**UNIVERSIDAD TECNOLÓGICA DE QUERÉTARO**

**CESEQ**



**Diplomado en Software Embebido**

Proyecto Integrador

**File Name: Software Development Plan.docx**

DOCUMENT: Software Development

Document No. #CESEQ\_SDP\_001

Scrum Master: Ramírez, Jesus

Developer. Sánchez, Luis

Developer Algemiro Gil

Date (YYYYMMDD): 20190405

Version: 1.0.0.

Project Version: x.x.x.

# Log

Document Version

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date (yyyymmdd) | Description | Reviewer |
| 1.0.0. | 20190405 | First release | Pérez, Adbeel |
|  |  |  |  |

Project Document Version

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date (yyyymmdd) | Description | Reviewer |
| 1.0.0. | 20190823 | Initial Release | A: Luis Sánchez  R: Jesús Ramírez |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date (yyyymmdd) | Description | Reviewer |
| 1.2.0. | 20191002 | Quality Review A | A: Luis Sánchez  R: Algemiro Gil |
|  |  |  |  |

# Index

Contenido

[1. Log 2](#_Toc23359186)

[2. Index 3](#_Toc23359187)

[2.1 Table of Figures 4](#_Toc23359188)

[3. Project 5](#_Toc23359189)

[3.1 Project Scope 5](#_Toc23359190)

[3.2 Software 5](#_Toc23359191)

[3.3 Documentation 5](#_Toc23359192)

[3.3.1 Project folder structure: 5](#_Toc23359193)

[3.3.2 Stakeholder requirements 6](#_Toc23359194)

[3.4 System requirements 6](#_Toc23359195)

[4. Deliverables 7](#_Toc23359196)

[4.1 Software delivering process 7](#_Toc23359197)

[5. Development methodology 8](#_Toc23359198)

[5.1 Process 8](#_Toc23359199)

[6. Estimates 9](#_Toc23359200)

[6.1 Hardware facts 9](#_Toc23359201)

[6.2 Activities Facts 9](#_Toc23359202)

[6.3 SW Facts 9](#_Toc23359203)

[6.4 Assumptions 10](#_Toc23359204)

[6.5 Activities 11](#_Toc23359205)

[6.5.1 Documentation 11](#_Toc23359206)

[6.5.2 Project development 11](#_Toc23359207)

[7. Planning 12](#_Toc23359208)

[8. Solving Problem Strategy 12](#_Toc23359209)

[9. Design 12](#_Toc23359210)

[9.1 System overview. 12](#_Toc23359211)

[9.1.1 Inputs 13](#_Toc23359212)

[9.1.2 Control 13](#_Toc23359213)

[9.1.3 Outputs 14](#_Toc23359214)

[9.2 Main Thread 14](#_Toc23359215)

[9.3 ADC Thread 14](#_Toc23359216)

[9.4 Controller Unit 15](#_Toc23359217)

[9.4.1 Pseudocode 16](#_Toc23359218)

[9.4.2 Algorithm 17](#_Toc23359219)

[9.5 Display Unit 17](#_Toc23359220)

[9.6 Encoder Unit 18](#_Toc23359221)

[9.7 PWM Unit. 19](#_Toc23359222)

[9.8 Standards 20](#_Toc23359223)

[9.8.1 Standards Result analysis 20](#_Toc23359224)

[9.9 Naming conventions 21](#_Toc23359225)

[9.9.1 Variables 21](#_Toc23359226)

[9.9.2 Constants 22](#_Toc23359227)

[9.9.3 Functions 22](#_Toc23359228)

[9.9.4 Threads 22](#_Toc23359229)

[9.9.5 Files 22](#_Toc23359230)

[9.9.6 Folders 22](#_Toc23359231)

[9.9.7 Types 22](#_Toc23359232)

[9.9.8 Comments 22](#_Toc23359233)

[10. Testing 23](#_Toc23359234)

[10.1. Verification strategy (black box test) 23](#_Toc23359235)

[10.2. White box strategy 23](#_Toc23359236)

[10.3. Cyclomatic Complexity Redundancy index 23](#_Toc23359237)

[11. Release 24](#_Toc23359238)

[11.1. Software Development Folder 24](#_Toc23359239)

[11.2. Integration Tests Strategy 24](#_Toc23359240)

[11.3. Validation Testing / Functional Testing 24](#_Toc23359241)

[11.4. Throughput and Flash and RAM measurement 25](#_Toc23359242)

[11.4.1 Memory Analysis 25](#_Toc23359243)

[12. Results 25](#_Toc23359244)

[13. Lessons Learned 25](#_Toc23359245)

## 2.1 Table of Figures

Figure 1. Project folder structure. 5

Figure 2 Process Diagram 9

Figure 3. System Diagram 13

Figure 4. Main Thread 14

Figure 5. ADC Thread 15

Figure 6. Controller Unit 16

Figure 7. Control Algorithm 17

Figure 8 Display Unit 18

Figure 9. Encoder Unit 18

Figure 10. PWM Unit 19

Figure 11 Statistics for standard analysis 20

Figure 12 Standard Results 21

# 3. Project

## 3.1 Project Scope

The Project: “*Proyecto Integrador”*, is an embedded solution which has software, hardware and, documents artifacts.

## 3.2 Software

The software portion are the code files which generates the application. The application is built using synergy with e2 studio GUI. The source code is written in C/C++ language and consists in .C and .H files.

The project also includes files and functions to configure ports, interruptions, and registers. Besides, the synergy project includes files for configuration, scripts, debug information, and other auxiliary files. For this reason, it is included a repository folder as part of the project folder structure.

## 3.3 Documentation

The documentation is located in the project folder structure (Figure 1). It has the material and references to support the solution. It has been uploaded to the following Git Hub repository:

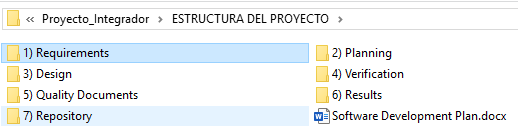
<https://github.com/Luis1250/Proyecto_Integrador>

Figure 1. Project folder structure.

### 3.3.1 Project folder structure:

The repository has the following folder structure

* Requirements: At this point there is not any deviation. So, the customer requirements described in the document “*20190527 Requisitos del proyecto integrador CESEQ.pdf”* located in the repository under de folder: *1) Requirements\stakeholder* **is fully accepted**.
* Planning: All the planning documentation is located in the repository under the folder: ***2) Planning:***
  + The file *Planning.docx* includes a Gantt with the tasks and they time window for completion.
* Design: The design artifacts are located in the repository under the folder ***3) Design****.*
  + The design includes UML diagrams to define the classes and modules of the project.
* Verification: The verification strategy is located in the repository under del folder ***4) Verification.***
  + The verification consists in a document with the verification strategy to demonstrate that the software has quality.
* Quality Documents: The quality documents are located in the repository under the folder:***5) Quality Documents.***
  + The quality documents include the standards to develop the software.
* Results: The results of the project are located in the repository under the folder: ***6) Results.***
* Repository: Contains the software solution for the project.
  + The results include material to demonstrate that the project followed the plan, design, and any other artifact mentioned in the documentation.
  + Since some compilation scripts have problems to handle folder names starting with numbers, the folder is called **code** instead of 7) code.

### 3.3.2 Stakeholder requirements

The stakeholder requirements are the inputs of this project, they contain the critical aspects that the client is expecting to be included in the final solution.

The document is located in the following path of the repository:

ESTRUCTURA DEL PROYECTO\1) Requirements\stakeholder

## 3.4 System requirements

The system team has decided that the solution is not big enough to have system requirements, high level requirements (subsystem) and low-level requirements. The decision is to have just software requirements. These software requirements have the enough level detail to be implemented by the development team. Furthermore, the software requirements are considering the testing to cover completeness and correctness of the project.

The software requirements are located in the Software Requirements Specifications documents located in the following path of the repository:

ESTRUCTURA DEL PROYECTO\1) Requirements\3. SWRA\_20190621.docx

# Deliverables

Since this Project is academic and the team wants to probe that the process has been followed, the following folders in the repository are considered inputs:

* + ESTRUCTURA DEL PROYECTO\1) Requirements
  + ESTRUCTURA DEL PROYECTO\2) Planning
  + ESTRUCTURA DEL PROYECTO\3) Design

The outputs are:

* + ESTRUCTURA DEL PROYECTO\4) Verification
  + ESTRUCTURA DEL PROYECTO\5) Quality Documents
  + ESTRUCTURA DEL PROYECTO\6) Results

Basically, the inputs define the final product. The requirements folder contains the customer requirements and the team’s requirements. These inputs allow to design a solution but this design is just another input for the development team. That is why the design is considered an input.

The verification folder has the evidence that the software implements the specification described in the requirements and design. The document containing the evidence and the strategy to get that evidence is called Software Verification Plan (*SoftwareVerificationPlan.docx*)

Quality Documents folder is considered other output since it has the evidence that the project followed all the expectations described in the *Lista de Cotejo de Auditoria de calidad V1.1.docx*

The results folder collects all the outputs which are not related with the other output folders. It includes but not limited to: evidence of meetings, design drafts, preliminary results, estimations, etc.

## 4.1 Software delivering process

The delivering process consist in three phases:

1. Developmental phase: This phase is for the development team and usually longer in time because during this phase the development team can deliver at any time code updates without restrictions. The development team is delivering in a branch to correspond with the functionality to complete.
2. Integration phase: This phase occurs after a developmental phase. The duration of this phase takes at much one day. The objective is to integrate the functionality worked during the developmental phase. So, the branch is merge with the master branch and a new branch is created to continue working with the next functionality.
3. The Final Release: This is a branch and the intention of this branch is to be last one worked. So, this is also the last integration with the master branch and it is reserved to the completion of the project. During this phase, the last development phase is completed (release 4), the final release branch is created and any pending task will be work here. During the final release, the quality team can request modifications to any artifacts in order to be corrected. The testing team will also check the correctness and completeness of the verification process.

Include in this section the delivering process and dates if it applies

All the artifacts worked and presented as results will be taken from the repository. The last state of the results will be the master branch after the Final Release branch has been merged with the master. This stage impacts to all the documentation and the e2 studio working area.

# Development methodology

The team has decided to use an agile methodology to accelerate the development process.

Scrum is the agile methodology selected and has the following controls:

* Scrum board:
  + Product Owner: Luis Sánchez
  + Scrum Master: Jesús Ramírez
  + Development team:
    - Luis Sánchez
    - Jesús Ramírez
    - Algemiro Gil
* Sprint length
  + Each sprint is scheduled in one week in order to be synchronized with the releases.
  + The time per sprint and por person is divided by:
    - 4 hrs on Fridays and can include laboratory time
    - 4 hrs on any other day with or without laboratory time.
* Task management
  + Small tasks are those task that can be completed in 1-2 hrs.
  + Medium tasks are those task that can be completed in 2-4 hrs.
  + Large tasks are those tasks that can be completed 4-8 hrs.
  + Extra Large are those task that can be completed in more than 8 hrs.
* Planning meetings
  + Are scheduled at the beginning of each release in order to plan the tasks for the next release.
  + The retrospective meeting will be done after the closure of the release and before the next planning in order to apply any improvement proposed during the retrospective meeting.
  + All the team members SHALL be required to attend all the meetings.

## 5.1 Process

The methodology described in the previous section help to the development tea to execute the project. However, the process is depicted in the Figure 2 and contains the main activities to complete the project.



Figure 2 Process Diagram

# Estimates

This section describes the estimations for the project. The estimations include equipment availability, laboratory time, human resources, and any other resource that is needed to develop the project.

## 6.1 Hardware facts

The hardware facts are those elements that are devices with direct or indirect impact to the project. Such as:

* CESEQ laboratory availability Is the place where the equipment is store.
* Board availability Is where the microcontroller lives and has interfaces to use it.
* Plant availability subsystem to provide signals to the inputs’ system
* PC availability Is the device where the application is developed.
* Oscilloscope Is a device to display and analyze signals.
* Signal generator Device to provide a specific signal to the system.
* Multimeter Device to measure volt-ohm-milliammeter
* Power supply Supplies the power to the Board.

## 6.2 Activities Facts

The activities facts include the roles needed to develop the project. Since the Scrum methodology is used in this project, the roles and responsibilities as well as the names of the persons are listed in the section [Development methodology](#_Development_methodology) of this document.

## 6.3 SW Facts

The software fact are also elements that are software and applications with direct or indirect impact to the project.

Such as:

* Software development kit
  + e2 studio with synergy customization for YSSKS7G2E30 RENESAS board
  + PC with minimum system requirements to install and use e2 studio.
* Internet access
* Git Hub count
* Git Hub desktop client
* Office suite for documentation

## 6.4 Assumptions

In this section are considered some aspects that can influence in the project. At this point are just assumptions but if the team feels like something is not in control a contingency plan should be taken.

* Hw Assumptions:
  + Hardware damaged. The hardware damage and malfunction is always a risk since the microcontroller is fragile to high voltage, wrong configuration, as a mechanism to reduce the risk, a double check before critical test will be implemented.
* Laboratory time availability.
  + Laboratory equipment. The laboratory has most of the devices needed to develop and test the project. It depends on UTEQ schedule. So, to mitigate unavailability, it will request

the calendar to consider the availability days.

* SW Assumptions
  + Programming language, SW IDE or Hw platform unknow. As any other embedded project, some functions, configurations, interrupts, etc. are unknow. To mitigate this risk, the development team has classes material and professor contacts to resolve any doubt or unclear topic.
* Activtities Assumptions
  + - Team time availability. The team has agreed to work 4 hours per Friday and 4 hours more during the week in order to have 8 hours per week (8 hrs. per sprint)
    - Hardware in good conditions. It is expected that the laboratory has equipment in good shape.
* Risks
  + Facilities
    - Holidays.
      * UTEQ does not open its facilities.
      * The laboratory is busy.
    - Externals
      * Loss of power.
      * Weather
    - Equipment
      * Malfunction.
  + Team
    - Illness
    - Vacations
    - Work
    - Teammate leaves the project.
  + Others
    - Adviser availability
  + Development environment.
    - PC malfunction
    - PC not configured
    - Wrong configuration

## 6.5 Activities

The activities start according with the activities described in the following subsections:

Note: The activity is done when the dependencies (if any) are not blocking the next activity.

### 6.5.1 Documentation

Create Software Development Plan. Starts on 02/08/2019

Software Requirements Analysis Starts on 03/08/2019

Planning

Project planning Starts on 09/08/2019

FMEA elaboration Starts on 10/08/2019

Gantt-chart elaboration Starts on 10/08/2019

#### Design

Software Standards analyses on 10/08/2019

Software Design Document Starts on 16/08/2019

#### Verification

Software Verification Plan Starts on 26/10/2019

#### Quality Documents

Project Evaluation Format

Starts on 04/10/2019

Formal review on 05/10/2019

Results Starts on 19/10/2019

### 6.5.2 Project development

The project development includes the tasks for project design, architecture design, design review, software validation, software verification. In the following sections are scheduled the dates to complete each development phase.

Module Design Starts on 16/08/2019

* PWM module Starts on 17/08/2019
* UART module Starts on 17/08/2019
* Connection Starts on 17/08/2019
* Control module Starts on 17/08/2019

Architecture Design Starts on 16/08/2019

Design Standards Review Starts on 04/10/2019

Software Validation Starts on 04/10/2019

Software Verification Starts on 05/10/2019

#### Results

Formal Results Starts on 19/10/2019

# Planning

The planning of this project is described in a separate document. The planning document is located at:

<PROJECT\_PATH>\2) Planning\7. Planning\_20190405.xlsx

# Solving Problem Strategy

This section is contained in a different document, the document is contained at:

<PROJECT\_PATH>\2) Planning\8. DFMEA\_20190405.xlsx

# Design

This section explains the design rational of the software implemented for the project. The project is a collection of algorithms that contribute with the solution.

The programming style selected due to its complexity, requirements, and feasibility is the procedural style. There is a main procedure and others to handle inputs, outputs, calculations, etc.

Furthermore, the nature of the solution implies the use of threads and modules to structure the solution.

## 9.1 System overview.

The system has Inputs, control, and Outputs modules to get, process and display information. The system diagram is depicted in the Figure 3.

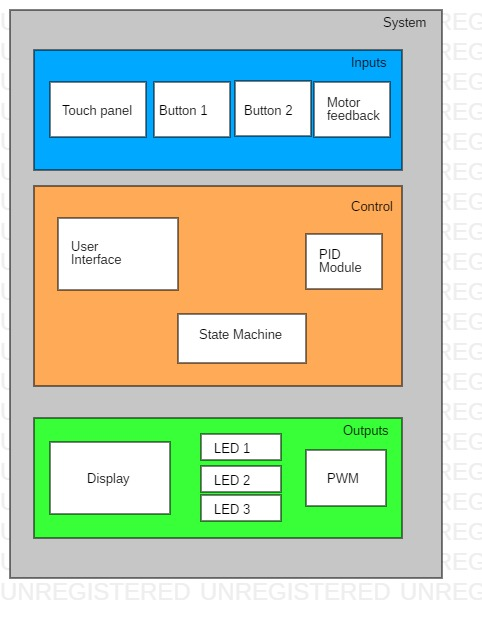


Figure 3. System Diagram

### 9.1.1 Inputs

All the inputs are considered. The user can interact using the touch display and using the buttons. It is expected also inputs from the motor, to know its current speed.

### 9.1.2 Control

The control module has elements to handle the user interface (touch display) the motor (PID module) and the flow of the program with the use of a state machine, to see Document [Controller](1)%20Requirements/stakeholder/Controller.docx).

### 9.1.3 Outputs

The outputs considered are: The display units with messages to the user, LEDs to indicate basic status of the system, and signals to control the motor.

## 9.2 Main Thread

The entry point of the solution is the main thread which calls the other threads. The Figure 4 depict the process flow. An important note is that event there is not a touch event received, the process thread is still receiving data from the ACD module and with that information is calculated a new speed demand for the motor, as well as the event response is another thread that is continually sending information to the display (by the display thread).

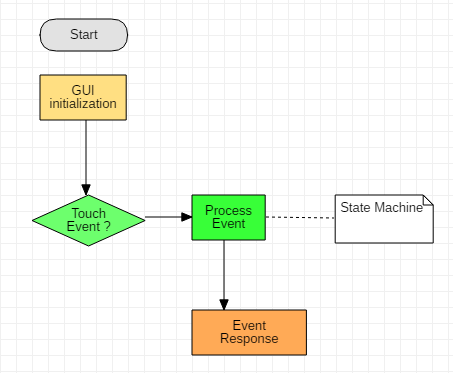


Figure 4. Main Thread

## 9.3 ADC Thread

This Thread is reading a message from the port continually. The objective is to calculate the current speed of the motor in RPMs. The following figure (Figure 5) is a reference of this design.

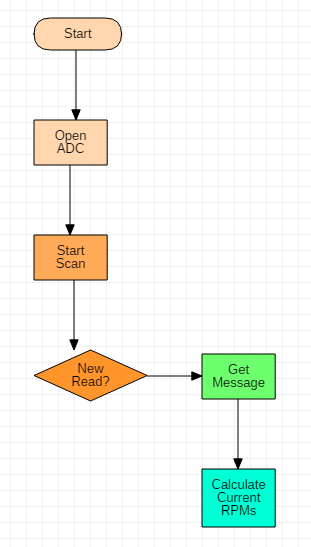


Figure 5. ADC Thread

## 9.4 Controller Unit

The controller unit is in charge of the calculations between the current seed and the target speed. The difference between current speed and target speed is the error. With the error, the controller unit will calculate a demand to reduce the error. This design can be observed in Figure 6.

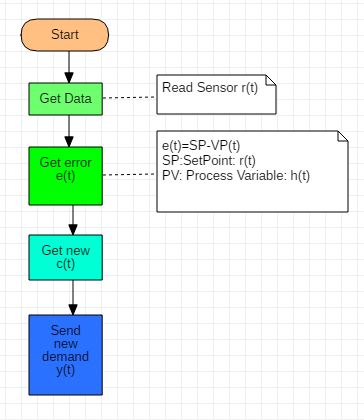


Figure 6. Controller Unit

### 9.4.1 Pseudocode

Since the control unit is a critical element of the system, the following pseudocode was generated to describe the algorithm proposes to this project.

1.- The input reference voltage shall be defined as r(t) and within the values of 0 to 3.5 v.

2.- control shall take reference input as well as receive the final voltage y(t) for error calculation

3.- Comparison shall be made to define error function e(t).

4.- The error function shall be input to the PID control to reevaluate.

4.1.- e(t) = r(t)- h(t) is the error (the setpoint\_ r(t), and h(*t*) is the process variable).

5.- A new corrected function c(t) shall be used to generate PWM signal

6.- PWM function u(t) shall be used as input to control speed

7.- Speed – voltage signal y(t) shall be fed back to the origin of the PID loop to reevaluate error.

### 9.4.2 Algorithm

Since the control unit is a key element of the product, a low-level diagram of the algorithm is provided instead of a textual description. The Figure 7 depicts the algorithm for the control unit and its calculations.

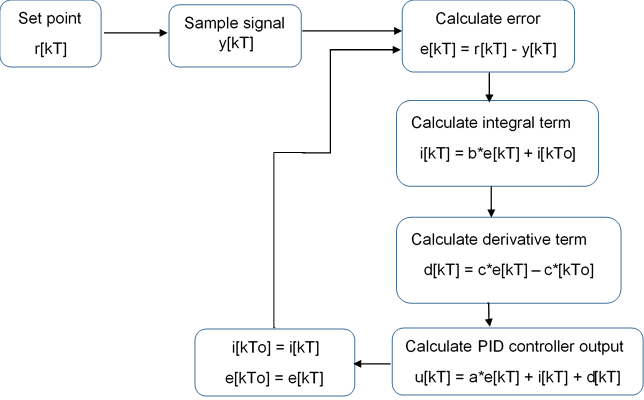


Figure 7. Control Algorithm

## 9.5 Display Unit

The display unit is the portion in the software that will send messages to the LCD. These messages will be continuously sent in order to have fresh data for the user. The information to be displayed includes: The Duty Cycle, the speed in RPMs, the current Set Point. This design is depicted in Figure 8.

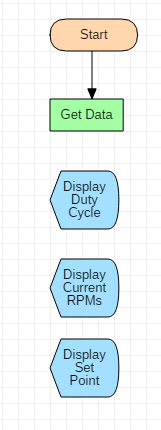


Figure 8 Display Unit

## 9.6 Encoder Unit

The encoder unit is need to know the current speed of the motor. This unit will take samples in a period of time. With those inputs, this module will calculate the current speed of the motor and will convert that speed in RPMs. See this design in Figure 9.

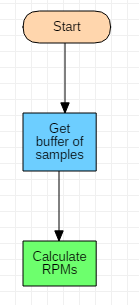


Figure 9. Encoder Unit

## 9.7 PWM Unit.

The PWM unit will control the motor with a new demand signal. The PWM unit needs to be called always because the motor needs to keep the controlled speed. So, the PWM unit uses the controller outputs to have a value to be sent to the motor. The Figure 10 describes the logic of how PWM works.

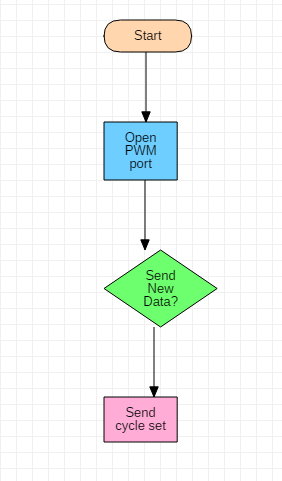


Figure 10. PWM Unit

This section **SHALL** contain Static and dynamic modeling diagrams like: block diagram flow diagram, call tree diagram, state machine diagram, sequence diagram and others depending on the programming paradigm.

This section MUST be contained in this document or in a different document linked to this section, the new document SHALL be contained at:

<PROJECT\_PATH>\3) Design\9. SoftwareDesignDocument\_20190405.docx

Sections 9.1. and 9.2. MUST be contained in this document or MUST be divided into different documents. With the naming defined in every section.

## 9.8 Standards

The standard selected is C11 due to it standardizes features already supported by common contemporary compilers, and includes a detailed memory model to better support multiple threads of execution.

For more information of the standard, it can be located at: <https://www.iso.org/standard/57853.html>

The standard tool that will be used to determine if the code complies with the standard is *Cppcheck*.

The results of the static analysis is located in the folder of results with the following path:

<PROJECT\_PATH>\6) Results\ standards\Results.xml

### 9.8.1 Standards Result analysis

This section contains an analysis of the standard results.

The Cppcheck software takes the code and analyze it. The given code are the files locate in the folder <PROJECT\_PATH>\code\Proyecto\_Integrador\src. The Cppcheck reports the findings of the analysis in a xml format. However, there is more useful information like statistics which count the errors, warnings, etc. The statistics are depicted in Figure 11.

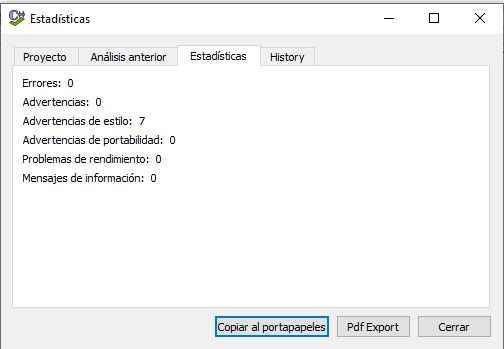


Figure 11 Statistics for standard analysis

As can be observed, there are only 7 warnings (*Advertencias*), those warnings are for style which they are not representing a risk for the system.

In the Figure 12 can be observed the main analysis with the 7 warnings. All of them are in autogenerated code by synergy. The configuration of the LCD display, has specifications for the use of the touch panel. Nevertheless, this capability is not used by this version of the project.

1. guiapp\_specifications.c has a justified warning because it was autogenerated by synergy.
2. common\_data.c has a justified warning because it was autogenerated by synergy.
3. Display\_Thread.c has a justified warning because it was autogenerated by synergy.
4. Display\_Thread\_entry.c has a justified warning because it was autogenerated by synergy.
5. main.c has three justified warnings because it was autogenerated by synergy.
6. guiapp\_event\_handlers.c has a justified warning because it was autogenerated by synergy.

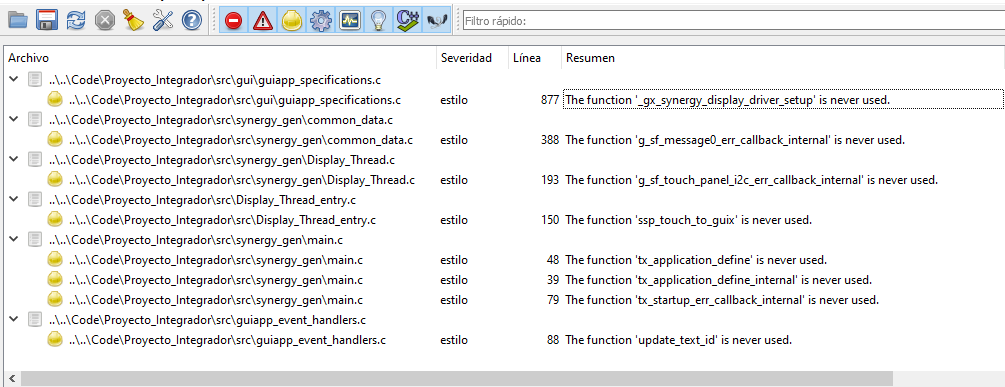


Figure 12 Standard Results

## 9.9 Naming conventions

Since the nature of the project, the naming convention for the variables, functions and other artifacts, will be simple because it is expected low line of code per file.

The Naming convention and considerations for the code are the following:

### 9.9.1 Variables

1. There are not magic numbers (all numbers will have a variable or constant)
2. Variables will in lower case.
3. Variable names will be meaningful. E.g.: *average*
4. If the variable needs a composite name, it will start in lower case and each word of the composite name will start capital. E.g.: *messageReceived*
5. Well-known words can be used in capital followed by an under score to separate the well-known variable with the next word. E.g.: *LED*\_*State*
6. If the well-know word is a suffix, it is not necessary the underscore. E.g.: *actualRPM*
7. If a variable needs a number to differentiate from other variable the number is separate with an under score. E.g.: *message*\_*3*
8. The use of suffix are allowed and the intention is to differentiate a related data. E.g.: speed, speed\_ref (where speed\_ref means a reference speed)
9. Variable and constant declarations will be located at the beginning of the function.
10. There are not global variables, functions will send the information as arguments. There is not restriction to use value or reference.
11. If it is needed calibration variable, it will be considered an adjustment and it will be defined in the adjustments.h

### 9.9.2 Constants

Since the project will use Renesas Synergy Software Package (SSP), and the developer team wants to differentiate between Renesas macros and local macros, the following rules will apply for macros and constants:

Note: The developer team has decided to only use macros in order to reduce the amount of memory.

1. All the Renesas macros will be used as is. All in capital and using under scores for composite words. E.g.: *MODULE*\_*STOP*
2. The macros defined by the development team will use the PI\_ prefix. E.g.: *PI*\_*Address*\_*Limit*
   1. *Note: PI stands for Project Integrator.*
3. All the constants will be located in the files constants.c and constants.h
4. Constants used by synergy are lower case and they not need changes.

### 9.9.3 Functions

1. The name of the functions will start with a verb and follow the rules of a variable. E.g.: *sendMessage*

### 9.9.4 Threads

1. The action of a thread will be specifying in present continue. E.g.: *updating*\_*LCD*

### 9.9.5 Files

1. It is expected one file per function or thread at least the function is overloaded.
2. If the file corresponds to a function or thread, the name will follow its corresponding naming convention.

### 9.9.6 Folders

1. The name of the folder will group files according its folder description. E.g.: *guiapp*
2. The name of folders will use the naming convention of variables. E.g: *readingThreads*

### 9.9.7 Types

1. The types will use the naming convention of a variable and also will use the suffix \_type. E.g.: *messageStructure\_type*.
2. All the types definitions will be in the file PI\_types.h

### 9.9.8 Comments

It is expected comments when:

1. The function/thread is described.
2. Variable declaration.
3. Type definition.
4. Conditions
5. Path conditions
6. Loops
7. Sections.
8. Logic flow.

The comments can be written in English plain.

# Testing

## Verification strategy (black box test)

This section SHALL be contained at:

<PROJECT\_PATH>\4) Verification\10.1. [BlackboxTest\_baseline.docx](4)%20Verification/10.1.%20BlackboxTest_baseline.docx)

…and its results SHALL be located with the date as suffix, as following is indicated:

<PROJECT\_PATH>\4) Verification\Results\10.1. BlackboxTest\_20190405.docx

Every time a module or feature is implemented, it SHALL contain their tests section and SHALL be contained with the reference to the requirement number in order to have traceability.

## White box strategy

It SHALL define the software which is going to be used, for instance: gtest, junit, sunit, etc.

A document baseline SHALL be created as a reference for all the project implementation. This document SHALL be located at:

<PROJECT\_PATH>\4) Verification\10.2. WhiteboxTest\_baseline.docx

…and its result SHALL be located at:

<PROJECT\_PATH>\4) Verification\Results\[10.2. WhiteboxTest\_20190405.docx](4)%20Verification/10.2.%20WhiteboxTest_baseline.docx)

Every time a module or feature is implemented, every test case SHALL contain a reference to the requirement number in order to have traceability.

## Cyclomatic Complexity Redundancy index

The cyclomatic complexity redundancy metric was obtained for the project.

The analyzer used was the *C and C++ Code* Counter which is a tool to analyze C++ and Java files and generates a report on various metrics of the code. Metrics supported include lines of code, McCabe's  
complexity and metrics proposed by Chidamber&Kemerer and Henry&Kafura.

McCabe's cyclomatic complexity is a software quality metric that quantifies the complexity of a software program. Complexity is inferred by measuring the number of linearly independent paths through the program. The higher the number the more complex the code.

The output report can be found in the following path:

<PROJECT\_PATH>\4) Verification\Results\[10.3. CCRI\_20190405.docx](4)%20Verification/10.3.%20CCRI_20190405.docx)

# Release

The release of this project includes all the artifacts in a final state. All the artifacts are located in the GITHUB repository.

Firmware version number is 1.0. All the development was done under decimal versions. The Integer versions were reserved for customer version.

All the documents are under control. This means that the release version is the latest commit to release branch as well as the merge with master. After this merge any update is considered out of the Release 1.0.

The code shall be controlled in GITHUB and path shall be defined here.

## Software Development Folder

The path for software development folder shall be defined in this section and be contained and controlled at GITHUB previous to the final release.

## Integration Tests Strategy

This section SHALL be contained in the planning and reflected in the schedule.

IT **SHALL** be defined a document baseline as a reference for all the project implementation. This document **SHALL** be located at:

<PROJECT\_PATH>\4) Verification\11.1. IntegrationTesting\_baseline.docx

…and its RESULT SHALL be located at:

<PROJECT\_PATH>\4) Verification\Results\11.1. IntegrationTesting\_20190405.docx

Every time a module or feature is implemented, every test case SHALL contain a reference to the requirement number in order to have traceability.

This test MUST contain the plant connected or not.

## Validation Testing / Functional Testing

This section SHALL be contained in the planning and reflected in the schedule.

IT **SHALL** be defined a document baseline as a reference for all the project implementation. This document **SHALL** be located at:

<PROJECT\_PATH>\4) Verification\11.2. ValidationTesting\_baseline.docx

…and its RESULT SHALL be located at:

<PROJECT\_PATH>\4) Verification\Results\11.2. ValidationTesting\_20190405.docx

Every time a module or feature is implemented, every test case SHALL contain a reference to the requirement number in order to have traceability.

This test SHALL contain the plant connected.

## Throughput and Flash and RAM measurement

### 11.4.1 Memory Analysis

The Throughput analysis was done when the code achieved a final state.

This section is contained in the planning and reflected in the schedule.

It defines the RAM, Flash and Throughtput measurements strategy at:

<PROJECT\_PATH>\4) Verification\ 11.3. [ThroughputRAMFlash\_procedure](4)%20Verification/11.3.%20ThroughputRAMFlash_procedure.docx)

# Results

All the pictures, outputs, drafts, utility reports as well as the evidence are located in the following path.

<PROJECT\_PATH>\ 6) Results

The results include some tool outputs like:

* Code statistics. Under Code Review folder
* Photos of some system/integration test. In media folder
* Standards results. In standards folder.
* Other files with diagrams.
* GITHUB statistics.

# Lessons Learned

The team has decided to exercise a retrospective as part of the Scrum methodology. The format used was to answer the following questions:

1. What did the team do well?
   1. The planning was very close to the execution, the main tasks were identified and planned with enough time to be started, worked and completed.
   2. Some Risks from the FMEA happened but the team applied the recommended actions.
   3. The architecture design was very useful. The architecture is flexible and allowed a smooth system testing.
   4. The use of a distributed state machine allowed to have a global state without having all the code inside the logic of the state machine.
   5. The number of threads was the correct one that we needed to have a good balance between process, functions and load.
   6. The number of branches in GITHUB allows to split the progress according with the milestones defined in the planning.
   7. Communication was a key for a fast deployment.
2. What did the team not do well?
   1. The team used the input capture functionality. However, by the nature of the inputs received, probably the use of interruptions could fit better. That is something that the team does not know and there was not time to change that part.
   2. Break big task in smaller tasks. The team identified big tasks but not necessarily all the tasks had the same length, complexity, etc. The team needed more sprints to have a better rhythm.
   3. Some tools were new for the team, and the team feels like they needed more time to get better results with the tools.
3. What are the recommendations for the next project? (assuming the team will start a new development cycle)
   1. Use iterations instead of complete something in the first pass. The team recommends to create software functional artifacts instead of perfect artifacts in order to achieve faster design problems or testing problems.
   2. Consider alternatives in the design in order to have flexible designs instead of starting from the scratch due to the architecture rigidity.
   3. Define acceptance criteria with tolerance in order to progress. In the final stages, the tolerance can be reduced, but we do not recommend from the beginning.
   4. Document the failures and wrong decisions in order to avoid them in the future designs and reduced the rework.