a synthesis and repackaging of those libraries.

## 2.5. Modes of Operation

As there is only one user class and because the end result will be a library for use by Data Scientists, the library itself is the only true “mode of operation”. There will be unit tests and example scripts for using the library, however, primary use will be by Data Scientists for use in their own scripts and analysis.

## 2.6. Operational Scenarios (Also Known as “Use Cases”)

- Ex: Create a tree, add several layers of preprocessing directives, a couple modeling options, and an evaluation strategy. Perhaps none of the results end up being acceptable and the DS wishes to insert a scaling preprocessing step early in the tree and change the modeling strategy for one of the nodes and test again. Perhaps that results in a satisfactory result for one of the tree paths and the DS wishes to save the path as a pipeline for later loading to use on other data sets.

“An operational scenario [also known as “Use Cases”] is a step-by-step description of how the proposed system should operate and interact with its users and its external interfaces under a given set of circumstances. Scenarios should be described in a manner that will allow readers to walk through them and gain an understanding of how all the various parts of the proposed system function and interact. The scenarios tie together all parts of the system, the users, and other entities by describing how they interact.” (IEEE Std 1362-1998)

‘Operational scenarios should describe operational sequences that illustrate the roles of the system, its interactions with users, and interactions with other systems. Operational scenarios should ideally be described for all operational modes and all classes of users identified for the proposed system. Each scenario should include events, actions, stimuli, information, and interactions as appropriate to provide a comprehensive understanding of the operational aspects of the proposed system. Prototypes, storyboards, and other media, such as video or hypermedia presentations, may be used to provide part of this information.’ (IEEE Std 1362-1998)

Include, in each operational-scenario (or use-case), a one-sentence description of the scenario, a list of the users or “actors” (from the user classes) involved in the scenario, the preconditions for starting the scenario, and the postconditions (the relevant status of the system and world) after the scenario is completed.

Structure the writing of operational scenarios, or use cases, so that they are easy to read, with headings and numbered steps rather than in paragraph form. For example, note how the following is well-structured to make it easy to read.

**Use Case: Get a copy of an online book.**

***Brief description:*** This use case describes how a student would make a digital copy of the required reading for a course, for a textbook that is made available to student online, but which the student cannot download to his or her computer.

***Actors:*** A student.

***Preconditions:***

1. The student has access to an online digital copy of the assigned reading.

2. The reading material can be viewed one full and sufficiently-high-resolution page at a time.

3. The page can be advanced one page with a consistent onscreen button, mouse command, or keyboard command.

4. Each page of reading material is positioned in the exact same rectangle on the computer screen.

5. The student can load applications on the machine accessing the online digital copy of the reading, and has access to substantial (such as 100MB) hard drive space on that machine.

***Steps to Complete the Task:***

1. The student gains access to the online digital copy of the assigned reading.

2. The user figures out:

(a) How to display the reading material one page at a time, with the page filling as much of the screen as possible (perhaps even turning the computer screen 90°).

(b) What commands are available to advance through the reading one page at a time.

3. The user starts up the required-reading-assistant software.

...

***Postconditions:***

The user has a PDF of all of the required reading, with OCR (optical character recognition) applied, and with a separate chapter within the PDF for each reading assignment or chapter in the book.

You can read more on use cases in the Oracle (2007) White Paper on “Getting Started With Use Case Modeling”, available at: <https://www.oracle.com/technetwork/testcontent/gettingstartedwithusecasemodeling-133857.pdf>

**Note that diagrams can assist with communication.**

“Graphical tools should be used wherever possible, especially since ConOps documents should be understandable by several different types of readers. Useful graphical tools include, but are not limited to, work breakdown structures (WBS), N2 charts, sequence or activity charts, functional flow block diagrams, structure charts, allocation charts, data flow diagrams (DFD), object diagrams, context diagrams, storyboards, and entity-relationship diagrams.” (IEEE Std 1362-1998).

# 3. Specific Requirements

This is where the actual requirements are specified. A requirement is a description of a behavior or property that a computer program must have, independent of how that behavior or property is achieved. Requirements must be complete, unambiguous, consistent, and objectively verifiable (see van Vliet, 2008, pp.241-242, for a discussion of these terms). Requirements describe what the system will do, but do not commit to specific *design* details of how the system will do it.

Requirements should be organized in a hierarchy. A good organization (a) makes requirements easier to read and understand because related requirements will be near each other in the document, (b) makes requirements easier to modify and update, and (c) makes it easier to find a specific requirement. There are a number of ways that requirements can be organized to help achieve these goals.

The following section headings provide one way to organize the requirements. You can adapt this organization. For example, sections 2.1, 2.2, 2.3, and 2.4 (see headings below) describe “behavioral requirements” (Faulk, 2013). If a system supports two major user activities, it might be best to describe the behavioral requirements for each activity separately. For example, in a digital deejaying system, the two major activities could be (a) load songs into the system and (b) use the system to play songs. It might be best to fully specify everything in sections 2.1 through 2.4 first for the song-loading activity, and then for the song-playing activity.

Though there is a tradition of distinguishing between “functional” and “non-functional” requirements, with the former describing services provided by the system, and the latter describing constraints on the system and its development, in actual practice this is not a very useful distinction, and is not a good basis for structuring requirements (Faulk, 2013).

Requirements should be prioritized, with each classified as (a) must have, (b) should have, (c) could have, and (d) won’t have. These can be recalled with the memory aid of MoSCoW (vanVliet 2008, p.237). When reading requirements, it should be very easy to see how each requirement is classified, such as by having them grouped by priority.

Throughout the document, lists and sublists of requirements should be indented and numbered to make it easy to read and reference the specification details. Such as:

1. *General Requirement*

1.1. *Specific Requirement*

1.1.1 *Requirement Detail*

Note how this permits reference to "SRS Item 1.1.1".

## 3.1. External Interfaces (Inputs and Outputs)

This section should describe inputs into and outputs from the software system. (ISO/IEC/IEEE 29148:2011)

Each interface description should include the following:

1. Name of item.

2. Description of purpose.

3. Source of input or destination of output.

4. Valid ranges of inputs and outputs.

5. Units of measure.

6. Data formats.

## 3.2. Functions

Define the actions that must take place in the software to accept and process inputs, and to generate outputs (ISO/IEC/IEEE 29148:2011). This should include:

1. Validity checks on the inputs.

2. Sequence of operations in processing inputs.

3. Responses to abnormal situations, including error handling and recovery.

4. Relationship of outputs to inputs, including

(a) input/output sequences

(b) formulas for input to output conversion

## 3.3. Usability Requirements

Define usability requirements and objectives for the software system, include measurable effectiveness, efficiency, and satisfaction criteria in specific contexts of use. (ISO/IEC/IEEE 29148:2011)

## 3.4. Performance Requirements

Specify the static and the dynamic numerical requirements placed on the software or on human interaction with the software. For example: (a) Static numerical requirements may include the amount and type of information to be handled. (b) Dynamic numerical requirements may include the amount of data to be processed within certain time periods.

Performance requirements should be stated in measurable terms. For example,

“95% of the transactions shall be processed in less than 1 second”

rather than

“An operator shall not have to wait for the transaction to complete.”

(ISO/IEC/IEEE 29148:2011)

## 3.5. Software System Attributes

Specify the required attributes of the software product, such as reliability, security, privacy, maintainability, or portability. (ISO/IEC/IEEE 29148:2011) Review a comprehensive list of software attributes, or software qualities, such as are provided in van Vliet (2008) Chapter 6. Decide on a relatively small number of attributes that are most important for this system. Explain why each attribute is important, and what steps or plan will be taken to achieve those attributes. This could include constraints on attributes of the system’s static construction, such as testability, changeability, maintainability, and reusability. (Faulk, 2013)

# 4. References

This section lists the sources cited in the creation of this template document. An SRS should reference all of the sources that it draws from. If sufficient citations are provided “in line” (at the point of reference) in the document, this section may not be necessary.

IEEE Std 1362-1998 (R2007). (2007). IEEE Guide for Information Technology–System Definition–Concept of Operations (ConOps) Document. <https://ieeexplore.ieee.org/document/761853>

IEEE Std 830-1998. (2007). IEEE Recommended Practice for Software Requirements Specifications. <https://ieeexplore.ieee.org/document/720574>

ISO/IEC/IEEE Intl Std 29148:2011. (2011). Systems and software engineering — Life cycle processes — Requirements engineering. <https://ieeexplore.ieee.org/document/6146379>

ISO/IEC/IEEE Intl Std 29148:2018. (2018). Systems and software engineering — Life cycle processes — Requirements engineering. <https://ieeexplore.ieee.org/document/8559686>

Faulk, Stuart. (2013). *Understanding Software Requirements*. <https://projects.cecs.pdx.edu/attachments/download/904/Faulk_SoftwareRequirements_v4.pdf>

Oracle. (2007). White Paper on “Getting Started With Use Case Modeling”. Available at: <https://www.oracle.com/technetwork/testcontent/gettingstartedwithusecasemodeling-133857.pdf>

van Vliet, Hans. (2008). *Software Engineering: Principles and Practice*, 3nd edition, John Wiley & Sons.

# 5. Acknowledgements

All sources used in the creation of the document should be listed here. This template builds slightly on a similar document produced by Stuart Faulk in 2017, and heavily on the publications cited within the document, such as IEEE Std 1362-1998 and ISO/IEC/IEEE Intl Std 29148:2018.

**Possible Minor Future Revision**

The ConOps portion of this template was written based on IEEE Std 1362-1998 before realizing that this standard has been subsumed by IEC/IEEE Intl Std 29148:2018. A future exercise could, in short, determine if the references to the former could be replaced with references to the latter; that is, if the ConOps description in each of the two documents is sufficiently identical.