Firmware Generator for IoT Devices

José Tabango-Castillo, Denisse Armijos-Reyes, Dixys Hernández- Rojas*, Bertha Mazón-Olivo,

Unidad Académica de Ingeniería Civil, Universidad Técnica de Machala, km 5.5, Av. Panamericana, Machala, El Oro, Ecuador.
**E-mail: dhernandez@utmachala.edu.ec*

SUMMARY

Due to the technologies incidence of Internet of Things (IoT) in our modern life, for example in telemetry applications for the measurement of temperature and humidity. Using smart sensors, the possibility of configuring and programming these devices according to the needs, with a minimum of programming skills it becomes more and more imperative. This work presents the development of a web application, with different levels of security, hardware abstraction and programming language that reduces the start-up time of a project and the learning curve of a new user. Through a friendly, simple and intuitive graphical interface, the sensors can be configured, and by applying combined pattern techniques, the code for the firmware is generated, allowing the integration of the sensor node in an IoT architecture. The Web application designed was subjected to quality tests, whose result shows that it obtained the necessary score required by ISO / IEC 9126-3.

Keywords: Code generator, firmware, web application, sensor node, IoT.

1. Introduction

The software factory, is a working model used in companies dedicated to the development of software products, [1] whose objective is the execution of development and maintenance projects of some companies, considering that besides developing they allow to give technological solutions, taking as requirements the needs, technology and standards, being it, its main source of income, in which the internet of things (IoT) is applied to the development of their products.

The use of IoT is present in many productive activities, such as irrigation control using humidity sensors, monitoring of aquaculture pools with salinity sensors, sealing control of containers with infrared sensors in manufacturing companies, among others. Changing the way in which organizations communicate with industrial procedures, through the implementation of this technology generates large profits and optimizes business procedures [2]. The use of IoT technology is also present in everyday life, in activities such as the lighting of lights with the proximity sensor, or temperature control in homes with temperature sensors, therefore the vast majority of industrial things and everyday use can be measured or monitored thanks to the IoT [3]. The monitoring of variables such as humidity, temperature, luminosity, salinity, among others, is also achieved through the use of IoT devices, which in turn are made up of sensors, actuators and sensor nodes, which form the main base of an architecture IoT [4]–[6].
One of the drawbacks of the implementation of an IoT architecture is the complexity of the programming IoT devices, since this requires highly trained personnel, in addition, there is also the limitation of adaptability of the code generating applications with the variety of existing programming languages, product of the constant technological changes. Among the code generating applications are the CASE tools, which use diagrams for their operation, generating code suitable for a single programming language [7].

It is very useful in economic terms that an industrial entity implements the use of IoT devices in their production processes, because they contribute to the optimization of resources by simplifying processes and representing savings. However, the cost and time represented by its implementation is high, in addition, due to the difficulty of programming devices, and its low adaptability, there is a high level in the learning curve of IoT device programming. In response to the detailed drawbacks and the need for its implementation, the development team of IOTMACH, has developed a Web Application for the generation of Firmware.

This article presents a software that details a process for the implementation and operation of the same. Besides

this system will be evaluated with ISO / IEC 9126, which is an international model focused on software evaluation [8], consisting of 4 parts model of quality, external metrics, internal metrics and quality of metrics in use. Model 9126-1 is defined for the quality model, and has a framework for evaluating the software. On the other hand, model 9126-2 is defined for external metrics, which measures the general behavior system, either the software is being developed or implemented, while the model 9126-3 has internal metrics to quantify the software characteristics, the latter model 9126-4 is involved with model 9126-1 because it measures the defined attributes of this standard, it also focuses on the feature performance and sub-characteristic measurements, and measures the effects of using specific software. The ISO / IEC 9126 standard has been applied successfully in several countries such as Australia, the United States, and Argentina [9], allowing to specify and evaluate the quality of the software taking into account its requirements, security, use, development and maintenance, among others. The present article first makes an analysis the art state and works related to the generation of code, then the proposed firmware generator system is presented, followed by this, the system quality is evaluated according to the international standard ISO / IEC 9126, culminating with the general conclusions.

2. Related Words.

The needs to generate programs or computer applications in an agile way, is a challenge for many developers or software factories. The accelerated growth of technology requires the development and deployment of quality programs in very short times, however, it is a complex task if you do not have the right tools. For example, a Content Management System (CMS) is a tool that allows you to create web applications without the need to be a specialist, but focused on a very basic field of application. Other cases are frameworks [10], which consist of a standard and reusable environment for programs with a higher level of personalization, however, requires the knowledge and skill of an expert. The generation of program code is a task that has been addressed by many researchers and software developers for several decades, mostly in the field of desktop and web applications. Several CASE tools and fourth-generation programming languages have been created, however, the code they generate is inefficient, difficult to understand and questionable to maintain [11]. Most of the code generators that currently exist in the market start from a model or technique to establish the starting requirements to generate the code of a program. Some of these techniques or models with their respective characteristics have been compiled and examples of tools are showed in **Table 1**. In the field of IoT, the agile generation of code is a challenge of greater importance, due to the diversity of components and technologies that are required to develop, deploy and integrate it. For example, the creation of firmware for IoT devices, development of applications and services (middleware) for both the IoT Gateway and the data processing center or cloud computing. In this sense, IBM has made an important contribution with its Node-Red tool [12], which consists of a visual programming language (graphic components) based on service flows or protocols.

Table 1: Code generation techniques

Techniques	Characteristics	Tools
MDA/MDD/MDE	Model-Driven Architecture (MDA), Model-Driven Development (MDD), Model-Driven Engineering (MDE) they are approaches to software development driven by models [13. MDA is proposed by the OMG [13], it comprises a subset of MDD that provides a series of guides or patterns expressed as models for creating software. The standards that apply are: UML, XMI (XML Metadata Interchange), MOF (Meta Object Facility) and CWM (Common Warehouse Metamodeling).] The working pattern of MDD consists of 3 components: language, editor and generator. MDE is based on models for engineering processes, but they are not the thread of code generation [14], [15]. The case of use for code generation are focused mainly in the creation of applications desk, web y mobiles.	Openxava[16], ArcStyler, OptimalJ,[17] AndroMDA,[18] Codagen,[19] Together Architect, XCode [20]
CIM	Computing Independent Model [21]. It is focused on the context and requirements of the system, without taking its structure or processing. Also known as a business model [22].	Genexus [23]
BPM / BPMN BPMN4WSN	Business process management (BPM) focused to improve the operation of the business. It is divided into phases such as process documentation, and modeling towards the execution, monitoring and optimization process.[24][25]	Genexus [23] Bizagi [26] Studio[27]

DSL	Domain-Specific Modeling (DSL) It is a framework of automatic generation of code executable directly from models (XML, UML, annotations). It is composed of: abstract, concrete and semantic syntax. DSL integrates with MDD. There are several tools that generate code based on DSL for desktop, web and mobile environments, Gateway and IoT devices and other general purpose applications.[22] [28]	Apache Camel [29] Eclipse Kura [30] Aceleo [31] EMF[32] FAP[33]
DSP	Domain Specific Patterns (DSP) Patterns**,** artefacts, Templates. Elevate from one to several levels of abstraction. Presents a meta-model describing the global needs**.**	Eclipse [34]
Visual Code, Flow	Focused on the integration of IOT Gateway applications. Create code through a web interface and visual elements .IOT – Gateway and IoT devices	Node Red [35] Flogo [36]
Patterns, artefacts, Templates	Focused on the automatic generation of code (firmware) for IOT devices [37]. Generation of code through web interface, and templates.	BleGen[38]
MDD and Rule-Based Inference System	The MDD has a method to generate the system, it does so from a model, it performs the transformation through rules, allowing to automate the manipulation of the models [39].	PAMPA[40]

At the end of the investigation of the aforementioned techniques, "Patterns, artefacts, Templates" was selected because, in the generation of source code it is necessary to use programming patterns of which we will highlight the abstraction and reuse of source code, with it you can create templates, so that the end user can generate source code easily, and intuitively without losing the trustworthiness and security of the source code. The remaining techniques previously described in **Table 1** are used less frequently, and also do not fully adapt to the requirements of the present software development, so they were discarded.

3. Firmware generator system

3.1. *General architecture of the system.*

The firmware generator of this work was developed with the objective of programming commercial or private hardware devices to be used as sensor nodes and to be inserted in an IoT system using plug and play mechanisms. Therefore, the generated firmware must contain all the information and configuration necessary for said sensor node to be discovered and enabled by a specific IoT application. In **Figure. 1,** the general architecture of the Firmware Generator System is shown within a conventional IoT scheme, formed by a WSN network containing the sensor nodes and the Gateway, establishing communication between the WSN and the Cloud Computing via Internet. In the Cloud, besides the applications, there are server clusters: two web server, databases and communications among others.

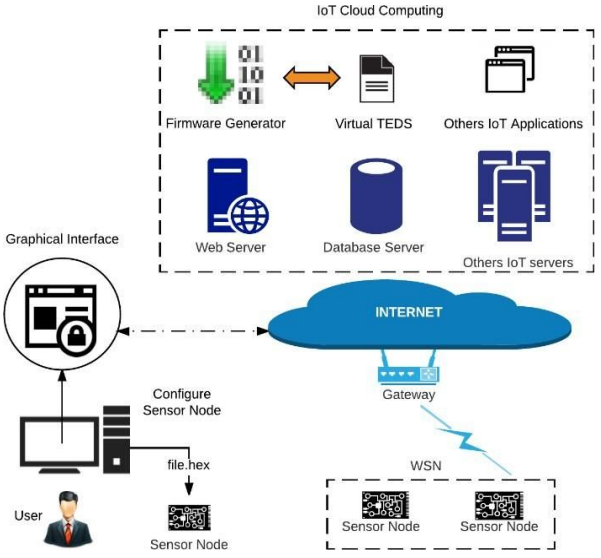


Figure. 1. System architecture firmware generator

The firmware generator system was designed as an application for IoT, it lies in the Cloud with the ability to communicate with other applications already established, obtain all the necessary information and to configure the sensor node with the parameters and settings that guarantee the plug and play mechanism. It can also be seen that the system has a web application, which allows the user to remotely configure and customize all the parameters of the sensor node through a simple and intuitive graphical interface from a browser in a remote terminal, PC, Tablet or Smartphone. Once, the system generates the firmware corresponding to the configurations already defined and with validated parameters, the resulting file (usually a hexadecimal) can be used safely and downloaded from the terminal used. Once there, the firmware is downloaded in to the sensor node using the was technicians and tools of the sensor node provider. The final user of our system should only fill in some templates, previously defined by an expert, to get the desired firmware. The methodology used is detailed in the next section.

3.2. *Workflow*

In this section we will describe the workflow of the firmware generator system to get a reliable source code with all the flexibility, ease and simplicity possible for the end user, which can be a technician or even a user without programming skills. To do this, you should only be clear about the requirements of the selected hardware for the sensor node, according to the IoT application to be implemented, such as precision agriculture, eHealth, smart home or others. In **Figure 2**, this workflow is summarized, which has as inputs three important elements that will be obtained from various sources and concentrated by FGE (Firmware Generator Engine) which are base firmware, validated project and parameters by domain of application. For the generation of reliable firmware, the information validation strategy was adopted in several system layers.

The first flow begins with the "Expert" user, which generates a base firmware for the specific hardware selected as the sensor node and ends in the "FGE".

The second flow begins with the "System Administrator" and ends with the "FGE". This user manages the projects that contain the previously generated firmware, guaranteeing that it contains all the necessary libraries and elements. As mentioned previously to enter the project into the system, it was validated and compiled, guaranteeing the reliability of the final code.

The third flow begins in the "IoT Server" and ends in the same "FGE" as the previous flows. In this case, the objective is to discover parameters of the application domains already existing in said server. These parameters will then be used by the end user to fill in the templates that the "FGE" will generate from the TAGS extracted automatically from the base firmware.

The last flow is born in the "FGE" and ends with the "File machine code" (File. hex) that will be finally recorded in the sensor node. For this the end user should only enter the adjustment values and parameters necessary according to the use that is desired to give the sensor node in the specific application domain, for example, IP address of the communication server, number of ports used, types of sensors, etc.

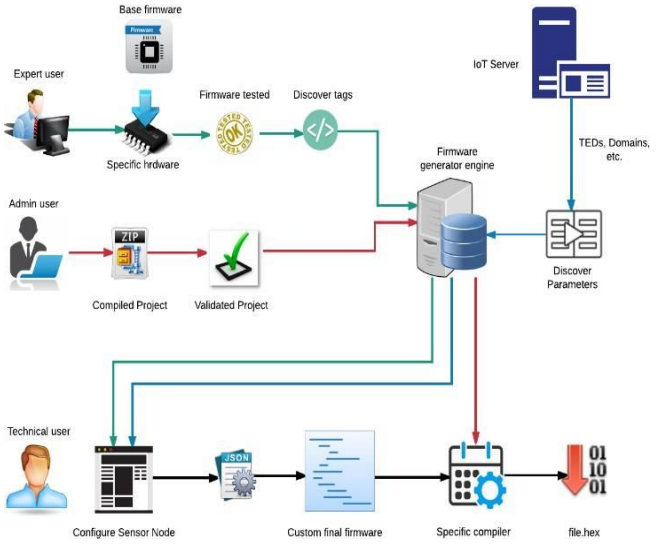


Figure 2. Workflow of the firmware generator system.

3.3. *Base firmware generation*

To guarantee the reliability and safety of the final firmware, the firmware generator system must read the base firmware, with the hardware parameters that must be configured and customized so that the sensor node can be inserted via plug and play. To achieve this, the expert programmer must include in the base firmware a block of labels, assigned to the global variables used in said code, which simplifies and optimizes the generation of final code. In **Figure 3**, the TAGS of the system are exemplified. The "Global Tags" allow configuring the sensor node both with specific information of the sensors and actuators and global information of the IoT system, for example: the application domain. The "Pines Tags" allow to make an abstraction of the hardware when declaring the quantity of pins and sensors associated to the physical pins in the sensor node. On the other hand, the "Modules Tags" allow to declare the communication modules and protocols to be used by the specific application domain.

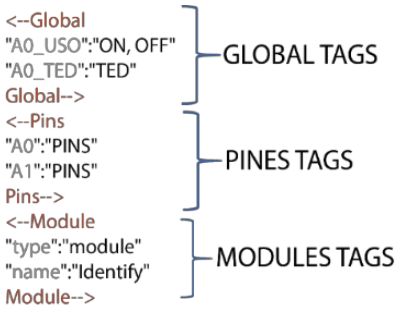


Figure 3. System tags firmware generator

Figure 4, shows the process of generating the firmware, performed by the end user. The process starts with the registration and validation of the technical project, then the user must choose the sensor to configure, this configuration is stored in a JSON file which will be sent to the server. The user fills in the template obtained with the desired data for the sensor node. For example, the pin configuration of the sensor. This template will also be validated before the compiler returns the final file in hexadecimal.

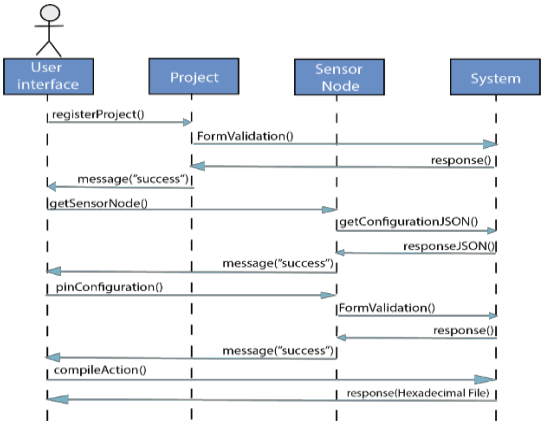


Figure 4. Sequence diagram (Technical User)

3.4. *Web interface of the firmware generation system.*

In order to meet the requirements and operation of the aforementioned application, the Scrum methodology was used for software development [41]. Additionally, different tools were used among which we have:

- PostgreSQL version 9.4, for the development of the database

- Python programming language with Django Framework, for the Server code development (Back-End).
- AngularJS Framework version 1, for the client development side (Front-End)

The main view of the Web application, is showed in **Figure 5**, where in the left sidebar are displayed the modules enabled for the user, for user technical support, Zendesk was used, allowing communication with end users, achieving constant improvement of the web application, which is seen in the lower right. In the upper part, we find the sensor nodes, configured by the expert user, ready for use by the end users.

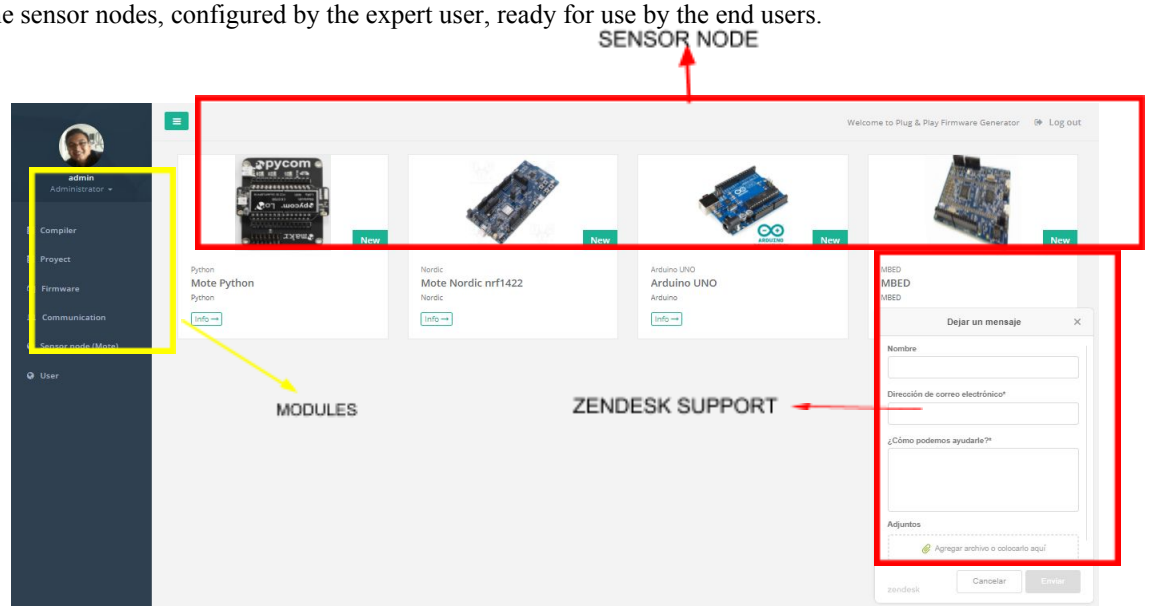


Figure. 5. Main screen of the Firmware Generator System

3.5. *Functionality of the firmware generator system*

In order to verify the functionality of the proposed system, it has been chosen to use two IoT devices, which are Arduino programmed with the language "C" [42] and Pycom, programmed with the Python language [43]. In **Figure 6** we can see the elements used in the experiments, which are Arduino IDE, Atom IDE to program the Arduino and Pycom sensor nodes respectively. The computer used was an HP Laptop 15-bs0xx with Windows 10.

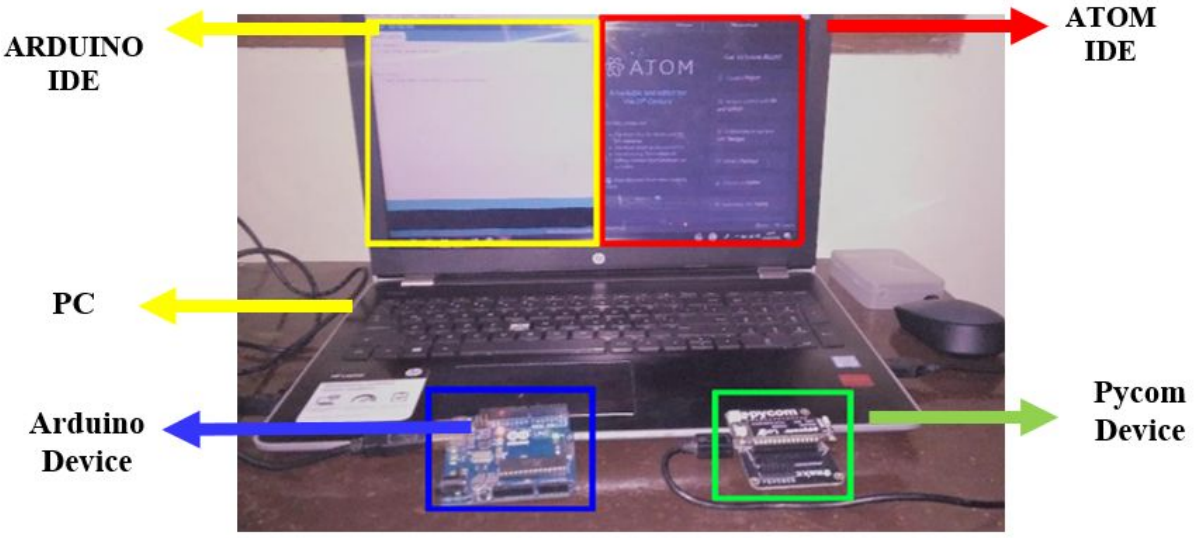


Figure 6. Plaque Arduino One and Pycom (Expansion Board v2).

For the system tests three sections have been established. The first section is the base generation firmware, the second is the web application configuration by the administrator, and the last section is the firmware file generation, which will be detailed below. In the first section, the expert user in IoT technologies performs the base

firmware, verifies and validates it, later the administrator user converts the source code in the TAG format of the proposed system. **Figure 7** shows an example of Base Firmware, which allows turn on the internal LED of Arduino with the variable "P13", and with the variable "Text", it will be displayed in the Arduino IDE console.

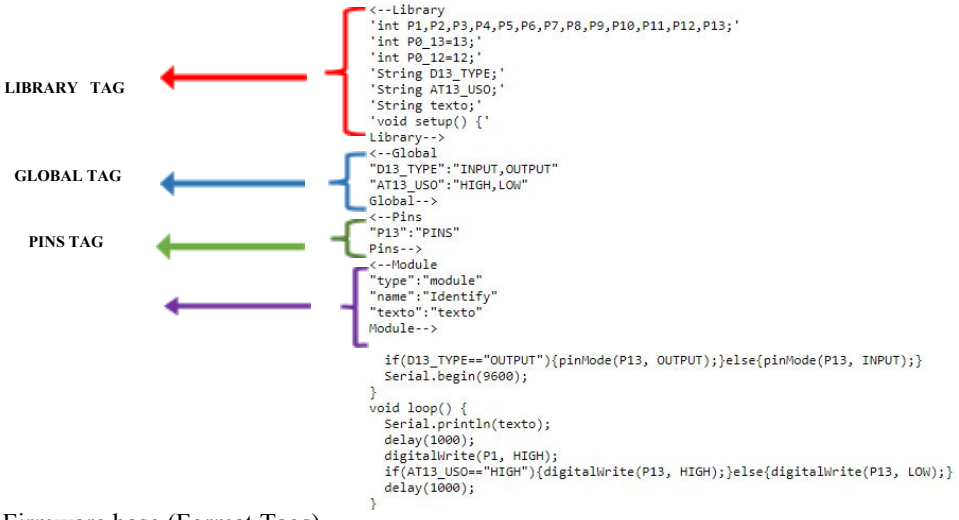


Figure 7 Firmware base (Format Tags)

In the second section, the system will read the TAGS firmware, and using the same data will create the template dynamically, in addition the administrator user must configure and verify the FGE server-side compilers operation. In the last section, the end user must fill in the template with valid information, which will be presented in the web interface as you can see in **Figure 6**. To generate the firmware code preview, the end user must be placed in the "Code" tab ". To compile and select the option" Compile ", with which the hexadecimal file will be downloaded. If you require any modification of the sensor node configuration, you must return to the "Module" or "Pines" tab and making the necessary adjustments in the forms. In **Figure 8**, shows generation of the sensor node code in the proposed application.

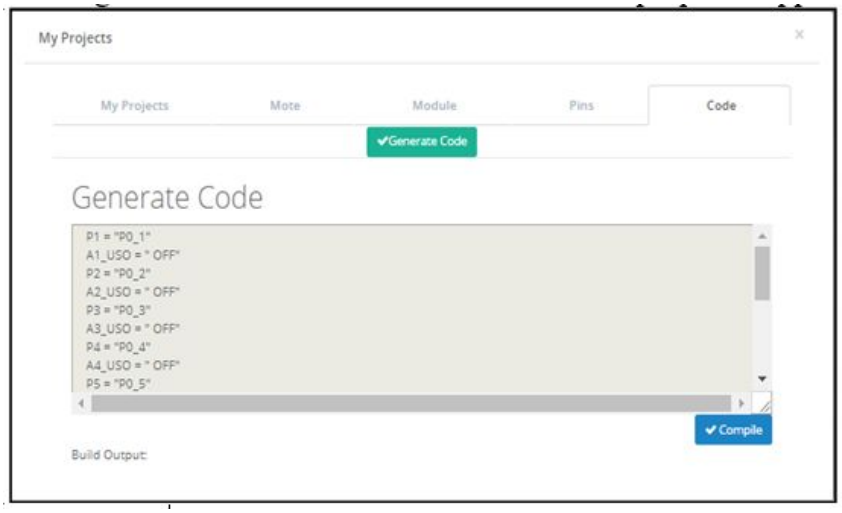


Figure 8. Preview of firmware code

Figure 9 shows the results obtained for the IoT Arduino device, in which the light of pin 13 (internal LED of the board) is visualized, and **Figure 10** shows the results obtained for the IoT Pycom device, in which the light of pin 9 (internal led of the board) is displayed. In addition, it was observed that the messages text registered in the "Modules" tab were correctly presented in the Atom and Arduino terminals.

Figure 9. Arduino one result

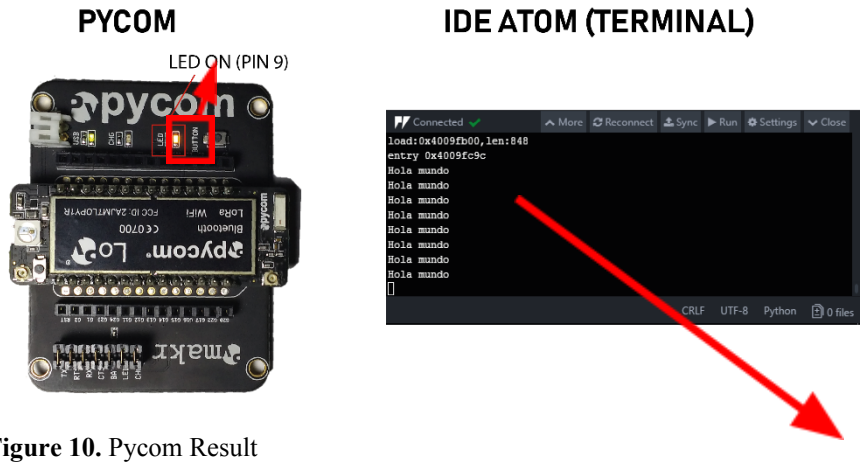
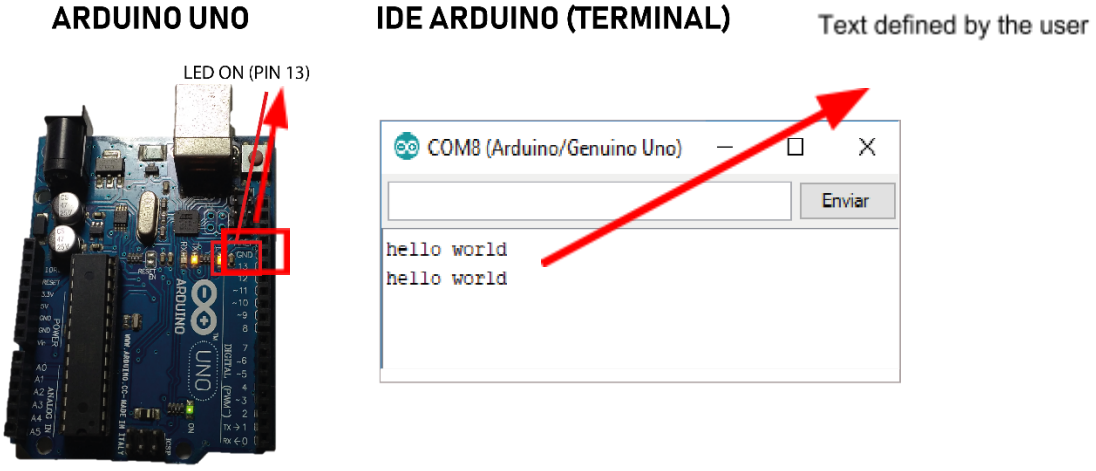


Figure 10. Pycom Result



Once the system functionality has been demonstrated, we will go on to evaluate the system quality in the next section, applying international standards ISO / IEC 9126-3.

4. **Software quality control evaluation.**

Software is one of the most useful tools in process optimization to resource saving and decision making, so its use has significant influence in the field in which it is implemented. For the quality evaluation of the proposed application control, we chose to use ISO / IEC 9126-3 [44], because this standard focuses on the evaluation of software products, and their quality measurement. The selected ISO standard allows to measure the degree of performance of the main system characteristics during its life cycle, which optimizes the operation and the system efficiency [45]. **Table 2** shows the characteristics and sub-characteristics that were used to evaluate the proposed system.

Table 2: Characteristics and sub features of the proposed model

CHARACTERISTICS	SUB FEATURES
Functionality	Suitability, Precision, Security, Compliance functionality.
Reliability	Maturity, Fault tolerance, Reliability compliance, Comprehensibility.
Usability	Learning ability, Operability, Usability.
Efficiency	Use of resources.
Maintainability	Analyzability, Mutability.
Portability	Adaptability, Installation capacity.

The detail of the characteristics to be evaluated in the system proposed is found in the table above. The first

parameter is the functionality, which verifies that the system complies with the requirements specification, the reliability ensures that the system has an adequate functioning in undesirable conditions, the usability verifies that the system is easy to understand, the efficiency determines the optimization of system resources, maintainability ensures that the system is adaptable to changes involving corrections or system improvements and portability analyzes the ability of the system to be used in different devices. In order to evaluate each of the aforementioned characteristics, it is necessary to follow the process flow, shown in **Figure 11**.

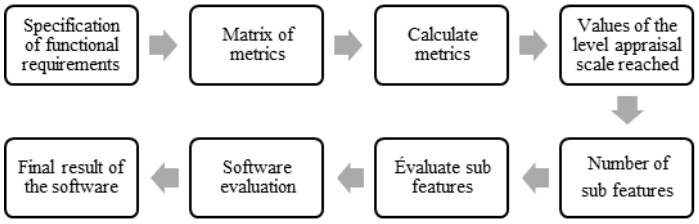


Figure 11. Development Specification of functional requirements

The first stage of the process consists in the specification of requirements, which are obtained from the requirements issued by the IoT user, which are found in **APPENDIX B. SPECIFICATION REQUIREMENTS.** In the second stage the Quality Metrics Matrix was developed, which takes into account the characteristics, sub-characteristics, metrics, formulas, etc., this matrix can be seen in the APPENDIX A. **QUALITY METRIC TABLE**.

The third stage consists of applying formulas established by the ISO standard, whose variables differ in each parameter evaluated. In **Table 3**, the application of a quality metric is displayed in a general way, which corresponds to the sub-characteristic suitability corresponding to the functionality characteristic, in which the score of 1 was obtained for the variable A, which indicates that there is a problem in the evaluated system, while in variable B the score obtained was 23, which represents the number of system requirements. The real measure (x) shows the result of the formula application (metric) that yielded a result of 0.957, considered as a qualitative assessment HIGH.

Table 3: Example of one ISO 9126-3 internal metrics

Characteristics	Sub characteristics	Metrics	Application Method	Formula	A	B	Real Measure (X)	Level reached
Functionality	Suitability	Functional adaptation	Count the number of implemented functions that are suitable for performing the specified tasks, then measure the relationship of the functions with the implemented functions.	**X= 1 - A/B** **A=** Number of functions in which problems are detected in the evaluation **B=** Number of verified functions	1	23	0.957	High

Following the process, the fourth stage consists of establishing the level at which the results obtained in the development of the metrics are found, for which the norm establishes 3 levels of classification: High, Medium and Low, each level has a range in decimal places and in percentages. To establish the ranges, the scale proposed by [46] was used as reference. In **Table 4**, the quantitative scale of the evaluated system is showed.

Table 4: Value level scale reached

Level	Rank	Rank (%)
High	0.76 - 1.00	76 - 100
Medium	0.34 - 0.75	34 - 75
Low	0.00 - 0.33	0 - 33

In the fifth stage formulas pre-established by the standard are applied using the results obtained in the previous stages. For each characteristic, a set called with the variable *C* is represented, the sub-characteristics are represented with the variable *Si*, which in turn have a set of metrics expressed with the variable *M*, and the set of real measures by X. **Table 5** shows the value obtained from the proposed system.

Table 5: Number of sub features and weights by characteristics of the standard ISO/IEC 9126-3

Characteristics (C)	Number of Sub characteristics (Ns_i) Ns_i)	Weigh (Pc_i Pc_i)	Amount (*Vsi*) %
Functionality	4	26	0.26
Reliability	3	21	0.21
Usability	4	17	0.17
Efficiency	2	10	0.10
Maintainability	2	16	0.16
Portability	2	10	0.10
		100	1.00

In the sixth stage the characteristics evaluation is carried out, for which the values of the aforementioned table are needed. **Table 6** shows the sub features evaluation of the proposed system.

Table 6: Evaluation of the sub characteristics

Characteristic (Ci)	Sub characteristics (*si*)	Weigh(Ps_i Ps_i)%	Average (x *si*) %	Weighted Value (*VPsi*)%
Functionality	Suitability	6.50	0.913	5.93
	Precision	6.50	1.000	6.50
	Security	6.50	1.000	6.50
	Compliance functionality	6.50	1.000	6.50
Reliability	Maturity	7.00	0.950	6.65
	Fault tolerance	7.00	0.641	4.48
	Compliance with reliability	7.00	1.000	7.00
	Comprehensibility	4.25	1.920	8.16
Usability	Learning capacity	4.25	0.125	0.53
	Operability	4.25	1.332	5.66
	Compliance usability	4.25	0.625	2.66
Efficiency	Behavior over time	5.00	0.633	3.17
	Utilization of resources	5.00	1.333	6.67
Maintainability	Analyzability	8.00	0.833	6.67
	Mutability	8.00	1.250	10.00
Portability	Adaptability	5.00	1.000	5.00
	Installation capacity	5.00	1.000	5.00
		100.00	16.556	**97.08**

Weight calculation of each sub features

$Ps_i = {}^{Ns_i}/_{Pc_i}$ $Ps_i = {}^{Ns_i}/_{Pc_i} * 100$

W

here Pc_i Pc_i represents a value of each weight of the characteristics and Ns_i Ns_i the number of sub characteristics

that has each one of characteristics.

Calculation for the average of sub features.

$s_i = (\sum_{i=1}^{n} M_i)\ /n\ s_i = (\sum_{i=1}^{n} M_i)\ /n$

The average of the metrics $M_i\ \ M_i$ of each one $s_i\ s_i$

Calculation for the weighted value of the sub features.

$$Vs_i = (Ps_i)\ Vs_i = (Ps_i)_{(s_i\ s_i)}$$

Where $s_i\ s_i$ is the weighted value, $Ps_i\ Ps_i$ represents the weight of each sub-characteristics.

In the seventh stage the values of the maximum scores, real valuation, real percentage of the software and the level reached, shown in **Table 7**, are obtained.

Table 7: Software evaluation according to standard metrics ISO/IEC 9126-3.

Characteristic (Ci)	Maximum Expected Score (*Pmi*)	Real Software Rating (*Vri*) %	Real appraisal percentage of the software (*Pvri*) (%)	Level reached
Functionality	26	25.43	97.83	High
Reliability	21	18.13	86.35	High
Usability	17	17.01	100.05	High
Efficiency	10	9.83	98.33	High
Maintainability	16	16.67	104.17	High
Portability	10	10.00	100.00	High

Calculation of the maximum score

$$Pm_i = Vs_i * 100\ Pm_i = Vs_i * 100$$

Where $Vs_i\ Vs_i$ is the value of each sub characteristic, represents the maximum score of each characteristic also, $Vs_i\ \ Vs_i$ they are taken from **Table 5**.

Calculation of the software actual valuation of each characteristic.

$Vri = Vps_1, Vps_2, \ldots, Vps_n\ Vps_1, Vps_2, \ldots, Vps_n$

Where $Vps_1, Vps_2, \ldots, Vps_n\ Vps_1, Vps_2, \ldots, Vps_n$ represents the weighted value of each sub characteristic

Calculation of the real value software percentage for each characteristic.

$$Pvr_i = {}^{Pm}/_{Vr} * 100$$

$Vr\ Vr$ which represent the real software value, $Pm\ \ Pm$ Maximum score for each feature.

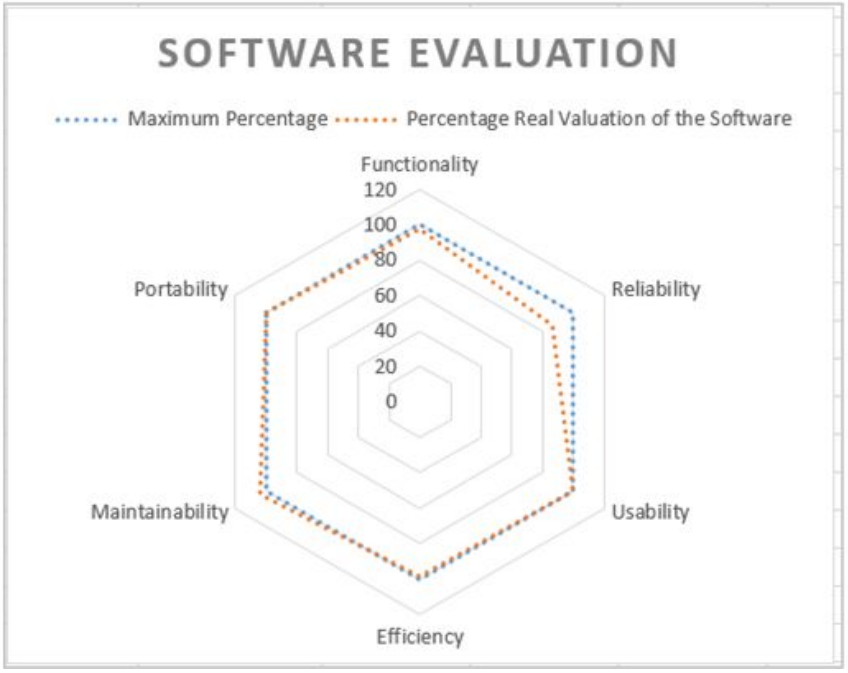
As a last step in **Table 8**, we present the final results of the metrics qualification, in which it can be seen that a good score was obtained because the five characteristics have a high score, which are functionality, portability, usability, maintainability and efficiency, while in the reliability problems were found in the detection and prevention of failures, however, this does not represent disadvantages of great influence. In conclusion, the developed system is suitable for use by end users**.**

Table 8: Final result of the software evaluation.

	Data (%)
Maximum Expected Total Score	100
Real Total Valuation of the software (Score obtained / 100)	97.08
Average percentage of compliance with the characteristics	97.79
General Achieved Level	Alto

In **Figure 12** the statistical table results of the proposed system evaluation is displayed.

Figure 12. Software evaluation



5. CONCLUSIONS

The Firmware Generator System was developed using the current tools, effective and recommended by the scientific community such as: Python, Django and AngularJS. In addition to implementing different levels of security in the system, making use of pre-installed data tested and validated by experts. The Django tool adds another level of security, ensuring that the information is encrypted in an integral and confidential manner. The developed system was implemented in the central servers of the Technical University of Machala on the Windows Server Operating System as part of a set of projects related to IoT. The active Firmware Generator System in this institution has demonstrated high performance for programming the IoT devices used in the Smart Agriculture projects, allowing the programming of the sensors to be done quickly and intuitively, considerably reducing the learning curve. The system allows the configuration of the hardware and the programming language for embedded systems such as C, Python and Node.js, therefore, the system can be adapted for a great variety of hardware types, commercial or private. It is recommended that compilers for future projects be distributed in different servers, guaranteeing multiplatform, thus achieving greater adaptability to new technologies.

References

[1] M. Florencia, "La Gestión del Conocimiento en Pequeñas y Medianas Fábricas de Software en el Área Metropolitana de Buenos Aires," *XIX Work. Investig. en Ciencias la Comput.*, pp. 575–579, 2017.

[2] E. Borgia, "The internet of things vision: Key features, applications and open issues," *Comput. Commun.*, vol. 54, pp. 1–31, 2014.

[3] R. Dhall and V. K. Solanki, "An IoT Based Predictive Connected Car Maintenance Approach," *Int. J. Interact. Multimed. Artif. Intell.*, vol. 4, no. 3, p. 16, 2017.

[4] D. Hernandez Rojas, B. Mazon Olivo, J. Novillo Vicuña, C. Escudero Cascon, A. Pan Bermudez, and G. Belduma Vacacela, "IoT Android Gateway for Monitoring and Control a WSN," *Int. Conf. Technol. Trends*, 2017.

[5] D. L. Hernández-Rojas, T. M. Fernández-Caramés, P. Fraga-Lamas, and C. J. Escudero, "Design and practical evaluation of a family of lightweight protocols for heterogeneous sensing through BLE beacons in

IoT telemetry applications," *Sensors*, vol. 18, no. 1, pp. 1–33, 2018.
[6] A. Campoverde, D. Hernández, and B. Mazón, "Cloud computing con herramientas open-source para Internet de las cosas," *Maskana*, vol. 6, pp. 173–182, 2015.
[7] a Noyer *et al.*, "Tool independent code generation for the UML closing the gap between proprietary models and the standardized UML model," *Eval. Nov. Approaches to Softw. Eng. (ENASE), 2014 Int. Conf.*, pp. 1–9, 2014.
[8] J. M. Ruiz, C. D. Pacifico, and M. M. Pérez, "Clasificación y Evaluación de Métricas de Mantenibilidad Aplicables a Productos de Software Libre," *XIX Work. Investig. en Ciencias la Comput.*, pp. 460–464, 2017.
[9] M. De, M. Callejas-cuervo, A. C. Alarcón-aldana, and A. M. Álvarez-carreño, "Modelos de calidad del software, un estado del arte*," *Entramado, 13(1)*, vol. 13, no. 1, pp. 236–250, 2017.
[10] N. Islam, S. Isam, and S. Sagorika, "Content Management System (CMS): Application of Joomla to website development in Libraries and Information Centers," in *International Seminar "Vision 2021,"* no. February 2011, pp. 83–116.
[11] R. Mall, D. Kundu, and D. Samanta, "Automatic code generation from unified modelling language sequence diagrams," *IET Softw.*, vol. 7, no. 1, pp. 12–28, 2013.
[12] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "Integration of Cloud Computing and Internet of Things: A Survey," *Futur. Gener. Comput. Syst.*, vol. 56, pp. 684–700, 2015.
[13] OMG, "Object Management Group," 2017. [Online]. Available: http://www.omg.org/mda/faq_mda.htm#whatismda. [Accessed: 20-Jul-2017].
[14] A. Rodrigues Da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Comput. Lang. Syst. Struct.*, vol. 43, pp. 139–155, 2015.
[15] G. Giachetti, B. Marín, L. López, X. Franch, and O. Pastor, "Verifying goal-oriented speci fi cations used in model-driven development processes," *Inf. Syst.*, vol. 64, no. August 2015, pp. 41–62, 2017.
[16] "AJAX Java Web Framework for Rapid Application Development of Enterprise Applications - OpenXava," 2017. [Online]. Available: http://openxava.org/es. [Accessed: 20-Jul-2017].
[17] "Objects by Design: OptimalJ Product Review," 2004. [Online]. Available: http://www.objectsbydesign.com/tools/oj/optimalj.html. [Accessed: 27-Sep-2017].
[18] "Home - What is AndroMDA?" [Online]. Available: https://www.andromda.org/. [Accessed: 27-Sep-2017].
[19] "Codagen Technologies," 2017. [Online]. Available: http://www.omg.org/mda/mda_files/codagen_technologies.htm. [Accessed: 27-Sep-2017].
[20] C. Gonzalez-perez and P. Martín-rodilla, "A Metamodel and Code Generation Approach for Symmetric Unary Associations," *Res. Challenges Inf. Sci. (RCIS), 2017 11th Int. Conf.*, 2017.
[21] S. Martínez, L. Lamoth, R. Moreno, and N. Jacho, "Análisis de la Transformación de Modelo CIM a PIM en el Marco de Desarrollo de la Arquitectura Dirigida por Modelos (MDA)," *Rev. Politec.*, vol. 36, no. 3, 2015.
[22] D. de J. Martínez Acosta, "Herramienta para la generación automática del código fuente para aplicaciones con arquitectura modelo vista controlador (MVC) bajo desarrollo dirigido por modelos textuales (MDD)," Universidad Nacional de Colombia, 2014.
[23] "GeneXus, Suite de Desarrollo de Sistemas Empresariales Multiplataforma." [Online]. Available: https://www.genexus.com/productos/genexus?es. [Accessed: 20-Jul-2017].
[24] S. Teixeira *et al.*, "Modeling and automatic code generation for Wireless Sensor Network Applications using Model-Driven or Business Process approaches: A systematic mapping study Modeling and automatic code generation for Wire- less Sensor Network Applications using Model-Dri," *J. Syst. Softw.*, vol. 132, pp. 50–71, 2017.
[25] V. B. Vukšić, M. I. Štemberger, and D. S. Vugec, "Insights into BPM maturity in Croatian and Slovenian companies," *Inf. Commun. Technol. Electron. Microelectron. (MIPRO), 2017 40th Int. Conv. on. IEEE*, no. 1, pp. 1391–1396, 2017.
[26] SYDLE, "BPM - SYDLE," *SYDLE*, 2017. [Online]. Available: https://www.sydle.com/es/bpm/?gclid=EAIaIQobChMIpcHpvdDA1gIV1UsNCh34UAxWEAAYASAAEgJaX_D_BwE. [Accessed: 25-Sep-2017].
[27] Bizagi Time to Digital, "Bizagi Oficial Plataforma de negocios digitales y BPMS," 2017. [Online]. Available: https://www.bizagi.com/. [Accessed: 20-Jul-2017].
[28] A. M. Sutii, M. van den Brand, and T. Verhoeff, "Exploration of modularity and reusability of domain-specific languages: an expression DSL in MetaMod," *Comput. Lang. Syst. Struct.*, 2017.
[29] The Apache Software Foundation, "Apache Camel: Index," 2015. [Online]. Available: http://camel.apache.org/. [Accessed: 20-Jul-2017].
[30] Kura, "Eclipse Kura ™ - Open Source framework for IoT." [Online]. Available: http://www.eclipse.org/kura/. [Accessed: 20-Jul-2017].

[31] The Eclipse Foundation., "Acceleo," 2017. [Online]. Available: https://www.eclipse.org/acceleo/. [Accessed: 20-Jul-2017].
[32] L. Haibing, Z. Ning, L. Yonglin, L. Xiaobo, and Z. Yifan, "EMF Based Validation Methods of the Static Semantics of Models," *2015 2nd Int. Conf. Inf. Sci. Control Eng.*, pp. 207–211, 2015.
[33] D. G. Morales *et al.*, "FAP-Tool : Generación de Código para la Administración Electrónica," pp. 2–5, 2016.
[34] The Eclipse Foundation., "Eclipse Oxygen," 2017. [Online]. Available: https://www.eclipse.org/. [Accessed: 26-Sep-2017].
[35] Node Red, "Node-RED." [Online]. Available: https://nodered.org/. [Accessed: 20-Jul-2017].
[36] "Project Flogo," 2017. [Online]. Available: http://www.flogo.io/. [Accessed: 20-Jul-2017].
[37] J. Pablo and R. Rosero, "Firmware architecture to support Plug and Play sensors for IoT environment," *Conference*, pp. 1–7, 2015.
[38] P. Oliveira and P. J. Matos, "BLEGen — A Code Generator for Bluetooth Low Energy Services," *Lect. Notes Softw. Eng.*, vol. 4, no. 1, pp. 7–11, 2016.
[39] E. E. Thu and N. Nwe, "Model Driven Development of Mobile Applications Using Drools Knowledge-based Rule," in *Software Engineering Research, Management and Applications (SERA), 2017 IEEE 15th International Conference*, 2017, pp. 179–185.
[40] L. Cuaderno *et al.*, "Herramientas de soporte al proceso de desarrollo dirigido por modelos y su implementación con DSL Tools," *Jornadas Investig. y Desarro. en Ing. Softw.*, 2007.
[41] I. E. Cient, E. Antonio, M. Avalia, D. B. Review, and O. J. S. Revis, "A Influência do Papel do Scrum Master no Desenvolvimento de Projetos Scrum," *Gestão e Proj. - GeP*, vol. 8, pp. 80–99, 2017.
[42] P. P. Ray, "A survey on Internet of Things architectures," *J. King Saud Univ. – Comput. Inf. Sci.*, vol. 1, no. 2, pp. 78–95, 2016.
[43] "Pycom - Next Generation Internet of Things Platform." [Online]. Available: https://pycom.io/. [Accessed: 21-Jan-2018].
[44] "ISO/IEC TR 9126-3:2003 - Software engineering -- Product quality -- Part 3: Internal metrics." [Online]. Available: https://www.iso.org/standard/22891.html. [Accessed: 22-Jan-2018].
[45] L. E. González, L. A. E. González, N. J. Acosta, and J. L. G. Tovar, "Estándares Para La Calidad De Software," *Tecnol. Investig. y Acad.*, vol. 5, no. 1, pp. 75–84, 2017.
[46] W. Siabato, "Métricas aplicadas a los modelos de calidad: caso de uso en los SIG," *Congr. Int. Ing. Geomatica y Tipográfica.*, no. 1, pp. 1–13, 2008.

APPENDIX A. QUALITY METRICS.

ISO 9126-3 INTERNAL METRICS								
Characteristics	**Sub characteristics**	**Metrics**	**Application method**	**Formulas**	**A**	**B**	**Actual measurement(X)**	**Level reached**
Functionality	Suitability	Functional adequacy	Count the number of implemented functions that are suitable for performing the specified tasks, then measure the ratio of it to functions implemented.	**X=1-A/B** A= Number of functions in which problems are detected in evaluation B= Number of functions checked	1	23	0,957	**High**
		Functional implementation completeness	Count the number of missing functions detected in evaluation and compare with the number of function described in the requirement specifications.	**X=1-A/B** A=Number of missing functions detected in evaluation. B=Number of functions described in requirement specifications	1	23	0,957	**High**
		Functional implementation coverage	Count the number of incorrectly implemented or missing functions and compare with the number of functions described in the requirement specifications	**X= 1 - A/B** A= Number of incorrectly implemented or missing functions detected. B= Number of functions described in requirement specifications.	1	23	0,957	**High**
		Functional specification stability (volatility)	Count the number of functions changed (added, modified, or deleted) during development life cycle phase, then compare with the number of functions described in the requirement specifications.	**X=1-A/B** A=Number of functions changed during development life cycle phases B=Number of functions described in requirement specifications	5	23	0,783	**High**

Functionality	Precision	Precision	Count the number of data items that meet the requirements of specific levels of precision and compare to the total number of data items with specific level of precision requirements.	**X=A/B** A= Number of data items implemented with specific levels of precision, confirmed in evaluation B= Number of data items that require specific levels of precision	2	2	1,000	**High**
	Security	Access auditability	Count the number of access types that are being logged correctly as in the specifications and compare with the number of access types that are required to be logged in the specifications.	**X=A/B** A= Number of access types that are being logged as in the specifications B= Number of access types required to be logged in the specifications	2	2	1,000	**High**
		Access controllability	Count the number of access controllability requirements implemented correctly as in the specifications and compare with the number of access controllability requirements in the specifications.	**X=A/B** A= Number of access controllability requirements implemented correctly as in the specifications. B= Number of access controllability requirements in the specifications.	3	3	1,000	**High**
		Data encryption	Count the number of implemented instances of encrypt able /decrypt able data items as specified and compare with the number of instances of data items requiring data encryption/decryption facility as in specifications.	**X=A/B** A=Number of implemented instances of encrypt able/decrypt able data items as specified confirmed in review B= Number of data items requiring data encryption/decryption facility as in specifications NOTE: Data encryption: e.g., data in open database, data in public communication facility	1	1	1,000	**High**
	Functionality compliance	Functional compliance	Count the number of items requiring compliance that have been met and compare with the number of items requiring compliance as in the specification	**X=A/B** A= Number of correctly implemented items related to functionality compliance confirmed in evaluation B= Total number of compliance items.	8	8	1,000	**High**
		Intersystem standard compliance	Count the number of interfaces that meet required compliance and compare with the number of interfaces requiring compliance as in the specifications. Note: All specified attributes of a standard must be checked	**X=A/B** A= Number of correctly implemented interfaces as specified, confirmed in review B= Total number of interfaces requiring compliance	1	1	1,000	**High**
Reliability	Maturity	Fault detection	Count the number of detected faults in review and compare it to the number of estimated faults to be detected in this phase	**X=A/B** A=Absolute number of faults detected in review B=Number of estimated faults to be detected in review (using past history or reference model)	3	5	0,600	**Medium**
		Fault removal	Count the number of faults removed during design/coding and compare it to the number of faults detected in review during design/coding.	**X=A** A=Number of corrected faults in design/coding Y=A/B A=Number of corrected faults design/coding B= Number of faults detected in review.	7	7	1,000	**High**
		Test adequacy	Count the number of test cases planned and compare it to the number of test cases required to obtain adequate test coverage.	**X=A/B** A=Number of test cases designed in test plan and confirmed in review B= Number of test cases required	25	20	1,250	**High**
Reliability	Fault tolerance	Failure avoidance	Count the number of avoided fault patterns and compare it to the number of fault patterns to be considered.	**X=A/B** A=Number of fault patterns having avoidance in design/code B=Number of fault patterns to be considered NOTE: Fault pattern examples out of range data deadlock NOTE: Fault tree analysis technique may be used to detect fault patterns.	10	20	0,500	**Medium**

		Incorrect operation avoidance	Count the number of implemented functions to avoid critical and serious failures caused by incorrect operations and compare it to the number of incorrect operation patterns to be considered. NOTE: Also, data damage in addition to system failure	**X=A/B** A=Number of functions implemented to avoid incorrect operation patterns. B=Number of incorrect operation patterns to be considered NOTE: Incorrect operation patterns Incorrect data types as parameters Incorrect sequence of data input Incorrect sequence of operation NOTE: Fault tree analysis technique may be used to detect incorrect operation patterns	25	32	0,781	**High**
	Reliability compliance	Reliability compliance	Count the number of items requiring compliance that have been met and compare with the number of items requiring compliance as in the specification.	**X=A/B** A= Number of correctly implemented items related to reliability compliance confirmed in evaluation B= Total number of compliance items	8	8	1,000	**High**
Usability	Understandability	Completeness of description	Count the number of functions which are adequately described and compare with the total number of functions in the product.	**X= A/B** A= Number of functions (or types of functions) described in the product description B= Total number of functions (or types of functions).	24	25	0,960	**High**
		Demonstration capability	Count the number of functions that are adequately demonstrable and compare with the total number of functions requiring demonstration capability	**X=A/B** A= Number of functions demonstrated and confirmed in review B= Total number of functions requiring demonstration capability.	32	8	4,000	**High**
		Function understandability	Count the number of user interface functions where purposes is understood by the user and compare with the number of user interface functions.	**X= A/B** A= Number of user interface functions whose purpose is understood by the user B= Number of user interface functions.	20	25	0,800	**High**
	Learnability	Completeness of user documentation and/or help facility	Count the number of functions implemented with help facility and/or documentation and compare with the total number of functions in product.	**X= A/B** A= Number of functions described B= Total of number of functions provided	4	32	0,125	**Low**
	Operability	User operation cancellability	Count the number of implemented functions, which can be cancelled by the user prior to completion and compare it with the number of functions requiring the pre cancellation capability.	**X=A/B** A=Number of implemented functions which can be cancelled by the user B= Number of functions requiring the pre cancellation capability	32	8	4,000	**High**
		User operation Undo ability	Count the number of implemented functions, which can be undone by the user after completion and compare it with the number of functions.	**X = A / B** A=Number of implemented functions which can be undone by the user B= Number of functions	32	32	1,000	**High**
		Customizability	Count the number of implemented functions, which can be customized by the user during operation and compare it with the number of functions requiring the customization capability	**X=A/B** A=Number of functions which can be customized during operation B=Number of functions requiring the customization capability	32	32	1,000	**High**
		Physical accessibility	Count the number of implemented functions, which can be customized and compare it with the number of functions	**X=A/B** A=Number of functions which can be customized. B=Number of functions.	8	8	1,000	**High**
		Operational consistency	Count the number of instances of operations with inconsistent behavior and compare it with the total number of operations	**X=1 - A/B** A=Number of instances of operations with inconsistent behavior B=Total number of operations	20	32	0,625	**Medium**
Usability		Message Clarity	Count the numbers of implemented messages with clear explanations and compare it with the total number of messages implemented.	**X=A/B** A=Number of implemented messages with clear explanations. B=Number of messages implemented.	45	50	0,900	**High**
		Interface element clarity	Count the number of interface elements which are self-explanatory and compare it with the total number of interface elements.	**X=A/B** A=Number of interface elements which are self-explanatory. B=Total number of interface elements.	20	25	0,800	**High**
	Usability compliance	Usability compliance	Count the number of items requiring compliance that have been met and compare with the number of items requiring compliance as in the specification.	**X=A/B** A= Number of correctly implemented items related to usability compliance confirmed in evaluation.	20	32	0,625	**Medium**

				B= Total number of compliance items.				
Efficiency	Time behavior	Response time	Evaluate the efficiency of the operating system and the application system calls. Estimate the response time based on this. The following may be measured, -all or parts of design specifications -test complete transaction path -test complete modules/parts of software product -complete software product during test phase.	**X = If(A/B>=1; 1;A/B)** A= Minimum expected response time seconds B= Actual response time in seconds.	2	3	0,667	**Medium**
		Throughput time	Evaluate the efficiency of handling resources in the system. Make a factor based upon the application calls to the system in handling the resources.	**X = If(A/B>=1; 1; A/B)** A= Minimum expected waiting time in seconds. B= Real timeout in seconds.	3	5	0,600	**Medium**
	Resource utilization	Memory utilization message density	Count the number of error messages pertaining to memory failure and warnings and compare it to the estimated number of lines of code responsible in system calls.	**X=A/B** A=Number of memory related error messages. B=Number of lines of code directly related to system calls	8	6	1,333	**High**
Maintainability	Analyzability	Activity recording	Count the number of items logged in the activity log as specified and compare it to the number of items required to be logged.	**X=A/B** A=Number of implemented data login items as specified confirmed in review B=Number of data items to be logged defined in the specifications	50	60	0,833	**High**
	Changeability	Change record ability	Record ratio of module change information.	**X=A/B** A=Number of changes in functions/modules having change comments confirmed in review B=Total number of functions/modules changed from original code.	5	4	1,250	**High**
Portability	Adaptability	Porting user friendliness	Count the number of implemented functions which are capable of supporting ease-of-adaptation by user as specified and compare it to the number of functions with easy-to-adapt capability requirements.	**X=A/B** A=Number of functions supporting ease-of-adaptation by user as specified, confirmed in review. B=Total number of functions with ease-to-adapt capability requirements.	32	32	1,000	**High**
	Install ability	Installation effort	Count the number of implemented installation automated steps and compare it to the number of prescribed installation steps.	**X=A/B** A=Number of automated installation steps confirmed in review. B=Number of installation steps required.	5	5	1,000	**High**
		Installation flexibility	Count the number of implemented customizable installation operations as specified and compare it to the number of installation operations with customization capability requirements.	**X=A/B** A=Number of implemented customizable installation operation as specified confirmed in review. B=Number of customizable installation operation required.	1	1	1,000	**High**

APPENDIX B. SPECIFICATION REQUIREMENTS

1. **System functions**

The processes that make up the computer system are the following:

- ✓ Compiler Management
  - o List of Compilers
  - o Compiler Registration
  - o Compiler Update
  - o Compiler Removal
- ✓ Project management
  - o Projects list
  - o Project Registration
  - o Project Update
  - o Project Elimination
- ✓ Firmware Management

- Firmware List
- Firmware Registration
- Firmware update
- Removal of Firmware

✓ Management of Sensor Nodes (Mote)
- Nodes Sensor List
- Sensor Node Register
- Sensor Node Update
- Sensor Node Elimination

✓ Technical Projects Management
- Technical Projects List
- Technical Project Registration
- Technical Project Update
- Technical Project Elimination

✓ Security Management
- User Control
- Roles Control
- Permissions Control

2. **Specific requirements**

2.1 **Functional Requirements**

2.2 Compiler Management

No. R	kind	Description	Data	Restrictions - Observations	Priority
R1	Registry	configuration of a new compiler	Name, type Compiler	They must not be null	High
R2	Modification	Modification of compiler data	Data R1	Work with R1	Low
R3	Query	Compiler Data Reference	Data R1	Work with R1	Low
R4	Low logic	Compiler Data Removal	Data R1	A file that already exists is selected	Low

2.3 Project management

No. R	Kind	Description	Data	Restrictions - Observations	Priority
R5	Registry	Input a new project	Python project - Name - Base firmware name - Detail - Command line - Project Keil project - Name - Base firmware name - Detail - Path - Command line - Project MBED project - Name - Base firmware name - Detail - User - Password - Project URL - target - program Local project - Name - Detail - Command line - Project	They must not be null Compiler comes from R1	High
R6	Modification	Modification of project data	Data R5	Table with R5	Low
R7	Query	Consultation of Project Data	Data R5	Table with R5	Low

R8	Low logic	Elimination of Project Data	Data R5	After the download Only consultations can be made Work with R5	Low

2.3.1 Firmware Administration

No. R	Kind	Description	Data	Restrictions - Observations	Priority
R9	Registry	Enter a new Firmware	Name, Programming language, Project Compiler, status, syntax, icon, firmware file	They should not be null, Project Compiler comes from R5	High
R10	Modification	Modification of Firmware data	Data R9	Work with R9	Low
R11	Query	Checking the firmware data	Data R9	Work with R9	Low
R12	Low logic	Removal of firmware data	Data R9	After the download Only consultations can be made Work with R9	Low

2.3.2 Sensor Node Management

No. R	Kind	Description	Data	Restrictions - Observations	Priority
R13	Registry	Entry of a new sensor	Name, model, manufacturer, processor, status, description, image, image Pin, Firmware, List Pin, list Module, list Ted	Firmware comes from R9 List Pin, list Modulo comes from R9 They must not be null	High
R14	Modification	Modification of sensor data	Data R13	Work with R13	Low
R15	Query	Check the sensor data	Data R13	Work with R13	Low
R16	Low logic	Removal of sensor data	Data R13	After the download Only consultations can be made Work with R13	Low

2.3.4 Technical Project Management

No. R	Kind	Description	Data	Restrictions - Observations	Priority
R17	Register	A new module entry	Project Name, State Description, Advanced Configuration, Node Sensor, list Module List Pin, Code Generated	Node Sensor comes from R13 List Pin, list Modulo comes from R13 List Ted comes from Virtual Teds servers They must not be null	High
R18	Modification	Modification of the module data	Data R17	Work withR17	Low
R19	Query	Consultation of the Data of the module	Data R17	Work with R17	Low
R20	Low logic	Data Removal from the module	Data R17	After the download Only consultations can be made Work with R17	Low

2.4 Security Management

2.4.1 User Control

No. R	Kind	Description	Data	Restrictions - Observations	Priority
R21	High	Input a new user	User, Password, first name, last name, email	It must not be null, It must be greater than 6 characters, The name and password must not be the same	High
R22	Modification	Modify a user	Data R25	work with the requirements R25	Medium
R23	Query (screen)	consult the data by screen	Name, surnames, mail	work with the requirements R25	Low
R24	Low logic	Removal of the user	User	work with the requirements R25	Low

2.4.2 Roles Control

No. R	Kind	Description	Data	Restrictions - Observations	Priority
R25	high	Input a new role	Rol name, rol description.	It should not be null, Role names should not be the same, should not be null, Role names should not be the same.	High
R26	Modification	Modify a Role	Modify the same data as the income	work with the requirements R29	Medium
R27	Low logic	Elimination of the role	Role identifier (Name)	work with the requirements R29	Low

2.4.3 Permissions Control

No. R	Kind	Description	Data	Restrictions - Observations	Priority
R28	High	Enter a permit Role	Name of the permit, description of the permit	Must not be null, Permission name should not be equal.	High
R29	Modification	Modify a permit	Data R39	work with the requirements R32	Medium
R30	Low logic	Elimination of the permit	Permit Identifier (Name)	work with the requirements R32	Low