CSSR for

Culturally Sensitive Social Robotics for Africa

D3.4 System Integration and Quality Assurance Manual

Due date: **31/03/2024**
Submission Date: **01/11/2023**
Revision Date: **03/12/2024**

Start date of project: **01/07/2023** Duration: **36 months**

Lead organisation for this deliverable: **Carnegie Mellon University Africa**

Responsible Person: **D. Vernon** Revision: **1.4**

Project funded by the African Engineering and Technology Network (Afretec) Inclusive Digital Transformation Research Grant Programme		
Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants (including Afretec Administration)	
RE	Restricted to a group specified by the consortium (including Afretec Administration)	
CO	Confidential, only for members of the consortium (including Afretec Administration)	

Executive Summary

Deliverable D3.4 sets out the procedures used in CSSR4Africa to validate and test software developed by the partners prior to integration into the CSSR4Africa software repository. It is, in essence, is a check-list of the *mandatory* standards set out in Deliverable D3.2 and software must satisfy all these checks before it can be integrated.

Developers can submit their software by sending it by email in a zip file to aakinade@andrew.cmu.edu and vernon@cmu.edu, with CSSR4Africa, the ROS package name, and the ROS node name of the component being submitted in the subject line,
e.g., `CSSR4Africa pepper_interface_tests sensorTest`.

The system integration team at CMU-Africa will validate the software against the check-list in this deliverable, compiling the code, and running the unit test application supplied with the submission. If the component satisfies all the checks and the test runs successfully, the ROS node will then be uploaded to the CSSR4Africa GitHub software repository.

The complete CSSR4Africa software system will also be subject to quality assurance procedures, including regression tests and system tests.

Contents

1 Files and Directories

Refer to Deliverable D3.2, Appendix A (Mandatory Standards for File Organization), for a definition of the standards on which this checklist is based.

1.1 C/C++Programming Language

- ☐ Files for a single component are stored in a directory or subdirectory named after the ROS package or node (in the case of the `cssr_system` and `unit_tests` package) in which it is to be integrated. Refer to Deliverable D3.1 System Architecture for details of the ROS package names and the associated ROS nodes.
- ☐ This directory has between four and seven sub-directories, depending on the software functionality: `config`, `data`, `include/<package name>`, `launch`, `msg`, `src`, and `srv`; with `config`, `data`, `include/<package name>`, and `src` necessarily present.
- ☐ The `config` directory contains at least one file, named as follows.
  - ☐ `<componentName>Configuration.ini`
  - ☐ The configuration file `<componentName>Configuration.ini` contains the key-value pairs that set the component parameters.
  - ☐ Each key-value pair is written on a separate line.
- ☐ The `data` directory contains at least two of three files (with the topics files mandatorily present), named as follows.
  - ☐ `<componentName>Input.dat`
  - ☐ `pepperTopics.dat`
  - ☐ `simulatortopics.dat`
  - ☐ The topics files `pepperTopics.dat` and `simulatorTopics.dat` contains the key-value pairs that set the topic names required by the component.
  - ☐ Each key-value pair is written on a separate line.
- ☐ The `include/<package name>` directory contains at least one file, named as follows.
  - ☐ `<componentName>Interface.h`
- ☐ The `launch` directory contains three files, named as follows.
  - ☐ `<componentName>LaunchRobot.launch`
  - ☐ `<componentName>LaunchSimulator.launch`
  - ☐ `<componentName>LaunchTestHarness.launch`
- ☐ The `src` directory contains at least two source files, named as follows.
  - ☐ `<componentName>Application.cpp`
  - ☐ `<componentName>Implementation.cpp`

- ☐ The package directory or node subdirectory contains a `README.md` file with instructions on how to run the test
- ☐ The package directory contains a `CMakeLists.txt` build file.
- ☐ The package directory contains a `package.xml` manifest file (if it is a node within a package, e.g., `cssr_system` or `unit_tests` package, there is no `package.xml` manifest file).

1.2 Python Programming Language

- ☐ Files for a single component are stored in a directory or subdirectory named after the ROS package or node (in the case of the `cssr_system` and `unit_tests` package) in which it is to be integrated. Refer to Deliverable D3.1 System Architecture for details of the ROS package names and the associated ROS nodes.
- ☐ This directory has between four and seven sub-directories depending on the software functionality: `config`, `data`, `launch`, `msg`, `model`, `src`, and `srv`; with `config`, `data`, and `src` mandatorily present.
- ☐ The `config` directory contains at least one of two files, named as follows.
  - ☐ `<component_name>_configuration.ini`
  - ☐ `<component_name>_configuration.json`
  - ☐ The configuration file `<component_name>_configuration.ini` or `<component_name>_configuration.json` contains the key-value pairs that set the component parameters.
  - ☐ Each key-value pair is written on a separate line.
- ☐ The `data` directory contains at least two of three files (with the topics files mandatorily present), named as follows.
  - ☐ `<component_name>_input.dat`
  - ☐ `pepper_topics.dat`
  - ☐ `simulator_topics.dat`
  - ☐ The topics files `pepper_topics.dat` and `simulator_topics.dat` contains the key-value pairs that set the topic names required by the component.
  - ☐ Each key-value pair is written on a separate line.
- ☐ The `launch` directory contains three files, named as follows.
  - ☐ `<component_name>_launch_robot.launch`
  - ☐ `<component_name>_launch_simulator.launch`
  - ☐ `<component_name>_launch_test_harness.launch`
- ☐ The `src` directory contains at least two source files, named as follows.
  - ☐ `<component_name>_application.py`
  - ☐ `<component_name>_implementation.py`

- ☐ The package directory or node subdirectory contains a `README.md` file with instructions on how to run the test.
- ☐ The package directory contains a `CMakeLists.txt` file specifying message generation and service building, if applicable.
- ☐ The package directory contains a `package.xml` manifest file (if it is a node within a package, e.g. `\cssr_system` or `unit_tests` package, there is no `package.xml` manifest file).
- ☐ The package directory or node subdirectory contains a `<component_name>_requirements.txt` file with the dependencies (and their versions) required to be installed for proper functionality of the node.

2 Internal Source Code Documentation

Refer to Deliverable D3.2, Appendix B (Mandatory Standards for Internal Source Code Documentation), for a definition of the standards on which this checklist is based.

2.1 C/C++Programming Language

All source files contain a documentation comment that gives the copyright notice, as follows.

```
/* <filename> <one line to identify the nature of the file>
 *
 * Author:
 * Date:
 * Version:
 *
 * Copyright (C) 2023 CSSR4Africa Consortium
 *
 * This project is funded by the African Engineering and Technology Network (Afretec)
 * Inclusive Digital Transformation Research Grant Programme.
 *
 * Website: www.cssr4africa.org
 *
 * This program comes with ABSOLUTELY NO WARRANTY.
 */
```

☐ <componentName>Application.cpp

☐ <componentName>Implementation.cpp

☐ <componentName>Interface.h

The <componentName>Application.cpp file contains a documentation comment with the following sections:

- [ ] /* <componentName>Application.cpp <one line to identify the nature of the file>
  *
  * <detailed functional description>
  *
- [ ] * Libraries
- [ ] * Parameters
  *
  * Command-line Parameters
- [ ] * Configuration File Parameters
- [ ] * Subscribed Topics and Message Types
- [ ] * Published Topics and Message Types
- [ ] * Services Invoked
- [ ] * Services Advertised and Request Message
- [ ] * Input Data Files
  *
  * <componentName>Input.dat
- [ ] * Output Data Files
  *
  * <componentName>Output.dat
- [ ] * Configuration Files
  *
  * <componentName>Configuration.ini
- [ ] * Example Instantiation of the Module
  *
  * rosrun <componentName> __name:=<alternativeComponentName> ...
- [ ] * Author: <name of author>, <author institute>
- [ ] * Email: <preferred email address>
- [ ] * Date:
- [ ] * Version:

2.2 Python Programming Language

All source files contain a documentation comment that gives the copyright notice, as follows.

```
"""
    <filename> <one line to identify the nature of the file>

    Author:
    Date:
    Version:

    Copyright (C) 2023 CSSR4Africa Consortium

    This project is funded by the African Engineering and Technology Network (Afretec)
    Inclusive Digital Transformation Research Grant Programme.

    Website: www.cssr4africa.org

    This program comes with ABSOLUTELY NO WARRANTY.

"""
```

- ☐ <component_name>_application.py
- ☐ <component_name>_implementation.py

The <component_name>_application.py file contains a documentation comment with the following sections:

"""

- [ ] <component_name>_application.py <one line to identify the nature of the file>

  <detailed functional description>

- [ ] Libraries
- [ ] Parameters
- [ ] Command-line Parameters
- [ ] Configuration File Parameters
- [ ] Subscribed Topics and Message Types
- [ ] Published Topics and Message Types
- [ ] * Services Invoked
- [ ] * Services Advertised and Request Message
- [ ] Input Data Files

  <component_name>_input.dat

- [ ] Output Data Files

  <component_name>_output.dat

- [ ] Configuration Files

  <component_name>_configuration.ini

- [ ] Example Instantiation of the Module

  rosrun <component_name> __name:=<alternative_component_name> ...

- [ ] Author: <name of author>, <author institute>
- [ ] Email: <preferred email address>
- [ ] Date:
- [ ] Version:

3 Component Unit Testing

3.1 C/C++Programming Language

- ☐ A unit test application named `<componentName>LaunchRobot.launch` is provided in the `launch` directory.
- ☐ A unit test application named `<componentName>LaunchSimulator.launch` is provided in the `launch` directory.
- ☐ A unit test application named `<componentName>LaunchTestHarness.launch` is provided in the `launch` directory.
- ☐ The `<componentName>LaunchRobot.launch` file launches the component being tested.
- ☐ The `<componentName>LaunchSimulator.launch` file launches the component being tested.
- ☐ The `<componentName>LaunchTestHarness.launch` file launches the component being tested.
- ☐ The component being tested outputs the copyright message on startup:

```
<Program name and version>

This project is funded by the African Engineering and Technology Network (Afretec)
Inclusive Digital Transformation Research Grant Programme.

Website: www.cssr4africa.org

This program comes with ABSOLUTELY NO WARRANTY.
```

- ☐ The component being tested write short messages to the terminal during the start-up phase to indicate the state of the node:

```
nodeName: start-up.
nodeName: subscribed to /topicName.
```

- ☐ The component being tested periodically (every ten seconds) writes a short heartbeat message to the terminal to indicate the state of the node:

```
nodeName: running.
```

- ☐ The `<componentName>LaunchRobot.launch` file connects the component a data source and a data sink on the physical robot, and produces the expected result as set out in the `README.md` file. This means this file launches the physical robot and all its sensors and actuators as required.
- ☐ The `<componentName>LaunchSimulator.launch` file connects the component a data source and a data sink on the simulator. This means this file launches the simulator robot and all its sensors and actuators as required.
- ☐ The `<componentName>LaunchTestHarness.launch` file connects the component a data source and a data sink using a driver and a stub. This means this file launches therequired drivers and stubs, and the node being tested.
- ☐ Unit test instructions are provided in a file named `README.md` in the package directory.

- ☐ The instructions explain how the communication and computation functionality are validated by describing the (sink) output data that will be produced from the (source) input data.
- ☐ The instructions explain how the configuration functionality is validated by describing what changes in behaviour will occur if the values for the component parameters in the component configuration (`.ini`) file are altered.

3.2 Python Programming Language

- ☐ A unit test application named `<component_name>_launchRobot.launch` is provided in the `launch` directory.
- ☐ A unit test application named `<component_name>_launchSimulator.launch` is provided in the `launch` directory.
- ☐ A unit test application named `<component_name>_launch_test_harness.launch` is provided in the `launch` directory.
- ☐ The `<component_name>_launch_robot.launch` file launches the component being tested.
- ☐ The `<component_name>_launch_simulator.launch` file launches the component being tested.
- ☐ The `<component_name>_launch_test_harness.launch` file launches the component being tested.
- ☐ The component being tested outputs the copyright message on startup:

```
<Program name and version>

This project is funded by the African Engineering and Technology Network (Afretec)
Inclusive Digital Transformation Research Grant Programme.

Website: www.cssr4africa.org

This program comes with ABSOLUTELY NO WARRANTY.
```

- ☐ The component being tested write short messages to the terminal during the start-up phase to indicate the state of the node:

```
nodeName: start-up.
nodeName: subscribed to /topicName.
```

- ☐ The component being tested periodically (every ten seconds) writes a short heartbeat message to the terminal to indicate the state of the node:

```
nodeName: running.
```

- ☐ The `<component_name>_launch_robot.launch` file connects the component a data source and a data sink on the physical robot, and produces the expected result as set out in the `README.md` file. This means this file launches the physical robot and all its sensors and actuators as required.
- ☐ The `<component_name>_launch_simulator.launch` file connects the component a data source and a data sink on the simulator. This means this file launches the simulator robot and all its sensors and actuators as required.
- ☐ The `<component_name>_launch_test_harness.launch` file connects the component a data source and a data sink using a driver and a stub. This means this file launches therequired drivers and stubs, and the node being tested.
- ☐ Unit test instructions are provided in a file named `README.md` in the package directory.

- ☐ The instructions explain how the communication and computation functionality are validated by describing the (sink) output data that will be produced from the (source) input data.
- ☐ The instructions explain how the configuration functionality is validated by describing what changes in behaviour will occur if the values for the component parameters in the component configuration (`.ini`) file are altered.

4 System Testing

Regression testing refers to the practice of re-running all integration tests periodically to ensure that no unintentional changes has been introduced during the ongoing development of the CSSR4Africa software release. These test check for backward compatibility, ensuring that what used to work in the past remains working. Regression tests will be carried out on all software in the CSSR4Africa release every two months.

Principal Contributors

The main authors of this deliverable are as follows (in alphabetical order).

Adedayo Akinade, Carnegie Mellon University Africa.
David Vernon, Carnegie Mellon University Africa.

Document History

Version 1.0

First draft.
David Vernon.
1 November 2023.

Version 1.1

Updated the standards checklist to accomodate software written in Python programming language.
Created two subsections for C/C++ and Python programming languages in sections 1 (Files and Directories and 2 (Internal Source Code Documentation).
Documented the checklist for Python programming language.
Relaxed the requirements for files and directories in packages because of packages which may not have all in their design.
Updated the submission email to include Adedayo Akinade, who is the current system integrator.
Adedayo Akinade.
29 August, 2024.

Version 1.2

Updated the software standards to accomodate standard practice for filenames when using the Python programming language, i.e., using an underscore _ to separate words, rather than using camel case, as is the recommended practice with C++.
The same practice is also applied to related files, e.g., input, output, and configuration files.
Added Adedayo Akinade to the list of contributors.
David Vernon.
9 September 2024.

Version 1.3

Corrected several errors and omissions in the changes made in the previous version.
David Vernon.
13 September 2024.

Version 1.4

Corrected some errors in the changes made in the previous version.
Added provision for the data directory.
Added provision for `.json` configuration files in the Python programming language standards.
Changed the file name for the requirements file for software written in Python programming language.
Added checklist for startup, copyright and heartbeat messages in the node operation.
Added clarification for the functions of the launch files.
Adedayo Akinade.
3 December 2024.