VU VRIJE UNIVERSITEIT AMSTERDAM

Software Testing 2020

Assignment PROJECT: Heart Monitor
Software Requirements Specification

Group: 11 - Korean Air

Members: Markus Funke (2644722)
Wouter Kok (2639768)
Sven Preng (2614131)
Pjotr Scholtze (2612369)

Master program: Computer Science - SEG

June 19, 2020

Contents

1 Introduction

1.1 Purpose

This Software Requirements Specification (SRS) is regarding the Heart Beat Monitor Simulation Software **HB-Sim2020** that is to be developed during the Software Testing course at the VU (2019-2020). The document is intended for future developers, testers, the product owner and other stakeholders.

1.2 Scope

The Heart Beat Monitor Simulation Software is to be developed as part of the PROJ assignment of the Software Testing course. The main purpose of the software is to be tested after development to discuss the possible bugs for educational purposes. The software will run a simulation of a generic heartbeat monitor used in a medical facility. The heartbeat monitor should be able to notify the medical team of any anomalies in the oxygen levels, heart beat and blood pressure as well as give a constant update on these current values via a command line interface. Since this software should simulate a heartbeat monitor, it is not connected to any actual sensors and thus will receive it's input through a test file.

1.3 Definitions, acronyms, and abbreviations

- HB-Sim2020 - Heart Beat Monitor Simulation Software
- SRS - Software Requirements Specification
- BPM - Beats Per Minute
- CSV - Comma-separated values
- CLI - Command Line Interface
- QAS - Quality Attribute Scenario
- mmHg - millimeter of mercury
- HR - Heart Rate
- BP - Blood Pressure

1.4 Technologies to be used

To adhere to the standards for the PROJ assignment, the software will be written in a recent version of python (3.7.7) while using minimal external libraries in order to reduce possible bugs that might be introduced by such libraries. A comprehensive overview of all relevant and used technologies/libararies can be found in section 5.1.

1.5 Overview

The SRS is organized as follows. Section 1 introduces the document, section 2 describes the general factors that affect the product and its requirements. It provides background for the actual requirements which are defined in section 3 and 4. The non-functional requirements describes *how* the system *should* work. The functional requirements examine the specific behavior of the *HB-Sim2020* system and what the system is supposed to *do*.

2 Overall description

2.1 Product perspective

The implemented system *HB-Sim2020* can be defined as virtual implementation of an actual hardware based heart rate monitor. Since a hardware monitor is connected to actual a) pulse-, b) blood-pressure-, and c) oxygen-sensors, the virtual implementation needs to work without any real sensor connections. Figure 1 and figure 2 distinguish the two different approaches in a visual manner. While the real world environment uses an actual wave analyzer to display the current values, the virtual environment uses a comma-separated-value (CSV) file to simulate incoming sensor data and prints all values and messages to a Command Line Interface (CLI) in a text based manner. The following sections all refer to the virtual environment as in figure 2.

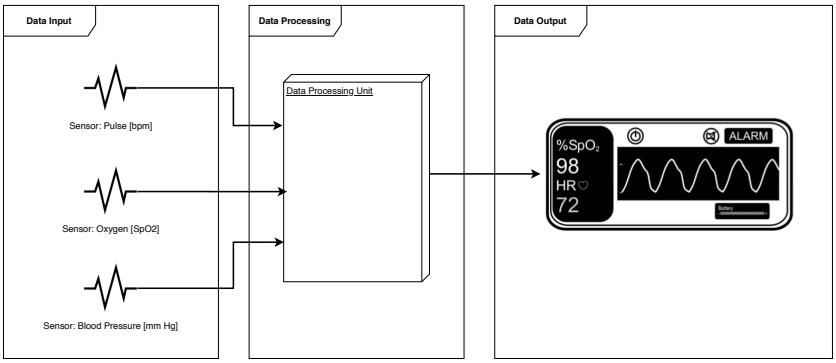

Figure 1: Heart Rate Monitor - real world environment (image: [1])

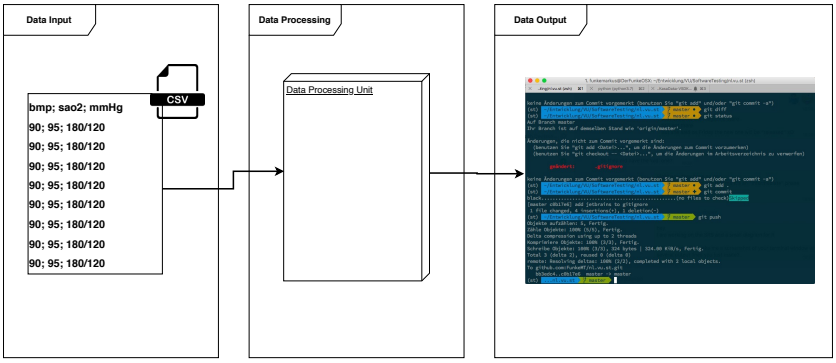

Figure 2: Heart Rate Monitor - virtual environment

2.1.1 System interfaces

The system collects data from a CSV file exclusively which simulates real sensor data. The necessary file structure and requirements are described in section 4.2.2. Beyond, the system is connected to the file system for logging. If the disk is full or if the user has no write access, the program will not stop; it will inform the user that it cannot write to disk and will still continue to operate. The CSV file is the only interface which is connected to the system.

2.1.2 User interfaces

The user interface is provided as CLI prompt. The CLI offers the option to specify the CSV path at program start. After initialisation, no further user inputs are available and valid. The program works autonomous and switches to the data processing mode. Whilst the program is in this mode, the CLI presents the processed data and various information. Detailed information about the user interface is explained in section SR#4.2.7

2.1.3 Operations

It is required that the user provides a CSV file for the program. The CSV file should have a header with the fields as denoted in 4.2.2, every other row should contain unsigned integer values with a data snapshot. Every row after the heading represents a snapshot of data, the program interprets every row as a unit of time defined in 4.2.2.

2.2 Medical Terms

Since the system deals with several medical units and medical terms, the following section provides a brief overview of the different units in use.

2.2.1 Heart Rate (pulse)

Heart Rate (pulse)	Abbreviation	HR
	Unit	`bpm` - beats per minute
	Value range	$0bpm...230bpm$ (exclusive)
	Description	"Heart rate is the speed of the heartbeat measured by the number of contractions (beats) of the heart per minute (bpm). The heart rate can vary according to the body's physical needs, including the need to absorb oxygen and excrete carbon dioxide. It is usually equal or close to the pulse measured at any peripheral point. Activities that can provoke change include physical exercise, sleep, anxiety, stress, illness, and ingestion of drugs."[2] Heart Rate and Pulse are used interchangeable throughout the document. Pulse is used in the application.

Table 1: Heart rate

2.2.2 Oxygen

Oxygen	Abbreviation	SpO_2
	Unit	% - percent
	Value range	0%...100%
	Description	"Oxygen saturation is the fraction of oxygen-saturated hemoglobin relative to total hemoglobin (unsaturated + saturated) in the blood. The human body requires and regulates a very precise and specific balance of oxygen in the blood. Normal arterial blood oxygen saturation levels in humans are 95–100 percent."[3]

Table 2: Oxygen

2.2.3 Blood Pressure

Blood pressure	Abbreviation	BP
	Unit	$mmHg$ - millimeter of mercury
	Value range	$0/0mmHg$...$300/300mmHg$
	Description	"Blood pressure (BP) is the pressure of circulating blood on the walls of blood vessels. Most of this pressure is due to work done by the heart by pumping blood through the circulatory system. Used without further specification, "blood pressure" usually refers to the pressure in large arteries of the systemic circulation. Blood pressure is usually expressed in terms of the **systolic pressure** (maximum during one heartbeat) over **diastolic pressure** (minimum in between two heartbeats) and is measured in millimeters of mercury (mmHg), above the surrounding atmospheric pressure."[4]

Table 3: Blood pressure

2.3 Product functions

This section examines the product features in a high level manner. Detailed and traceable functional requirements are described and examined in the actual requirements section 4. Functions can be grouped into three different groups, i.e. i) data input, ii) data processing, and iii) data output. The groups contain multiple sub-functions.

2.3.1 Data input

Data input	CLI	CSV location can be specified by a command-line argument at program start.
		CLI provides a *help* menu.
	CSV	CSV file simulates the sensor data.
		The CSV file contains all necessary sensors (blood pressure, heart rate, oxygen) and their possible circumstances (no data, corrupted data, etc.)

Table 4: Functions - data input

2.3.2 Data processing

Data processing	validation	Input values will be validated against specific rules (i.e. numeric values, strings, etc.)
	error handling	Corrupted incoming values will be handled without terminating the program.
	blood pressure	Processing and evaluation.
	oxygen	Processing and evaluation.
	heart rate	Processing and evaluation.
	statistic	A statistic keeps track of all processed values.

Table 5: Functions - data processing

2.3.3 Data output

Data output	CLI	All messages will be printed to the CLI.
		The messages will be formatted in a proper manner such that the messages can be read easily.
	status messages	Status messages contain the *status, value, unit* of oxygen, heart rate, and blood pressure.
	statistic	A statistic of all processed values will be displayed if the program ends or terminates.

Table 6: Functions - data output

2.4 Entity Relation Diagram

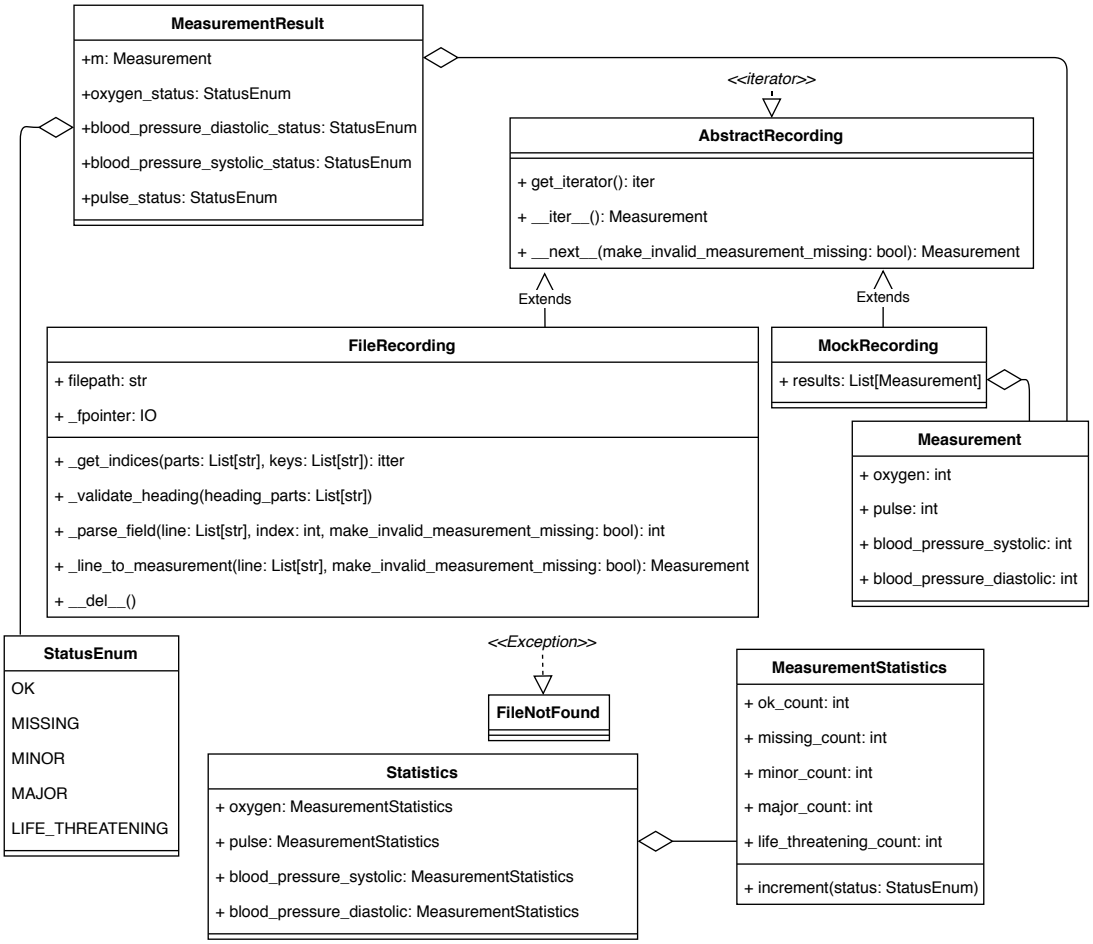

Figure 3: ER Diagram

2.5 Constraints

This section lists restrictions that had to be made due to time and scope constraints.

1. The scope of systems the program needs to support are:
    - Windows 10 with Linux subsystem version: Linux 4.4.0-18362-Microsoft x86_64
    - Windows 10 10.0.18363 Build 18363
    - Linux Ubuntu 20.04 x64
    - MacOS 10.13.6
2. The architecture of the operating system has to be: x64.
3. The program will be written in Python 3. Other operating systems (version) that are supported by Python but are not listed above are not officially supported by our application.
4. The program will be written on the assumption that the system has enough RAM available. This is a requirement to run the application, and any hardware that will run our application needs at least 256MB of free ram before the program should be executed. However, the program should not have to check this requirement.
5. The program does not require any special permissions on the host operating system other then those required to read the intended file from the file system and permission to write if logging is required. However, if writing permissions are not available, the system will omit logging and inform the user.
6. The program does not check for credentials before running.
7. The program does not need to check whether the file has been manipulated.
8. The program does not need to work with real sensors, a CSV file will simulate the sensors.
9. The program does not support network paths.
10. The program does not check the CSV file size or out-of-memory problems.
11. For this system we do not have access to actual sensors for the required measurements. For this reason we choose to work with CSV input under the assumption that this file is in the correct format available in SR#4.2.2.

3 Specific non-functional requirements

The non-functional requirements are expressed as Quality Attribute Scenarios (QAS). The scenarios are defined through a specific source, which triggers the whole scenario and leads into the response measure.

3.1 Reliability

ID: QA-#3.1

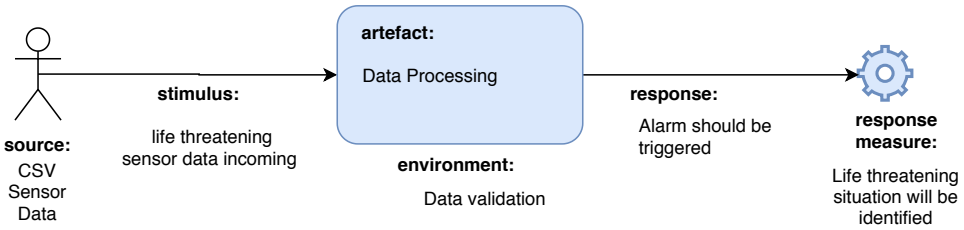

Figure 4: ASR Reliability

The QAS for the quality attribute *Reliability* is based on incoming life threatening sensor data. A life threatening situation could be caused by high blood pressure (i.e. hypertensive crisis) if the systolic and/or diastolic value is higher than 180 or 120, respectively. The goal is that this situation is immediately marked as life threatening. This allows a fast intervention by medical personnel.

3.2 Availability

ID: QA-#3.2

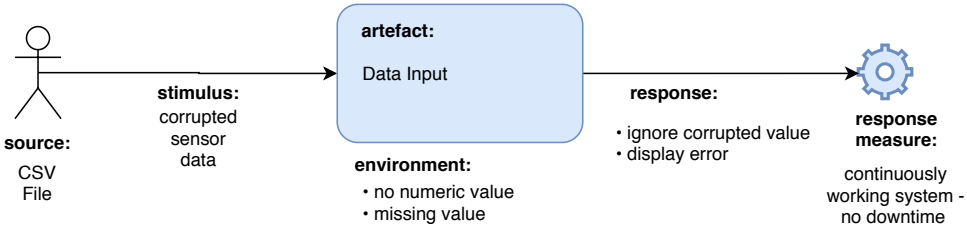

Figure 5: ASR Availability

The QAS for the quality attribute *Availability* is triggered by corrupted sensor data inside the CSV file. The error could be caused by a malformed header or non-numeric values. Through a sufficient validation process, misleading values are ignored and the status MISSING will be displayed. This behavior ensures a continuously working system.

3.3 Functional Completeness

ID: QA-#3.3

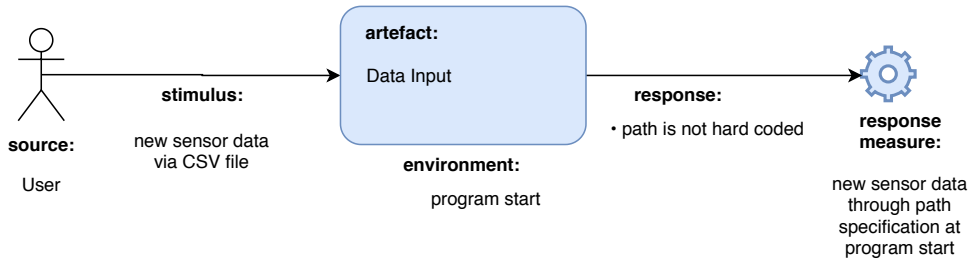

Figure 6: ASR Functional Completeness

The QAS for the quality attribute *Functional Completeness* is based on the concern of a user to add own sensor data as CSV file. The CSV file selection should be possible through a simple command line argument to allow the usage of any file location.

4 Specific requirements

According to Lehtola et al. [5] and other researches [6, 7], requirements prioritization is specified as an informal process and relies highly on individual conditions. To meet the project stakeholders' expectations and keep the project scope along with time constraints in mind, we used the following factors to prioritize the identified requirements in the following order:

1. Non-functional requirements (see section 3)
2. Project deadline
3. Cost-value relation
4. Developer skill level
5. Technical dept

After an informal discussion between the involved team members and the consideration of the previously mentioned factors, we identified the three priorities a) high, b) medium, and c) low, which are explained in detail in table 7.

Priority	Definition
High	The requirement has to be implemented in all circumstances and has to be delivered with the final release (must have). The actual implementation has to be done exactly as described in this document. Without this requirement, the system does not work.
Medium	The requirement has to be implemented and has to be delivered with the final release (must have). The requirement does not affect the actual functionality of the system directly. If the actual implementation differs from the requirement specification, the system still works as specified but with less (e.g.) usability.
Low	The requirement is defined as optional. If all requirements with prioritization high and medium are implemented and the deadline is not yet reached, this requirement could be taken into consideration. Besides, this requirement does not affect the main functionality of the system, but adds upon that.

Table 7: Requirements priorities

4.1 External interface requirements

4.1.1 User interfaces

The command line interface (CLI) is used as an user interface. Therefore all user inputs are received and accepted by the CLI. There is no dedicated graphical user interface. The system feature *User interface* 4.2.7 examines the user interface in detail.

4.1.2 Hardware interfaces

Hardware interface for our application are not directly available, you could however use the CSV to provide captured data from the hardware interface to the application. Which allows external hardware to provide data to our application. This interfacing is limited because it is a playback of readings.

4.1.3 Software interfaces

The application uses the software interface for the file system, to read recorded data and write the log if possible to disk.

4.1.4 Communications interfaces

As communication interface, the program uses the CSV-file format. The exact specifications of this format can be found at 4.2.2, 4.2.1 and an example at 5.2.

4.2 System features

4.2.1 Data input - Input recording location

Attribute	Content
Title	Input recording location
ID	SR-#4.2.1
Priority	High
Implemented [in version]	Yes [1.0]
Cross-Reference	QA-#3.3
Description	The location of the recording should be given to the application as a command-line parameter. This is required to run the application. The format of the recording should be CSV and the first row should be a heading row denoting in which column which value should be.
Stimulus/Response sequence	• **Argument not given**: Help message should be shown. • **Argument given**: The program should parse the first line of the argument. This line should contain all fields with information of the sensory data. If not, it should show an error message and exit. If there are no errors, the program should continue.
Functional requirements	• **FR-#1**: The location of the recording should be given by a command-line argument, *--path*. • **FR-#2**: The recording should be written in CSV format using ”,” (comma) as a separator. (see SR-#4.2.2) • **FR-#3**: Incomplete or incorrect header information will result in showing the same help information as missing *--path*, with information showing how to point to a correct CSV.

4.2.2 Data input - CSV file

Attribute	Content
Title	Comma-separated values (CSV) file
ID	SR-#4.2.2
Priority	High
Implemented [in version]	Yes [1.0]
Cross-Reference	QA-#3.3; SR-#4.2.1; SR-#4.2.7
Description	The system collects the data from a CSV file exclusively which simulates real sensor data. The CSV file is given as command line argument at program start. The file will also be used to simulate broken-, corrupted- or defect-sensors.
Stimulus/Response sequence	• **CSV file is given**: The program starts as excepted and starts reading from file and starts with data processing. • **CSV file is NOT given**: The program provides the help message. • **CSV file is given but does not exists**: The program provides the help message. • **CSV file contains a corrupted header**: The program provides the help message. • **CSV file contains corrupted data/values**: The program starts and reads from file. If the values are out of range (see processor for detailed ranges) the value will be counted as 0 and MISSING (see SR-#4.2.7). • **Not enough rights to read the file**: The program does not have enough rights to read the file, the program should show the help message and exit.

Functional requirements	• **FR-#1**: The recording should be written in CSV format according **rfc4180** [9]. • **FR-#2**: The CSV format should be using "," (comma) as a separator. • **FR-#3**: The CSV format should be able to handle line endings in Windows (CRLF), Linux (LF) and MacOS (LF) format. • **FR-#4**: The first row of the CSV is a heading row, containing all of the following values in any order: *oxygen,pulse,blood_pressure_systolic,blood_pressure_diastolic*. These values denote the order of the columns, and are used by the program to determine which column has which data so it can parse it correctly. • **FR-#5**: Every row in the CSV file represents one second of recording.
Example	An example CSV file listing can be found in appendix 5.2

4.2.3 Data processing - Oxygen measurements

Attribute	Content
Title	Oxygen measurements
ID	SR-#4.2.3
Priority	High
Implemented [in version]	Yes [1.0]
Cross-Reference	QA-#3.1; SR-#4.2.7
Description	The heartbeat monitor needs to sensor multiple measurements. One of them is oxygen. When an oxygen measurement comes in, it needs to be checked if the oxygen saturation levels are correct in the blood.
Stimulus/Response sequence	• **Incoming measurement**: The measurement is processed by a processor, which analyses the value and returns the right status message.
Functional requirements	• **FR-#1**: Return correct status message for the current oxygen level. The correct messages are: – OK for oxygen 95 or higher – MINOR for oxygen 90 to 94 (inclusive) – MAJOR for oxygen 61 to 89 (inclusive) – LIFE_THREATENING for oxygen 60 or below • **FR-#2**: Return MISSING status message when the value for oxygen is not a number or outside the range of 0 and 100 (numbers are inclusive). • **FR-#3**: The statistics of each value should be kept track of. • **FR-#4**: Values can only be positive integers.
References	Oxygen stages: [10]

4.2.4 Data processing - Pulse measurements

Attribute	Content
Title	Pulse measurements
ID	SR-#4.2.4
Priority	High
Implemented [in version]	Yes [1.0]
Cross-Reference	QA-#3.1; SR-#4.2.7
Description	The heartbeat monitor needs to sensor multiple measurements. One of them is pulse. When the measurement comes in it needs to be checked if it is a valid measurement and in which category the measurement falls.
Stimulus/Response sequence	• **Incoming measurement**: The measurement is processed by a processor, which analyses the value and returns the right status message.
Functional requirements	• **FR-#1**: Return the correct status messages for the current pulse. The correct message are, numbers are inclusive: – Min pulse 60, max pulse 100 = OK – Min pulse 50, max pulse 120 = MINOR – Min pulse 40, max pulse 160 = MAJOR – Min pulse 1, max pulse 230 = LIFE_THREATENING – Any other values should be considered MISSING • **FR-#2**: The statistics of each value should be kept track of. • **FR-#3**: Values can only be positive integers.
References	• Minimum heart rate: [11] • After the age of 10, 60 to 100 is considered normal in a resting position (not including factors such as stress and exercise etc...) [12, 13]

4.2.5 Data processing - Blood pressure measurements

Attribute	Content
Title	Blood pressure measurements
ID	SR-#4.2.5
Priority	High
Implemented [in version]	yes [1.0]
Cross-Reference	QA-#3.1; SR-#4.2.7
Description	The heartbeat monitor needs to sensor multiple measurements. One of them is blood pressure. When the measurement comes in it needs to be checked if it is a valid measurement and in which category the measurement falls.
Stimulus/Response sequence	• **Incoming measurement**: The two measurement values diastolic and systolic are processed by a processor, which analyses the value and returns the right status message.
Functional requirements	• **FR-#1**: Return the correct status messages for the current **diastolic** blood pressure. Numbers are inclusive: – higher then or equal to 120 = LIFE_THREATENING – lower then 40 = LIFE_THREATENING – higher then or equal to 90 = MAJOR – lower then 50 = MAJOR – higher then or equal to 80 = MINOR – lower then 60 = MINOR – between 60 (inclusive) and 79 (inclusive) = OK – values cannot be parsed, or are missing = MISSING.

Functional requirements	• **FR-#2**: Return the correct status messages for the current **systolic** blood pressure. Numbers are inclusive: – higher then or equal to 180 = LIFE_THREATENING – lower then 40 = LIFE_THREATENING – higher then or equal to 140 = MAJOR – lower then 60 = MAJOR – higher then or equal to 130 = MINOR – lower then 90 = MINOR – between 90 (inclusive) and 129 (inclusive) = OK – value that cannot be parsed, or are missing = MISSING. • **FR-#3**: The statistics of each value should be kept track of. • **FR-#4**: Values can only be positive integers.
References	Blood pressure stages: [8]

4.2.6 Data output - Statistic

Attribute	Content
Title	Data output - statistic
ID	SR-#4.2.6
Priority	Medium
Implemented [in version]	Yes [1.0]
Cross-Reference	SR-#4.2.7; SR-#4.2.3; SR-#4.2.4; SR-#4.2.5
Description	After running the simulation a statistics overview is given of each measurement and how often they occurred. Beyond, the total number of recordings should be shown (up until that point).
Stimulus/Response sequence	• **Just before the program ends**: If all data rows are processed, the statistics should be shown.
Functional requirements	• **FR-#1**: The statistics for pulse, oxygen, blood pressure systolic and blood pressure diastolic should be shown. • **FR-#2**: When a statistics is shown it shows the number of OK, MISSING, MINOR, MAJOR, and LIFE_THREATENING measurement values. • **FR-#3**: It shows the number of processed measurements.

4.2.7 Data output - User interface

Attribute	Content
Title	User interface
ID	SR-#4.2.7
Priority	Medium
Implemented [in version]	yes [1.0]
Cross-Reference	QA-#3.3; QA-#3.1
Description	The user interface provides arguments at program start. After this phase the system doesn't take any other input. While the program is running, the interface will update the user with a constant flow of messages at the interval defined in 4.2.2 FR-#5. The messages will contain information according to table 16. The status of a particular value (Pulse, Oxygen saturation, Systolic, Diastolic) is one of: OK, MISSING, MINOR, MAJOR, LIFE_THREATENING which is generated according to the processing of the system. These values are explained in table 17.
Stimulus/Response sequence	• **Measurement-Result ready to present**: If there are more results to display, the system will output these results as soon as they are ready in the order they are coming in (FIFO). • **No Measurement-Result ready to present**: If there are no more results to display, the system will output statistics after which the system will shutdown.
Functional requirements	• **FR-#1**: All status messages will have a timestamp. • **FR-#2**: Status message should contain the status of oxygen, pulse, blood_pressure_systolic and blood_pressure_diastolic. • **FR-#3**: Status message should contain the value of oxygen, pulse, blood_pressure_systolic and blood_pressure_diastolic. • **FR-#4**: Status message should appear on a constant level. At the end of the program it should show the statistics of the monitoring.

Line	Format	Elaboration
Timestamp	`[dd-mm-yy HH:MM:SS]`	The time in the format d-m-y H:M:S (little-endian) according to Dutch standards with a 24 hour clock according to ISO 8601.
Pulse	`Pulse: 000 bpm`	The pulse of the patient in BPM
Pulse Status	`status: STATUS`	The status of the pulse sensor according to the simulation.
Oxygen saturation	`SaO2: 000%`	The oxygen saturation of the patient in SaO2 percentage.
Oxygen Status	`status: STATUS`	The status of the oxygen sensor according to the simulation.
Blood pressure	`Blood pressure: 0/0 mm Hg`	The blood pressure of the patient in systolic over diastolic mm Hg.
Systolic Status	`status: STATUS`	The status of the systolic sensor according to the simulation.
Diastolic Status	`status: STATUS`	The status of the diastolic sensor according to the simulation.

Table 16: The values required in the output

Status	Elaboration
OK	A reading that is not considered dangerous
MISSING	Missing indicates any value that cannot be considered a valid value from the sensor. This includes, readings that are too high, too low, missing or of incorrect format.
MINOR	A reading that is of interest to medical personnel but cannot be considered a threat yet.
MAJOR	A reading that is of interest to medical personnel and has to be considered a threat.
LIFE_THREATENING	A reading that is of interest to medical personnel because it can have severe effects on the chance of survival of the patient.

Table 17: Notification Levels

4.2.8 Data output - Logging

Attribute	Content
Title	Logging
ID	SR-#4.2.8
Priority	Low
Implemented [in version]	yes [1.0]
Cross-Reference	SR-#4.2.7
Description	The system puts every status and statistic that is to be printed on the screen in a file. The filename will change according to the date. The files will be stored in the module folder logs. If the system cannot log due to lack of space or lack of rights for writing in the location, it will provide a message.
Stimulus/Response sequence	• **New result to log without log folder**: If a new result is ready the logger will try to log this and create a new log folder with a new log file to parse the logs in. • **New result to log with log folder with existing log file**: If a new result is ready the logger will try to log this an create a new log in the existing folder and append to the existing log file that corresponds to the current date. • **New result to log with log folder without existing log file**: If a new result is ready the logger will try to log this an create a new log in the existing folder and create a new log file according to the current date. • **New result to log without writing access**: If a new result is ready the logger will try to log this. If it fails due to insufficient access rights, the system will update the user about this and no logging will be available for the session until the user gets access to write in the folder. • **New result to log with no space left**: If a new result is ready the logger will try to log this. If it fails due to insufficient space, the system will update the user about this and no logging will be available for the session until the user makes space for logs.

Functional requirements	• **FR-#1**: The logger should create a log for every status message and statistic. • **FR-#2**: The logger should create a logfile for the current date if it does not exist • **FR-#3**: The logger should append to the logfile for the current date if it does exist • **FR-#4**: The logger should create a log folder if this folder does not exist • **FR-#5**: The logger should show a single message if it cannot log due to lack of space • **FR-#6**: The logger should show a single message if it cannot log due to lack of access rights

4.3 Performance requirements

No performance requirements are specified since the system adhere rather to a sequential process than a parallel or time critical process.

4.4 Other requirements

4.4.1 Python Version

The system shall be developed with the Python version 3.7.7. Used and recommended Python packages and libraries are specified in appendix 5.1.

4.4.2 Code Style-guide

The code development shall adhere to the "PEP 8 – Style Guide for Python Code"[1] and shall be fulfilled by using the Python code formatter *Black*[2] (version 19.10b0).

4.4.3 Usage

System start

- The system executable should only be opened via a terminal (Windows, Unix, MacOS see section 2.5).
- **Opening the executable via double-click or any other way is *not* supported.**
- Step-by-Step:
    1. Preconditions:
        (a) Make sure you have the correct executable for your platform `HB-Sim2020.exe` (Windows filename) or `HB-Sim2020` (Linux and MacOS filename).
        (b) Make sure the executable and `simulations.csv` are in the same folder.
        (c) Make sure the executable has executable rights on MacOS and Linux.
        (d) For logging the folder the executable resides in needs to have write permissions and enough free space, this is the case for all platforms. If the program does not have write permissions it will still run but give a message about it and not generate any logs files.
    2. Open a command-line interface
    3. Go to the folder holding both files
    4. Run (Windows): `./HB-Sim2020.exe --path simulation.csv`
    5. Run (Unix): `./HB-Sim2020 --path simulation.csv`
    6. Run (MacOS): `./HB-Sim2020 --path simulation.csv`

[1] https://www.python.org/dev/peps/pep-0008/
[2] https://github.com/psf/black

System termination

The system is terminated/ends in two different ways:

- System reaches the end of the CSV. There are no more measurements to process. The program will be end automatically.
- The user can terminate the program at any time by pressing `ctrl + c`

4.4.4 Python Executable

The releases are created with the Python library *PyInstaller*[3] (version 3.6) to create executables for Windows, Unix, and MacOS. Every release should be packed as ZIP and contain the executable itself and one CSV-file with simulation data.

Known Bug

PyInstaller itself contains of a known bug (see [4] and [5]) by handling `KeyboardInterrups`. Due to this issue, it could be that the program terminates with an exception while pressing `ctrl + c`, for instance `Failed to execute script heartbeatmonitor`. Since this is a known issue, this should be taken into account for next releases.

[3] https://pypi.org/project/PyInstaller/
[4] https://github.com/pyinstaller/pyinstaller/issues/2349
[5] https://github.com/pyinstaller/pyinstaller/issues/3646

5 Appendix

5.1 Used Python Libraries

Library	Notes
`Python`	Version 3.7.7
`os`	Os is used for making sure a file or folder does (not) exist, creating the paths to files and folders with the correct separators depending on your system, making directories, to get the line separator for this operating system/platform.
`sys`	Sys is used for getting the command-line arguments, and setting the exit code for the program.
`typing`	Typing is used for type hinting with denoting build-in types as Dict and List.
`datetime`	Used for getting the current date and time and formatting it correctly for human display.
`csv`	Is used for reading from a CSV file.
`time`	Used for making the program "sleep"/wait for a certain time and so simulating that we're waiting on new sensor readings.
`errno`	Used for determining the error when wanting to write a file to disk and something went wrong.
`pathlib`	Used for getting to the parent directory, in a file path string.
`enum`	Used for making enum objects in the code.

5.2 Simulation Data - CSV

```
oxygen,pulse,blood_pressure_systolic,blood_pressure_diastolic
95,80,100,70
96,87,101,74
94,85,101,75
95,83,101,74
97,85,102,72
94,89,103,68
94,88,104,64
93,82,105,62
94,75,106,60
93,73,106,59
92,70,105,60
90,68,108,55
90,67,109,54
85,65,110,52
86,63,118,49
85,61,115,47
86,59,116,46
84,57,120,42
85,53,121,38
```

Listing 1: CSV example

References

[1] World Health Organization, *Pulse Oximetry Training Manual*, World Heal. Organ., pp. 1–23, 2011.

[2] Wikipedia contributors, *Heart rate*, Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Heart_rateoldid=952483498 (accessed April 23, 2020).

[3] Wikipedia contributors, *Oxygen saturation (medicine)*, Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Oxygen_saturation_(medicine)oldid=952492535 (accessed April 23, 2020).

[4] Wikipedia contributors, *Blood pressure*, Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Blood_pressureoldid=951946682 (accessed April 23, 2020).

[5] L. Lehtola, M. Kauppinen, and S. Kujala, *Requirements Prioritization Challenges in Practice*, in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3009, no. April, 2004, pp. 497–508.

[6] A. S. I. Mohamed, B. I. A. El-Maddah, and C. A. M. Wahba, *Criteria-based requirements prioritization for software product management*, Proc. 2008 Int. Conf. Softw. Eng. Res. Pract. SERP 2008, no. January, pp. 587–593, 2008.

[7] J. Karlsson and K. Ryan, *A cost-value approach for prioritizing requirements*, IEEE Softw., vol. 14, no. 5, pp. 67–74, 1997.

[8] T. Kirkpatrick and K. Tobias, *Health System PEDIATRIC AGE SPECIFIC Self Learning Module*, pp. 1–11, 2009.

[9]]Y. Shafranovich, *RFC 4180 - Common Format and MIME Type for CSV Files*, iet.org, 2005. [Online]. Available: https://www.ietf.org/rfc/rfc4180.txt. [Accessed: 25-Apr-2020].

[10] Abdo, Wilson F, and Leo M A Heunks. *Oxygen-induced hypercapnia in COPD: myths and facts.* Critical care (London, England) vol. 16,5 323. 29 Oct. 2012, doi:10.1186/cc11475

[11] https://thenextchallenge.org/resting-heart-rate/

[12] https://www.medicalnewstoday.com/articles/235710target-training-heart-rates

[13] https://www.heart.org/en/healthy-living/fitness/fitness-basics/target-heart-rates