TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE APIZACO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"METODOLOGÍA DE ASEGURAMIENTO DE LA CALIDAD (VEyVAA) DE UN HEAD END SYSTEM (HES) DE UNA RED DISTRIBUIDA DE MEDIDORES INTELIGENTES"

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRÍA EN SISTEMAS COMPUTACIONALES

PRESENTA

ING. MANUEL FLORES NAVA

DIRECTORES

M. C. MARÍA GUADALUPE MEDINA BARRERA

M.C. JOSÉ JUAN HERNÁNDEZ MORA

Apizaco, Tlaxcala, Agosto 2018

Dedicatoria

El trabajo de tesis que aquí se presenta lo dedico a las personas que han aportado en mi crecimiento como persona y como profesional, pero de manera especial se lo dedico a los siguientes seres de mí vida:

A Dios por permitirme cumplir con mis objetivos y no abandonarme en los momentos difíciles y regalarme momentos maravillosos durante esta etapa de aprendizaje. Gracias Dios por escuchar mis oraciones, por darme la fortaleza y entendimiento para cumplir con este trabajo.

A mi pequeña traviesa Suri Sahori, que es hoy en día mi principal motor de alegrías y de motivaciones para cumplir mis metas. Hija mía tú me has regalado la más bella experiencia de ser papa, por lo que día a día me esfuerzo para que te sientas orgullosa de tu cariño y pueda darte las mejores enseñanzas de vida, mi deseo es ser tu mejor ejemplo de vida, te amo hija.

A mi cariño Cecy, a ti que me has motivado día a día y que has creído siempre en mí, agradezco muchísimo tus consejos y tu apoyo incondicional siempre lo tengo presente, gracias por estar en el lugar y en el momento exacto para impulsarme a cumplir con mis objetivos en la ingeniería y ahora en la maestría, que tu bien sabes lo duro que trabaje y los sacrificios que se tuvieron que hacer durante estos años para lograr mis metas del cual me siento muy satisfecho, una vez más gracias siempre te llevo en mi mente y corazón.

A mi mama Hortensia, a ti que eres mi inspiración de coraje y de fortaleza para confrontar los problemas y superarlos, desde que decidiste apoyarme en continuar con mis estudios a pesar de las adversidades me he esforzado para que siempre te sientas orgullosa de mí, gracias mami porque tú desde siempre has creído en mí, has confiado en mis capacidades y sé que mis logros también son los tuyos, porque gracias a ti he podido llegar muy lejos por tus consejos y tu amor, no olvidare nunca lo que has hecho por mí.

A mis hermanos y familia, por escuchar (o aguantar jaja) las muchas cosas que siempre tengo que decir y tratar de compartir los conocimientos que durante estos años he aprendido, gracias por comprender mi situación en el estudio que espero sea un motivante bueno para todos. Somos una familia con muchas cosas por compartir nuestra unión siempre debe permanecer para ser felices.

A mi papa que estoy seguro que desde donde te encuentres estas orgulloso de lo que he logrado y se que en todo momento estas a mi lado para sentirme seguro de superar todos los obstáculos que se presentan en la vida, gracias por todo lo que nos compartiste.

"Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber." - Albert Einstein.

"Libera tu mente y te sorprenderás de lo que tu mente tenía en secreto" – Manuel Flores

Agradecimientos

Como primera instancia agradezco a CONACYT por el apoyo económico que ha aportado en mis estudios de maestría, estos apoyos fueron fundamentales para lograr los objetivos de esta tesis y agradezco mucho la confianza que han depositado en mí, que por mi parte he correspondido y cumplido con mis compromisos y obligaciones.

También agradezco al Instituto Tecnológico de Apizaco por abrirme las puertas de sus instalaciones, por poner a disposición todo su material de información para llevar acabo mis investigaciones y por ofrecer sus servicios de calidad para satisfacer mi preparación académica.

Agradezco a mi comité revisora, a la maestra María Guadalupe Medina Barrera que fungió como mi directora de tesis, gracias a sus consejos, exigencias, dedicación y apoyo he podido culminar la tesis que aquí se presenta, al maestro José Juan Hernández Mora que fungió como codirector, gracias por motivarme y guiarme durante mis estancias así como dedicar tiempo para mejorar este trabajo, a la maestra María Janaí Sánchez Hernández que fungió como mi tutora y al maestro Higinio Nava Bautista que fungió como mi revisor, agradezco por su dedicación y tiempo para mejorar este trabajo de tesis.

A los maestros que aportaron sus conocimientos durante esta etapa de aprendizaje académico en la maestría, les agradezco por su dedicación y empeño a su buena labor de enseñanzas de forma profesional.

Agradezco a mis compañeros y amigos de la maestría en sistemas computacionales, así como de las diferentes líneas de investigación por los buenos consejos, por las charlas amenas que contribuyen a fortalecer conocimientos diversos y por compartir sus experiencias profesionales, así como de agradables convivencias.

Agradezco a la empresa Eos Tech por darme la oportunidad y abrirme las puertas de sus instalaciones para llevar a cabo el proyecto que en esta tesis se presenta, gracias por las buenas experiencias vividas y las buenas enseñanzas que nos proporcionaron en sus diversos cursos de preparación, gracias por las buenas convivencias laborales y sociales.

A los compañeros y jefes de Eos tech, por su apoyo, por sus consejos, por las risas, pero sobre todo por su amistad que más de uno me lo ofreció incondicionalmente, cada uno de ustedes son un mundo por conocer que han aportado buenas cosas a mi vida como profesional y como ser humano, les agradezco por las excelentes convivencias que no se olvidaran.

A mis compañeros amigos que compartimos la aventura y experiencia en Querétaro durante 6 meses, no olvidare los buenos momentos que vivimos juntos en la empresa y en la casa, Donde compartimos muchas alegrías y una que otra situación difícil que supimos superar, además todos logramos alcanzar nuestros objetivos con la satisfacción de cumplir con nuestros proyectos.

Resumen

En el trabajo de investigación sobre el aseguramiento de calidad de software, se diseña y se implementa una metodología *VEyVAA* (Verificación y validación Ágil) en *Scrum*, que permite evaluar la calidad de un software mediante dos enfoques diferentes que son: la verificación de los componentes en base a los requerimientos técnicos y la validación del sistema terminado en base al estándar *ISO/IEC FDIS 9126*.

Para aplicar los dos enfoques de evaluación se propone dos tipos de *tester* el interno y externo. El *tester* interno trabaja a lado del programador y se encarga de verificar mediante pruebas objetivas, además de formar parte del equipo scrum en la toma de decisiones. El *tester* externo se encarga de validar el sistema completo mediante pruebas subjetivas y estándar de calidad, además de velar por los interese del cliente. En cada tipo de evaluación se describen los procesos, la planeación de pruebas, la interacción con el equipo *Scrum* o con los *stakeholders* y la utilización de formatos para el registro de datos.

La aplicación de la metodología se realizó en el proyecto *Head End System (HES)*, este sistema está planeado para ser implementado en ciudades inteligentes (*Smart Cities*), por lo que es muy importante evaluar la calidad debido a la gran responsabilidad social que conlleva su utilización.

La función del *HES* es gestionar todo el flujo de datos de las *utility* (sistemas de servicios públicos) para procesar y enviar las tareas correspondientes a la red de medidores inteligentes de energía eléctrica. El sistema está estructurado en la arquitectura Modelo-Vista-Controlador y *building blocks*, por lo que el *tester* clasifica los módulos en las tres capas para organizar las pruebas y herramientas a utilizar.

Los resultados de la metodología se dividen en dos partes: la primera parte son los resultados de la verificación como son las fallas detectadas, las notificaciones al programador de todas esas fallas, las correcciones de las fallas por parte del programador y la liberación de los módulos. La segunda parte son los resultados de la validación como son el porcentaje de implementación de las características de calidad y el cumplimiento de los requerimientos del cliente.

Abstract

In the research work on software quality assurance, a VEyVAA methodology (Agile Verification and Validation) is designed and implemented in Scrum, which allows evaluating the quality of a software using two different approaches: verification of the components based on the technical requirements and the validation of the finished system based on the ISO / IEC FDIS 9126 standard.

To apply the two evaluation approaches two types of internal and external tester are proposed, the internal tester works with the programmer and is responsible for verifying through objective tests, as well as being part of the scrum team in the decision making and the tester external is responsible for validating the complete system through subjective tests and quality standards, in addition to ensuring the interests of the client.

In each type of evaluation, the processes, the planning of tests, the interaction with the Scrum team or with stakeholders and the use of formats for data recording are described.

The application of the methodology was carried out in the Head End System (HES) project, this system is planned to be implemented in smart cities, so it is very important to evaluate the quality due to the great social responsibility that comes with its utilization.

The function of the HES is to manage all the data flow of the utility (public service systems) to process and send the corresponding tasks to the network of smart electric energy meters. The system is structured in the Model View Controller architecture and building blocks, so the tester classifies the modules in the three layers to organize the tests and tools to be used.

The results of the methodology are divided into two parts: the first part is the results of the verification such as the detected faults, the notifications to the programmer of all those faults, the corrections of the failures by the programmer and the release of the modules. The second part is the results of the validation such as the percentage of implementation of the quality characteristics and compliance with the client's requirements.

Tabla de Contenidos

Capítulo 1 Introducción 1
1.1 Planteamiento del problema 3
1.2 Justificación 4
1.3 Pregunta de investigación 5
1.4 Objetivo general 5
1.4.1 Objetivos específicos 5
1.5 Alcances y limitaciones 5
1.6 Estado del arte 6
1.6.1 Antecedentes 6
1.6.2 Sistemas de control de consumo de energía eléctrica en el mundo 9
1.6.3 Sistemas de aseguramiento de la calidad de software 15
Capítulo 2 19
Marco teórico 19
2.1 Modelos de desarrollo de software 19
2.2 Pruebas de software 29
2.3 Calidad de software 42
2.3.1. Calidad estructurada (modelos y estandares) 47
2.4 Tecnologías entorno a *Head End System* (Caso de prueba) 56
Capítulo 3 66
Metodología 66
3.1 Esquema general de la metodología *VEyVAA* 66
3.2. Verificación 70
3.2.1 Proceso de verificación 70
3.2.2 Diseño de un plan de pruebas de verificación 73
3.2.3 Interacción del tester interno con el equipo de Scrum 75
3.3 Validación 76
3.3.1 Estándar ISO/IEC FDIS 9126 y Métricas 76
3.3.2 Proceso de validacion 83
3.3.3 Diseño del plan de pruebas de validación 85
3.3.4 Pruebas de aceptación 87
3.3.5 Interacción del tester externo con los Stakeholders 88
3.4 Formatos 89
3.4.1 Reporte de plan de pruebas 90
3.4.2 Reporte de resultados de verificación 91
3.4.3 Reporte de Resultados de validación 92
Capítulo 4 102
Caso de prueba: aplicación de la metodología *VEyVAA* a *Head End System (HES)* 102
4.1 Análisis del *Head End System* y Análisis de requerimientos 103
4.1.1. Head End System (HES) 103
4.1.2 Modelo-Vista-Controlador *(MVC)* en *Head End System* 104
4.1.3. Tecnologías de desarrollo para el *Head End System* 105
4.1.4 Análisis de requerimientos 106

4.2 Evaluación enfocado a verificación....109
4.2.1 Verificacion del módulo de encriptación y desencriptación *AES128*....109
4.2.2 Verificación al módulo de *CRC* (Calculo de Redundancia Cíclica) 123
4.2.3 Verificacion al módulo de web service 133
4.2.4 Verificación al modulo de Servicio de Windows....139
4.2.5 Verificación al módulo de Base de datos 142
4.2.6 Verificacion al módulo de Interfaces....148
4.3 Evaluación de validación del sistema....152
4.3.1 Creación del servidor (cliente-servidor)....152
4.3.2 Evaluacion del sistema 153
4.4 Resultados generales del *Head End System* 160
4.4.1 Resultados del módulo de *Advanced Encryption Standard AES 128*....160
4.4.2 Resultados del módulo del *CRC*....161
4.4.3 Resultados del módulo de la *Web Service*....163
4.4.4 Resultados del módulo de servicio de Windows.....165
4.4.5 Resultados del módulo de Base de datos....167
4.4.6 Resultados del módulo de interfaces 169
4.4.7 Resultados preventivos y correctivos de los módulos....172
4.4.8 Resultados de la evaluación del sistema....173
Capítulo 5 174
Conclusiones y trabajos a futuro 174
5.1 Conclusiones....174
5.2 Trabajos futuros....177
Bibliografia y referencias 179
Referencias de figuras 184
Referencias de tablas 185
Glosario de términos....186
ANEXO 1 190
ANEXO 2 208

Índice de tablas

Tabla 2.1.- Fases del modelo Waterfall (Pressman, 2010) 21
Tabla 2.2.- Fases del modelo RAD y su descripción. (Elaboración propia) 23
Tabla 2.3.- Roles en scrum y su función (elaboración propia) 26
Tabla 2.4.- Eventos de Scrum y su descripción (Lara, 2014) 27
Tabla 2.5.- Herramientas de Scrum y su descripción (elaboración propia) 28
Tabla 2.6.- Tipos de pruebas de caja blanca (elaboración propia) 38
Tabla 2.7.- Comparativa de sqa, sqc y pruebas (Elaboración propia). 45
Tabla 2.8.- Enfoques de SQA (Vargas y Biagioli, 2009) 45
Tabla 2.9.- Comparativa entre verificación y validación (Elaboración propia). 47
Tabla 2.10.- Niveles de madures de CMMI (Pressman, 2010) 48
Tabla 2.11.- Categorías de MoProSoft y sus procesos (NYCE, 2016) 49
Tabla 2.12.- Nivel de capacidad de MoProSoft (NYCE,2016) 49
Tabla 2.13.- Factores en calidad de software de MCCALL (Pressman, 2010) 50
Tabla 2.14.- Descripción de las características FURPS+ (Vargas y Biagioli, 2009) 51
Tabla 2.15.- Características de calidad interna y externa de la ISO/IEC FDIS 9126- 2 y 3 (Elaboración propia basada en la ISO 9126) 53
Tabla 2.16.- Características de calidad de la ISO/IEC 25010 (Elaboración propia basado en ISO 25010) 54
Tabla 3.1.- Métricas de funcionalidad (Elaboracion propia basada en ISO 9126,2000) 78
Tabla 3.2.- Métricas de confiabilidad (Elaboracion propia basada en ISO 9126,2000) 79
Tabla 3.3.- Métricas de usabilidad (Elaboracion propia basada en ISO 9126,2000) 80
Tabla 3.4.- Métricas de eficiencia (Elaboracion propia basada en ISO 9126,2000) 81
Tabla 3.5.- Métricas de mantenibilidad (Elaboracion propia basada en ISO 9126,2000) 81
Tabla 3.6.- Métricas de portabilidad (Elaboracion propia basada en ISO 9126,2000) 82
Tabla 3.7.- Checklist de Funcionalidad (Elaboración propia) 96
Tabla 3.8.- Checklist de Confiabilidad (Elaboración propia) 97
Tabla 3.9.- Checklist de Usabilidad (Elaboración propia) 98
Tabla 3.10.- Checklist de Eficiencia. (Elaboración propia) 99
Tabla 3.11.- Checklist de mantenibilidad. (Elaboración propia) 100
Tabla 3.12.- Checklist de portabilidad. (Elaboración propia) 101
Tabla 4.1.- Sprintss de Scrum clasificados en módulos y requerimientos (Elaboración propia). 106
Tabla 4.2.- Datos generales de evaluación del módulo AES128 (Elaboración propia). 110
Tabla 4.3.- Plan de pruebas del módulo AES 128 (elaboración propia). 110
Tabla 4.4.- Datos Generales del módulo CRC (Elaboración propia). 124
Tabla 4.5.- Plan de pruebas del módulo CRC (Elaboración propia). 124
Tabla 4.6.- Datos generales de evaluación del módulo Web Service (Elaboración propia) 134
Tabla 4.7.- Plan de pruebas para el módulo Web Service (Elaboración propia). 134
Tabla 4.8.- Datos generales de evaluación del módulo Servicio Windows (Elaboración propia). 140
Tabla 4.9.- Resultados del servicio de Windows (Elaboración propia). 140
Tabla 4.10.- Casos de prueba para el Thread pool (Elaboración propia). 141

Tabla 4.11.- Datos generales de evaluación del módulo de Base de Datos (Elaboración propia).....142
Tabla 4.12.- Plan de pruebas de la base de datos (Elaboración propia). 143
Tabla 4.13.- Datos generales de evaluación del módulo de interfaces (Elaboración propia)..148
Tabla 4.14.- Plan de Pruebas de Interfaces (Elaboración propia). 148
Tabla 4.15.- Resultados al módulo de encriptación AES 12 (Elaboración propia).....160
Tabla 4.16.- Resultados del módulo CRC (Elaboración propia).....162
Tabla 4.17.- Resultados de la web service (Elaboración propia). 164
Tabla 4.18.- Resultados del módulo servicio Windows (Elaboración propia).....166
Tabla 4.19.- Resultados del módulo de base de datos (Elaboración propia).....168
Tabla 4.20.- Resultados del módulo de interfaces (Elaboración propia).....170
Tabla 4.21.- Total de casos de pruebas y pruebas ejecutadas en módulos (Elaboración propia). 172

Índice de figuras

Figura 1.1.- Costo en dólares es relativo al corregir errores y defectos (pressman,2010) 2
Figura 1.2 simulación del sistema Advanced Metering Infraestructure (AMI) (El financiero, 2015).....8
Figura 1.3.- Modelo de funcionamiento de Leviton (Leviton, 2017).....9
Figura 1.4.- Modelo de Enterprise Energy Management (EEM) (Leviton, 2017). 10
Figura 1.5.- Modelo de funcionamiento de Efergy (Minue, 2013). 11
Figura 1.6.- Graficas de consumo eléctrico con Efergy (Minue, 2013). 12
Figura 1.7.- Dispositivo de ODEnergyHome (Santamaria, 2012). 12
Figura 1.8.- Grafica de consumo eléctrico con EnviR (Corrales, 2012) 13
Figura 1.9.- Características del dispositivo EnviR (Corrales, 2012). 13
Figura 1.10.- Esquema del sistema AMI. (ENERI, 2013).....14
Figura 1.11.- Servicios que brinda el sistema AMI (ENERI, 2013). 15
Figura 2.1.- Modelo Waterfall (Pressman, 2010).....21
Figura 2.2.- Modelo de desarrollo rápido de aplicaciones DRA (Elaboración propia).....23
Figura 2.3.- Ciclo de vida de Scrum (Lara, 2015).....25
Figura 2.4.- El equipo de Scrum consiste en tres diferentes roles (Elaboración propia).....26
Figura 2.5.- Prueba de caja negra (elaboración propia).....35
Figura 2.6.- Prueba de caja blanca (elaboración propia) 37
Figura 2.7.- Modelo V. (Pressman, 2010). 40
Figura 2.9.- Tecnologías en Smart cities (Elaboración propia).....57
Figura 2.10 Advanced Metering Infrastructure Networks and Components (Brunschwiler, 2013).....58
Figura 2.11.- Modelo-Vista-Controlador (Flores, Medina, Hernández y Sánchez, 2017).....60
Figura 2.12.- Arquitectura Cliente-Servidor (Elaboración propia) 62
Figura 3.1.- Esquema de la metodología VEyVAA (Elaboración propia).....68
Figura 3.2.- Esquema del funcionamiento de la verificación (Elaboración propia).....71
Figura 3.3.- Proceso de aplicación de pruebas a módulo (Elaboración propia). 72
Figura 3.4.- Análisis y creación del plan de pruebas para la verificación (Elaboración propia). 74
Figura 3.5.- Interacción entre el equipo de scrum y el tester interno (Elaboración propia).....76
Figura 3.6.- Modelo de calidad interna y externa (Elaboración propia en base a ISO/IEC FDIS 9126-1:2000) 77
Figura 3.7.- El proceso de aplicación de pruebas (Elaboración propia).....84
Figura 3.8.- Proceso de aplicación de pruebas a módulo (Elaboración propia). 85
Figura 3.9.- Elaboración del plan de pruebas para la validación del software (Elaboración propia).....86
Figura 3.10.- Proceso de pruebas de aceptación del tester externo (Elaboración propia).....87
Figura 3.11.- Interacción de autores de Scrum con el tester externo (Elaboración propia). 89
Figura 3.12.- Formato del plan de pruebas (Elaboración de pruebas).....90
Figura 3.13.- Formato de resultados de pruebas de verificación pagina 1 (Elaboración propia). 91

Figura 3.14.- Formato de resultados de pruebas de verificación pagina 2 (Elaboración propia).....92
Figura 3.15.- Formato de resultados de pruebas de verificación pagina 3 (Elaboración propia). 93
Figura 3.16.- Formato de resultados de pruebas de validación Pagina 1(Elaboración propia). 94
Figura 3.17.- Formato de resultados de pruebas de validación pagina 2 (Elaboración propia).95
Figura 4.1.- Esquema del Head End System (Flores, Medina, Hernández y Sánchez, 2017). 104
Figura 4.2.- MVC en funcionamiento con Head End System (Elaboración propia).....105
Figura 4.3.- Clasificación de tecnologías por capa MVC (Elaboración propia). 105
Figura 4.4.- Herramienta y entorno a utilizar en el módulo AES128 (Elaboración propia). ..112
Figura 4.5.- Cadena con caracteres (Elaboración propia). 113
Figura 4.6.- Cadena Mayúscula y minúscula. (Elaboración propia) 113
Figura 4.7.- Cadena con números (Elaboración propia).....114
Figura 4.8.- Cadena con signos (Elaboración propia). 114
Figura 4.9.- Cadena con campo vacío (Elaboración propia). 114
Figura 4.10.- Cadena sin elementos (Elaboración propia). 115
Figura 4.11.- Cadena con valor cero (Elaboración propia). 115
Figura 4.12.- Cadena con 7093 caracteres (Elaboración propia). 116
Figura 4.13.- Comparación de cadenas (Elaboración propia). 116
Figura 4.14.- Cadena con combinación de caracteres (Elaboración propia). 116
Figura 4.15.- Comparativa de cadenas (Elaboración propia). 117
Figura 4.16.- Cadena de 12221 caracteres combinados (Elaboración propia). 117
Figura 4.17.- Comparativa de grandes cadenas (Elaboración propia).....117
Figura 4.18.- Cadena de 12286 caracteres incluyendo espacios vacíos (Elaboración propia). 118
Figura 4.19.- Comparativa de dos cadenas grandes combinadas (Elaboración propia). 118
Figura 4.20.- Cadena con doble comilla (Elaboración propia). 119
Figura 4.21.- Cadena con diagonal inversa (Elaboración propia). 119
Figura 4.22.- Cadena con salto de línea (Elaboración propia). 120
Figura 4.23.- Cadena con retorno de carro (Elaboración propia). 120
Figura 4.24.- Cadena con tabulador (Elaboración propia). 121
Figura 4.25.- Cadena con BackSpace (Elaboración propia).....121
Figura 4.26.- Cadena con form feed (Elaboración propia).....122
Figura 4.27.- Cadena con comilla simple (Elaboración propia).....122
Figura 4.28.- Cadena con secuencia de escape (Elaboración propia). 123
Figura 4.29.- Herramientas y entornos para ejecutar pruebas para CRC (Elaboración propia). 125
Figura 4.30.- Generación de un frame desde EOLO (Elaboración propia). 126
Figura 4.31.- Ejecución del CRC en consola (Elaboración propia). 126
Figura 4.32.- Se genera un frame desde EOLO sin espacios (Elaboración propia). 127
Figura 4.33.- Se genera frame sin espacios en CRC y coincide con EOLO (Elaboración propia).....127
Figura 4.34.- Se genera un frame desde EOLO con espacios múltiples (Elaboración propia). 128

Figura 4.35.- Se genera frame con espacios en CRC y coincide con EOLO (Elaboración propia).....128
Figura 4.36.- Se genera un frame desde EOLO de comienzo de paquete (Elaboración propia). 128
Figura 4.37.- Ejecución de cadena mayúscula, minúscula y combinada (Elaboración propia). 129
Figura 4.38.- Paquete con carácter invalido (Elaboración propia). 130
Figura 4.39.- Paquete corrompido (Elaboración propia). 130
Figura 4.40.- Excepciones de CRC ante alguna falla (Elaboración propia). 131
Figura 4.41.- Excepción de CRC ante un frame corrompido (Elaboración propia). 131
Figura 4.42.- Alertas de CRC ante algún fallo (Elaboración propia). 132
Figura 4.43.- Alertas de CRC ante algún paquete corrompido (Elaboración propia). 132
Figura 4.44.- Módulo de web service en ejecución en eclipse (Elaboración propia). 135
Figura 4.45.- Colocación de archivo de petición de algún servidor externo (Elaboración propia).....135
Figura 4.46.- Carga del servicio de la web service (Elaboración propia). 136
Figura 4.47.- Petición guardado en carpeta asignada para web service (Elaboración propia). 136
Figura 4.48.- Archivo de tareas (petición) con la estructura correcta (Elaboración propia). ..137
Figura 4.49.- Esquema para validar archivos XSD por web service (Elaboración propia).....137
Figura 4.50.- Validación y guardado de archivo XML (Elaboración propia). 137
Figura 4.51.- Sugerencia de la web service para revisar el registro de errores (Elaboración propia).....138
Figura 4.52.- Registro de errores (HesActivityLog.txt) en la carpeta ArchivosD (Elaboración propia).....138
Figura 4.53.- Visualización de un registro de errores (Elaboración propia). 138
Figura 4.54.- Conexión y guardado de archivo de petición en Hibernate (Elaboración propia). 139
Figura 4.55.- Conexión a la base de datos de oracle (hibernate) (Elaboración propia).....143
Figura 4.56.- Consulta a la tabla task. (Elaboración propia). 144
Figura 4.57.- Consulta a la tabla device (Elaboración propia). 144
Figura 4.58.- Consulta a la tabla Attribute (Elaboración propia). 145
Figura 4.59.- Consulta a la tabla attribute (Elaboración propia). 145
Figura 4.60.- Consulta a la tabla dotask (Elaboración propia). 145
Figura 4.61.- Consulta a la tabla device_attribute (Elaboración propia). 146
Figura 4.62.- Consulta a la tabla Device_attribute (Elaboración propia). 146
Figura 4.63.- Consulta a la tabla task_attribute (Elaboración propia). 146
Figura 4.64.- Consulta a la tabla Task_device (Elaboración propia). 147
Figura 4.65.- Interfaz de alarmas (Elaboración propia).....150
Figura 4.66.- Interfaz de conexión y desconexión de medidores (Elaboración propia). 150
Figura 4.67.- Usuario Administrador (Elaboración propia). 150
Figura 4.68.- Interfaz de escalabilidad (tiene interfaz disponible) (Elaboración propia). 151
Figura 4.69.-Menú principal (elaboración propia) 151
Figura 4.70.- Mensaje de alerta cuando se intenta cancelar una operación (Elaboración propia). 151

Figura 4.71.- Interfaces responsivas (elaboración propia). 152
Figura 4.72.- Servicio disponible para consumir (Elaboración propia). 153
Figura 4.73.- Sistema montado en el servidor (Elaboración propia).....153
Figura 4.74.- Advertencia de usuario no autorizado (Elaboración propia). 154
Figura 4.75.- Validacion de login (Elaboración propia).....154
Figura 4.76.- Menú principal (Elaboración propia).....155
Figura 4.77.- Interfaz de alarmas (Elaboración propia).....155
Figura 4.78.- Interfaz del archivo XML a seleccionar en una ruta local (Elaboración propia). 155
Figura 4.79.- Interfaz del archivo XML seleccionado es agregado al sistema (Elaboración propia).....156
Figura 4.80.- Interfaz para configurar NIC´S (Elaboración propia).....156
Figura 4.81.- Interfaz para crear nuevos Jobs (Elaboración propia). 156
Figura 4.82.- Uso de calendario eficiente (Elaboración propia).....157
Figura 4.83.- Mensaje de alerta cuando un dato es requerido (Elaboración propia).....157
Figura 4.84.- Mensaje de alerta cuando un medidor es invalido (Elaboración propia).....158
Figura 4.85.- Interfaz de registro y de modificación de NIC (Elaboración propia). 158
Figura 4.86.- Interfaz de conexión y desconexión de medidores (Elaboración propia).....158
Figura 4.87.- Interfaz sobre eventos y horarios (elaboración propia) 159
Figura 4.88.- Interfaz para agregar nuevos dispositivos (elaboración propia) 159
Figura 4.89.-Interfaz para agregar el acceso de un nuevo cliente (Elaboración propia) 159
Figura 4.90.- Interfaz del número de serie del cliente (RPU) (Elaboración propia) 159
Figura 4.91.- Resultado de pruebas a módulo AES128 (Elaboración propia). 161
Figura 4.92.- Resultados de pruebas a modulo CRC (Elaboración propia). 163
Figura 4.93.- Resultados de pruebas a web service (Elaboración propia).....165
Figura 4.94.-. Resultado de pruebas a servicio Windows (Elaboración propia). 167
Figura 4.95.- Resultado de pruebas a Base de datos (Elaboración propia). 169
Figura 4.96.- Grafica de resultados del módulo interfaces (Elaboración propia).....171
Figura 4.97.- Resultado de total de casos de fallos notificados, resueltos y validados (Elaboración propia).....172
Figura 4.98.- Resultados de evaluación de calidad ISO/IEC FDIS 9126 (Elaboración propia). 173

Capítulo 1

Introducción

Las empresas que ofrecen algún tipo de servicio o producto de software, hoy en día se encuentran envueltos en un mundo de competencia gracias al auge que existe en las distintas tecnologías dentro de la sociedad, donde un error potencial en sus servicios o productos, se corre el riesgo de afectar de manera irreversible tanto a los usuarios que depositan su confianza, como a la empresa al realizar un gasto imprevisto por solucionar los errores.

Debido a los riesgos que pueden surgir dentro de un proyecto de software, las empresas tienen la necesidad de implementar filtros de calidad en sus productos, que permita llevar el control de sus procesos de desarrollo y obtener softwares de buena calidad que cumpla con las necesidades de los clientes.

En la presente tesis se propone una metodología que asegure la calidad de un *Head End System (HES)*, el sistema gestiona todo el flujo de datos que genera un sistema de red de dispositivos inteligentes de medición de energía eléctrica, para ser procesados y enviados a un sistema de servicios públicos (*Utility*) donde presentara los datos para ser leídos por un usuario.

El *Head End System* fue solicitado por una empresa mexicana que se encarga de distribuir el servicio de energía eléctrica a todo el territorio mexicano, debido al alcance que tiene la distribución del servicio de electricidad, ha surgido una necesidad de llevar un control más específico en el consumo de energía eléctrica de cada medidor y la manipulación de manera remota de los dispositivos que pertenecen a la empresa.

La evaluación del *Head End System*, contiene un grado de complejidad debido a la magnitud de múltiples funciones que debe realizar el sistema, además el impacto social que tendrá al implementar dicho sistema sobre el consumo eléctrico en la sociedad, por esas razones se deben evaluar de una manera efectiva cada uno de sus funciones.

Pocas veces se da importancia en la calidad del software, por la errónea idea de evitar más gastos en el presupuesto o para ahorrar tiempo de entrega del producto, pero estas acciones pueden tener graves consecuencias, ya que existe el riesgo de desarrollar software defectuoso, como fallos durante su operación o que no realice lo deseado por el usuario, además de provocar un gasto enorme en la solución del problema.

Es común observar que la calidad es tomada como una fase más en el desarrollo (la de pruebas) y no como una característica que debe contener en todo el proceso de desarrollo, por lo que puede ser muy

arriesgado esperar hasta que el sistema se encuentre terminado para poder evaluarlo, porque si se llega detectar problemas graves pueden generar más gastos e incluso la cancelación del proyecto.

Debido a la importancia y responsabilidad que tienen las funciones del *Head End System*, se debe asegurar el funcionamiento correcto de sus operaciones antes de que sea implementado en campo, por lo que se propone aplicar un conjunto de acciones para verificar la calidad en fases tempranas y validar la calidad del sistema antes de entregarlo al cliente.

La propuesta para evaluar la calidad del software, es una metodología de aseguramiento de calidad de software (ACS) que nos permita evaluar desde el más mínimo componente que conforma el sistema hasta cuando el sistema se encuentre terminado.

La metodología de ACS se adapta al proceso de desarrollo de software ágil *Scrum* interviniendo en las fases de desarrollo de *sprints* evaluando los módulos de software y la fase de sistema terminado, de esta forma se tiene más control en la detección de errores y fallas para resolverlos a tiempo.

Los responsables de evaluar el sistema deberán obtener todo el conocimiento necesario entorno al *HES*, por ejemplo, los requerimientos del cliente, los detalles técnicos, así como las tecnologías, herramientas y metodologías que se ocuparán para su desarrollo y evaluación, esto con la finalidad de que el equipo ACS (*Testers*) puedan adaptar la aplicación de pruebas y revisiones de calidad.

Para identificar posibles fallos o defectos en el software, se pueden implementar una serie de filtros de calidad durante el desarrollo, con el objetivo de prevenir en fases tempranas cualquier situación que ponga en riesgo la calidad del software.

En ingeniería del software se sabe que es menos costoso una detección de fallas temprana que en una fase final cuando el sistema ya se encuentra terminado, siendo mucho más costoso si el software ya ha sido lanzado al mercado o entregado al cliente. En la figura 1.1 se muestra una gráfica sobre como los costos de corrección aumentan dependiendo de la fase de desarrollo en la que se detecta.

Figura 1.1.- Costo en dólares es relativo al corregir errores y defectos (pressman,2010)

La calidad puede intervenir de diferentes maneras durante todo el desarrollo del software, esas intervenciones pueden ser, por ejemplo: inspección de calidad, *testing* de software, control de desarrollo etc. Aunque signifique un gasto más dentro del presupuesto de desarrollo, al implementarlo puede asegurar la calidad del software y prevenir fallos en situaciones irreversibles.

La metodología ACS implementado en *Scrum* se propone la participación de dos tipos de *tester* uno que forma parte del equipo *Scrum* que se encargara de verificar y otro que en la parte final de desarrollo se encargue de validar al software terminado.

Las fases de scrum son flexibles al cambio y con agilidad en el desarrollo, por lo que la metodología debe adaptarse a esos principios de ser eficiente ante cualquier situación y no afectar el proceso de desarrollo, más al contrario debe ser un apoyo importante en la creación de software con buena calidad.

1.1 Planteamiento del problema

Debido al alcance que tiene la distribución del servicio de electricidad, ha surgido una necesidad de llevar un control más específico del consumo de energía eléctrica de cada medidor que pertenece a una compañía importante que distribuye el servicio de energía eléctrica en México, así como también detectar interrupciones de energía y solucionar el defecto de esa interrupción.

El sistema de gestión *Advanced Metering Infrastructure (AMI)* está conformado por 3 sistemas informáticos que se encuentran integrados para la gestión y procesamiento de los datos de consumo de energía eléctrica y de las tareas que realizan los medidores inteligentes. Esos sistemas son los siguientes:

- Sistema de red de medidores inteligentes, también llamado *System Object Network (SON).*
- Sistema de gestión de datos del medidor *Meter Data Management System (MDMS).*
- Sistema de control de flujo de datos, también llamado *Head End System (HES).*

Cada uno de ellos tiene una finalidad en su funcionamiento, el primero de ellos es la red de medidores que controlan el consumo de energía eléctrica de los consumidores de electricidad, este sistema es llamado *System Object Network (SON).*

El *Meter Data Management System (MDMS)* se encarga de procesar toda la información que se genera en la *SON* para ponerlo a disposición de los usuarios que gestionan y controlan el consumo de energía eléctrica.

El *Head End system (HES)* sirve como un puente de comunicación entre la información generada por una red de medidores inteligentes y de un sistema informático que gestiona a la *MDMS* estos sistemas generan las tareas que deben ser ejecutadas por los medidores inteligentes.

El *HES* gestiona todo el flujo de datos, para procesar y enviar las tareas correspondientes a la red de medidores inteligentes, la gestión de los datos funciona gracias a un conjunto de operaciones que realiza de manera interna. por lo que tiene mucha relevancia y responsabilidad en su funcionamiento.

Por la complejidad del *Head End System* es necesario asegurar su calidad antes de ponerlo a campo en funcionamiento, del cual debe evaluar todas las funciones que debe realizar el sistema, además debe cumplir con ciertas normas de la calidad del software.

Para evaluar la calidad del *HES*, es necesario desarrollar una metodología de aseguramiento de calidad, que permita evaluar en base a la verificación de las funcionalidades y el cumplimiento de requerimientos, así como la validación en base a características de calidad que deben estar implementadas en el sistema.

1.2 Justificación

La implementación de las tecnologías inteligentes en nuestra vida cotidiana ha evolucionado la forma de controlar nuestra vida social, así como realizar compras online hasta en la forma de realizar pagos por nuestros servicios públicos, y cada vez más las empresas ofrecen mejores servicios con el apoyo de tecnologías inteligentes, por lo que surge la necesidad de generar software confiable y de buena calidad.

Las empresas se preparan por las nuevas tendencias de ciudades inteligentes *(smart cities),* con nuevas propuesta e innovadoras herramientas para abastecer sus servicios de manera eficiente y mejorar la calidad de vida de millones de usuarios que en este caso es sobre la energía eléctrica.

El *Head End System (HES)* es una de las propuestas innovadoras que están ideadas para controlar la gestión de datos que generan los medidores inteligentes sobre el consumo eléctrico de millones usuarios, que controla la mayor empresa de servicio eléctrico en México.

Desarrollar el *HES* conlleva una gran responsabilidad ya que el producto será utilizado para resolver las mediciones de consumo de energía eléctrica de manera remota, por lo que se encontrará en constante funcionamiento y en caso de alguna medición errónea afectaría tanto al usuario como a la empresa. Es por ello que tiene una gran importancia que los softwares desarrollados cumplan con las funciones de manera correcta y efectiva.

Existen casos sobre los fallos que han surgido en sistemas informáticos que han causado afectaciones catastróficas a los usuarios, tales como perdidas de información, perdidas económicas o incluso pérdidas de vidas humanas, por lo que siempre es necesario aplicar todo tipo de pruebas a los sistemas informáticos para prevenir situaciones irreversibles.

Es así que se plantea una metodología que evalué la calidad del *Head End System* para evitar algún fallo que pueda afectar a los usuarios, además de verificar y validar que el sistema realice correctamente sus funciones.

1.3 Pregunta de investigación

¿Con la implementación de una metodología que permita la verificación y validación de calidad durante todo el desarrollo del software y sus componentes, se puede lograr el aseguramiento de calidad y crear un producto confiable que satisfaga las necesidades del usuario final?

1.4 Objetivo general

Diseñar e implementar una metodología que asegure la calidad del software en el *Framework* Scrum mediante la verificación de módulos funcionales en base a requerimientos del cliente y mediante la validación del sistema terminado en base a las características calidad interna y externa de la *ISO 9126.*

1.4.1 Objetivos específicos

- Clasificar los módulos en base a la arquitectura Modelo-Vista-Controlador (MVC) para su correcta evaluación.
- Diseñar y aplicar esquemas del proceso de evaluación para la verificación de módulos funcionales.
- Diseñar y aplicar esquemas del proceso de evaluación para la validación del sistema terminado.
- Analizar los requerimientos del cliente para crear los criterios de evaluación.
- Diseñar y aplicar plan de pruebas para evaluar la implementación de requerimientos por módulo.
- Diseñar y aplicar plan de pruebas para evaluar el sistema terminado basado en el estándar *ISO/IEC FDIS 9126.*
- Diseñar y aplicar plan de pruebas para evaluar al cliente y conocer el grado de satisfacción.
- Configurar las herramienta y entornos para la ejecución de pruebas.
- Detectar y notificar las posibles fallas del sistema para su pronta corrección.
- Evaluar la calidad del software para la toma de decisiones sobre el producto.

1.5 Alcances y limitaciones

La metodología *VEyVAA* (Verificación y Validación Ágil) se implementa dentro del desarrollo del *Head End System* en base al *Framework Scrum,* donde se enfoca a evaluar el sistema por medio de la verificación y validación del software.

La verificación se implementa en la evaluación de módulos del sistema en base a requerimientos técnicos y la validación se enfoca en evaluar la calidad y el cumplimiento de los requerimientos del cliente todo el proceso se lleva acabo de manera ágil.

La metodología tiene la finalidad de adaptarse en modelos de desarrollo que permitan dividir un sistema en módulos funcionales para su fácil evaluación y acepte la evaluación de calidad de la *ISO 9126.*
La metodología permite realizar pruebas a módulos funcionales, pruebas de sistema terminado, pruebas de aceptación y revisiones de calidad.
Los procesos de evaluación es detectar y notificar los errores, fallos y defectos para su pronta corrección, además de evaluar el porcentaje de calidad contenida en el sistema y la satisfacción del cliente.

1.6 Estado del arte

Siempre es importante saber el origen del proyecto, conocer las razones que dieron a desarrollar un nuevo sistema informático, además de conocer toda información suficiente que envuelve el proyecto.

En este apartado se realiza una investigación sobre los sistemas que existen para medir el consumo eléctrico que han sido desarrollados por empresas extranjeras, este con el fin analizar las funciones, los beneficios a los usuarios y los dispositivos que se utilizan.

Por otro lado, también se hace una investigación y análisis sobre propuestas que sean implementado para evaluar la calidad de los productos de software y sobre los procesos de desarrollo de software, así como la certificación de empresas de calidad, con el objetivo de visualizar posibles soluciones y diseñar una nueva metodología de aseguramiento de calidad evaluar la calidad de un sistema.

1.6.1 Antecedentes

El comienzo de renovación de dispositivos comunes a dispositivos inteligentes ha permitido que la aplicación de software sean las soluciones más eficientes para el control del consumo eléctrico.
En la siguiente información del periódico digital la jornada de la UNAM hace referencia sobre el macro proyecto que se pretende desarrollar.

Roberto de la Mora director de Mercadotecnia, Innovación y Nuevos Negocios de Ho1a[1] señala que: “La instalación de los medidores inteligentes ofrecen múltiples beneficios para los habitantes del norte del país, ya que integran tecnología para la medición automática del consumo, de manera sencilla y transparente, aumentando la precisión de la facturación del servicio”. (Posada, 2016)

[1] Ho1a: es una compañía mexicana de telecomunicaciones y TI, originaria de Guadalajara, Jalisco; líder en la implementación de soluciones de tecnología para negocios.

La comisión federal de electricidad (CFE) desarrolla las bases para futuros negocios con la colocación de medidores inteligentes en todo el país, proyecto que concluirá hacia 2020 y le permitirá rentar la infraestructura que estos requieren para otros servicios, como agua potable, gas y alumbrado público. (Mendieta, 2015)

Siguiendo con información sobre el proyecto que se desarrollara por parte de "Ho1a", también se cuenta la participación de Tecnologías *EOS*[2], la empresa asigno parte del proyecto a *EOS TECH*[3] por lo que el sistema integral Advanced Metering Infrastructure (AMI) se está construyendo en las áreas de desarrollo de la empresa. El periódico "Milenio" nos da más datos relevantes.

En entrevista con Milenio, Roberto de la Mora, director general de "Ho1a", división de negocios de Megacable, afirmo que el proyecto en el que trabajan con la CFE, el cual tiene el nombre de Reducción de pérdidas no Técnicas, es de primer mundo, no solo para generar ahorros al evitar pérdidas a la empresa productiva del estado, sino para generar la oferta de servicios avanzados. (Mendieta, 2015)

El proyecto entrara en operación en enero de 2016 y "Ho1a" a innovación, con su socio Tecnologías EOS, hizo la oferta más conveniente por 8 millones 585 mil 881 dólares. (Mendieta, 2015)

El "financiero" también publica un artículo sobre el proyecto que ya se ha puesto en marcha, en este artículo da mucha información relevante tanto para la sociedad como para el equipo de desarrollo, ya que informa la intención que tiene CFE de controlar esta tecnología, ya que proporciona datos técnicos y tarifas que manejara CFE mediante esta tecnología.

Roberto de la Mora director de nuevos proyectos de "Hola" Innovación, empresa que instala medidores en el Valle de México explica un ejemplo del funcionamiento del sistema "Tú me das ese contrato y me das permiso que cuando yo necesite te desconecto el refrigerador nada más entre esas horas, y te doy una tarifa preferente y me ayudas a mí a re balancear mi carga". (Meana, 2015)

Una vez mostrado esta información sabremos hacia dónde va dirigido este mega proyecto, los beneficios que dará a las familias mexicanas, saber lo bueno que será este proyecto es un motivante para realizar un gran trabajo.

El objetivo del proyecto AMI es reducir los robos de la energía eléctrica y el control de todos los dispositivos inteligentes para ofrecer un servicio de calidad a sus consumidores.

---

[2] Tecnologías EOS: Son especialistas en medición eléctrica para servicios residenciales, comerciales e industriales; adquisición remota de datos; Infraestructura de Medición Avanzada (AMI).

[3] EOS TECH: es una empresa mexicana del sector eléctrico que tiene la misión de ser una de las Empresas líderes en Tecnología e Innovación de productos y servicios en su campo, orientada a la excelencia de calidad para mejorar la vida de los consumidores.

En la figura 1.2 se muestra un esquema que fue presentado en su página web del periódico el financiero, donde se explica cómo sería el funcionamiento del proyecto Advanced Metering Infraestructure (AMI) en campo, en el esquema se representan 3 grandes subproyectos que conforma el sistema AMI, el primero de ellos es la red mesh del cual implementa los medidores inteligentes, el MDMS que es un sistema que gestiona todas las operaciones que el usuario solicita y que debe llevar acabo la red mesh, por último el Head End System es un sistema que controla y valida todo el flujo de datos que genere el usuario y los medidores inteligentes.

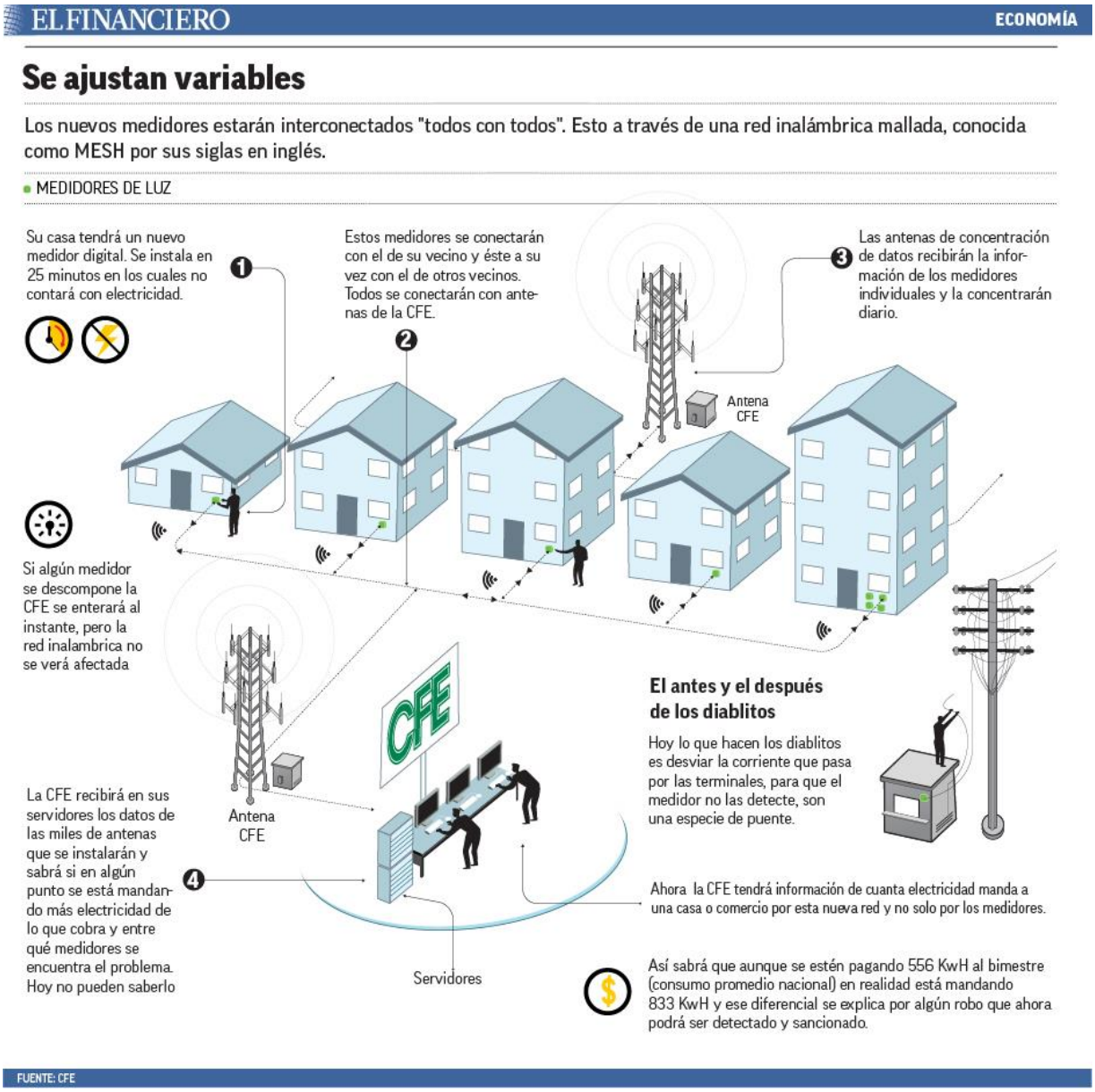


Figura 1.2 simulación del sistema Advanced Metering Infraestructure (AMI) (El financiero, 2015)

1.6.2 Sistemas de control de consumo de energía eléctrica en el mundo

El uso del medidor inteligente se ha expandido a la mayor parte del país, por lo tanto, se tiene una ventaja que se puede aprovechar en el uso de su tecnología, pero también es cada vez más complicado tener un control de cada uno de los medidores por la cantidad de dispositivos distribuidos en parte del país, por ello se han propuesto un proyecto de un sistema informático que controle la información de esos millones de medidores.

Ahora bien, se debe conocer si existen en el mercado productos de software similares o que alguna empresa utilice un software con características similares.

Se tiene historial de dispositivos que pueden realizar el control del consumo eléctrico, pero todos esos dispositivos son proyectos de otros países como Estados unidos, España, Francia Inglaterra entre otros. A continuación, se citarán algunos dispositivos y sus características que se comercializan en los países mencionados con anterioridad.

Leviton Manufacturing Company, Inc.

Es una empresa de estados unidos de mayor fabricación de cableado eléctrico. En la plataforma web que se puede tener acceso desde cualquier dispositivo que pueda entrar a la internet, en ella podrás visualizar el consumo de energía eléctrica, también podrás tener el control de lo que consumes.

En la figura 1.3 se muestra el modelo de funcionamiento de *Leviton*, básicamente se sostiene por medio de un servidor que administrara los medidores de levitón para llevar un control de información que será mostrada cuando el usuario lo solicite por medio de una página web.

Figura 1.3.- Modelo de funcionamiento de Leviton (Leviton, 2017).

El administrador de energía *Leviton*, una plataforma web avanzada que proporciona información en tiempo real para la empresa, brinda las herramientas más sofisticadas para impulsar eficiencias energéticas, reducir costos operativos y crear instalaciones más sostenibles y sólidas a nivel medioambiental. (Leviton, 2017)

De acuerdo a la página de L*eviton* (2017) el centro de monitorización de energía *Enterprise Energy Management (EEM)*, es un modelo que muestra cómo interactúan los dispositivos con el usuario, así como la comunicación de los medidores. Su modelo de EEM se muestra en la figura 1.4.

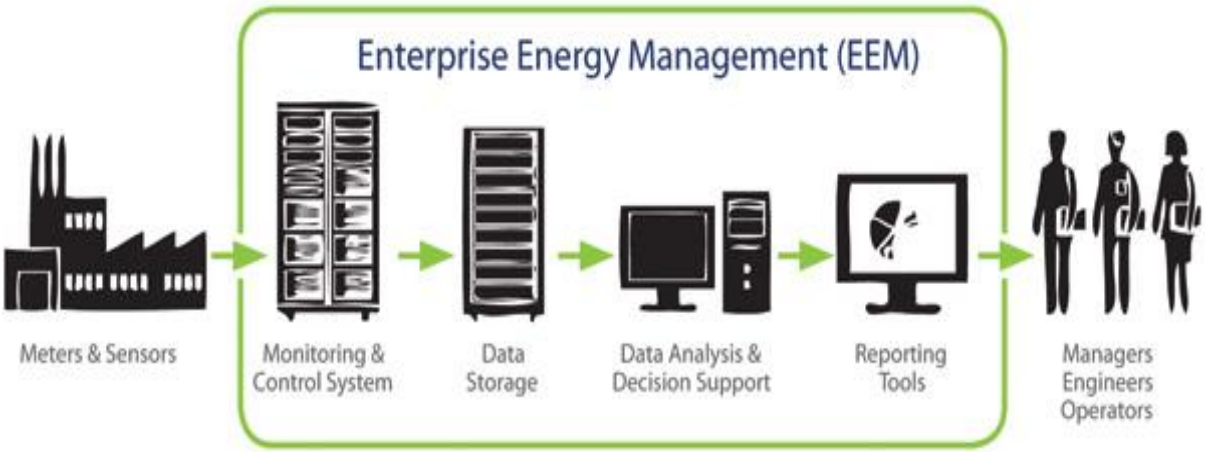


Figura 1.4.- Modelo de Enterprise Energy Management (EEM) (Leviton, 2017).

Los principales beneficios que ofrece los dispositivos de *Leviton* (2017) son:

- Fácil instalación.
- 8 entradas de pulso, ampliable a 30.
- Acepta todos los tipos de medidores.
- Compatible con el sistema de medición *Modbus.*
- Enchufar y listo: – no es necesario programar.
- Recopila datos de energía en tiempo real en intervalos de 15 minutos.
- Automáticamente envía los datos al centro de datos del administrador de energía.
- creación de informes.
- Informe en tiempo real.
- Informe diario.
- Informe semanal.
- Informe anual.
- Informe comparativo.
- Informe de emisiones de carbono.
- Informe del centro de energía.
- Informe del centro de energía.
- Informe de medidor múltiple.

Efergy

Efergy es una empresa británica que ofrece toda una gama de aparatos y servicios para el control del gasto eléctrico, desde el clásico monitor de enchufe hasta un completo sistema de monitorización con estadísticas y la posibilidad de acceder a la información remotamente desde el ordenador o el teléfono móvil. (Minue, 2013)

De acuerdo a Minue (2013), El *Engage e2 Hub Kit*, que incluye tanto un monitor *offline* como la mencionada posibilidad de consultar nuestro consumo a través de su plataforma web o su aplicación móvil para iOS y Android de las cuales se observa en la figura 1.5.

Según Minue (2013) Dentro de este kit se incluye lo siguiente:

- Un sensor, que se debe colocar en el cable de fase de nuestro cuadro eléctrico. No hace falta cortar ni hacer empalmes, es como un clip.
- Un transmisor, al que se conecta el sensor para enviar la información a la centralita. Se debe colocar cerca del cuadro eléctrico y funciona a pilas.
- Una centralita, que se conecta por ethernet a nuestro *router* y envía los datos al servicio *online*.
- Un monitor, que sirve para visualizar instantáneamente nuestro consumo, la media del día, de la semana, del mes, así como otros datos de interés.

Figura 1.5.- Modelo de funcionamiento de Efergy (Minue, 2013).

Sin embargo, lo más interesante de este kit es la opción de acceder gratuitamente a la plataforma *web Engage*, donde podremos consultar todos nuestros datos, que se han transformado en gráficas y valores fáciles de entender, no sólo del consumo eléctrico sino también de la cantidad de dinero que eso supone en nuestra factura, ver en la figura 1.6. (Minue, 2013)

Figura 1.6.- Graficas de consumo eléctrico con Efergy (Minue, 2013).

Odenergy home.

ODEnergyHome es un dispositivo electrónico que nos permite monitorizar el consumo de energía que se produce en un hogar en tiempo real, el dispositivo se puede observar en la figura 1.7, con este producto, podremos conocer el consumo de KW, CO2 y €. También detectar picos de consumo, anomalías o ineficiencias. Sin olvidar que, lo más interesante de todo, nos servirá para conocer de forma individualizada el consumo de cada aparato. (Santamaria, 2012)

Figura 1.7.- Dispositivo de ODEnergyHome (Santamaria, 2012).

Envir

Los monitores de energía nos permiten, de forma sencilla, saber cuánta electricidad estamos gastando en todo momento y de que aparato proviene dicho consumo.

El monitor de energía *EnviR* de *Current Cost* que se muestra en la figura 1.8, es uno de los más sofisticados y avanzados de los monitores de energía disponibles en el mercado, permitiendo un control minucioso del gasto energético. Este gestor energético inteligente, se está convirtiendo en una herramienta imprescindible para empresas, dónde el control del consumo se traduce en un gran ahorro. (Corrales, 2012)

Figura 1.8.- Grafica de consumo eléctrico con EnviR (Corrales, 2012)

El gestor energético *EnviR* incluye:

- Un monitor que le permite visualizar los datos sobre su consumo.
- Un transmisor que envía los datos de su consumo desde el cuadro eléctrico al monitor.
- Una pinza sensor para conectar el transmisor al cuadro eléctrico de manera simple.
- En la pantalla del monitor se puede visualizar en todo momento los siguientes datos y características que se muestran en la figura 1.9.
- Otra función única del *EnviR* es que es capaz de monitorizar datos de hasta 10 canales, por ejemplo, se puede controlar el consumo de una red eléctrica trifásica (3 canales) y 7 aparatos, cómo el termo eléctrico, la lavadora, la nevera, el lavavajillas, la televisión, el ordenador y la estufa eléctrica. O si así se desea, conectar otro transmisor con tres sensores y medir el consumo de dos líneas trifásicas (6 canales) y hasta 4 aparatos. (Corrales, 2012)

Figura 1.9.- Características del dispositivo EnviR (Corrales, 2012).

ENERI-AMI

Eneri es una empresa de Guadalajara México que brinda soluciones para la medición y administración de la energía eléctrica, a través de dispositivos de control y medición automatizada, desarrollada por la misma empresa.

La empresa ha desarrollado un sistema de Infraestructura Avanzada de Medición Inteligente *(AMI)* este sistema es "bidireccional de alta precisión para la medición de consumo de energía eléctrica basada en gabinete, con capacidad de lectura, corte y reconexión remota desde las oficinas de la empresa de distribución eléctrica para el propósito de la prevención del robo de energía". (*ENERI*, 2013)

En la figura 1.10 se puede observar el esquema de funcionamiento *AMI* del cual la infraestructura está conformado por 4 elementos clave; red de medidores Inteligentes, el concentrador, *MDMS* y el sistema automático de facturación de la compañía eléctrica.

Figura 1.10.- Esquema del sistema AMI. (ENERI, 2013)

En la página web de la empresa *ENERI* (2013) hace mención sobre el sistema *ESMS* (*Eneri Smart Metering System*) es una aplicación basada en arquitectura web que provee de un lugar centralizado para los datos de los medidores ya que permite en tiempo real o automatizado la recolección de datos de los medidores (diferentes variables eléctricas, alarmas), así como su administración (corte, reconexión remota), permitiendo de esta manera la disminución de cartera vencida, reducción de costo y tiempo en el proceso de facturación.

ESMS permite la integración con otros sistemas comerciales y/o de órdenes de trabajo de la eléctrica, por medio de protocolos estándar, web-service o archivos planos. Una de las fortalezas del *ESMS* es su diseño que cumple con los más altos estándares de seguridad cibernética para evitar intrusiones externas ajenas al sistema; la información está asegurada todo el tiempo. (*ENERI*, 2013)

En la figura 1.11 se muestran los servicios principales que ofrece el sistema *AMI* a las compañías de energía eléctrica, por ejemplo: corte y reconexión, balance de energía, interfaces, entre otros, además las funciones se realizan de manera remota por lo que sistema facilita el control de consumo de energía de los usuarios de la compañía.

Figura 1.11.- Servicios que brinda el sistema AMI (ENERI, 2013).

Es importante conocer las características y el funcionamiento de este tipo de sistemas, debido a que el caso de prueba de esta tesis es sobre un sistema que controla y gestiona el flujo de datos que arrojan medidores inteligentes de energía eléctrica por lo que es importante saber cómo controlan los datos por medio de dispositivos y plataformas web.

Aunque la mayoría de los sistemas son diseñados para el consumidor, el sistema *Eneri* tiene la atención debido a que va dirigido a las compañías que ofrecen el servicio eléctrico, por lo que ha sido interesante conocer su esquema de funcionamiento ya que tiene una gran similitud al *Head End System* (el caso de prueba de esta tesis).

1.6.3 Sistemas de aseguramiento de la calidad de software

En el área de desarrollo de software han surgido una variedad de propuestas que benefician a las compañías en la mejora de procesos, en la reducción de tiempos, en la reducción de costos, pero sobre todo en el aumento de calidad de sus productos, este último ha sido gracias a las diversas propuestas que se han diseñado para evaluar un software.

El aseguramiento de calidad es hoy en día fundamental para las empresas de desarrollo de software para lograr la calidad suficiente en sus procesos o en sus productos. Por lo tanto, las áreas de ingeniería de software se interesan en la implementación de nuevos métodos de aseguramiento de calidad para lograr los objetivos de sus proyectos.

A continuación, se presenta un conjunto de investigaciones sobre las nuevas formas y eficientes de evaluar un producto de software.

En la investigación se presenta un trabajo sobre la construcción de un modelo de predicción de fallas con la implementación de un conjunto de nuevas métricas de cambio (*LOC-worked-on*; líneas de código, *max-changeset*; conjunto de cambios, *avg-codechurn* valor promedio de código, entre otros) y de métricas de código, en algoritmos de aprendizaje automático, con el objetivo de mejorar el rendimiento de los modelos de predicción de fallas de software. En la aplicación del modelo se obtienen resultados positivos debido a las métricas de cambio que proporciona un rendimiento adicional en la predicción de fallas (Choudhary, et al., 2018).

> Los resultados de la clasificación muestran que las nuevas métricas proporcionan un importante aumento en la recuperación, es decir, aproximadamente un 10% sobre las métricas de códigos estáticos y aproximadamente un 23% sobre las métricas de cambios existentes. Incluso un pequeño aumento en el valor de recuerdo es crucial en términos de los recursos que se pueden guardar en una organización. El modelo sugerido de aprendizaje automático que usa métricas de códigos estáticos y métricas de cambio supera al modelo que usa solo métricas de código estático. (Choudhary, et al., 2018)

En la siguiente investigación se propone un modelo para la mejora de calidad de software en base a *Customer Knowledge Management (CKM)* que administra las aportaciones y conocimiento que proporciona el cliente a la empresa de desarrollo de software empresarial para obtener un producto de alta calidad que cumpla con las necesidades del cliente (Khosravi, et al., 2017).

Se desarrolló un modelo de evaluación de *CKM* y se obtuvieron resultados que permite conocer la importancia que tiene la participación y confianza del cliente, así como la infraestructura tecnología de *CRM* y cooperación interfuncional, el impacto de *CKM* en la calidad del software es significativo (Khosravi, et al., 2017)

> Los resultados confirmaron que los habilitadores de *CKM* se clasifican en los factores organizativos, humanos y tecnológicos que tienen un impacto efectivo en la *CKM*. La relación entre el *CKM* (dimensión de procesos) y la calidad del software (dimensión de resultados) es estadísticamente significativa. Por lo tanto, los resultados de este estudio confirmaron y respaldaron enérgicamente el Marco genérico de *CKM*. (Khosravi, et al., 2017)

En otro artículo se describe un marco de evaluación de aseguramiento de calidad de software en base a los procesos del estándar *IEEE 12207-2008* para ingeniería de software y aseguramiento de la calidad del software, el sistema *INTER CODAC* (control, acceso a datos y comunicación) es el responsable de implementar el estándar y otras normas de calidad para crear estrategias para llevar acabo los procesos de evaluación de software de alta calidad (Pande, et al.2013).

Un dato importante es la implementación de la verificación y validación para reunir evidencia sobre el rendimiento del proceso y la conformidad del producto, con la intención de registrar los datos del proceso para auditorías de calidad y en caso de ser necesario realizar mejoras en el proceso, cabe mencionar que el plan de validación y verificación del software *INTER CODAC* se basa en el Estándar 1012-2004 para *SVVP* (Plan verificación y validación de software) (Pande, et al.2013).

La implementación de modelos de calidad de software beneficia a las empresas interesadas en las múltiples opciones de evaluación como en el caso de anterior se menciona que "El caso de estudio de *CODAC SQA* tiene como objetivo en mejorar la confiabilidad y el ahorro de costos a través del descubrimiento temprano de problemas y la mantenibilidad del software". (Pande, et al.2013)

En este otro caso de estudio se enfoca en evaluar la calidad sobre sistemas de gestión de la información *(PLM, Product Lifecycle Management*) que integra datos, procesos, sistemas comerciales y empresariales. Para su evaluación se propone un Marco de evaluación de calidad *(QuEF)* que se ha utilizado para gestionar la calidad de metodologías de desarrollo web y actualmente trabaja en la gestión de requisitos de entidades (productos, procesos, servicios u organizaciones).
El objetivo de la implementación de *QuEF* es ayudar a las empresas elegir la solución *PLM* más útil para sus entornos particulares por lo que permite tomar decisiones en base a las necesidades de la empresa (Enríquez, J. G. Sánchez, J. M., Domínguez F. J., García, J. A., Escalona, M. J., 2018).

> Este marco describe plantillas para definir un modelo de calidad específico para el dominio en estudio. También ofrece un método para crear instancias del modelo de calidad, evaluarlo y calcular las preferencias de los elementos que lo forman. Además, el marco incluye la definición de un conjunto de fases para hacer cumplir la mejora continua de calidad en el modelo de calidad. El aspecto más importante es que la gestión de la calidad se centra en el modelo de calidad. Además, también se implementa un soporte de herramientas para promover esta solución en entornos reales. Por lo tanto, podemos tener una administración de calidad de manera automática utilizando *QuEF*, automatizando la gestión de calidad de las entidades para reducir los costos y el tiempo, y mejorando la calidad en el proceso de gestión de la calidad. (Enríquez, J. G. Sánchez, J. M., Domínguez F. J., García, J. A., Escalona, M. J., 2018)

La implementación de una certificación de calidad en una empresa proporciona un ambiente de confianza para el cliente y retos para los desarrolladores, con un objetivo en común por obtener un producto de software de alta calidad, pero también para mejorar los procesos que se aplican en la empresa.

Las certificaciones hoy en día para las empresas tienen altos costos o mantienen una rigurosa evaluación en sus procesos que pocas son las empresas que alcanzan a cumplir con todas las fases, mientras no se logre alcanzar las metas de certificaciones internacionales las empresas recurren a modelos y certificaciones que avalen el desempeño y calidad de los productos o servicios.

En la siguiente investigación se enfoca en la certificación de calidad de software que promueve y maneja las perspectivas de proceso y calidad del producto. También involucra la supervisión de calidad y la constante madurez en sus evaluaciones. Para la certificación no solo se enfoca en código fuente o en el sistema sino también en los artefactos de especificaciones de usuario y planeación (Darwish, 2016).

El trabajo de este articulo busca mejorar el esfuerzo en el proceso de certificación de software, particularmente al delinear un sistema para evaluar el software y determinar el estado de la calidad del software.

La certificación integra los atributos de calidad del proceso y del producto a través del dominio *fuzzy* basado en ACO. Dentro del enfoque de lógica difusa, el primer paso es transformar los valores discretos en uno continuo, este proceso se llama *fuzzificación.* las reglas semánticas difusas *IF-THEN* y se explica un procedimiento de *fuzzificación,* inferencia y *defuzzificación* que lleva a la decisión final del sistema de construir un modelo real de certificación de software. (Darwish, 2016)

El estudio de caso se llevó a cabo con una organización gubernamental (Universidad de Alejandría-Egipto) y uno de los sistemas en la universidad fue seleccionado para ser evaluado y certificado se obtuvieron resultados positivos. En general, la aplicación del estudio de caso ha demostrado la viabilidad y la practicidad del modelo de evaluación de software propuesto. El modelo proporciona información beneficiosa para los desarrolladores, propietarios y partes interesadas sobre el estado de calidad del sistema. (Darwish, 2016)

Conclusión

En este capítulo se logró establecer las metas a cumplir para comenzar a estructurar toda la información necesaria que nos ayudará a fundamentar la propuesta de solución sobre la evaluación de calidad de un sistema. Además, se realizó una investigación sobre casos que nos permite visualizar los posibles resultados, así como conocer las soluciones que se han propuesto en otras partes del mundo.

Capítulo 2

Marco teórico

Introducción

En el capítulo 2 se definen varios conceptos importantes que nos ayudan a comprender el trabajo de evaluación, además se describirán algunos modelos para el desarrollo de software y metodologías de pruebas, este con el fin de fundamentar la propuesta de solución en base a la información general y de trabajos existentes que nos pueden proporcionar datos reales y resultados reales.

Todo ello con el objetivo de diseñar un modelo de trabajo y una metodología de pruebas que nos ayude a asegurar la calidad del software y se pueda adaptar a la metodología de desarrollo ágil.

2.1 Modelos de desarrollo de software

Es de vital importancia saber la variedad de modelos de desarrollo de software para conocer los procesos y actividades que permiten crear un software de alta competitividad, además si se aprende a manejar las estructuras de software te da una amplia posibilidad de controlar cualquier proyecto de software.

Un proceso es un conjunto de actividades, acciones y tareas que se ejecutan cuando va a crearse algún producto del trabajo. Una actividad busca lograr un objetivo amplio (por ejemplo, comunicación con los participantes) y se desarrolla sin importar el dominio de la aplicación, tamaño del proyecto, complejidad del esfuerzo o grado de rigor con el que se usará la ingeniería de software. (Pressman, 2010)

EL modelo del proceso establece el fundamento para el proceso completo de la ingeniería de software por medio de la identificación de un número pequeño de actividades estructurales que sean aplicables a todos los proyectos de software, sin importar su tamaño o complejidad. (Pressman, 2010)

Ingeniería de software basada en componentes CBSE

La ingeniería de software basada en componentes *(CBSE, por las siglas de Component-Based Software Engineering)* surgió a finales de la década de 1990 como un enfoque al desarrollo de sistemas de software basado en la reutilización de componentes de software. (Sommerville 2011)

Sommerville (2011) menciona los fundamentos de la *CBSE* que son:

- Componentes independientes que se especifican por completo mediante sus interfaces. Debe existir una separación clara entre la interfaz del componente y su implementación. (Sommerville 2011)
- Estándares de componentes que facilitan la integración de éstos. Tales estándares se incrustan en un modelo de componentes. Definen, a un nivel mínimo, cómo deben

especificarse las interfaces de componentes y cómo se comunican estos últimos. Los componentes escritos en diferentes lenguajes pueden integrarse en el mismo sistema. (Sommerville 2011)

- Middleware que brinda soporte de software para integración de componentes. Para hacer que componentes independientes distribuidos trabajen en conjunto, es necesario soporte de middleware que maneje las comunicaciones de componentes. Además, el middleware para soporte de componentes puede brindar apoyo para la asignación de recursos, la gestión de transacciones, la seguridad y concurrencia. (Sommerville 2011)
- Un proceso de desarrollo que se engrana con la ingeniería de software basada en componentes. Se necesita un proceso de desarrollo que permita la evolución de requerimientos, dependiendo de la funcionalidad de los componentes disponibles. (Sommerville 2011)

Según Sommerville (2011) en la base de la *CBSE* existen firmes principios de diseño que apoyan la construcción de software comprensible y mantenible:

- Los componentes son independientes, de manera que sus ejecuciones no interfieren entre sí. Se ocultan los detalles de la implementación. La implementación de componentes puede cambiar sin afectar al resto del sistema. (Sommerville 2011)
- Los componentes se comunican a través de interfaces bien definidas. Si dichas interfaces se mantienen, es posible sustituir un componente por otro, lo que ofrece funcionalidad adicional o mayor. (Sommerville 2011)
- Las infraestructuras de componentes ofrecen varios servicios estándar que pueden usarse en sistemas de aplicación. Esto reduce la cantidad de código nuevo que debe desarrollarse. (Sommerville 2011)

Modelo en *waterfall*

El modelo *waterfall* también llamada de la cascada, a veces llamado ciclo de vida clásico, sugiere un enfoque sistemático y secuencial para el desarrollo del software, que comienza con la especificación de los requerimientos por parte del cliente y avanza a través de planeación, modelado, construcción y despliegue para concluir con el apoyo del software terminado. (Pressman 2010)

Aunque el modelo de la cascada propuesto originalmente por *Winston Royce* prevé los "bucles de retroalimentación", la gran mayoría de organizaciones que aplican este modelo de proceso lo tratan como si fuera estrictamente lineal. (Pressman 2010)

Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida. (Pressman 2010)

De esta forma, cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo. El progreso fluye de arriba hacia abajo, como una cascada como se muestra en la figura 2.1.

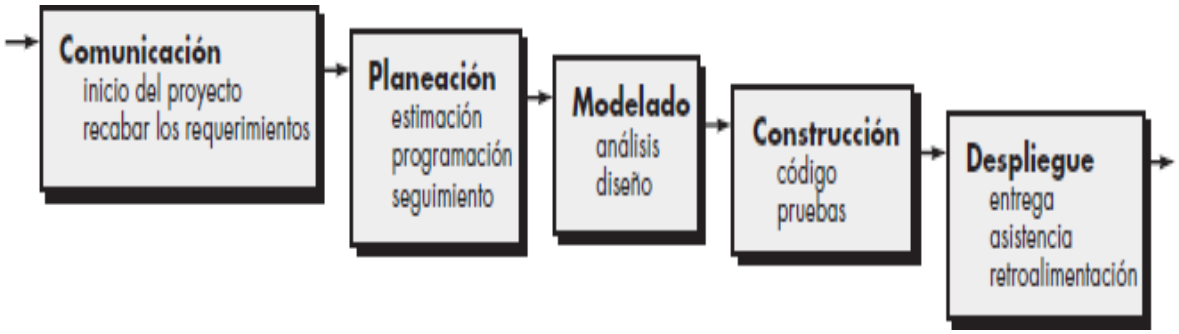


Figura 2.1.- Modelo Waterfall (Pressman, 2010).

En la tabla 2.1 se describen las fases del modelo waterfall este modelo clásico contiene la base de la ingeniería de software debido a que tiene las bases principales que debe contener un proceso de desarrollo de software.

Tabla 2.1.- Fases del modelo Waterfall (Pressman, 2010)

Fases waterfall	**Descripción**
Comunicación	Es la fase de inicio del proyecto donde existe una constante comunicación con el cliente para recabar los requerimientos que se deben cubrir en el desarrollo de software.
Planeación	En esta fase se realizan las estimaciones necesarias para conocer la factibilidad del proyecto en base a los requerimientos del cliente y se realiza una programación de actividades y seguimiento.
Modelado	En esta fase se realiza un análisis de los requerimientos funcionales y no funcionales del cliente y técnicos, de las cuales se deben reflejar como primera instancia en diagramas o bocetos que muestren de una manera abstracta el software.
Construcción	En la fase se comienza a desarrollar el software solicitado y se aplican pruebas cuando ya lo tengan terminado.
Despliegue	En esta fase se realiza el proceso de entrega al cliente cuando el software ha sido liberado y se da continuidad durante el uso con el usuario y en caso de alguna mejora o correcciones se aplica una fase mantenibilidad.

Las siguientes ventajas y desventajas se retoman del libro de Pressman 2010 un enfoque practico, del cual el autor describe el modelo waterfall.

Ventajas

- Realiza un buen funcionamiento en equipos débiles y productos maduros, por lo que se requiere de menos capital y herramientas para hacerlo funcionar de manera óptima.
- Es un modelo fácil de implementar y entender por lo que es utilizado con frecuencia.
- Está orientado a documentos.
- Promueve una metodología de trabajo efectiva: Definir antes que diseñar, diseñar antes que codificar.

Desventajas

- Es común que en el desarrollo de un software rara vez sigue una secuencia lineal, esto crea una mala implementación del modelo, lo cual hace que lo lleve al fracaso.
- El proceso de creación del software tarda mucho tiempo ya que debe pasar por el proceso de prueba y hasta que el software no esté completo no se puede implementar.
- Cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo.
- Una etapa determinada del proyecto no se puede llevar a cabo a menos de que se haya culminado la etapa anterior.

Modelo *Rapid Application Development (RAD)*

Él modelo RAD se originó en los enfoques de prototipado rápido y fue formalizado por primera vez por James Martin (1991), quien se refería a un ciclo de vida de desarrollo diseñado para sistemas de alta calidad con un desarrollo más rápido y menores costos que el ciclo de vida tradicional.

> A mediados de la década de los 90s, la definición de RAD se utilizó como un término general para abarcar una cantidad de métodos, técnicas y herramientas de muchos proveedores diferentes que aplican su propia interpretación y enfoque. Esta evolución ad hoc no estructurada e improvisada de RAD significa que la razón detrás de su uso no siempre es clara. (Berger, H., Beynon-Davies, P., y Cleary, P.,2004)

Segun Maner (1997) define al modelo RAD como un proceso de desarrollo de software que permite la construcción de sistemas utilizables en tan solo 60-90 días, a menudo con algunos compromisos. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso RAD permite al equipo de desarrollo crear un “sistema completamente funcional” dentro de periodos cortos de tiempo.

Cuando se utiliza principalmente para aplicaciones de sistemas de información, el enfoque *RAD* se muestra en la figura 2.2 comprende las fases que se describen en la tabla 2.2.

Tabla 2.2.- Fases del modelo RAD y su descripción. (Elaboración propia)

Fases DRA	Descripción
Modelado de gestión	El flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el proceso de gestión? ¿Qué información se genera? ¿Quién la genera? ¿A dónde va la información? ¿Quién la proceso?
Modelado de datos	El flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.
Modelado de proceso	Los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir, o recuperar un objeto de datos. Es la comunicación entre los objetos.
Generación de aplicaciones	En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario).
Pruebas de entrega	Como el proceso *DRA* enfatiza la reutilización, ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas. Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.

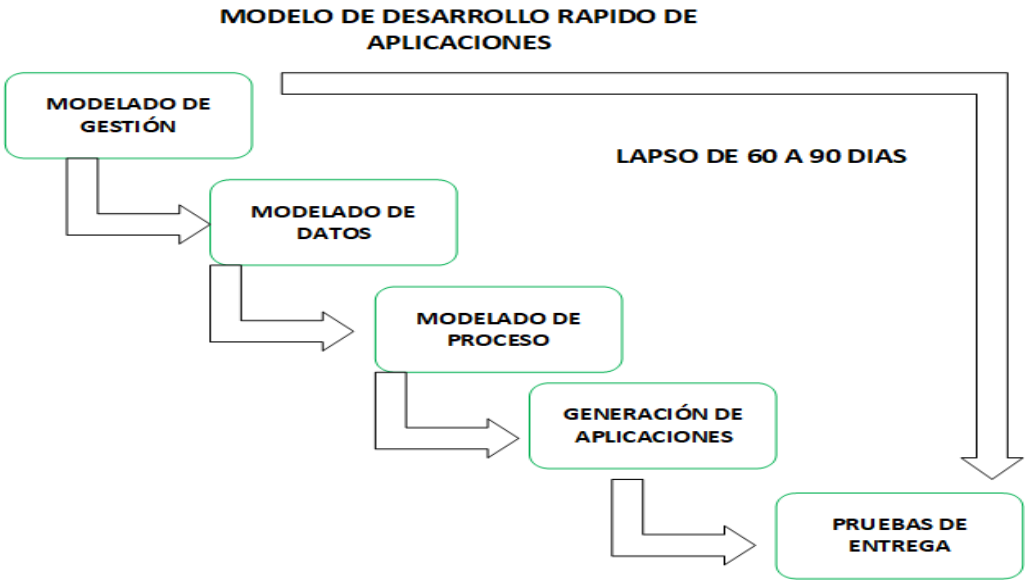


Figura 2.2.- Modelo de desarrollo rápido de aplicaciones DRA (Elaboración propia).

No todos los tipos de aplicaciones son apropiados para *RAD*. Si un sistema no se puede modularizar adecuadamente. La construcción de los componentes necesarios para *RAD* será problemática.

Segun Maner (1997) realiza un listado de las ventajas y desventajas del modelo RAD

Ventajas de RAD

- Entregables a veces más fáciles de portar (porque hacen un mayor uso de abstracciones de alto nivel, scripts, código intermedio)
- Visibilidad temprana (debido a la creación de prototipos)
- Mayor flexibilidad (porque los desarrolladores pueden rediseñar casi a voluntad)
- Codificación manual muy reducida (debido a los asistentes, generadores de código, reutilización de código)
- Mayor participación del usuario (porque están representados en el equipo en todo momento)
- posiblemente un costo reducido (porque el tiempo es dinero, también debido a la reutilización)
- Ciclos de desarrollo más cortos (porque el desarrollo se inclina hacia el cronograma y lejos de economía y calidad)

Desventajas de RAD

- Costo de conjunto de herramientas y hardware integrados para ejecutarlo
- Más difícil de medir progreso (porque no hay hitos clásicos)
- Menos eficiente (porque el código no está hecho a mano)
- Pérdida de precisión científica (porque no se usan métodos formales)
- El prototipo puede no ampliarse, un problema grande
- Funciones reducidas (debido al timeboxing, reutilización del software)
- La dependencia de componentes de terceros puede sacrificar la funcionalidad necesaria
- agregar funcionalidad innecesaria
- crear problemas legales
- Los requisitos pueden no converger (porque los intereses de los clientes y los desarrolladores pueden divergir de una iteración a la siguiente)
- Esfuerzos exitosos difíciles de repetir (no hay dos proyectos que evolucionen de la misma manera)
- Características no deseadas (a través de la reutilización de componentes existentes).

Framework de desarrollo ágil Scrum

Scrum (nombre que proviene de cierta jugada que tiene lugar durante un partido de rugby) es un método de desarrollo ágil de software concebido por *Jeff Sutherland* y su equipo de desarrollo a principios de la década de 1990. En años recientes, Schwaber y Beedle (2001) han desarrollado más los métodos *Scrum*.

Los principios *Scrum* son congruentes con el manifiesto ágil y se utilizan para guiar actividades de desarrollo dentro de un proceso de análisis que incorpora las siguientes actividades estructurales: requerimientos, análisis, diseño, evolución y entrega. El ciclo de scrum se observa en la figura 2.3.

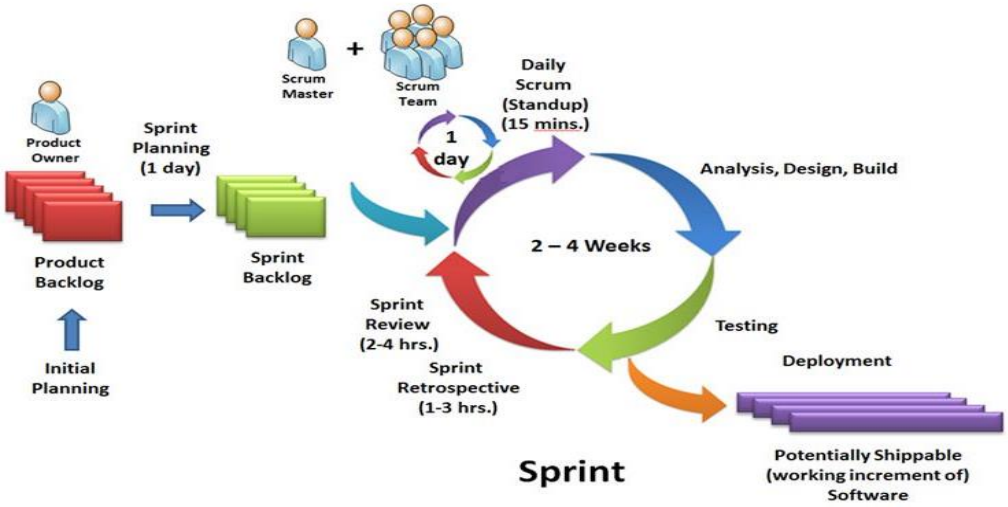


Figura 2.3.- Ciclo de vida de Scrum (Lara, 2015).

Dentro de cada actividad estructural, las tareas del trabajo ocurren con un patrón del proceso llamado *Sprint*. El trabajo realizado dentro de un *Sprint* (el número de éstos que requiere cada actividad estructural variará en función de la complejidad y tamaño del producto) se adapta al problema en cuestión y se define y con frecuencia se modifica en tiempo real por parte del equipo *Scrum*. (Pressman, 2010)

Scrum es un *framework* para trabajar en equipo en una serie de interacciones. Las fases en las que se divide y define un proceso de *Scrum* con sus respectivos responsables, las fases son las siguientes:

- ¿Quién? y el ¿Qué?: identifica los roles de cada uno de los miembros del equipo y define su responsabilidad en el proyecto. En la tabla 2.3 describe los roles que existen en Scrum y en la figura 2.4 se puntualiza cada rol.
- El ¿Dónde? y el ¿Cuándo?: que representan el *Sprint*
- El ¿Por qué? y el ¿Cómo?: representan las herramientas que utilizan los miembros de *Scrum*

Tabla 2.3.- Roles en scrum y su función (elaboración propia)

Roles en Scrum ¿Quién?	**Descripción ¿Qué?**
Product Owner	Dueño del producto es la "voz del cliente" y el responsable de desarrollar, mantener y priorizar las tareas en el *backlog*.
Scrum Master	Es responsable de asegurarse que el trabajo del equipo vaya bien siguiendo las bases de Scrum. Además, se encarga de remover cualquier obstáculo que pueda encontrar el equipo de desarrollo.
Development Team Members	Miembros del Equipo de desarrollo son los encargados de escribir y probar el código.
Stakeholders	Son las personas u organizaciones interesadas en el proyecto del software como puede ser el cliente y usuarios.

El *Sprint* es la unidad básica de trabajo para un equipo *Scrum*. Esta es la característica principal marca la diferencia entre *Scrum* y otros modelos para el desarrollo ágil. Es una simple iteración llevada a cabo por los miembros del equipo. Un equipo puede completar varios *Sprints* durante el desarrollo del proyecto. (Lara, 2014)

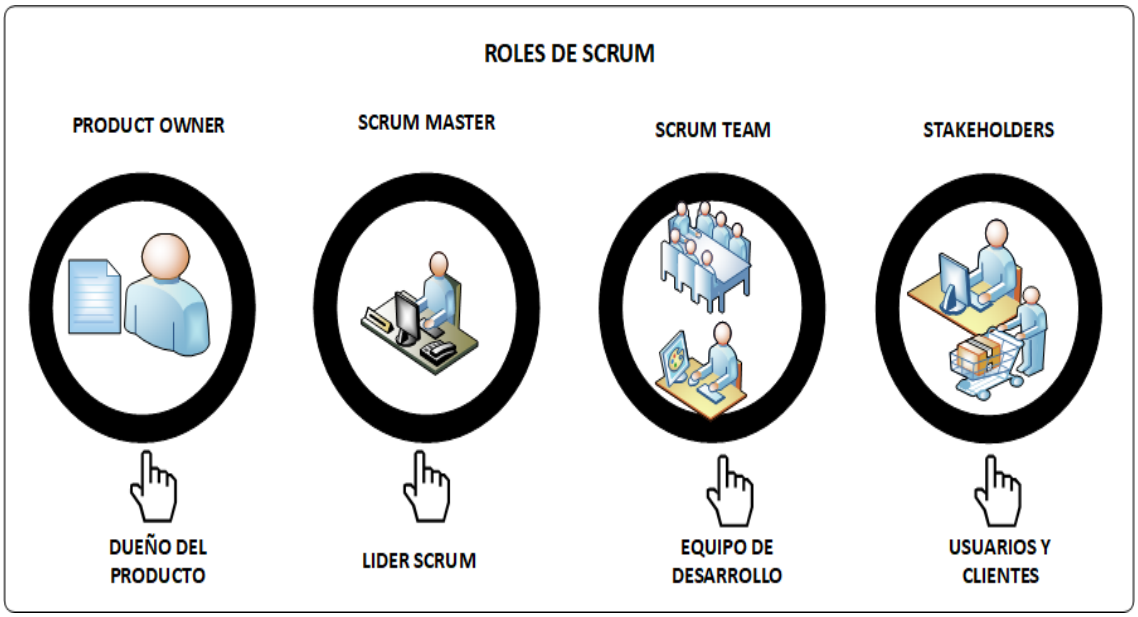


Figura 2.4.- El equipo de Scrum consiste en tres diferentes roles (Elaboración propia).

Un *Sprint* inicia con un equipo que se compromete a realizar el trabajo y finaliza con la demostración de un entregable. El tiempo mínimo para un *Sprint* es de una semana y el máximo es de 4 semanas.

Dentro del desarrollo de un *Sprint* se llevan a cabo ciertos eventos, estos reciben el nombre de *Scrum Events* o Eventos *Scrum*. Como se observa en la tabla 2.4 información tomada de Lara (2014).

Tabla 2.4.- Eventos de Scrum y su descripción (Lara, 2014)

Scrum Events	**Descripción**
Sprint Planning/plan del sprint	Todos los involucrados en el equipo se reúnen para planificar el Sprint. Durante este evento se decide qué requerimientos o tareas se le asignará a cada uno de los elementos del equipo. Cada integrante deberá asignar el tiempo que crea prudente para llevar a cabo sus requerimientos. De esta manera se define el tiempo de duración del Sprint
Scrum team meeting/ Reunión de Equipo de Scrum	Estas reuniones se deben realizar diariamente con un máximo de 15 minutos. Siempre en el mismo horario y lugar. En ellas, cada miembro del equipo deberá responder tres simples preguntas: • ¿Qué hiciste ayer? • ¿Qué tienes planeado hacer hoy? • ¿Qué obstáculos encontraste en el camino? Estas reuniones sirven para que todos los miembros del equipo se apoyen entre ellos. Si alguno de ellos tiene algún inconveniente que tome más tiempo del asignado en resolverse; este debe tratarse más a fondo en una reunión enfocada en buscar la mejor solución para ello.
Backlog Refinement/ Refinamiento del Backlog	El *Product Owner* revisa cada uno de los elementos dentro del *Product Backlog* con el fin de esclarecer cualquier duda que pueda surgir por parte del equipo de desarrolladores. También sirve para volver a estimar el tiempo y esfuerzo dedicado a cada uno de los requerimientos.
Sprint Review/ Revisión del Sprint	Los miembros del equipo y los clientes se reúnen para mostrar el trabajo de desarrollo de software que se ha completado. Se hace una demostración de todos los requerimientos finalizados dentro del Sprint. En este punto no es necesario que todos los miembros del equipo hablen, pueden estar presentes pero la presentación está a cargo del *Scrum Master* y el *Product Owner*.
Retrospective/ Retrospectiva del Sprint	En este evento, el *Product Owner* se reúne con todo su equipo de trabajo y su *Scrum Master* para hablar sobre lo ocurrido durante el *Sprint.* Los puntos principales a tratar en esta reunión son: • Qué se hizo mal durante el Sprint para poder mejorar el próximo • Qué se hizo bien para seguir en la misma senda del éxito • Qué inconvenientes se encontraron y no permitieron poder avanzar como se tenía planificado.

Las herramientas que se describen en la tabla 2.5 son útiles no sólo durante un *Sprint;* sino que ayudan a lo largo del proyecto, ya que permite al equipo a entender por qué hacen lo que están haciendo. Son visibles para cada uno de los miembros del equipo y para las personas que están fuera también.

Tabla 2.5.- Herramientas de Scrum y su descripción (elaboración propia)

Herramientas Scrum ¿Por qué?	**Descripción ¿Cómo?**
Product Backlog/Pila del producto	En general es una lista de Sprint´s priorizados para ser desarrollados en un corto tiempo y potencialmente entregables. El sprint puede ser un nuevo componente o puede ser corrección de un bug, una referencia o parte de un requerimiento.
User Stories/ Historias de Usuario	Es un elemento especial del product Backlog. Son llamados Historias porque en ellos se proporciona información sobre cómo debe ser el comportamiento del requerimiento que se está trabajando. De igual manera, proporciona información directa del cliente en caso de existir algún cambio.
Sprint Backlog	Es el conjunto de elementos tomados del *Product Backlog* que fueron priorizados, medidos y aceptados en las reuniones de *Sprint Planning.* Estos, en conjunto con sus respectivos *User Stories*, forman oficialmente los requerimientos a elaborar en cada uno de los *Sprint´s* que tendrá el proyecto.
The Taskboard/ el panel de tareas	El panel de tareas muestra todas y cada una de las tareas que tienen asignadas cada uno de los miembros del equipo. Esta tabla se divide en tres columnas que representan el estado de la tarea: 1. Por hacer 2. Haciendo 3. Terminado Al inicio del Sprint todas están en la primera columna. Al momento de pasar una tarea a la columna número dos, indicará al Scrum Master y al Product Owner qué está haciendo cada miembro del equipo y cuánto tiempo lleva trabajando en dicha tarea. Al finalizar la tarea, esta debe cambiarse a la última columna. Esto quiere decir que está listo para que QA haga las pruebas necesarias.
Definition of Done/Defunción de "listo"	Todo equipo eficaz y ágil tiene ciertos acuerdos que deben cumplirse antes de dar por finalizado un Proyecto. Estos son: • Todas las tareas están completas • Revisión de Código / *Code Reviewed* • Pruebas realizadas a cada elemento desarrollado • Revisión por parte de los clientes (que cumpla sus necesidades) • La revisión de las condiciones de Aceptación por parte del *Product Owner*.

Los principios del Manifiesto Ágil

Los principios del manifiesto ágil según Beck et al. (2001), presentan la filosofía para desarrollar un software de manera rápida y con un entregable eficiente para el cliente. A continuación, se enlista los principios dando origen al enfoque de desarrollo de software ágil:

- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.

- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

2.2 Pruebas de software

Las pruebas son una de las formas para evaluar el buen funcionamiento de un software, ya sea por componentes o por el sistema completo. Con las pruebas podremos conocer si el software cumple con los requerimientos del cliente o estándares de calidad.

Por esa razón es muy importante conocer los conceptos y variantes que contiene las pruebas, esto con la razón de comprender aún más las actividades de *testing.*

Según Dijkstra, (1970) “La prueba de software puede ser usada para mostrar la presencia de bugs, pero nunca su ausencia”.

En la *ANSI/IEEE 610.12* (1990) se define la prueba: “Es el proceso de operar un sistema o componente bajo condiciones específicas, observar o registrar los resultados, y realizar una evaluación de algún aspecto del sistema o componente.”

Las pruebas tienen diferentes objetivos dependiendo del enfoque de aplicación según Pressman (2010) cita en su libro “ingeniería de software: un enfoque práctico” a Glen Myers (1979) donde menciona que establece varias normas que pueden servir acertadamente como objetivos de las pruebas:

1. La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.
2. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
3. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

La prueba de software es un proceso que se enfoca sobre la lógica interna del software y las funciones externas. Es un proceso de ejecución de un programa con la intención de descubrir un error, no puede asegurar la ausencia de defectos; sólo puede demostrar que existen defectos en el software (Ecured, 2017)

Según Villareal, D., Gamboa, S., & Gómez, F. L. (2015). Se definen cuatro niveles de pruebas: pruebas de módulos, pruebas de integración, pruebas del sistema y pruebas de aceptación.

Un plan de prueba subraya los tipos de pruebas que se van a realizar y define casos de prueba específicos que se diseñan para garantizar que: se satisfacen todos los requerimientos de funcionamiento, se logran todas las características de comportamiento y de rendimiento, todo el contenido es preciso y se presenta de manera adecuada, y se satisfacen la facilidad de uso. (Pressman,2010).

El caso de prueba: Es un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y postcondiciones de ejecución, desarrollados con un objetivo particular o condición de prueba, tal como ejercitar un camino de un programa particular o para verificar que se cumple un requerimiento especifico (IEEE, 1990).

Procedimiento de prueba: Instrucciones detalladas para la configuración, ejecución y evaluación de los resultados para un caso de prueba determinado (IEEE, 1990).

Existen herramientas que apoyan en diversos aspectos de la prueba. A continuación, se presenta una clasificación de Pérez (2006) sobre algunas herramientas:

- **Administración de las pruebas y el proceso de pruebas:** Herramientas para la administración de las pruebas, para el seguimiento de incidentes, para la gestión de la configuración y para la administración de requerimientos.
- **Pruebas estáticas:** Herramientas para apoyar el proceso de revisión, herramientas para el análisis estático y herramientas de modelado.
- **Especificación de las pruebas:** Herramientas para el diseño de las pruebas y para la preparación de datos de prueba
- **Ejecución de las pruebas:** herramientas de ejecución de casos de prueba, herramientas de pruebas unitarias, comparadores, herramientas de medición del cubrimiento, herramientas de seguridad.
- **Desempeño y la monitorización:** Herramientas de análisis dinámico, herramientas de desempeño, de carga y de estrés, herramientas de monitorización Comprar o alquilar una herramienta no garantiza el éxito con ella. Cada tipo de herramienta requiere esfuerzo adicional para poder alcanzar ventajas a largo plazo.
- **Entre las ventajas de usar herramientas se encuentran:** reducir el trabajo repetitivo (por ejemplo, en las pruebas de regresión), mayor consistencia y capacidad de repetición, evaluación objetiva, facilidad para el acceso a la información de las pruebas.
- **Entre los riesgos de usar las herramientas, se encuentran:** expectativas poco realistas para la herramienta, subestimar el tiempo, costo y esfuerzo de inducción inicial en la herramienta, subestimar el tiempo y el esfuerzo necesarios para alcanzar ventajas significativas y continuas con la herramienta, subestimar el esfuerzo requerido para mantener los elementos de prueba generados por la herramienta, tener demasiada confianza en la herramienta sustituyendo todas las pruebas manuales.

Tipos de pruebas

Es importante como *tester* conocer las diferentes técnicas de pruebas para evaluar diferentes especificaciones, características o funcionalidades de un software. Debido a las múltiples funcionalidades que contiene un software existen una variedad de pruebas que nos ayuda a comprobar la buena calidad del software.

A continuación, se enlista algunas pruebas importantes para el *testing.*

- **Pruebas de condición:** La prueba de condición es un método de prueba de caja blanca en la que se ejercitan las condiciones lógicas contenidas en el módulo de un programa (Pressman, 2010).

- **Pruebas de estrés:** Estas pruebas someten al programa a cargas pesadas. Una carga pesada es un volumen máximo de datos, o de actividad, en un plazo corto de tiempo. La prueba de estrés implica como elemento el tiempo. (Pérez, 2006)
- **Pruebas de usabilidad *(Usability Testing)*:** Las pruebas de usabilidad consisten en comprobar la adaptabilidad del sistema a las necesidades de los usuarios, tanto para asegurar que se acomoda a su modo habitual de trabajo, como para determinar las facilidades que aporta al introducir datos en el sistema y obtener los resultados. Las pruebas de usabilidad están dentro de las pruebas de caja negra y en la categoría de las pruebas no funcionales. (Estrada, 2010)
- **Pruebas de seguridad *(Security Testing)*:** La prueba de seguridad intenta verificar que los mecanismos de protección que se construyen. (Pressman, 2010)
- Las pruebas de seguridad pretenden medir la confidencialidad, integridad y disponibilidad de los datos, desde la perspectiva del aplicativo, es decir, partiendo de identificar amenazas y riesgos desde el uso o interface de usuario final. (Estrada, 2010)
- **Pruebas de Desempeño:** Muchos programas tienen objetivos específicos de desempeño o de eficiencia, indicando características tales como tiempos de respuesta y rendimiento bajo ciertas condiciones de carga de trabajo y configuración. En la prueba de desempeño, los casos de prueba se deben diseñar para demostrar que el programa no satisface sus objetivos de desempeño. (Pérez 2006)
- **Pruebas de Recuperación:** La recuperación es una prueba del sistema que fuerza al software a fallar en varias formas y que verifica que la recuperación se realice de manera adecuada. Si la recuperación es automática (realizada por el sistema en sí), se evalúa el reinicio, los mecanismos de puntos de verificación, la recuperación de datos y la reanudación para correcciones. Si la recuperación requiere intervención humana, se evalúa el tiempo medio de reparación (TMR) para determinar si está dentro de límites aceptables. (Pressman, 2010)

Niveles de pruebas

Así como existen una variedad de técnicas de pruebas también se pueden identificar diferentes fases en el proceso de evaluación o aplicación de pruebas durante el ciclo de desarrollo de software o al termino de construcción del software. A continuación, se mencionan cada una de las fases de evaluación de un software.

- **Pruebas unitarias *(Unit Testing)*:**

  Se focaliza en ejecutar cada módulo (o unidad mínima a ser probada, ej. = una clase) lo que provee un mejor modo de manejar la integración de las unidades en componentes mayores. Busca asegurar que el código funciona de acuerdo con las especificaciones y que el módulo lógico es válido. (Abad, 2005)

  Algunas de las acciones que se realiza en este tipo de pruebas según Abad (2005) menciona lo siguiente:

  A. Particionar los módulos en pruebas en unidades lógicas fáciles de probar.
  B. Por cada unidad hay que definir los casos de prueba (pruebas de caja blanca).
  C. Para esto los casos de prueba deben diseñarse de forma tal que se recorran todos los caminos de ejecución posibles dentro del código bajo prueba; por lo tanto, el diseñador debe construirlos con acceso al código fuente de la unidad a probar.
  D. Los aspectos a considerar son los siguientes: Rutinas de excepción, Rutinas de error, Manejo de parámetros, Validaciones, Valores válidos, Valores límites, Rangos, Mensajes posibles. (Abad, 2005)

- **Pruebas de integración *(Integration Testing):*** El objetivo de las pruebas de integración es verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes. (Pérez, 2006)

  En las pruebas de integración incremental se combina el siguiente componente que se debe probar con el conjunto de componentes que ya están probados y se va incrementando progresivamente el número de componentes a probar.

  En las pruebas de Integración no incremental: Se prueba cada componente por separado y posteriormente se integran todos de una vez realizando las pruebas pertinentes. (Pérez, 2006)

- **Prueba del sistema:** Refiere al comportamiento del sistema entero. La mayoría de las faltas funcionales deben haber sido identificadas ya durante las pruebas de unidad e integración. La prueba del sistema generalmente se considera apropiada para probar los requerimientos no funcionales del sistema, tales como seguridad, desempeño, exactitud, y confiabilidad. Las interfaces externas, los dispositivos de hardware, o el ambiente de funcionamiento también se evalúan a este nivel. (Pérez, 2006)

- **Pruebas de Regresión:** Su objetivo es verificar que no ocurrió una regresión en la calidad del producto luego de un cambio, asegurándose que los cambios no introducen un comportamiento no deseado u errores adicionales. Implican la re-ejecución de alguna o todas las pruebas realizadas anteriormente (Pérez, 2006)

  Después que los *testers* reportan los defectos, los desarrolladores generan una nueva versión del software, en la cual los defectos supuestamente han sido removidos. La cuestión de fondo es: ¿Cuánta prueba de la versión n es necesario hacer usando las pruebas ejecutadas contra la versión n-1? (Pérez, 2006)

Cualquier arreglo de un problema detectado puede:

- Solucionar solo el problema que fue reportado.
- Fallar en solucionar el problema que fue reportado.
- Solucionar el problema, pero arruinar otra cosa que antes funcionaba.
- Fallar en solucionar el problema que fue reportado y romper otra cosa que antes funcionaba.
  Hay tres tipos de pruebas de regresión:
- Regresión de defectos solucionados: Cuando se reporta un defecto y vuelve una nueva versión que supuestamente lo soluciona, el objetivo es probar que no fue solucionado.
- Regresión de defectos viejos: Se prueba que un cambio en el software causó que un defecto que ya había sido solucionado vuelva a aparecer.
- Regresión de efectos secundarios: Implica volver a probar una parte del producto. El objetivo es probar que el cambio ha causado que algo que funcionaba ya no funciona.

Métodos de pruebas de software

Existen dos formas de realizar pruebas al software en dos enfoques diferentes y esas son:

1. La parte funcional o comportamiento del software (caja negra)
2. la parte interna del software a nivel código donde se revisa la estructura, y la lógica del sistema. (Caja blanca).

Existe una combinación de las dos formas de prueba al mismo tiempo llamada caja gris, pero eso dependerá de la necesidad de los módulos a probar. A continuación, se describirán con más detalles el funcionamiento de estas pruebas.

Pruebas de caja negra *(Black box testing)*

Este tipo de pruebas es también conocido como pruebas funcionales o pruebas de comportamiento, se centran en obtener conjuntos de condiciones de entrada que ejerciten completamente los requisitos funcionales de un programa (Pressman, 2010).

La funcionalidad lo podemos observar en la figura 2.5 donde se representa un caso de prueba que se aplica prueba de caja negra del cual se enfoca a la funcionalidad del módulo o del sistema y como salida se obtiene resultados que pueden ser positivos o negativos.

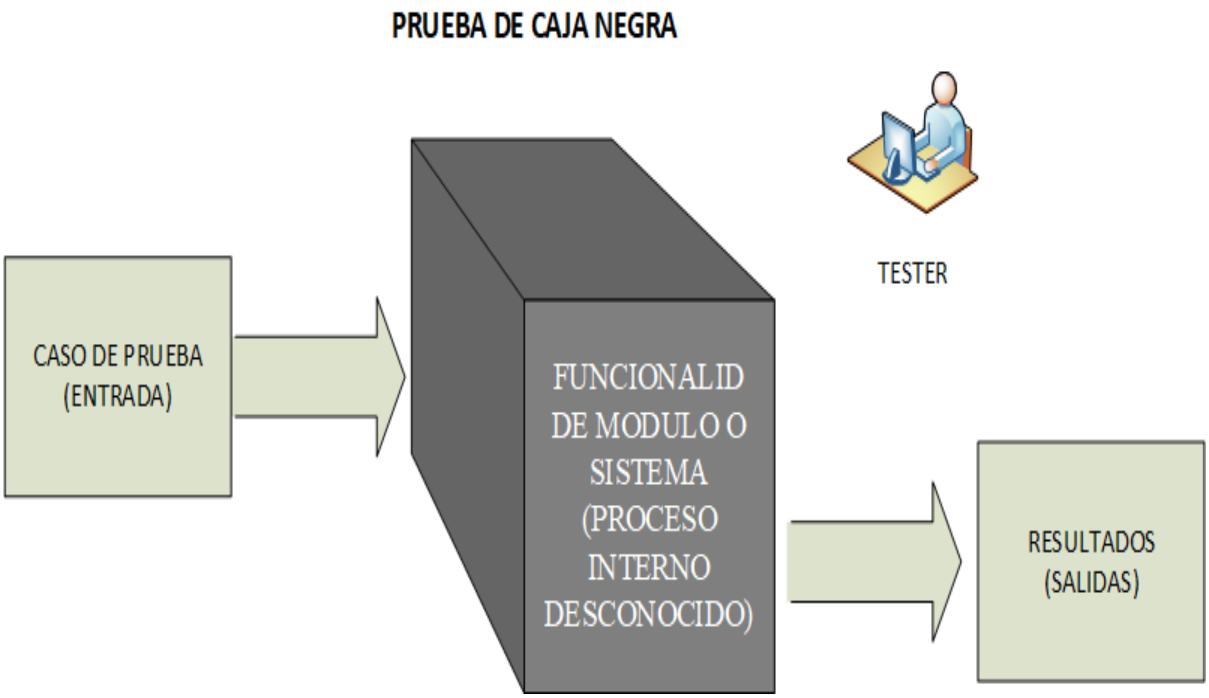


Figura 2.5.- Prueba de caja negra (elaboración propia)

Según Pressman (2010), El objetivo principal de las pruebas de caja negra es encontrar ítems de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en acceso a base de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Partición equivalente:

- Técnica para llevar a cabo la cobertura de entradas y salidas.
- Se utiliza para encontrar datos válidos e inválidos.
- Las particiones pueden ser identificadas como salidas, valores internos, relación de tiempo (Por ejemplo, antes o después de un evento) y parámetros de interfaz (*Testing* de Integración).

Análisis de Valor Límite:

- El valor máximo y el valor mínimo son los valores límites. Un valor límite de una partición válida es un valor límite válido, el valor de una partición inválida es un valor límite inválido.
- Es aplicable para todos los niveles de *testing*.
- Es considerada una extensión de la Partición Equivalente.
- Puede ser utilizada para la prueba de selección de dato

Tabla de Decisiones:

- Sirven para capturar los requerimientos del sistema que contienen las condiciones lógicas y documentar el diseño interno del sistema.
- Muchas veces, condiciones y acciones de entrada son estados que pueden ser Verdaderos o Falsos.
- Cada columna corresponde a una regla de negocio y es una única combinación de condiciones y el resultado de la ejecución se asocia con esta regla.
- La cobertura estándar es tener al menos una prueba por columna, lo que implica cubrir todas las combinaciones de pruebas.

Transición de Estado:

- Un sistema puede mostrar una respuesta diferente en función de las condiciones actuales o de su historial (su estado).
- Permite al *tester* poder ver el software en estados, las transiciones en los estados, la entrada o eventos que provocan cambios de estado (transiciones) y las acciones que puedan resultar de estas transiciones.
- Se muestran las relaciones entre: entradas y estados, y pueden destacarse posibles transiciones que son inválidas. Las pruebas pueden ser diseñadas para cubrir una secuencia típica de estados, para poder cubrirlos todos, para ejercer cada transición, ejercer secuencias específicas de transiciones o pruebas de transiciones.

Pruebas de caja blanca *(White box testing)*

Las cajas blancas también suelen ser llamadas estructurales o de cobertura lógica. En ellas se pretende investigar sobre la estructura interna del código, exceptuando detalles referidos a datos de entrada o salida, para probar la lógica del programa desde el punto de vista algorítmico.

En la figura 2.6, se puede observar un caso de prueba que entra en la aplicación de prueba de caja blanca, del cual se encarga de evaluar la estructura del código, así como coberturas lógicas, obteniendo de salida los resultados que pueden ser positivos o negativos.

Realizan un seguimiento del código fuente según se va ejecutando los casos de prueba, determinándose de manera concreta las instrucciones, bloques, etc. que han sido ejecutados por los casos de prueba.
En las pruebas de caja blanca se desarrollan casos de prueba que produzcan la ejecución de cada posible ruta del programa o módulo, considerándose una ruta como una combinación específica de condiciones manejadas por un programa.

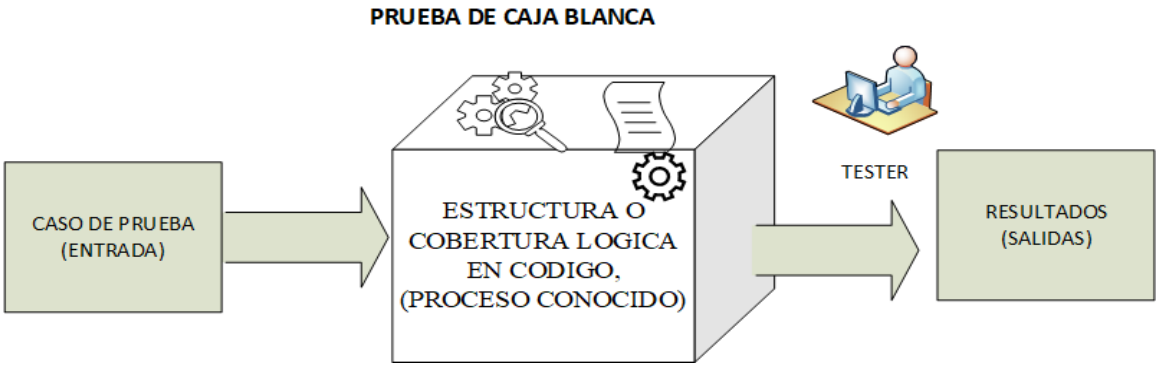


Figura 2.6.- Prueba de caja blanca (elaboración propia)

Características de las pruebas de Caja Blanca.

Estas se basan en el diseño de Casos de Prueba que usa la estructura de control del diseño procedimental para derivarlos. Mediante las pruebas de Caja Blanca el ingeniero de software puede obtener Casos de Prueba que:

- Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
- Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

Las pruebas de caja blanca son consideradas entre las más importantes que se aplican a los sistemas, con la que se obtienen como resultados la disminución en un gran porciento el número de errores existentes en el software y por ende una mayor calidad y confiabilidad en la codificación.

En la tabla 2.6 se describe los diferentes tipos de pruebas de caja blanca, como podemos ver las pruebas de caja blanca se enfoca de manera específica en la revisión al código tanto en estructura como en clases, variables, condiciones, caminos entre otros.

Tabla 2.6.- Tipos de pruebas de caja blanca (elaboración propia)

Tipos de pruebas de Caja Blanca	**Descripción**
De estructura de datos locales	Se centran en el estudio de las variables del programa. Busca que toda variable esté declarada y que no existan con el mismo nombre, ni declaradas local y globalmente, que haya referencias a todas las variables y para cada variable, analiza su comportamiento en comparaciones.
De Cobertura de Sentencias	Comprueba que todas las sentencias se ejecuten al menos una vez.
De Cobertura de Decisión	Ejecuta casos de prueba de modo que cada decisión se pruebe al menos una vez a Verdadero (*True*) y otra a Falso (*False*).
De Cobertura de Condición:	Ejecuta un caso de prueba a True y otro a False por cada condición, teniendo en cuenta que una decisión puede estar formada por varias condiciones.
De Cobertura de Condición/Decisión	Se realizan las pruebas de cobertura de condición y las de decisión a la vez.
De Condición Múltiple	Cada decisión multicondicional se traduce a una condición simple, aplicando posteriormente la cobertura de decisión.
De Cobertura de Caminos	Se escriben casos de prueba suficientes para que se ejecuten todos los caminos de un programa. Entendiéndose camino como una secuencia de sentencias encadenadas desde la entrada del programa hasta su salida.

Modelo V

El modelo V, se aprecia la relación entre las acciones para el aseguramiento de la calidad y aquellas asociadas con la comunicación, modelado y construcción temprana. A medida que el equipo de software avanza hacia abajo desde el lado izquierdo de la V, los requerimientos básicos del problema mejoran hacia representaciones técnicas cada vez más detalladas del problema y de su solución. Una vez que se ha generado el código, el equipo sube por el lado derecho de la V, y en esencia ejecuta una serie de pruebas (acciones para asegurar la calidad) que validan cada uno de los modelos creados cuando el equipo fue hacia abajo por el lado izquierdo. (Pressman, 2010)

El lado izquierdo de la V representa la descomposición de las necesidades, y la creación de las especificaciones del sistema. El lado derecho de la V representa la integración de las piezas y su verificación. V significa «Verificación y validación».
Los siguientes objetivos están destinados a ser alcanzados durante la ejecución del proyecto:

- **Minimización de los riesgos del proyecto:** permite una detección temprana de las desviaciones y los riesgos y mejora la gestión de procesos, reduciendo así los riesgos del proyecto.
- **Mejora y Garantía de Calidad:** los resultados provisionales definidos se pueden comprobar en una fase temprana. La uniformidad en el contenido del producto mejora la legibilidad, comprensibilidad y verificabilidad.
- **Reducción de los gastos totales durante todo el proyecto y sistema de Ciclo de Vida:** el esfuerzo para el desarrollo, producción, operación y mantenimiento de un sistema puede ser calculado, estimado y controlado de manera transparente mediante la aplicación de un modelo de procesos estandarizados.
- **Mejora de la comunicación entre todos los inversionistas:** la descripción estandarizada y uniforme de todos los elementos pertinentes y términos es la base para la comprensión mutua entre todos los inversionistas. De este modo, se reduce la pérdida por fricción entre el usuario, comprador, proveedor y desarrollador.

La figura 2.7 se muestra el Modelo en V, o Modelo de Cuatro Niveles, del ciclo de vida de un proyecto de desarrollo de software. El modelo representa, en forma de V, las relaciones temporales entre las distintas fases del ciclo de desarrollo de un proyecto.

En los niveles lógicos del 1 al 4, para cada fase del desarrollo, existe una fase correspondiente o paralela de verificación o validación. Esta estructura obedece al principio de que para cada fase del desarrollo debe existir un resultado verificable.

- El nivel 1 está orientado al “cliente”. El inicio del proyecto y el fin del proyecto constituyen los dos extremos del ciclo. Se compone del análisis de requisitos y especificaciones, se traduce en un documento de requisitos y especificaciones.
- El nivel 2 se dedica a las características funcionales del sistema propuesto. Puede considerarse el sistema como una caja negra, y caracterizarla únicamente con aquellas funciones que son directa o indirectamente visibles por el usuario final, se traduce en un documento de análisis funcional.

- El nivel 3 define los componentes hardware y software del sistema final, a cuyo conjunto se denomina arquitectura del sistema.
- El nivel 4 es la fase de implementación, en la que se desarrollan los elementos unitarios o módulos del programa.

Ventajas:

- La relación entre las etapas de desarrollo y los distintos tipos de pruebas facilitan la localización de fallos.
- Es un modelo sencillo y de fácil aprendizaje
- Especifica bien los roles de los distintos tipos de pruebas a realizar
- Involucra al usuario en las pruebas

Desventajas:

- Es difícil que el cliente exponga explícitamente todos los requisitos
- El cliente debe tener paciencia pues obtendrá el producto al final del ciclo de vida
- Las pruebas pueden ser caras y, a veces, no lo suficientemente efectivas
- El producto final obtenido puede que no refleje todos los requisitos del usuario

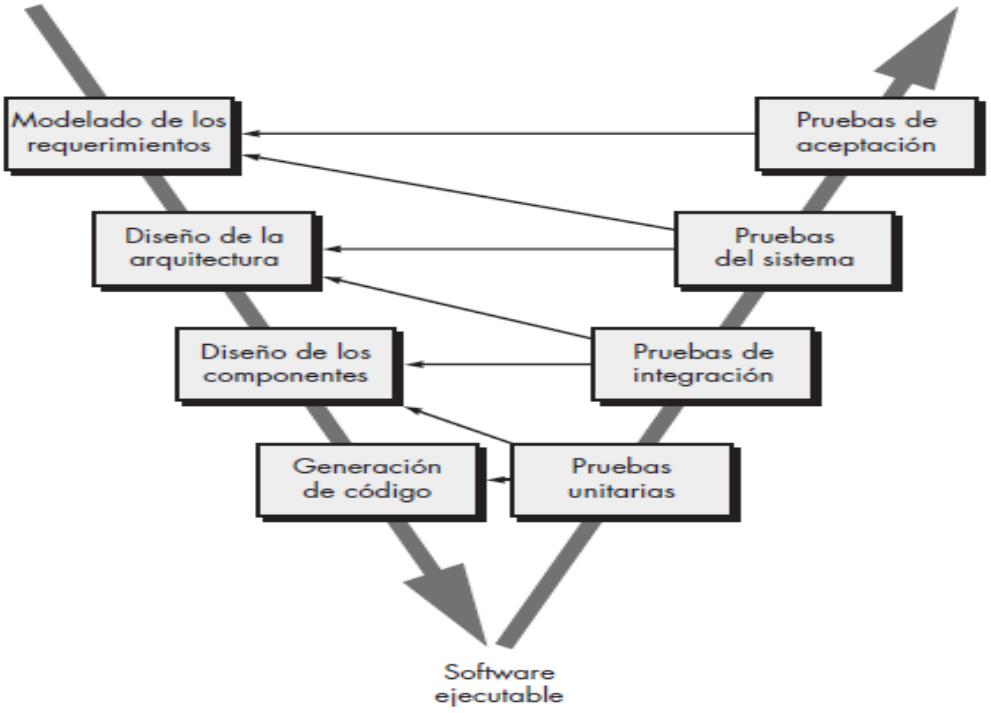


Figura 2.7.- Modelo V. (Pressman, 2010).

Planeación de Pruebas.

Es la etapa en donde se ejecutan las primeras actividades correspondientes al proceso de pruebas y tiene como resultado un entregable denominado plan de pruebas el cual debe estar conformado en cuando menos por aspectos tales como:

Alcance de la prueba: determina que funcionalidades del producto y/o software serán probadas durante el transcurso de la prueba, así como la prioridad con la que las funcionalidades deben probarse y la profundidad de las pruebas. (Zapata,2013)

Tipos de Prueba: se debe determinar qué tipos de pruebas requeriría el producto. No todos los productos de software requieren la aplicación de todos los tipos de pruebas que existen, por esta razón, es estrictamente necesario que el líder de pruebas se plantee preguntas que le permitan determinar qué tipos de prueba son aplicables al proyecto en evaluación. Los posibles tipos de prueba a aplicar son: pruebas de stress, pruebas de rendimiento, pruebas de carga, pruebas funcionales, pruebas de usabilidad, pruebas de regresión, entre otros. (Zapata,2013)

Estrategia de Pruebas: teniendo en cuenta que no es viable probar con base a todas las posibles combinaciones de datos, es necesario determinar a través de un análisis de riesgos sobre que funcionalidades debemos centrar nuestra atención. Adicionalmente, una buena estrategia de pruebas debe indicar los niveles de pruebas (ciclos) que aplicaremos y la intensidad o profundidad a aplicar para cada nivel de pruebas definido. En este punto también es importante definir los criterios de entrada y salida para cada ciclo de pruebas a ejecutar. (Zapata,2013)

Criterios de Salida: entre las partes involucradas en el proceso, se define de manera formal, bajo qué condiciones se puede considerar que una actividad de pruebas fue finalizada. Los criterios de salida se deben definir para cada nivel de pruebas a ejecutar. Algunos ejemplos de criterios de salida que pueden ser utilizados son: porcentaje de funcionalidades de alto riesgo probadas con éxito, número defectos críticos y/o mayores aceptados, etc. (Zapata,2013)

Diseño de Pruebas: los entregables claves para iniciar este diseño pueden ser: casos de uso, historias de usuario, arquitectura del sistema, diseños, manuales de usuario (si existen), manuales técnicos (si existen). Los casos de prueba negativos permiten validar cómo se comporta el sistema ante situaciones atípicas y permite verificar la robustez del sistema, atributo que constituye uno de los requerimientos no funcionales indispensable para cualquier software. Por último, es necesario definir cuáles son los datos de prueba necesarios para la ejecución de los casos de prueba diseñados. (Zapata,2013)

Implementación y Ejecución de Pruebas: la ejecución de pruebas debe iniciar con la creación de los datos de prueba necesarios para ejecutar los casos de prueba diseñados. La ejecución de estos casos, puede realizarse de manera manual o automatizada; en cualquiera de los casos, cuando se detecte un fallo en el sistema, este debe ser documentado. Es indispensable ejecutar un ciclo de regresión que nos permita garantizar, que los defectos corregidos en el proceso de depuración de la firma, no hayan desencadenado otros tipos de defectos en el sistema. (Zapata,2013)

Evaluación de Criterios de Salida: los criterios de salida son necesarios para determinar si es posible dar por finalizado un ciclo de pruebas. Para esto, es conveniente definir una serie de métricas que permitirán al finalizar un proceso de pruebas, comparar los resultados obtenidos contra las métricas definidas, sí los resultados obtenidos no superan la métrica definida, no es posible continuar con el siguiente ciclo de pruebas. (Zapata,2013)

Cierre del proceso: Durante este periodo de cierre el cual históricamente se ha comprobado que se le destina muy poco tiempo en la planeación, se deben cerrar las incidencias reportadas, se debe verificar si los entregables planeados han sido entregados y aprobados, se deben finalizar y aprobar los documentos de soporte de prueba, analizar las lecciones aprendidas para aplicar en futuros proyectos, etc. (Zapata,2013)

2.3 Calidad de software

Definiciones sobre la Calidad

La característica calidad es muy importante que toda empresa competitiva debe ofrecer en sus productos o servicios para obtener la confianza de los usuarios y la fidelidad de sus clientes. Para obtener el grado aceptable de calidad en el mercado se debe invertir tanto económicamente y en esfuerzos de mejora en las evaluaciones de calidad.

Debido a los múltiples procesos de desarrollo que existen en la actualidad, los procesos de evaluación también suelen ser versátiles por lo que no puede haber pretexto para implementar el aseguramiento de calidad de software en los proyectos de desarrollo de software.

A continuación, se presentan algunas definiciones sobre la calidad:

“Conjunto de propiedades y características de un producto o servicio, que confiere su aptitud para satisfacer las necesidades dadas” (Instituto Alemán para la Normalización, DIN 55 350-11, 1979).

Deming W. (1992) definió la calidad como un “grado predecible de uniformidad y fiabilidad a bajo coste, adecuado a las necesidades del mercado”.

Otro de las definiciones de calidad es de Armand V. Feigenbaum que es citado por lozano (1998) sobre la calidad “Un sistema eficaz para integrar los esfuerzos de mejora de la calidad de los distintos grupos de una organización, para proporcionar productos y servicios a niveles que permitan la satisfacción del cliente”.

En la norma *ISO 8402 (1994)* se define la calidad como el conjunto de propiedades y características de un producto o servicio que le confiere su aptitud para satisfacer unas necesidades expresadas o implícitas.

David Garving (1984), dice que “la calidad es un concepto complejo y de facetas múltiples” y puede describirse desde 4 puntos de vista diferentes:

- El punto de vista del usuario concibe la calidad en términos de las metas específicas del usuario final y si el producto las satisface tiene calidad.
- El punto de vista del fabricante la define en términos de las especificaciones originales del producto. Si éste las cumple tiene calidad.
- El punto de vista del producto sugiere que la calidad tiene que ver con las características inherentes de un producto.
- El punto de vista basado en el valor la mide de acuerdo con lo que un cliente éste dispuesto a pagar por un producto.

Ahora algunas definiciones sobre la calidad del software

Una definición propuesta por Pressman (2010), sugiere que “la calidad de software es el cumplimiento de los requisitos de funcionalidad y desempeño explícitamente establecidos, de los estándares de desarrollo explícitamente documentados y de las características implícitas que se esperan de todo software desarrollado profesionalmente”.

Pérez (2016) cita a Glass donde afirma que “la calidad es importante, pero que si el usuario está insatisfecho nada de lo demás importa”.

La calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad. (Acimed, 1995)

La obtención de un software con calidad implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software. (Acimed, 1995)

Software Quality Assurance (SQA) vs Software Quality Control (SQC)

Al igual que ocurrió con la definición de calidad hay varios puntos de vista desde donde se puede definir el aseguramiento de la calidad del software. |

Una de las definiciones es de Galin (2004) define *SQA* como "Un conjunto, sistemático y planificado, de acciones necesarias para proveer la evidencia adecuada de que el proceso de desarrollo o mantenimiento de un sistema de software cumple los requerimientos técnicos funcionales tan bien como los requerimientos gerenciales para cumplir la planificación y operar dentro del presupuesto confinado"

Desde el punto de vista de la IEEE 610.12 (1990) define el aseguramiento de la calidad como: "Una guía planificada y sistemática de todas las acciones necesarias para proveer la evidencia adecuada de que un producto cumple los requerimientos técnicos establecidos". el SEI define *SQA* como "El aseguramiento de la calidad del software provee claro control del proceso que está siendo usado por el proyecto y del producto que se está construyendo."

Desde el punto de vista del aseguramiento, Reifer (1985) define *SQA* como "El aseguramiento de la calidad del software es el sistema de métodos y procedimientos usados para asegurar que el producto de software alcanza sus requerimientos. El sistema involucra la planificación, estimación y monitoreo de las actividades de desarrollo realizadas por otros."

Desde el punto de vista de la capacidad de uso Schulmeyer y McManus (1992) definen *SQA* como: "Las actividades sistemáticas que proveen evidencia de la capacidad o disponibilidad de uso del producto de software total."

A menudo hay una confusión entre *SQA* y el *testing* (el cual actualmente forma parte del área de control de calidad del software *SQC*). En el desarrollo de software la diferencia entre *SQC* y *SQA* no está clara y estos términos a menudo se confunden, *SQA* se encarga de controlar el cumplimiento del proceso, mientras que *SQC* son aquellas acciones del aseguramiento de la calidad que proporcionan un medio para controlar y medir las características de un elemento, proceso o facilidad respecto a los requisitos establecidos, ver la tabla 2.7. ahí se realiza una comparación sencilla suficiente para comprender sus diferencias entre *SQA*, *SQC* y pruebas.

Como responsable de la elaboración del proceso el trabajo del equipo de *SQA* es participar en la definición de los planes, procesos, estándares y procedimientos para asegurar que se ajustan a las necesidades del proyecto y que pueden ser usados para realizar las evaluaciones de *QA* y cumplir los

requerimientos del proyecto y las políticas de la organización. Para cumplir este rol el aseguramiento de la calidad debería comenzar en las fases tempranas del proyecto.

Tabla 2.7.- Comparativa de sqa, sqc y pruebas (Elaboración propia).

SQA	***SQC***	**Pruebas**
Es un conjunto de actividades diseñadas para asegurar que el proceso de desarrollo y/o mantenimiento es adecuado para asegurar que un sistema cumplirá los objetivos para los que fue diseñado.	Es un conjunto de actividades diseñadas para evaluar un producto. Que debe contener las funciones solicitadas por el cliente	Es el proceso de ejecutar un sistema con la intención de encontrar defectos incluyendo una planificación previa a la ejecución de los Casos de Test.
Se centra en los procesos y Procedimientos de revisión e inspección.	Ejecutar el software con un objetivo para identificar Error / defecto a través de implementación de Procedimientos y procesos.	Se centra en pruebas reales para validar la funcionalidad del software
Actividades orientadas a proceso	Actividades orientado al producto	Actividades orientadas al producto
Actividades preventivas	Actividades correctivas	Es un proceso preventivo

En la tabla 2.8 se visualiza una descripción de los enfoques que ofrece el aseguramiento de la calidad del software, en cada uno de las distintas formas de evaluar la calidad del software tienen ciertos objetivos y responsabilidades que deben cumplir.

Tabla 2.8.- Enfoques de SQA (Vargas y Biagioli, 2009)

Enfoque SQA	**Descripción**
Satisfacer al cliente	• Identificar la funcionalidad que al cliente le gustaría encontrar. • Ayudar a la organización a sensibilizarse con las necesidades del cliente. • Actuar como un cliente de prueba para obtener una alta satisfacción del cliente
analista	• Juntar muchos datos sobre todos los aspectos del producto y del proceso. • Con esta información ayudar a mejorar los procesos y los productos.
proveedor de información	• Proveer información técnica objetiva para que la gerencia pueda usarla para tomar mejores decisiones. • Proveer información apropiada de las clases de productos y de los riesgos asociados con estos. • Concentrarse más en la reducción de los riesgos que en el cumplimiento del proceso.

Cuando el equipo de *SQA* esta embebido en el proceso, se deben resolver varios inconvenientes para garantizar la objetividad:

- Todo aquel que realice actividades de aseguramiento de la calidad debería estar entrenado en el aseguramiento de la calidad.
- Las actividades de aseguramiento de la calidad realizadas para un producto deberían ser separadas de aquellas involucradas en el desarrollo y mantenimiento del mismo.
- Debe estar disponible un canal de comunicación independiente en el nivel apropiado de la gerencia para poder escalar las no conformidades cuando sea necesario.
- Los procesos de aseguramiento de calidad de un producto de software suelen dividirse en lo que respecta a su componente analítico en pruebas estáticas y dinámicas.

Verificación y Validación (V&V)

La definición dada por *IEEE 610.12* (1990) sobre la Verificación y Validación es la siguiente: Verificación: proceso de evaluación de un sistema o componente para determinar si un producto de una determinada fase de desarrollo satisface las condiciones impuestas al inicio de la fase. Validación: proceso de evaluación de un sistema o componente durante o al final del proceso de desarrollo para determinar cuándo se satisfacen los requerimientos especificados.

La verificación se refiere al conjunto de tareas que garantizan que el software implementa correctamente una función específica. La validación es un conjunto diferente de tareas que aseguran que el software que se construye sigue los requerimientos del cliente y se obtiene información mediante las siguientes preguntas: en la verificación: ¿Estamos construyendo el producto correctamente? El papel de la verificación comprende comprobar que el software está de acuerdo con su especificación. Se comprueba que el sistema cumple los requerimientos funcionales y no funcionales que se le han especificado. (Boehm, 1984).

Validación: ¿Estamos construyendo el producto correcto? La validación es un proceso más general. Se debe asegurar que el software cumple las expectativas del cliente. Va más allá de comprobar si el sistema está acorde con su especificación, para probar que el software hace lo que el usuario espera a diferencia de lo que se ha especificado. (Boehm, 1984).

CMMI (2002) define el propósito de la Validación de la siguiente manera: El propósito de la Validación es demostrar que un producto o componente de producto cumple su uso previsto cuando es puesto en su ambiente previsto.

Según Kit (1995), la Verificación es el proceso de evaluar, revisar, inspeccionar y hacer controles de escritorio de productos intermedios, tales como especificaciones de requerimientos, especificaciones de diseño y código. La verificación se aplica más al proceso de evaluación del sistema o componentes ya

que permite determinar si los productos de una determinada fase del desarrollo satisfacen las condiciones impuestas en el inicio de la etapa. En la tabla 2.9 se muestra una comparativa sobre la verificación y validación.

En la validación también es una evaluación del sistema o componentes solo que es en el transcurso o al final del proceso del desarrollo para determinar si cumple con lo especificado. El resultado final del desarrollo software se debe ajustar a lo que el usuario quería (sus necesidades) (Kit, 1995).

Tabla 2.9.- Comparativa entre verificación y validación (Elaboración propia).

Verificación	**Validación**
¿Estamos construyendo el producto correctamente? El software debería ajustarse a la especificación del cliente.	¿Estamos construyendo el producto correcto? El software debería hacer lo que el cliente realmente pide
Asegura que el sistema de software cumple con todas las funcionalidades.	Asegura que las funcionalidades sean el comportamiento deseado.
La verificación se aplica primero e incluye la comprobación de Documentación, código, etc.	La validación ocurre después de la verificación y Principalmente el control del producto en general.
Tiene actividades estáticas como revisiones, e Inspecciones para verificar un software además de pruebas dinámicas como unitarias y de integración.	Tiene actividades dinámicas, ya que incluye ejecución del software contra los Requisitos.
Es un proceso objetivo y no decisión subjetiva debe ser necesaria para verificar un software.	Se trata de un proceso subjetivo, sobre lo bien que funciona el software.

2.3.1. Calidad estructurada (modelos y estandares)

Existen modelos y estándares establecidos para evaluar la calidad del software, en algunos casos se enfocan en evaluar los procesos que se llevan a cabo para el desarrollo y en otros casos se enfocan en evaluar únicamente el producto final. También existen modelos que se encargan evaluar a las organizaciones y altas direcciones de empresas desarrolladoras de software.

Estos modelos y estándares se encargan de certificar mediante una estructura de procesos, características y documentación que comprueban la calidad del software, A continuación, se realiza una descripción de algunos modelos y estándares conocidos en el mundo y ampliamente usado por diversas empresas.

The Capability Maturity Model Integration CMMI

El *CMM* original se desarrolló y actualizó por parte del *Software Engineering Institute (SEI)* a lo largo de los años de 1990 como un marco conceptual MPS completo. Más adelante, evolucionó en la Integración del Modelo de Madurez de Capacidades *(CMMI)*, un metamodelo de proceso exhaustivo que se impulsa en un conjunto de sistemas y capacidades de ingeniería del software que deben presentarse conforme las organizaciones alcanzan diferentes niveles de capacidad y madurez del proceso. (Pressman, 2010)

La *CMMI* es un conjunto de mejores prácticas reconocido a nivel mundial que permite a las organizaciones mejorar el rendimiento, las capacidades clave y los procesos comerciales críticos. (CMMI Institute, 2018)

En la tabla 2.10 se describe de manera breve el modelo *CMMI* del cual evalúa de acuerdo con los siguientes niveles de Madurez:

Tabla 2.10.- Niveles de madures de CMMI (Pressman, 2010)

Nivel de Madurez	**Descripción**
Nivel 0: Incompleto	El área del proceso (por ejemplo, gestión de requisitos) no se realiza o no logra todas las metas y objetivos definidos por la CMMI para la capacidad nivel 1 del área de proceso.
Nivel 1: Realizado	Todas las metas específicas del área de proceso (como se define mediante la CMMI) están satisfechas. Se realizan las tareas de trabajo requeridas para producir productos operativos definidos.
Nivel 2: Administrado	Se satisfacen todos los criterios del nivel 1 de capacidad. Además, todo el trabajo asociado con el área de proceso se encuentra acorde con una política definida de manera organizacional; todo el personal que realiza el trabajo tiene acceso a recursos adecuados para tener listo el trabajo; los participantes se involucran de manera activa en el área de proceso según se requiera; todas las tareas del trabajo y los productos operativos se "monitorean, controlan y revisan, y se evalúan para su adhesión a la descripción del proceso".
Nivel 3: Definido	Se logran todos los criterios del nivel 2 de capacidad. Además, el proceso se "hace a la medida, a partir del conjunto de procesos estándar y de acuerdo con los lineamientos de producción de la organización; contribuye con productos operativos, medidas y otra información para mejorar los procesos activos del proceso organizacional".
Nivel 4: Administrado cuantitativamente	Se logran todos los criterios del nivel 3 de capacidad. Además, el área de proceso se controla y mejora, usando medición y valoración cuantitativa. "Los objetivos cuantitativos para el rendimiento cualitativo y de proceso se establecen y usan como criterios para gestionar el proceso".
Nivel 5: Optimizado	Se logran todos los criterios del nivel 4 de capacidad. Además, el área de proceso se adapta y optimiza, usando medios cuantitativos (estadísticos) para satisfacer las necesidades cambiantes del cliente y para mejorar continuamente la eficacia del área de proceso bajo consideración.

Regulado por *Software Engineering Institute (SEI)*

MoProSoft (Modelo de Procesos de Software)

Fue desarrollado a solicitud de la Secretaría de Economía para servir de base a la Norma Mexicana para la Industria de Desarrollo y Mantenimiento de Software bajo el convenio con la Facultad de Ciencias, Universidad Nacional Autónoma de México.

Es un Modelo de Referencia de Procesos conformado por un conjunto de buenas prácticas y procesos de gestión e ingeniería de software, que contribuyen a que las organizaciones dedicadas al desarrollo y mantenimiento de software mejoren su forma de trabajar y gestionar sus proyectos y por consiguiente incrementar sus niveles de capacidad y competitividad tanto nacional como internacionalmente. (NYCE, 2016)

En la tabla 2.11 se enlista las categorías dentro de la gestión de una empresa desarrolladora y en la tabla 2.12 se muestran los niveles de evaluación dentro de las pymes del cual *NYCE* se encarga de certificar.

Tabla 2.11.- Categorías de MoProSoft y sus procesos (NYCE, 2016)

Categoría	Procesos
Categoría alta dirección (DIR)	• Gestión de Negocio
Categoría Gerencia (GER)	• Gestión de Procesos • Gestión de Proyectos • Gestión de Recursos • Recursos Humanos y Ambiente de Trabajo • Bienes Servicios e Infraestructura • Conocimiento de la Organización.
Categoría Operación (OPE)	• Administración de Proyectos Específicos • Desarrollo y Mantenimiento de Software

Tabla 2.12.- Nivel de capacidad de MoProSoft (NYCE,2016)

Nivel de capacidad	Descripción
1. Realizado	El proceso se implementa y alcanza su propósito, Amarillo
2. Gestionado	El proceso realizado se administra sus productos de trabajo están establecidos, controlados y mantenidos, Azul
3. establecido	El proceso realizado y gestionado se implementa por medio de un proceso definido, Verde
4. predecible	El proceso establecido opera bajo límites definidos y conocidos, Rosa
5. optimizado	El proceso predecible se mejora continuamente, N.A.

Nota; Moprosoft Está regulado por NYCE

Modelo McCAll

McCall, Richards y Walters (1977) proponen una clasificación útil de los factores que afectan la calidad del software. se centran en tres aspectos importantes del producto de software: sus características operativas, su capacidad de ser modificado y su adaptabilidad a nuevos ambientes.

En relación con los factores mencionados McCall, Richards y Walters (1977), hacen las descripciones de su modelo de evaluación al producto de software que en la tabla 2.13 se puede observar.

Tabla 2.13.- Factores en calidad de software de MCCALL (Pressman, 2010)

Factores	Descripción
Corrección	Grado en el que un programa satisface sus especificaciones y en el que cumple con los objetivos de la misión del cliente.
Confiabilidad.	Grado en el que se espera que un programa cumpla con su función y con la precisión requerida [debe notarse que se han propuesto otras definiciones más completas de la confiabilidad
Eficiencia	Cantidad de recursos de cómputo y de código requeridos por un programa para llevar a cabo su función.
Integridad	Grado en el que es posible controlar el acceso de personas no autorizadas al software o a los datos.
Usabilidad	Esfuerzo que se requiere para aprender, operar, preparar las entradas e interpretar las salidas de un programa.
Facilidad de recibir mantenimiento	Esfuerzo requerido para detectar y corregir un error en un programa (ésta es una definición muy limitada).
Flexibilidad	Esfuerzo necesario para modificar un programa que ya opera.
Susceptibilidad de someterse a pruebas	Esfuerzo que se requiere para probar un programa a fin de garantizar que realiza la función que se pretende.
Portabilidad	Esfuerzo que se necesita para transferir el programa de un ambiente de sistema de hardware o software a otro.
Reusabilidad	Grado en el que un programa (o partes de uno) pueden volverse a utilizar en otras aplicaciones (se relaciona con el empaque y el alcance de las funciones que lleva a cabo el programa).
Interoperabilidad	Esfuerzo requerido para acoplar un sistema con otro.

Modelo FURPS+

Para clasificar los atributos de calidad del software se definieron varios modelos, uno de ellos fue el modelo FURPS+. Este modelo fue desarrollado por Robert Grady y Deborah Caswell de Hewlett Packard. Bajo el acrónimo *FURPS+*, por sus siglas en inglés (Vargas y Biagioli, 2009)

En la tabla 2.14 se describen las características del modelo *FURPS+* teniendo una cierta similitud en los atributos de McCall, pero con un plus en los atributos de documentación y licencia.

Tabla 2.14.- Descripción de las características FURPS+ (Vargas y Biagioli, 2009)

SIGLA	TIPO DE REQUERIMIENTO		DESCRIPCIÓN
F	Funtional	Funcional	Características, capacidades y algunos aspectos de seguridad
U	Usability	Facilidad de uso	Factores Humanos (interacción), ayuda, documentación.
R	Reliability	Fiabilidad	Frecuencia de fallos, capacidad de recuperación de un fallo, y grado de previsión.
P	Perfomance	Rendimiento	Tiempos de respuesta, productividad, precisión, disponibilidad, uso de los recursos
S	Supportability	Soporte	Adaptabilidad, facilidad de mantenimiento, internacionalización, facilidad de configuración.
+	Plus	Implementación	Limitación de recursos, lenguajes y herramientas, hardware
		Interfaz	Restricciones impuestas para la interacción con sistemas externos (No es GUI)
		Operaciones	Gestión del sistema, pautas administrativas, puesta en marcha.
		Empaquetamiento	Formas de distribución
		Legales	Licencia, derechos de autor, etc.

Nota: El signo más (+) en el acrónimo FURPS+ nos permite especificar restricciones, incluyendo restricciones de diseño, implementación e interfaces. *Fuente: Adaptado de Craig Larman (2003): "UML y Patrones" 2Ed.*

Norma ISO 9126

El estándar *ISO 9126* se desarrolló con la intención de identificar los atributos clave del software de cómputo. los factores *ISO 9126* no necesariamente conducen a una medición directa. Sin embargo, proporcionan una base útil para hacer mediciones indirectas y una lista de comprobación excelente para evaluar la calidad del sistema. (Pressman, 2010)

El estándar *ISO/IEC FDIS 9126* (2001), contiene cuatro documentos que describen las características de calidad en el software:

- Parte 1. Ingeniería de software-Calidad del producto-Modelo de calidad (*ISO/IEC 9126-1,2001*).
- Parte 2. Ingeniería de software-Calidad del producto-Métricas externas (*ISO/IEC 9126-2,2003*).
- Parte 3. Ingeniería de software-Calidad del producto-Métricas internas (*ISO/IEC 9126-3,2003*).
- Parte 4. Ingeniería de software-Calidad del producto-Métricas de calidad en el uso *(ISO/IEC 9126-1,2004)*

En el documento de la *ISO 9126-1* (2001), Propone un modelo de calidad categorizando la calidad de los atributos software en seis características (funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad), las cuales son subdivididas en sub características. Las sub características pueden ser medidas con métricas internas o externas.

En el documento de la *ISO 9126-2* (2003), Contiene terminología relacionada con las medidas de las métricas. El uso de las métricas en el proceso del ciclo de la vida y unos conjuntos básicos introductorios de métricas externas para cada característica y sub característica de calidad de software.

En el documento de la *ISO 9126-3* (2003), Proporciona métricas internas para medir los atributos de las características de calidad definidas en la norma *9126-1*.

En el documento de la *ISO 9126-4* (2004) Proporciona métricas para evaluar un software cuando ya se encuentra en funcionamiento y las métricas son las siguientes:

- Efectividad.
- Productividad.
- Seguridad.
- Satisfacción.

En la tabla 2.15 se muestra una descripción de las características de calidad interna y externa. El estándar es regulado por *International Organization for Standardization (ISO).*

Tabla 2.15.- Características de calidad interna y externa de la ISO/IEC FDIS 9126- 2 y 3 (Elaboración propia basada en la ISO 9126)

Característica	Descripción
Funcionalidad	El sistema debe tener la capacidad para proporcionar funciones que cumplan con las necesidades bajo condiciones especificadas. sus atributos son las siguientes: Idoneidad, exactitud, interoperabilidad, seguridad y cumplimiento.
Confiabilidad	El sistema debe tener la capacidad de mantener un nivel de rendimiento cuando se usa bajo condiciones especificadas. sus atributos son las siguientes: madurez, tolerancia a fallos, recuperabilidad y cumplimiento.
Usabilidad	El sistema debe tener un fácil entendimiento, fácil aprendizaje, fácil uso y atractivo para el usuario y se adapte a las distintas condiciones especificadas. sus atributos son las siguientes: Comprensibilidad, capacidad de aprendizaje operabilidad, atractivo y cumplimiento.
Eficiencia	El sistema debe tener la capacidad de proporcionar un rendimiento adecuado, en relación con la cantidad de recursos utilizados, bajo condiciones establecidas. sus atributos son las siguientes: comportamiento del tiempo, utilización de recursos y cumplimiento.
Mantenibilidad	El sistema debe tener la capacidad de ser modificado, estos pueden incluir correcciones, mejoras o adaptación del software a los cambios en el entorno, en los requisitos y especificaciones funcionales. sus atributos son las siguientes: analizable, cambiable, estable, testeable y cumplimiento.
Portabilidad	El sistema debe tener la capacidad de mantener un nivel de rendimiento cuando se usa bajo condiciones especificadas. sus atributos son las siguientes: adaptabilidad, instabilidad, coexistencia, reemplazable y comportamiento

Norma ISO 25010

La norma *ISO/IEC 25000* proporciona una guía para el uso de las nuevas series de estándares internacionales, llamados Requisitos y Evaluación de Calidad de Productos de Software *(SQuaRE)*. Constituyen una serie de normas basadas en la *ISO 9126* y en la *ISO 14598* (Evaluación del Software), y su objetivo principal es guiar el desarrollo de los productos de software con la especificación y evaluación de requisitos de calidad. Regulado por Organización Internacional para la Estandarización (ANSI)

La *ISO/IEC 25010* de *Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models*. Es Un modelo de calidad del producto compuesto por ocho características (que se subdividen en sub características) que se relacionan con las propiedades estáticas del software y las propiedades dinámicas del sistema informático. El modelo es aplicable tanto a sistemas informáticos como a productos de software. (ISO/IEC 25010, 2011)

Las características y sub características proporcionan terminología coherente para especificar, medir y evaluar la calidad del producto de sistema y software. También proporcionan un conjunto de características de calidad contra las cuales los requisitos de calidad establecidos se pueden comparar para que estén completos. (ISO/IEC 25010, 2011)

Aunque el alcance del modelo de calidad del producto está destinado a ser software y sistemas informáticos, muchas de las características también son relevantes para sistemas y servicios más amplios. (ISO/IEC 25010, 2011)

SQuaRE, se trata de una revisión de la *ISO 9126-1*, y tiene las mismas características de calidad de software. Hay dos aspectos importantes en el campo de la calidad del software, el producto y el proceso. *SQuaRE* se centra en el lado del producto. *SQuaRE* hereda el modelo de calidad de la *ISO 9126-1*. El modelo de ciclo de vida de la calidad del producto software se basa en la calidad del producto software en tres fases principales del ciclo de vida del producto software:

- Producto bajo desarrollo: está sujeto a la calidad interna del software
- Producto en operación: está sujeto a la calidad externa del software
- Producto en uso: está sujeto de calidad de uso.

El modelo de calidad del producto definido por la *ISO/IEC 25010* se encuentra compuesto por las ocho características de calidad que se muestran en la tabla 2.16.

Tabla 2.16.- Características de calidad de la ISO/IEC 25010 (Elaboración propia basado en ISO 25010)

Características	**Sub características**	**Descripción**
Adecuación Funcional	**Completitud funcional**	Grado en el cual el conjunto de funcionalidades cubre todas las tareas y los objetivos del usuario especificados.
	Corrección funcional	Capacidad del producto o sistema para proveer resultados correctos con el nivel de precisión requerido.
	Pertinencia funcional.	Capacidad del producto software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados.
Eficiencia de desempeño	**Comportamiento temporal**	Los tiempos de respuesta y procesamiento y las ratios de *throughput* de un sistema cuando lleva a cabo sus funciones bajo condiciones determinadas en relación con un banco de pruebas (*benchmark*) establecido.
	Utilización de recursos	Las cantidades y tipos de recursos utilizados cuando el software lleva a cabo su función bajo condiciones determinadas.
	Capacidad	Grado en que los límites máximos de un parámetro de un producto o sistema software cumplen con los requisitos.

Compatibilidad	**Coexistencia**	Capacidad del producto para coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes sin detrimento.
	Interoperabilidad	Capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.
Usabilidad	**Capacidad para reconocer su adecuación.**	Capacidad del producto que permite al usuario entender si el software es adecuado para sus necesidades.
	Capacidad de aprendizaje	Capacidad del producto que permite al usuario aprender su aplicación.
	Capacidad para ser usado	Capacidad del producto que permite al usuario operarlo y controlarlo con facilidad.
	Protección contra errores de usuario	Capacidad del sistema para proteger a los usuarios de hacer errores.
	Estética de la interfaz de usuario	Capacidad de la interfaz de usuario de agradar y satisfacer la interacción con el usuario.
	Accesibilidad.	Capacidad del producto que permite que sea utilizado por usuarios con determinadas características y discapacidades.
Fiabilidad	**Madurez**	Capacidad del sistema para satisfacer las necesidades de fiabilidad en condiciones normales.
	Disponibilidad	Capacidad del sistema o componente de estar operativo y accesible para su uso cuando se requiere.
	Tolerancia a fallos	Capacidad del sistema o componente para operar según lo previsto en presencia de fallos hardware o software.
	Capacidad de recuperación.	Capacidad del producto software para recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallo.
Seguridad	**Confidencialidad**	Capacidad de protección contra el acceso de datos e información no autorizados, ya sea accidental o deliberadamente.
	Integridad	Capacidad del sistema o componente para prevenir accesos o modificaciones no autorizados a datos o programas de ordenador.
	No repudio	Capacidad de demostrar las acciones o eventos que han tenido lugar, de manera que dichas acciones o eventos no puedan ser repudiados posteriormente.
	Responsabilidad	Capacidad de rastrear de forma inequívoca las acciones de una entidad.
	Autenticidad	Capacidad de demostrar la identidad de un sujeto o un recurso.
Mantenibilidad	**Modularidad**	Capacidad de un sistema o programa de ordenador (compuesto de componentes discretos) que permite que un cambio en un componente tenga un impacto mínimo en los demás.
	Reusabilidad	Capacidad de un activo que permite que sea utilizado en más de un sistema software o en la construcción de otros activos.
	Analizabilidad	Facilidad con la que se puede evaluar el impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar.
	Capacidad para ser modificado	Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.

	Capacidad para ser probado	Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.
Portabilidad	**Adaptabilidad**	Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de hardware, software, operacionales o de uso.
	Capacidad para ser instalado	Facilidad con la que el producto se puede instalar y/o desinstalar de forma exitosa en un determinado entorno.
	Capacidad para ser reemplazado	Capacidad del producto para ser utilizado en lugar de otro producto software determinado con el mismo propósito y en el mismo entorno.

El estándar es regulado por *International Organization for Standardization (ISO)*

2.4 Tecnologías entorno a *Head End System* (Caso de prueba)

Es necesario conocer sobre las nuevas tecnologías que están innovado la forma de ofrecer servicios públicos para mejorar la calidad humana que permiten a las ciudades poder controlar de manera eficiente las demandas de millones de usuarios facilitando el acceso a los diversos servicios públicos con excelente calidad (electricidad, gas, agua, etc.).

La metodología de aseguramiento de calidad tiene la misión de evaluar un sistema de tipo *Head End* como un caso de prueba para implementar una nueva metodología que propone en esta tesis. A continuación, se describirán las tecnologías que tienen relación con el *Head End System.*

Smart cities

Smart cities es un proyecto para transformar servicios generales en servicios inteligentes, agregando módulos de firmware en las tablas de control de servicios junto con la tecnología inalámbrica, haciendo uso de internet de las cosas *(IoT).*

Una *Smart cities* se define como un sistema complejo e interconectado que aplica las nuevas tecnologías para gestionar desde el correcto funcionamiento de los sistemas de transporte público y privado, hasta el uso eficiente de los recursos energéticos o hídricos, pasando por los planos de protección civil, o aspectos socio-económicos, como la vitalidad de los espacios públicos y del tejido comercial, o la comunicación de incidencias a habitantes y visitantes. (Rueda, 2017)

Los gobiernos están decidiendo adoptar la idea de ser una ciudad inteligente para ofrecer mejores servicios, generar economía y mejorar su infraestructura. (Vásquez y Estrada, 2013).

Debido al aumento de habitantes en las áreas metropolitanas se tiene la necesidad de controlar los servicios de manera eficiente para ello surgieron los *Smart cities* "Una ciudad inteligente detecta las necesidades de sus ciudadanos, y reacciona a estas demandas transformando las interacciones de los ciudadanos con los sistemas y elementos de servicio público en conocimiento. Así, la ciudad basa sus acciones y su gestión en dicho conocimiento, idealmente en tiempo real, o incluso anticipándose a lo que pueda acaecer", explica Juan Murillo, responsable de Análisis Territoriales de *BBVA Data & Analytics*. (Rueda, 2017)

En la figura 2.9 se puede observar las distintas tecnologías que se relacionan en el mundo de los *Smart cities.*

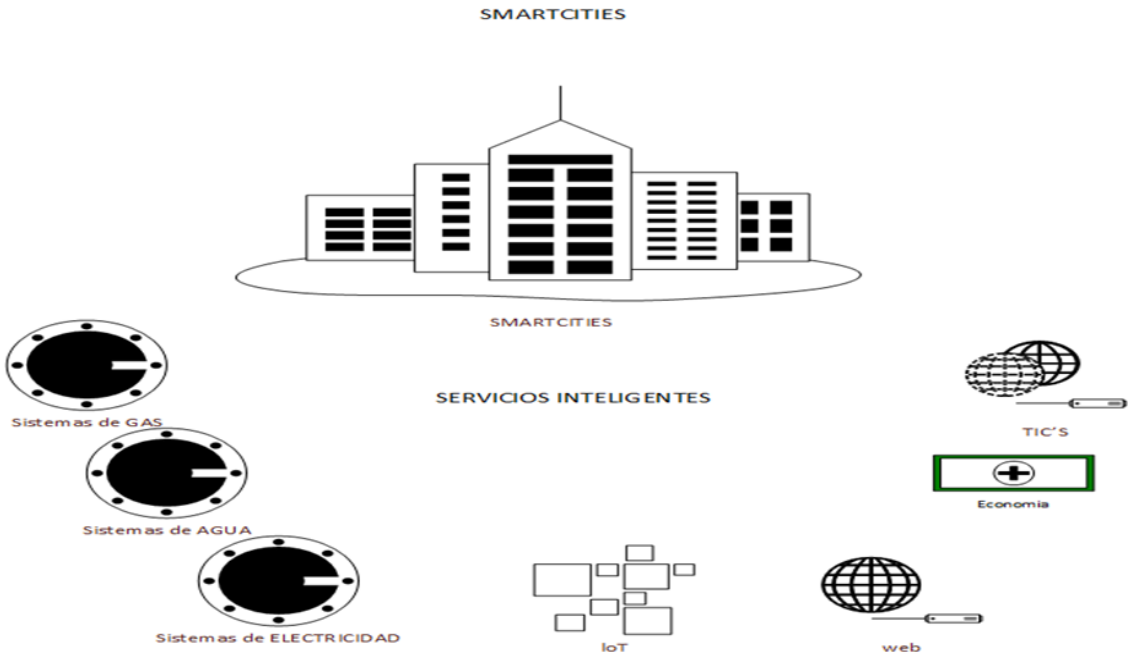


Figura 2.9.- Tecnologías en Smart cities (Elaboración propia).

The Advanced Metering Infrastructure (AMI)

La infraestructura de medición avanzada *(AMI)* se estructura típicamente en un conjunto de redes y se compone de algunos componentes principales. La Figura 2.10 proporciona una descripción general de todos los componentes y la mayoría de las redes. Está compuesto por el medidor, el recopilador y los sistemas de servidor en el operador del sistema de distribución (DSO) o en el lado de la compañía de medición. (Brunschwiler, 2013)

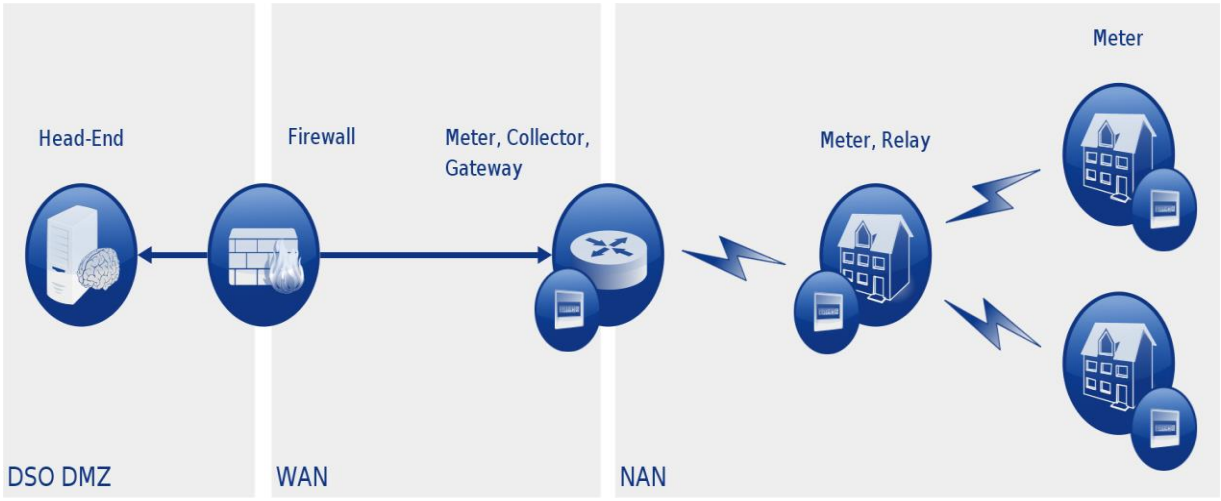


Figura 2.10 Advanced Metering Infrastructure Networks and Components (Brunschwiler, 2013).

AMI es una tecnología en transformación que tiene un amplio impacto en el mercado de la energía y sus consumidores. *AMI* permite a las empresas de servicios públicos equilibrar el suministro, la demanda y la capacidad, lo que hace que la red sea más inteligente y eficiente al impulsar aspectos de control de la red hasta los puntos finales de la entrega. Las partes interesadas están implementando los sistemas y las tecnologías requeridas para implementar *AMI*. (SmartGrid, 2010)

AMI es un sistema de comunicación bidireccional que puede llegar a todos los dispositivos en el espacio de distribución. El enfoque de la industria al adoptar *AMI* en lugar de *Automatic Meter Reading (AMR)* es que el sistema de comunicación no está dedicado a una sola aplicación. En cambio, *AMI* es un sistema de comunicación flexible y de propósito general que se puede usar para muchas aplicaciones, incluida la lectura de medidores, la automatización de la distribución, la conexión / desconexión, prometen proporcionar monitoreo y registro de energía avanzado, sofisticada recopilación de datos de tarifas, programas tarifarios, capacidades de comando y control de gestión de carga. Además, estos poderosos mecanismos permitirán a los consumidores administrar mejor su energía. (SmartGrid 2010)

Head End System

El *Head End System (HES)*, es un sistema que se presenta como una alternativa de nivel tecnológico en *Smart cities*, para tener la comunicación y el control de los servicios prestados a la ciudadanía con el apoyo de medidores inteligentes, facilitando el trabajo a las compañías responsables a la gestión de *Utility´s* (servicios públicos como el agua, el gas, la electricidad, etc.)

El *HES*, también conocido como sistema de control del medidor, se encuentra dentro de una red de empresas de medición. En la mayoría de los casos, la empresa de medición es el operador del sistema de distribución *(DSO)* responsable. El *HES* se está comunicando directamente con los medidores. Por lo tanto, el *HES* se encuentra en alguna zona desmilitarizada (*DMZ*) ya que los servicios y la funcionalidad se proporcionarán al exterior. (Brunschwiler, 2013)

Hay mucha más infraestructura en DSO o en el lado de la compañía de medición. Los datos recopilados se gestionarán dentro de un sistema de administración de datos de medición (MDM) que también mapea los datos al consumidor relevante. Según el nivel de automatización, los datos de medición tendrán influencia en las acciones de DSO para equilibrar la cuadrícula. (Brunschwiler, 2013)

Un *HES* es hardware y software que recibe la secuencia de datos del medidor que se devuelve a la utilidad a través de la *AMI.* Los sistemas de cabecera pueden realizar una cantidad limitada de validación de datos antes de hacer que los datos estén disponibles para otros sistemas para solicitar o enviar los datos a otros sistemas. (Brunschwiler, 2013)

Building Blocks

Building Blocks es un propuesta de un *Framework* para diseñar un entorno o sistema que Ayuda a modularizar components de softeware en bloques indpendiente, los bloques pueden ser duros, blandos y conectores. Además se pueden dividir en bloques sistémicos y de nivel de aplicación. Los bloques duros son una combinación de *software* y *hardware*. Los bloques blandos son entidades de *software.* (Radhakrishnan, 2004).

Según Radhakrishnan (2004) los Building Blocks se basa en las siguientes propiedades:

- Escalable horizontalmente: es posible replicar el *Building block*s varias veces para escalar el nivel de servicio que proporciona.
- Basado en estándares: *Building blocks* es capaz de soportar muchos tipos diferentes de aplicaciones.
- Personalización: *Building blocks* se puede usar para implementar nuevas aplicaciones con una personalización mínima.
- Reutilizables: los *Building blocks* livianos se pueden reutilizar para construir otras aplicaciones.
- Portable: los *Building blocks* livianos se pueden transportar fácilmente a otras plataformas.
- Integrable: los *Building blocks* tienen interfaces estandarizadas que los hacen fáciles de integrar con otros componentes arquitectónicos.

Modelo Vista Controlador (MVC)

Es un patrón de arquitectura de las aplicaciones software que separa la lógica de negocio de la interfaz de usuario y facilita la evolución por separado de ambos aspectos, además de Incrementa reutilización y flexibilidad, la arquitectura fue descrito por primera vez en 1979 para Smalltalk por Trygve Reenskaug.

Reenskaug (1979) define el modelo como "una representación activa de una abstracción en forma de datos en un sistema informático".

Reenskaug (1979) describe la vista como "Para cualquier Modelo dado, se adjunta una o más Vistas, cada Vista puede mostrar una o más representaciones pictóricas del Modelo en la pantalla y en copia impresa. Una Vista también puede realizar tales operaciones sobre el Modelo que está asociado de manera razonable con esa Vista".

Reenskaug (1979) define al controlador como "El enlace entre un usuario y el sistema. Proporciona al usuario información al organizar vistas relevantes para que se presenten en lugares apropiados en la pantalla. Proporciona medios para la salida del usuario presentando al usuario los menús u otros medios para dar comandos y datos. El controlador recibe dicha salida de usuario, la traduce a los mensajes apropiados y pasa estos mensajes a una o más de las vistas".

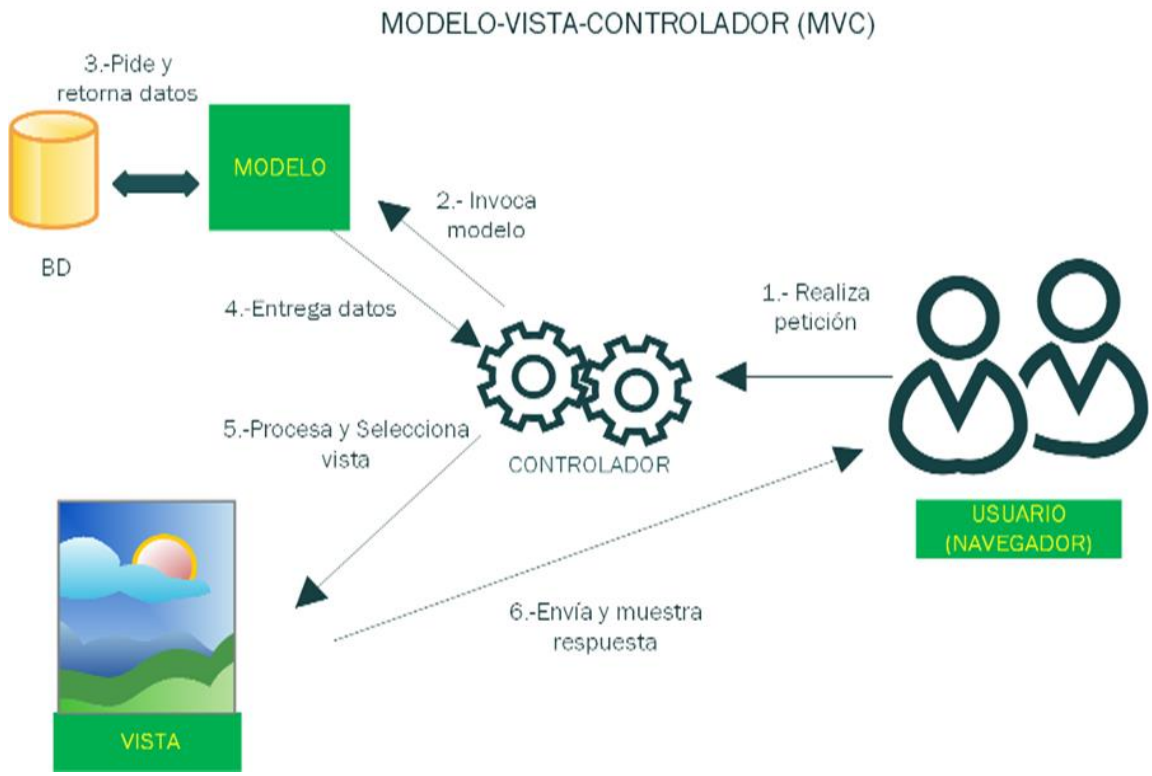


Figura 2.11.- Modelo-Vista-Controlador (Flores, Medina, Hernández y Sánchez, 2017).

Segun Pavón, (2009) el Modelo-Vista-Controlador está constituida por:

- Un modelo.
- Varias vistas.
- Varios controladores.
- Las vistas y los controladores suelen estar muy relacionados.
- Los controladores tratan los eventos que se producen en la interfaz gráfica (vista).
- Esta separación de aspectos de una aplicación da mucha flexibilidad al desarrollador.

Segun Pavón, (2009) el flujo de control de *MVC* es:

- El usuario realiza una acción en la interfaz
- El controlador trata el evento de entrada (Previamente se ha registrado)
- El controlador notifica al modelo la acción del usuario, lo que
- puede implicar un cambio del estado del modelo (si no es una mera consulta)
- Se genera una nueva vista.
- La vista toma los datos del Modelo
- El modelo no tiene conocimiento directo de la vista
- La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo

Cliente servidor

La computación cliente/servidor es un intento de equilibrar el proceso de una red hasta que se comparta la potencia de procesamiento entre computadoras que llevan a cabo servicios especializados tales como acceder a bases de datos (servidores), y aquellos que llevan a cabo tareas tales como la visualización GUI que es más adecuado para el punto final dentro de la red. (pressman, 2002)

Categorías de servidores

Existe una gran variedad de servidores que han sido desarrollados para ofrecer servicios de procesamiento, almacenamiento, comunicación entre otros, a continuación, se enlistan algunos tipos de servidores que pressman (2002) describe de manera puntual.

1.- Servidores de archivos.

Un servidor de archivos proporciona archivos para clientes. Estos servidores se utilizan todavía en algunas aplicaciones donde los clientes requieren un procesamiento complicado fuera del rango normal de procesamiento que se puede encontrar en bases de datos comerciales.

2.- Servidores de bases de datos.

Los servidores de bases de datos son computadoras que almacenan grandes colecciones de datos estructurados. Por ejemplo, un banco utilizaría un servidor de bases de datos para almacenar registros de clientes que contienen datos del nombre de cuenta, nombre del titular de la cuenta, saldo actual de la cuenta y límite de descubierto de la cuenta.

En un entorno de bases de datos cliente-Servidor los clientes envían las consultas a la base de datos, normalmente utilizando alguna GUI, estas consultas se envían al servidor en un lenguaje llamado SQL (Lenguaje de Consultas Estructurado). El servidor de bases de datos lee el código SQL, lo interpreta y, a continuación, lo visualiza en algún objeto tal como una caja de texto. El punto clave aquí es que el servidor de bases de datos lleva a cabo todo el procesamiento, donde el cliente lleva a cabo los procesos de extraer una consulta de algún objeto de entrada, tal como un campo de texto, enviar la consulta y visualizar la respuesta del servidor de la base de datos en algún objeto de salida, tal como un cuadro de desplazamiento. En la figura 2.12 se observa un esuqema del funcionamiento de la arquitectura Cliente-Servidor.

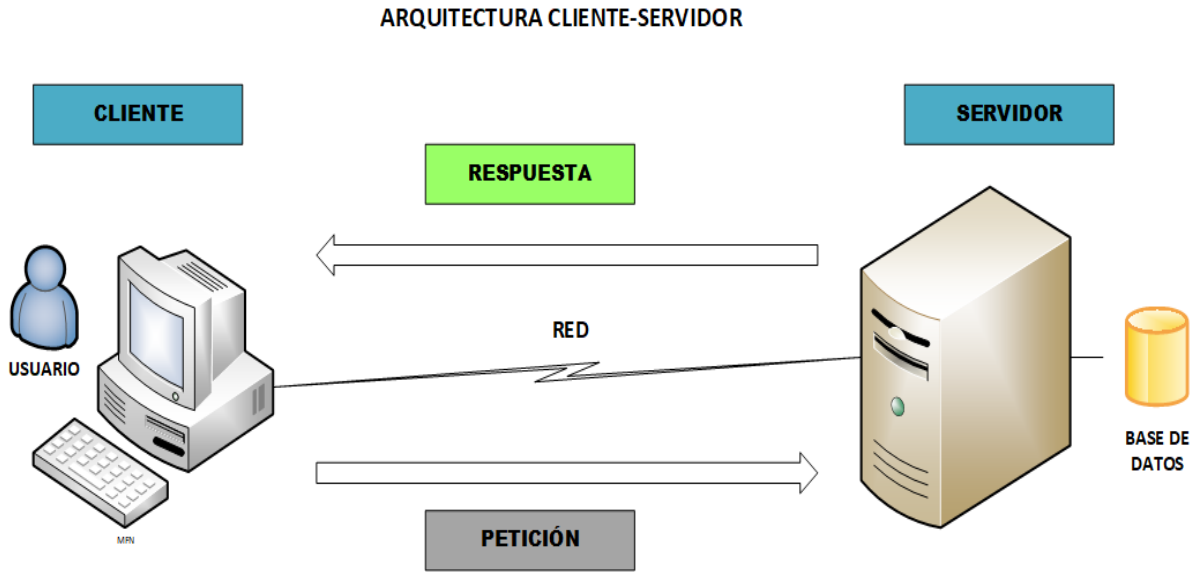


Figura 2.12.- Arquitectura Cliente-Servidor (Elaboración propia)

3.- Servidores de software de grupo.

Software de grupo es el término que se utiliza para describir el software que organiza el trabajo de un grupo de trabajadores.

Un sistema de software de grupo normalmente ofrece las siguientes funciones:

- Gestionar la agenda de los individuos de un equipo de trabajo.

- Gestionar las reuniones para un equipo.
- Gestionar el flujo de trabajo, donde las tareas se distribuyen a los miembros del equipo y el sistema de software de grupo proporciona información sobre la finalización de la tarea y envía un recordatorio al personal que lleva a cabo las tareas.
- Gestionar el correo electrónico, por ejemplo, organizar el envío de un correo específico a los miembros de un equipo una vez terminada una tarea específica.

Un servidor de software de grupo guarda los datos que dan soporte a estas tareas, por ejemplo, almacena las listas de correos electrónicos y permite que los usuarios del sistema de software de grupo se comuniquen con él, notificándoles, por ejemplo, que se terminado una tarea o proporcionándoles una fecha de reunión en la que ciertos empleados puedan acudir.

4.- Servidores Web.

Los documentos Web se almacenan como páginas en una computadora conocida como servidor Web. Cuando se utiliza un navegador *(browser)* para ver las páginas Web normalmente pincha sobre enlace en un documento Web existente. Esto dará como resultado un mensaje que se enviará al servidor Web que contiene la página. Este servidor responderá entonces enviando una página a su computadora, donde el navegador pueda visualizarlo. De esta manera los servidores Web actúan como una forma de servidor de archivos, administrando archivos relativamente pequeños a usuarios, quienes entonces utilizan un navegador para examinar estas páginas. Para comunicarse con un navegador Web, un cliente que utiliza un navegador está haciendo uso a su vez de un protocolo conocido como HTTP.

5.- Servidores de correo.

Un servidor de correo gestiona el envío y recepción de correo de un grupo de usuarios. Para este servicio normalmente se utiliza un *PC* de rango medio. Existen varios protocolos para el correo electrónico. Un servidor de correo estará especializado en utilizar solo uno de ellos.

6.- Servidores de objetos.

Los objetos que se pueden almacenar en una computadora, normalmente un servidor, con clientes capaces de activar la funcionalidad del objeto enviando mensajes al objeto, los cuales se corresponden con métodos definidos por la clase de objeto. Esta tecnología liberará finalmente a los programadores de la programación de bajo nivel basada en protocolos requerida para acceder a otras computadoras de una red. En efecto, esto permite que el programador trate a los objetos a 'distancia como si estuvieran en su computadora local. Un servidor que contiene objetos que pueden accederse a distancia se conoce como servidor de objetos.

7.- Servidores de impresión.

Los servidores de impresión dan servicio a las solicitudes de un cliente remoto. Estos servidores tienden a basarse en *PCs* bastante baratos, y llevan a cabo las funciones limitadas de poner en cola de espera las peticiones de impresión, ordenar a la impresora que lleve a cabo el proceso de impresión e informar a las computadoras cliente que ya ha finalizado una petición de impresión en particular.

8.- Servidores de aplicaciones.

Un servidor de aplicaciones se dedica a una aplicación única. Tales servidores suelen escribirse utilizando un lenguaje tal como *Java o C++*. Un ejemplo típico del servidor que se utiliza en el dibujo de un fabricante de aviones que gestionaba las versiones diferentes de dibujos producidos por el personal técnico iría dirigido a algún proceso de fabricación.

Conclusión

En el capítulo se pudieron conocer una variedad de metodologías de desarrollo de software que se implementan dependiendo de la magnitud y del tipo de software, lo más común son los llamados metodologías clásicas y metodologías agiles.

Las metodologías clásicas implementan procesos formales donde cada fase se deben cumplir para poder continuar con el siguiente, bien definido y diseñado los requerimientos como base antes de codificar. Estas son implementadas en proyectos a largo plazo, su evolución de desarrollo es de manera lineal.

Las metodologías agiles implementan procesos informales por los constantes cambios en los requerimientos y las constantes soluciones, estos se implementan en proyectos a corto plazo, su evolución de desarrollo es de manera iterativa e incremental, el equipo de desarrollo mantiene reuniones diarias para conocer las situaciones para resolver algún problema que impida avanzar.

Lo mencionado anteriormente, es para explicar que no existe una metodología que se pueda ocupar en todas las diferentes formas de desarrollar un software, así también de la misma forma existen una variedad de procesos y metodologías de calidad que se adaptan a los distintos marcos de desarrollo.

Existen diferentes soluciones que han permitido atacar la problemática de proyectos fallidos o clientes insatisfechos, algunas soluciones han sido mejorando procesos, presentando nuevas propuestas como modelos o metodologías de calidad que han sido implementados con éxito, pero debido a las versatilidades que existen para desarrollar un software, han hecho que los equipos de aseguramiento de calidad tengan la necesidad de rediseñar sus propuestas de evaluación de calidad para poder implementarlo en proyectos específicos para su buen funcionamiento.

Otra de las situaciones que surgen durante una evaluación del software, es que el equipo de desarrollo no ve al equipo de calidad como un aliado más en la mejora de los productos que desarrollan, sino como un equipo a vencer, por lo que es necesario la comunicación entre estos equipos para llevar el proyecto al éxito y en caso de fallas tomar las decisiones en común de manera eficiente para su pronta corrección. De acuerdo al análisis realizado en los estándares y modelos de evaluación de calidad, se comprende que los modelos de *CMMI Y MOPROSOFT* no cumplen con las necesidades que se tiene para evaluar un producto de software porque la esencia de los modelos se enfoca en evaluar el nivel de madurez de los procesos de calidad en el desarrollo de software para empresas pymes y de gran envergadura por lo cual no puede ser implementado en la propuesta de metodología.

Respecto con la información sobre el modelo de *McCALL* se puede saber que es pionero en la forma de evaluar la calidad de un producto de software y de acuerdo con los atributos que propone como características de calidad, se puede confirmar que el modelo es la base principal de *FURPS+, ISO 9126* este último fue influenciado en la *ISO 25010 (SQUARE)* que retoma parte de sus características y sub características de calidad. Aunque es importante mencionar que en los estándares mencionados contienen grandes mejoras en sus atributos de evaluación fortaleciendo así la calidad del software.

El objetivo de este análisis es poder implementar una evaluación estandarizada en la metodología de aseguramiento de calidad de esta tesis, por lo tanto se toma la decisión de optar por la *ISO 9126*, debido a que el estándar proporciona suficiente información sobre la forma de evaluar un software con características de calidad interna y externa, pero sobre todo que contiene un seguimiento al producto con la evaluación de calidad de uso que permite dar seguimiento al software cuando ya se encuentra en funcionamiento real siendo manipulado por el usuario final.

Por último, el responsable en evaluar la funcionalidad y calidad del sistema debe tener el conocimiento suficiente sobre las tecnologías que se involucran en le sistema *HES*, esto con la finalidad de visualizar el entorno donde estará funcionando para generar pruebas mas exactas, así como para conocer las tecnologías que se utilizarán para desarrollar dicho sistema con la intención de preparar las herramientas correctas para ejecutar las pruebas que comprobaran el buen funcionamiento del *HES*.

Capítulo 3

Metodología

Introducción

En este capítulo 3 se describe una propuesta de metodología de aseguramiento de calidad *VEyVAA* (Verificación y Validación Ágil) que será capaz de adaptarse en el desarrollo Scrum. El capítulo se divide en 3 partes importantes que son:

- La verificación
- La validación
- Formatos a utilizar

En la primera parte se describe el proceso que se debe llevar acabo en la verificación de módulos funcionales del sistema, también se describe el cómo se debe elaborar el plan de pruebas, la utilización de herramientas y la ejecución de las pruebas, por último, se describe quien es el responsable en aplicar las actividades de verificación, así como las interacciones que tendrá con el equipo de Scrum.

En la segunda parte se describe el proceso que se debe llevar acabo en la validación del sistema en base al estándar de calidad *ISO/IEC 9126* del cual se explica cada característica del estándar, también se describe como se debe elaborar el plan de pruebas para evaluar la calidad del sistema y el cumplimiento de los requerimientos, así como también la utilización de las herramientas y la aplicación de pruebas, por último se describe quien es el responsable de aplicar las actividades de validación y las interacciones que tiene con los *Stakeholders* y equipo *Scrum*.

En la tercera parte se describe la utilización de los formatos para cada tipo de evaluación, donde se registrarán los datos generales de las pruebas, planeación de las pruebas y los resultados obtenidos en la verificación y validación.

3.1 Esquema general de la metodología *VEyVAA*

Se propone una metodología de aseguramiento de calidad de software, diseñado para ser implementado dentro del desarrollo de software ágil *Scrum* con la arquitectura *MVC* y con el enfoque de *Building Blocks.*

Se propone la implementación de la verificación y validación en el aseguramiento de calidad de forma ágil, pero con un enfoque diferente en la evaluación del sistema y sus componentes. Por lo que se obtuvo un nuevo modelo de aseguramiento de calidad de software llamado *VEyVAA* (Verificación y Validación Ágil).

En la metodología *VEyVAA* se propone dos responsables que fungen como *tester*, uno se enfocara en la evaluación de verificación y el otro en la evaluación de validación, cada tipo de evaluación integra sus respectivos procesos, su plan de pruebas, la interacción con el equipo de *Scrum* y los *stakeholders*, así como sus propios formatos para registrar datos y resultados.
La implementación de la metodología *VEyVAA* contiene diversas utilidades para evaluar un software que a continuación se mencionan:

- En la metodología se describen los procesos que se deben llevar acabo para evaluar de manera eficiente la calidad del software y para detectar de manera temprana los errores existentes para su pronta corrección.
- La metodología evalúa la calidad de un software mediante procesos, estándares, planificación y ejecución de actividades que se deben llevar acabo, para poder lograr una buena toma de decisiones en la verificación y validación del software, que son parte fundamental dentro de la metodología, siendo estas las bases para evaluar durante el desarrollo y finalización del producto.
- La metodología se adapta a la filosofía de *building blocks* sobre la modularización del software en bloques funcionales e independientes, por lo que la metodología enfoca una parte de sus procesos en evaluar cada uno de esos módulos desarrollados en scrum, que constituyen los llamados *Sprint*.
- La metodología también se adapta al patrón de arquitectura de software modelo-vista-controlador que separa en tres capas a un sistema, como son los datos, las interfaces y los controladores, esta separación es un beneficio para la metodología que logra clasificar cada módulo a su correspondiente capa y realizar pruebas conforme a su funcionamiento.
- La metodología debe ser capaz de guiar a los responsables para evaluar la calidad de un sistema, así como en la toma de decisiones para notificar correcciones o mejoras en el producto.
- La metodología se desarrolla en la evaluación de la calidad de los componentes de un software y la evaluación de calidad de un sistema completo, teniendo así dos enfoques diferentes para poder abarcar lo más posible de pruebas que nos permita detectar errores.
- La metodología tiene como filosofía de romper con las barreras que existen entre áreas de desarrollo de software y el equipo de aseguramiento de calidad para aumentar la comunicación entre ellas, teniendo como consecuencia una eficiente solución a los problemas que se lleguen a suscitar y que los procesos de desarrollo sean evaluados por *testers* con pensamiento objetivo y subjetivo.

Los dos enfoques que se implementan en los procesos de pruebas de la metodología, son la verificación de los componentes y la validación del sistema completo como se observa en la figura 3.1.

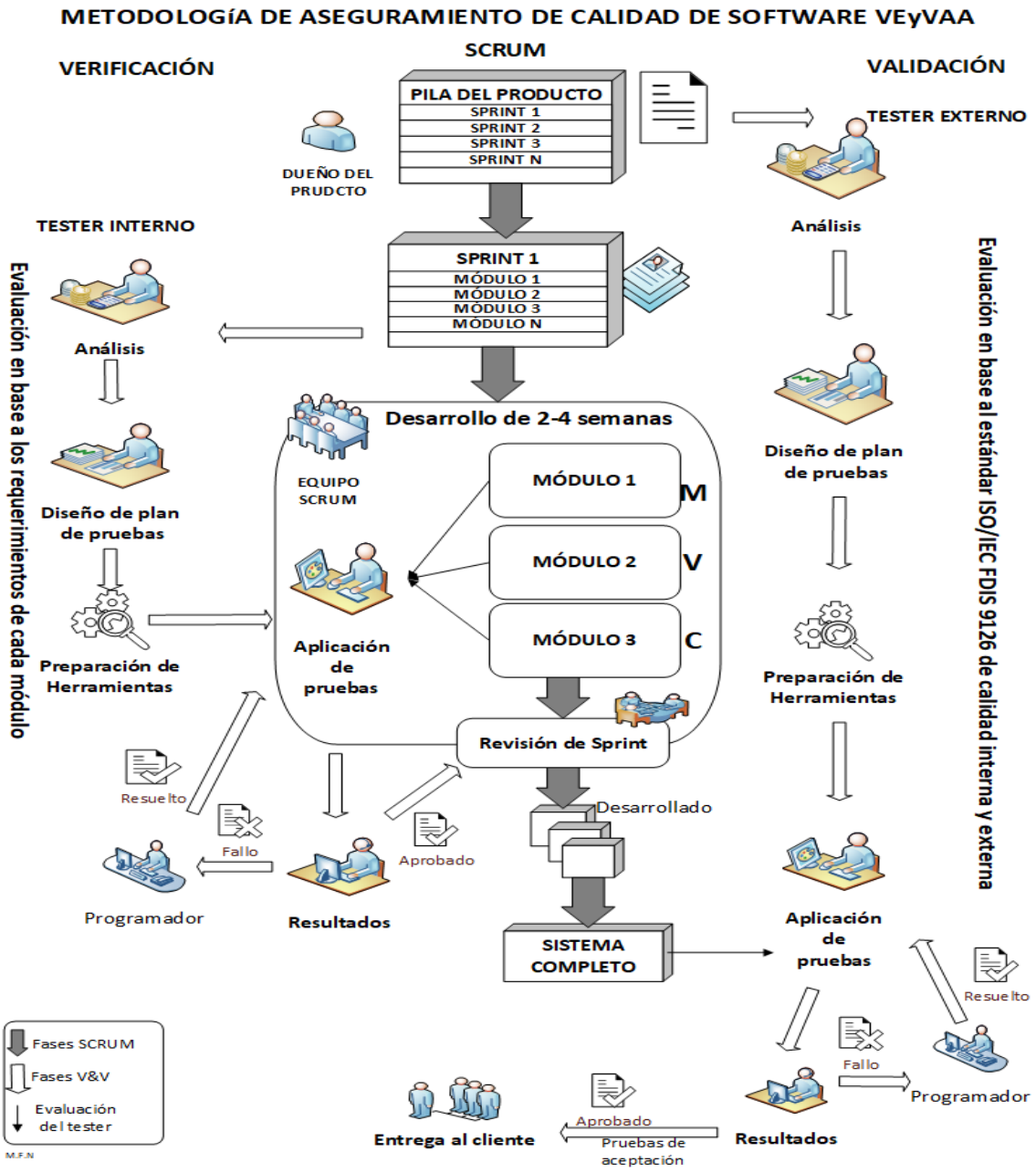


Figura 3.1.- Esquema de la metodología VEyVAA (Elaboración propia).

Cada una de las formas de evaluación tiene sus propios criterios, el primero de ellos es verificar que los requerimientos de cada componente se encuentren implementados y el segundo se encarga de validar que el sistema cumpla con el estándar de calidad ISO 9126.

Framework de desarrollo ágil *SCRUM*

En la parte central de la metodología que se muestrea en la figura 3.1 contiene las funciones que conlleva el *framework* de desarrollo ágil scrum que en el capítulo 2 de esta tesis se ha explicado de manera puntual cada una de las fases, aunque se han hecho algunos ajustes para el desarrollo del sprint en la forma de clasificar cada módulo a desarrollar, por la implementación de las tres capas de *MVC* y el enfoque de *Building blocks* para crear bloques (módulos) independientes.
De manera resumida se explica el funcionamiento de las actividades que se involucran en el *framework Scrum*, para comprender la interacción que mantiene el *framework* con la metodología *VEyVAA*.

En *Scrum* intervienen 3 tipos de autores que son responsables en todo el desarrollo del software: el dueño del producto, el *Scrum Master* y el equipo *Scrum*, todos tienen objetivos en común al desarrollar un software, como trabajar en equipo en el menor tiempo posible, también el de satisfacer las necesidades del cliente y superar sus expectativas sobre su producto.

El proyecto en *Scrum* inicia con la creación de una lista priorizada de características deseables que debe contener el software en la finalización de su desarrollo, esta lista llamada pila del producto está bajo la responsabilidad del dueño del producto, en su función también representa los intereses del cliente, por lo que se encarga de desarrollar tareas que cubren las necesidades más importantes del cliente para su pronta creación.

En la Pila del Producto se crean bloques de trabajo de mayor a menor priorización, cada bloque es nombrado como *Sprint*, cada *Sprint* se deben desarrollar en un lapso de tiempo de 1 a 4 semanas, el *sprint* contiene un conjunto de componentes o módulos de software.
Cuando se realiza la planificación del *Sprint* en el lapso de un día, el equipo de *Scrum* selecciona y analiza cada característica y cada requerimiento como un objetivo que debe cubrir cada módulo que conforma el *Sprint* y asignar responsabilidades a cada integrante del equipo para que al término del tiempo se obtengan entregables funcionales para el cliente.

Durante el desarrollo del *Sprint* el *Scrum Master* funge como líder de los desarrolladores de los *sprint* llamados equipo de *Scrum*, ellos mantienen una constante comunicación para la toma de decisiones, una de las formas de comunicación que realiza el equipo y el scrum master son las reuniones de 15 minutos cada día. En las reuniones diarias con el equipo de *Scrum* se obtiene información sobre el progreso y limitaciones de cada integrante, el *Scrum master* realiza 3 preguntas importantes, ¿qué has hecho? ¿qué harás? y ¿qué te impide hacerlo? y en base a las respuestas por parte del equipo de desarrollo el *Scrum master* aconseja y toma decisiones en conjunto para alcanzar los objetivos.

Cuando él *sprint* se termina de desarrollar se realiza un evento llamado Revisión del *Sprint* con todos los integrantes de *Scrum* y *Stakeholders*, la actividad consiste en una reunión donde el *Scrum Master* y el dueño del producto realizan una demostración del funcionamiento del sistema en base a los requerimientos del *Sprint*, los *stakeholders* dan su opinión sobre el *Sprint* además de su visto bueno para continuar con el siguiente *Sprint.*

Cabe mencionar que en la implementación de la arquitectura *MVC* en *Scrum* se clasificaron los módulos en sus capas correspondientes de la arquitectura y fueron asignados a cada programador una capa según su especialidad.

Antes de continuar con el siguiente *Sprint* el equipo *Scrum* y el dueño del producto realiza un último evento llamado retrospectiva *Scrum*, esto consiste en analizar lo sucedido durante el desarrollo sobre las cosas que se hicieron mal para mejorarlo en el siguiente *Sprint*, también se toma en cuenta las cosas que se hicieron bien para mantenerlas y por último sobre los asuntos que impidieron avanzar afectando lo que se había planeado, para tomar medidas de mejora en el próximo trabajo.

3.2. Verificación

La verificación es una de las formas de evaluación que contiene la metodología (la otra es la validación), que se encarga de verificar que el desarrollo del *Sprint* sea conforme a los requerimientos solicitados.

El enfoque de la verificación es evaluar de manera objetiva cada módulo del *sprint* que desarrolle el equipo de *Scrum*, por lo que la ejecución de pruebas será para comprobar que el módulo realiza las funciones de manera correcta y en caso contrario tomar medidas o soluciones prontas para avanzar en la evaluación.

La clasificación de los módulos que contiene cada *Sprint* se basa en las tres capas del modelo *MVC* (Modelo-Vista-Controlador), de esta forma el *tester* organiza e identifica las pruebas que aplicará para comprobar el funcionamiento de cada módulo dependiendo del tipo de capa en la que será implementado.

3.2.1 Proceso de verificación

El proceso de verificación inicia a partir de la fase de la planeación del sprint, en esta fase el *Sprint* ya ha sido seleccionado, por lo tanto, el *tester* interno que es el responsable de esta fase empieza su trabajo analizando e identificando los requerimientos que deben cumplir cada uno de los módulos que conforma el *Sprint*.

Después del análisis el *tester* continua con la planeación de pruebas que debe ejecutar para demostrar la implementación correcta de los requerimientos en cada módulo, en esta fase se toman en cuenta, las técnicas de pruebas que se van a ejecutar y las herramientas que se van a utilizar para la ejecución de las pruebas, los criterios de evaluación y parámetros de resultados.

Cuando ya se tienen identificados los requerimientos, así como las pruebas para comprobar los requerimientos y las herramientas que ejecutaran las pruebas, se necesita un espacio y tiempo para preparar el entorno donde se ejecutaran las pruebas con las herramientas seleccionadas con anticipación. Es muy necesario esta fase debido a que el *tester* invierte un tiempo considerable en instalar y configurar las herramientas, además de aprender o mejorar su manipulación.

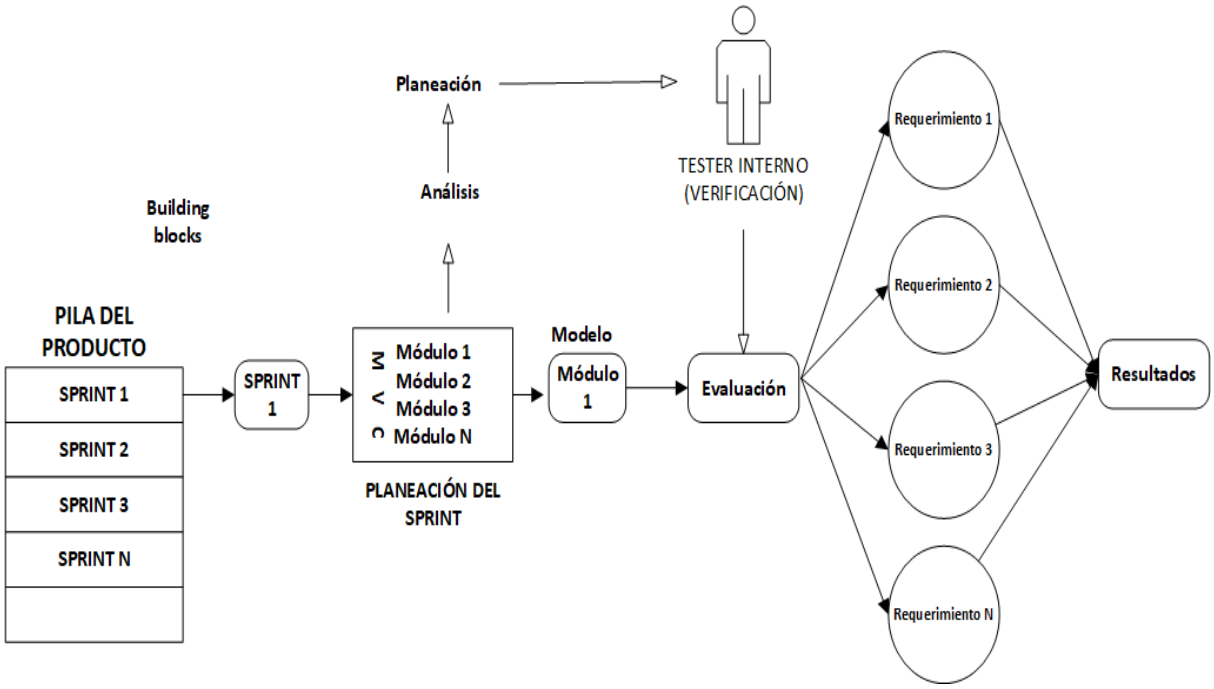


Figura 3.2.- Esquema del funcionamiento de la verificación (Elaboración propia).

Como se puede observar en la figura 3.2 la evaluación del *tester* de verificación enfoca sus esfuerzos en los módulos de cada sprint dando una buena ventaja para detectar fallas en fases tempranas, además de que la verificación se integra al equipo de scrum para estar en comunicación directa con los programadores en caso de realizar acciones correctivas.

En la clasificación MVC el *tester* toma ventaja para evaluar los módulos, realizando una clasificación de sus pruebas para definir el tipo de pruebas que puede aplicar en los módulos de control, en los módulos de la vista y en los módulos del modelo, de esa forma organiza de manera eficiente su plan de pruebas.

Cuando las fases anteriores se han cumplido entonces el *tester* se encuentra listo para aplicar el plan de pruebas, en esta fase se ejecutan todas las pruebas planeadas a cada módulo por lo que significa que existirá siempre un plan de pruebas exclusivo para cada módulo, debido a que los módulos se diferencian en sus funciones y en sus distintos requerimientos.

En la figura 3.3 se observa el proceso que debe llevar acabo el *tester* interno para la aplicación de las pruebas a un módulo funcional.

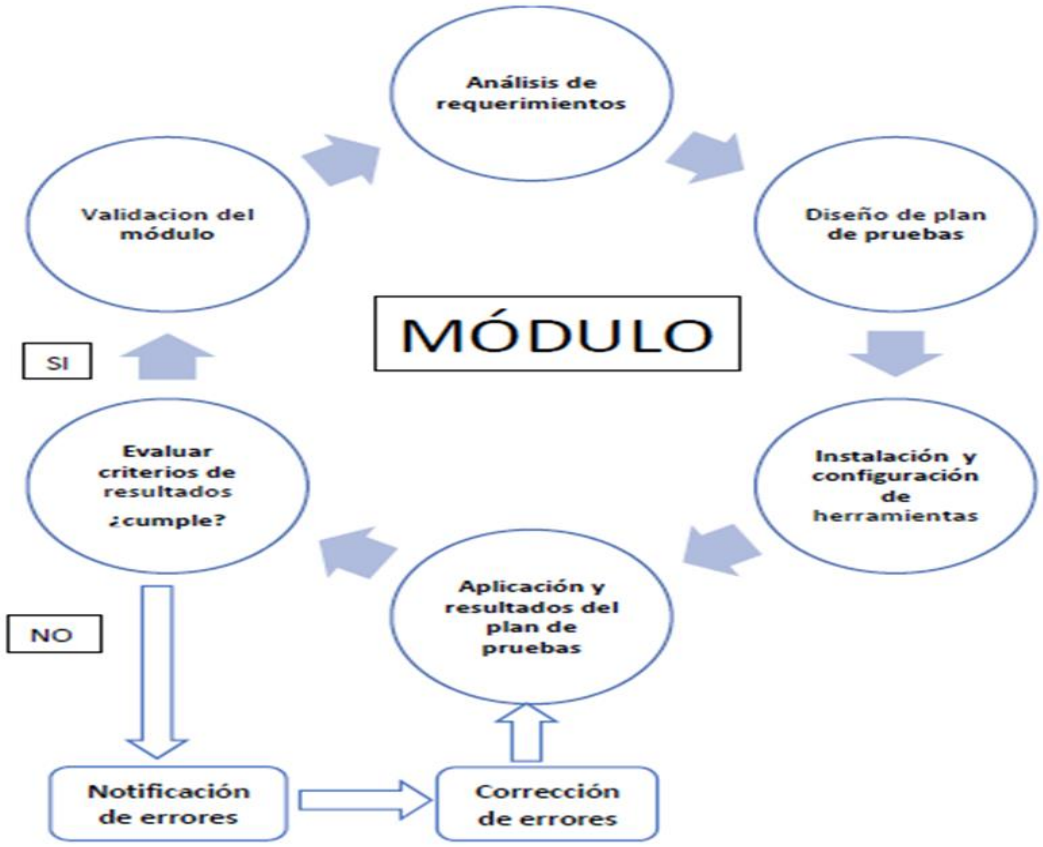


Figura 3.3.- Proceso de aplicación de pruebas a módulo (Elaboración propia).

Cuando se termina de aplicar el plan de pruebas obtenemos los resultados y en base a los criterios de cada plan se toman las decisiones de continuar con los siguientes *sprint* o se tendrán que notificar la existencia de errores al programador encargado del módulo para su pronta corrección.

Cuando se realiza las correcciones por parte de los programadores, estos solicitan nuevamente la aplicación del plan de pruebas para confirmar que ya se cumple con lo deseado, si en esta ocasión los resultados si cumplen con los criterios de evaluación entonces se toma la decisión de liberar el módulo para continuar con el siguiente módulo.

3.2.2 Diseño de un plan de pruebas de verificación

La elaboración del plan de pruebas para la verificación de los módulos depende de los requerimientos y de las pruebas, por lo tanto, la magnitud del plan dependerá de la cantidad de requerimientos que deben estar implementados en el módulo a evaluar y de la cantidad de pruebas necesarias para comprobar en el módulo el buen funcionamiento y el cumplimiento de los requerimientos.

Como se puede observar en la figura 3.4, un módulo puede contener un conjunto de requerimientos comúnmente funcionales por lo que es importante verificarlo, un requerimiento puede ser un criterio de evaluación y para poder evaluarlo se necesita uno o más técnicas de pruebas y dentro de esas pruebas se pueden desglosar los casos de prueba y así confirmar que cumple con lo especificado.

Las pruebas seleccionadas con sus respectivos casos de prueba conforman los puntos más importantes que el *tester* debe evaluar, pero también es la parte fundamental de los conocimientos que debe tener el *tester* responsable de la evaluación, ya que debe conocer un catálogo suficiente de técnicas de pruebas para que la evaluación sea rápida y eficiente porque así lo exige *Scrum.*

La evaluación no debe afectar los tiempos de entrega del módulo, por lo que en la selección de pruebas deben ser directas y claras, se pueden aplicar pruebas que permitan explorar de manera manual el funcionamiento correcto del módulo o en su caso aplicar pruebas automatizadas.

Los criterios de evaluación dependerán de la exigencia de cada requerimiento, el criterio se formula en base en que uno o más casos de prueba cumplen con el requerimiento entonces se realiza la afirmación de un “si cumple” y en caso contrario es “no cumple” con sus respectivas observaciones.

Los errores detectados se notifican lo más pronto posible al programador para que tome cartas en el asunto de corrección y sean nuevamente evaluadas hasta que cumpla los criterios de la prueba.

Por ejemplo, en el requerimiento 1 se solicita que el módulo *“login”* debe contener una seguridad de acceso, para comprobar la seguridad aplicamos pruebas de caja negra como la de ingresar datos en los campos usuario y contraseña, en el desglosamos tres casos de prueba el primero que podrían ser:

- Insertar números y letras en el campo usuario y contraseña.
- Inyección de código para comprobar que el sistema rechaza caracteres inválidos.
- No insertar datos y dar clic en el botón aceptar para el acceso al sistema (no permite acceso).

Los criterios serian: si en el primer caso de prueba el *login* no contiene el campo de *password* entonces se define como una prueba fallida. En el segundo caso si el sistema no acepta código en los campos entonces la prueba es exitosa y por último si da acceso sin insertar datos existe un error de seguridad.

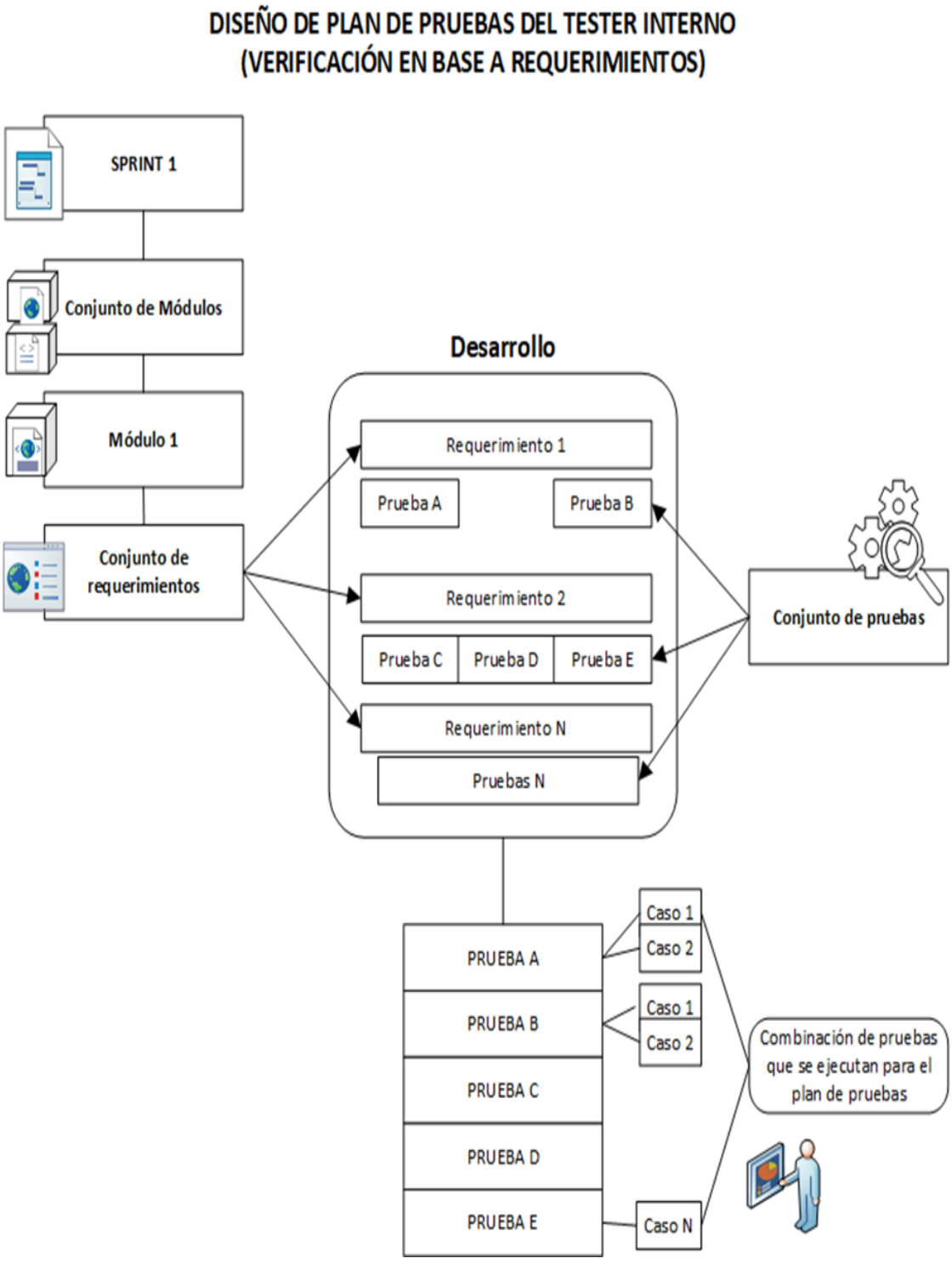


Figura 3.4.- Análisis y creación del plan de pruebas para la verificación (Elaboración propia).

3.2.3 Interacción del tester interno con el equipo de Scrum

La comunicación en el equipo scrum es de vital importancia para la realización de actividades o eventos para que el equipo este informado sobre los avances o cambios que hayan suscitado, además que permite agilizar los procesos de desarrollo y de soluciones prontas cuando surge algún problema.

La intervención del *tester* interno dentro del equipo scrum, debe ser visto como un aliado más que llega a sumar y mejorar todo lo que se está desarrollando, por lo que se propone que debe formar parte de las reuniones diarias que tiene el equipó, además también debe ser tomado en cuenta en la toma de decisiones, ya que su forma de evaluar es comprobar que los módulos están funcionando conforme el equipo lo ha planeado.

En la figura 3.5 se observa como interactúan los autores responsables de scrum con el *tester* interno, el primer autor con el que tiene comunicación el *tester* interno es con el scrum master durante la fase de la planeación del sprint, durante esta fase trabajan en conjunto para identificar los módulos a desarrollar y como consecuencia los módulos a evaluar, los dos individuos tienen objetivos en común en lograr crear un sprint de buena calidad.

El equipo de desarrollo de scrum y dueño del producto también tienen interacción con el *tester* interno durante los eventos de reuniones diarias, en las revisiones del *Sprint* y en las retrospectivas del S*print*, esto con el fin de comunicar todos los pormenores del sprint a todo el equipo que interviene en el proceso de construcción. El *tester* enfoca de manera más efectiva las evaluaciones con los objetivos del equipo.

El *tester* interviene como un visor desde una perspectiva diferente a los programadores al conocer las debilidades y fortalezas de cada integrante del equipo, donde de manera objetiva el *tester* se esfuerza en apoyar con ideas en la solución de problemas o en la identificación de manera rápida las debilidades cada desarrollo.

Durante el proceso de evaluación de cada módulo, el *tester* mantiene una constante comunicación con el programador que lo desarrolló, por la razón de que en caso de existir algún tipo de error o falla, el *tester* notifique de forma eficiente al programador de las correcciones que debe realizar lo más pronto posible para volver a aplicar pruebas y no se pierda tiempo útil en las evaluaciones.

El trabajo en conjunto entre desarrolladores y *testers* es lo que apuesta esta metodología para abarcar de manera más amplia la evaluación de calidad durante el desarrollo, de esta forma rompe las fronteras que existen entre áreas y permite trabajar como un solo equipo para lograr los objetivos de lograr un software de calidad que supere las expectativas del cliente.

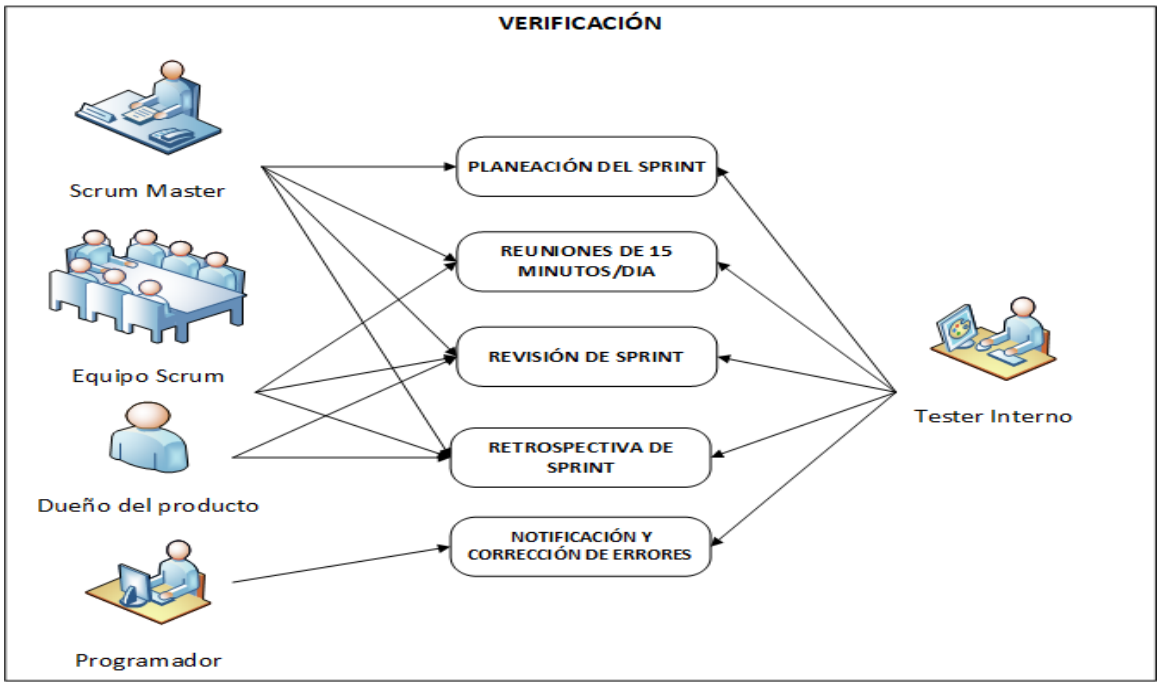


Figura 3.5.- Interacción entre el equipo de scrum y el tester interno (Elaboración propia).

3.3 Validación

La validación es la otra parte del proceso de evaluación que contiene la metodología *VEyVAA*, este proceso se encargara de evaluar cuando el software se encuentre terminado.

La evaluación para validar el sistema se enfocará en comprobar el cumplimiento de los requerimientos solicitados del cliente y la existencia de características de calidad interna y externa de la *ISO/IEC FDIS 9126.*

3.3.1 Estándar ISO/IEC FDIS 9126 y Métricas

Las características de calidad interna y externa de la *ISO/IEC 9126*, clasifica los atributos de calidad del software en seis características (funcionalidad, fiabilidad, usabilidad, eficiencia, facilidad de mantenimiento y portabilidad), que se subdividen en sub características. Las sub características pueden medirse mediante métricas internas o externas.

En la figura 3.6 se muestra un esquema de las características y las sub características claramente identificadas como las métricas que comprueban cada uno de los atributos de calidad.

Para cada característica y sub característica, la capacidad del software está determinada por un conjunto de atributos internos que pueden medirse.

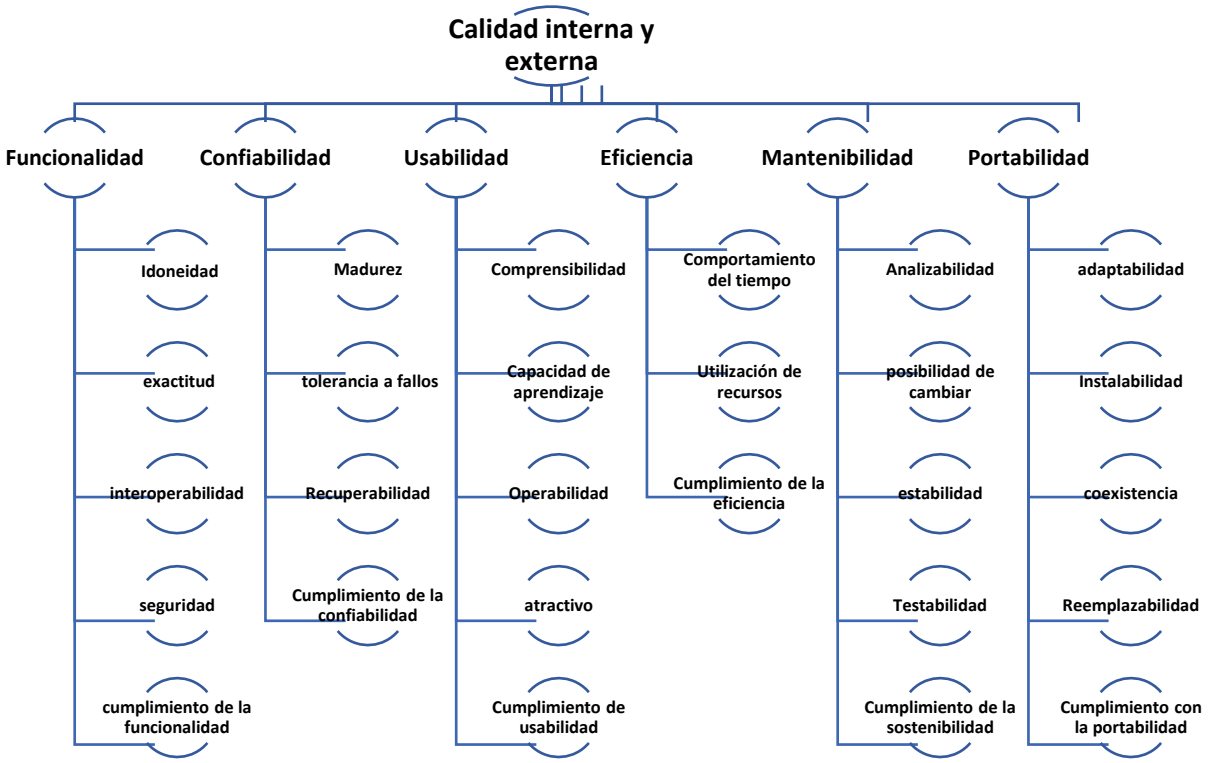


Figura 3.6.- Modelo de calidad interna y externa (Elaboración propia en base a ISO/IEC FDIS 9126-1:2000)

A continuación, se describe cada característica de calidad y las sub características del estándar que influyen en la calidad del software.

Funcionalidad

La capacidad del producto de software para proporcionar funciones que cumplan con las necesidades declaradas e implícitas cuando el software se utiliza bajo condiciones especificadas.

Esta característica se refiere a lo que hace el software para satisfacer las necesidades, mientras que las otras características se refieren principalmente a cuándo y cómo satisface las necesidades como se puede ver en la tabla 3.1

Tabla 3.1.- Métricas de funcionalidad (Elaboracion propia basada en ISO 9126,2000)

Funcionalidad	
Métricas	**Descripción**
Idoneidad	• La capacidad del producto de software para proporcionar un conjunto adecuado de funciones para las tareas especificadas y los objetivos del usuario. • Los ejemplos de adecuación son la composición orientada a las tareas de las funciones de las subfunciones constitutivas y las capacidades de las tablas. La idoneidad también afecta la operatividad.
Precisión	• La capacidad del producto de software para proporcionar los resultados o efectos correctos o acordados con el grado de precisión necesario.
Interoperabilidad	• La capacidad del producto de software para interactuar con uno o más sistemas especificados. • La interoperabilidad se utiliza en lugar de la compatibilidad con el fin de evitar posibles ambigüedades con la sustituibilidad
Seguridad	• La capacidad del producto de software para proteger la información y los datos para que personas o sistemas no autorizados no puedan leerlos o modificarlos y las personas o sistemas autorizados no se les niega el acceso a ellos. [ISO / IEC 12207: 1995] • Esto también se aplica a los datos en transmisión. • La seguridad se define como una característica de la calidad de uso, ya que no se refiere al software solo, sino a todo un sistema.
Cumplimiento de la funcionalidad	• La capacidad del producto de software para adherirse a las normas, convenciones o reglamentos en leyes y prescripciones similares relacionadas con la funcionalidad.

Nota: *Información tomada del estándar ISO/IEC FDIS 9126,* para un sistema que es operado por un usuario, la combinación de funcionalidad, fiabilidad, facilidad de uso y eficiencia se puede medir externamente por calidad de uso.

Confiabilidad

La capacidad del producto de software para mantener un nivel de rendimiento especificado cuando se usa bajo condiciones especificadas.

Las limitaciones en la fiabilidad se deben a fallos en los requisitos, diseño e implementación. Las fallas dependen de la forma en que se utilice el producto de software y de las opciones de programa seleccionadas en lugar del tiempo transcurrido, como se describe en la tabla 3.2.

Tabla 3.2.- Métricas de confiabilidad (Elaboracion propia basada en ISO 9126,2000)

Confiabilidad	
Métrica	**Descripción**
Madurez	• La capacidad del producto de software para evitar fallos como resultado de fallas en el software.
Tolerancia a fallos	• La capacidad del producto de software para mantener un nivel de rendimiento especificado en casos de fallas de software o de infracción de su interfaz especificada. • El nivel de rendimiento especificado puede incluir una capacidad de seguridad contra fallos.
Recuperabilidad	• La capacidad del producto de software para restablecer un nivel especificado de rendimiento y recuperar los datos directamente afectados en el caso de un fallo. • A raíz de un fallo, un producto de software a veces se reducirá durante un cierto período de tiempo, cuya duración se evalúa por su capacidad de recuperación. • La disponibilidad es la capacidad del producto de software de estar en un estado para realizar una función requerida en un momento dado, bajo condiciones de uso establecidas. • Externamente, la disponibilidad puede ser evaluada por la proporción del tiempo total durante el cual el producto de software se encuentra en estado ascendente. Por lo tanto, la disponibilidad es una combinación de madurez (que gobierna la frecuencia del fallo), tolerancia a fallos y capacidad de recuperación (que regula la duración del tiempo de inactividad después de cada falla).
Cumplimiento de la fiabilidad	• La capacidad del producto de software para cumplir con las normas, convenciones o reglamentos relacionados con la fiabilidad.

Nota: *Información tomada del estándar ISO/IEC FDIS 9126,* la definición de fiabilidad en ISO / IEC 2382-14: 1997 es "La capacidad de la unidad funcional para realizar una función requerida". En este documento, la funcionalidad es sólo una de las características de la calidad del software. Por lo tanto, la definición de fiabilidad se ha ampliado a "mantener un nivel de rendimiento especificado..." en lugar de "... realizar una función necesaria".

Usabilidad

La capacidad del producto de software para ser entendido, aprendido, usado y atractivo para el usuario, cuando se usa bajo condiciones especificadas.

Algunos aspectos de funcionalidad, confiabilidad y eficiencia también afectarán la usabilidad, pero para los propósitos de *ISO / IEC 9126* no se clasifican como usabilidad ver la tabla 3.3.

Tabla 3.3.- Métricas de usabilidad (Elaboracion propia basada en ISO 9126,2000)

Usabilidad	
Métrica	**Descripción**
Comprensibilidad	• La capacidad del producto de software para permitir al usuario comprender si el software es adecuado y cómo se puede utilizar para tareas y condiciones de uso específicas. • Esto dependerá de la documentación e impresiones iniciales proporcionadas por el software.
Aprendizaje	• La capacidad del producto de software para permitir al usuario aprender su aplicación.
Operatividad	• La capacidad del producto de software para permitir al usuario operar y controlarlo. • Los aspectos de idoneidad, capacidad de cambio, adaptabilidad e instalación pueden afectar la operatividad. • La operatividad corresponde a la capacidad de control, tolerancia de error y conformidad con las expectativas del usuario según se define en ISO 9241-10. • Para un sistema que es operado por un usuario, la combinación de funcionalidad, fiabilidad, facilidad de uso y eficiencia se puede medir externamente por calidad en uso.
Atractivo	• La capacidad del producto de software para ser atractivo para el usuario. • Se refiere a los atributos del software destinados a hacer el software más atractivo para el usuario, como el uso del color y la naturaleza del diseño gráfico.
Cumplimiento de la usabilidad	• La capacidad del producto de software para adherirse a normas, convenciones, guías de estilo o regulaciones relacionadas con la usabilidad.

Nota: *Información tomada del estándar ISO/IEC FDIS 9126,*

Los usuarios pueden incluir operadores, usuarios finales e indirectos que están bajo la influencia o dependen del uso del software. La usabilidad debe abordar todos los diferentes entornos de usuario que el software puede afectar, lo que puede incluir la preparación para el uso y la evaluación de los resultados.

Eficiencia

La capacidad del producto de software para proporcionar un rendimiento adecuado, en relación con la cantidad de recursos utilizados, bajo condiciones establecidas.

Los recursos pueden incluir otros productos de software, la configuración de software y hardware del sistema y materiales (por ejemplo, papel de impresión, discos).

Para un sistema que es operado por un usuario, la combinación de funcionalidad, fiabilidad, facilidad de uso y eficiencia se puede medir externamente por calidad en uso como se ve en la tabla 3.4.

Tabla 3.4.- Métricas de eficiencia (Elaboracion propia basada en ISO 9126,2000)

Eficiencia	
Métrica	**Descripción**
Comportamiento del tiempo	• La capacidad del producto de software para proporcionar la respuesta apropiada y los tiempos de procesamiento y tasas de rendimiento al realizar su función, bajo condiciones establecidas.
Utilización de recursos	• La capacidad del producto de software para usar cantidades apropiadas y tipos de recursos cuando el software realiza su función bajo condiciones establecidas. • Los recursos humanos se incluyen como parte de la productividad.
Cumplimiento de la eficiencia	• La capacidad del producto de software para adherirse a las normas o convenciones relacionadas con la eficiencia.

Nota: Información tomada del estándar ISO/IEC FDIS 9126.

Mantenibilidad

La capacidad del producto de software de ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptación del software a los cambios en el entorno, y en los requisitos y especificaciones funcionales, ver en la tabla 3.5.

Tabla 3.5.- Métricas de mantenibilidad (Elaboracion propia basada en ISO 9126,2000)

Mantenibilidad	
Métrica	**Descripción**
Análisis	• Se puede identificar la capacidad del producto de software para detectar deficiencias o causas de fallas en el software o para identificar las partes que se van a modificar.
Cambios	• La capacidad del producto de software para permitir que se implemente una modificación especificada. La implementación incluye codificación, diseño y documentación de cambios. • Si el software ha de ser modificado por el usuario final, la variabilidad puede afectar la operatividad
Estabilidad	• La capacidad del producto de software para evitar efectos inesperados de las modificaciones del software.
Testabilidad	• La capacidad del producto de software para permitir que el software modificado sea validado.

Cumplimiento de la sostenibilidad	• La capacidad del producto de software para adherirse a las normas o convenciones relacionadas con la mantenibilidad.

Nota: Información tomada del estándar ISO/IEC FDIS 9126,

Portabilidad

La capacidad del producto de software de ser transferido de un entorno a otro. El entorno puede incluir el entorno de organización, hardware o software en la tabla 3.6 se describe aún más los conceptos.

Tabla 3.6.- Métricas de portabilidad (Elaboracion propia basada en ISO 9126,2000)

Portabilidad	
Métrica	**Descripción**
Adaptabilidad	• La capacidad del producto de software de ser adaptado para diferentes entornos especificados sin aplicar acciones o medios distintos de los proporcionados para este propósito para el software considerado. • La adaptabilidad incluye la escalabilidad de la capacidad interna (por ejemplo, campos de pantalla, tablas, volúmenes de transacciones, formatos de informes, etc.). • Si el software ha de ser adaptado por el usuario final, la adaptabilidad corresponde a la idoneidad para la individualización según se define en la norma ISO 9241-10, y puede afectar a la operatividad.
Instalación	• La capacidad del producto de software de ser instalado en un entorno especificado. • Si el software debe ser instalado por un usuario final, la instalación puede afectar a la idoneidad y operatividad resultantes.
Coexistencia	• La capacidad del producto de software de coexistir con otro software independiente en un entorno común que comparte recursos comunes.
Reemplazable	• La capacidad del producto de software de ser utilizado en lugar de otro producto de software especificado para el mismo propósito en el mismo entorno. • Por ejemplo, la sustitución de una nueva versión de un producto de software es importante para el usuario durante la actualización. • La reemplazabilidad se utiliza en lugar de la compatibilidad con el fin de evitar posibles ambigüedades con la interoperabilidad. • La reemplazabilidad puede incluir atributos de instalación y adaptabilidad. El concepto se ha introducido como una sub característica propia debido a su importancia.
Cumplimiento de la portabilidad	• La capacidad del producto de software para cumplir con las normas o convenciones relacionadas con la portabilidad.

Nota: Información tomada del estándar ISO/IEC FDIS 9126.

3.3.2 Proceso de validacion

El responsable para validar el sistema terminado es el *tester* externo al equipo de desarrollo de scrum, donde aplicara un plan de pruebas diseñado en base a las características de calidad, con un enfoque de evaluación subjetiva ya que de cierta forma simulara las acciones que puede realizar un usuario común frente a un software.

En la evaluación subjetiva se realizarán ejecuciones de pruebas que intenten provocar algún fallo en el sistema para conocer las reacciones del software que contrarresten a la existencia de los fallos, también se evaluara el correcto funcionamiento del software.

La intervención del *tester* externo comienza a partir de la creación de la Pila del Producto que está bajo la responsabilidad del Dueño del producto quien genera y prioriza las necesidades del cliente, cuando ya existen los sprint priorizados en la lista, el *tester* de manera inmediata analiza las necesidades principales que debe cubrir cada uno de esos *Sprint*.

Las necesidades obtenidas en el análisis se convierten en un objetivo que debe comprobar el *tester* externo en la prueba de aceptación, ya que en esta prueba se evalúa la satisfacción del cliente ante el producto de software, del cual el cliente espera encontrar en el software las funcionalidades deseadas que ayudaran a resolver sus necesidades.
Cuando se cumpla con las evaluaciones de características de calidad y de aceptación con el visto bueno del cliente, entonces se da por terminado la evaluación general de la metodología de aseguramiento de calidad de software.

Superar las expectativas del cliente es una misión primordial de todo el equipo de desarrollo y del equipo de calidad, si se logra cumplir la misión, entonces podemos decir que el trabajo en conjunto del *framework* de desarrollo ágil *Scrum* y la metodología de aseguramiento de calidad ha sido exitoso.

En la figura 3.7 se muestra el proceso de pruebas de validación tiene una gran similitud al proceso de pruebas de verificación, con la diferencia que el primero enfoca sus pruebas conforme a las características de calidad interna y externa de la *ISO/IEC 9126* y el segundo enfoca sus pruebas en la comprobación de requerimientos implementados en cada módulo.

El proceso de evaluación inicia a partir de que se encuentra la lista priorizada del *product backlog* en él se realiza un análisis en las necesidades del cliente, en base a esto se diseña un plan de pruebas combinándolos con las características de calidad.
Las herramientas que se utilizan para aplicar pruebas serán en base a varios *checklist* que debe enfocar la evaluación en cada uno de las características y sub características de calidad.

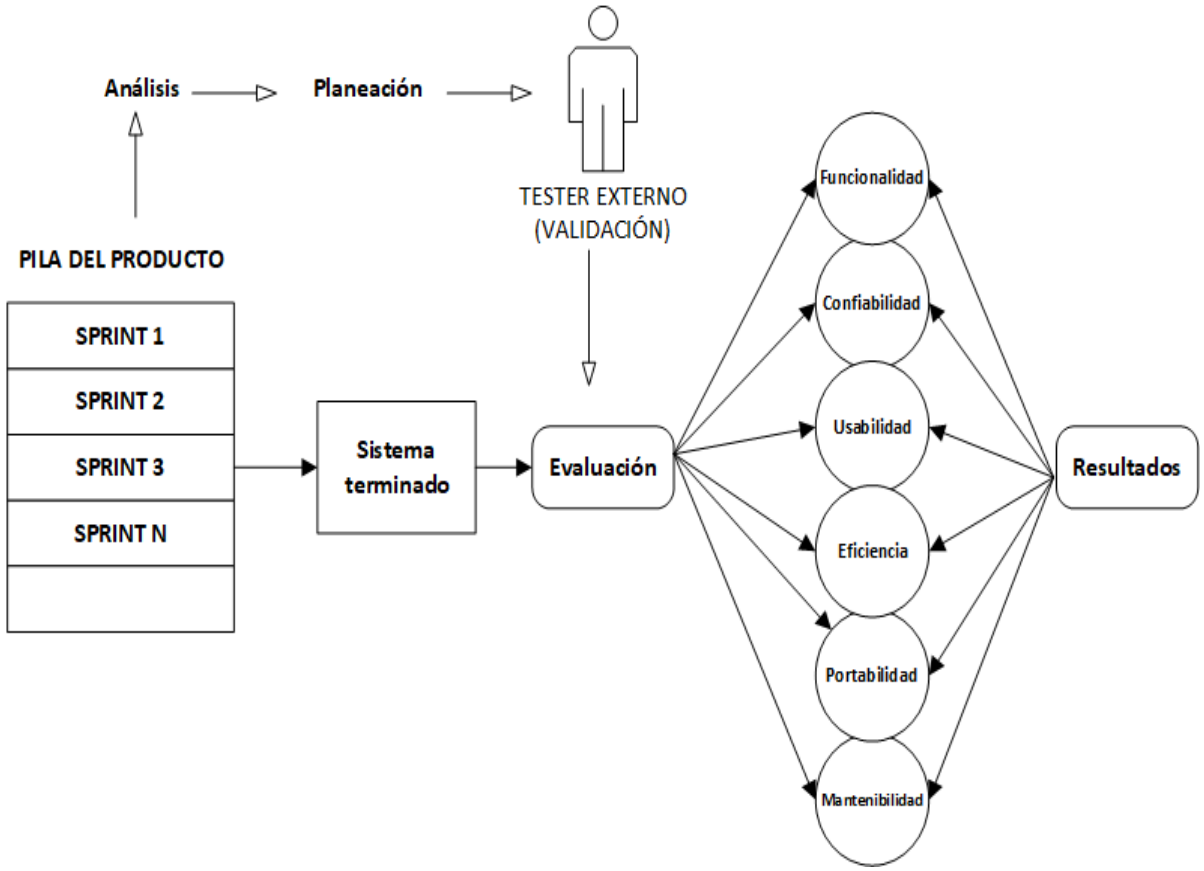


Figura 3.7.- El proceso de aplicación de pruebas (Elaboración propia).

La evaluación se apoya de la técnica de caja negra, con datos de entradas y salidas, abarcando así cada atributo del estándar *ISO/IEC 9126*, como se ve en la figura 3.8.

Cuando se obtienen los resultados se compara con los criterios de evaluación para la toma de decisiones de validar o no el sistema, el no validarlo significa que se detectaron fallas en el sistema de las cuales deben ser notificados a los programadores.

De la misma forma que el *tester* interno, cuando se identifican los fallos en el sistema de manera inmediata se debe notificar a los programadores para que realicen acciones correctivas lo más pronto posible, para que se vuelva aplicar el plan de pruebas hasta que cumpla con lo deseado.

Cuando el sistema ha pasado con la evaluación de validación se realiza el evento de entrega del producto al cliente en presencia de todo el equipo de desarrollo incluyendo el *tester* interno y el *tester* externo.

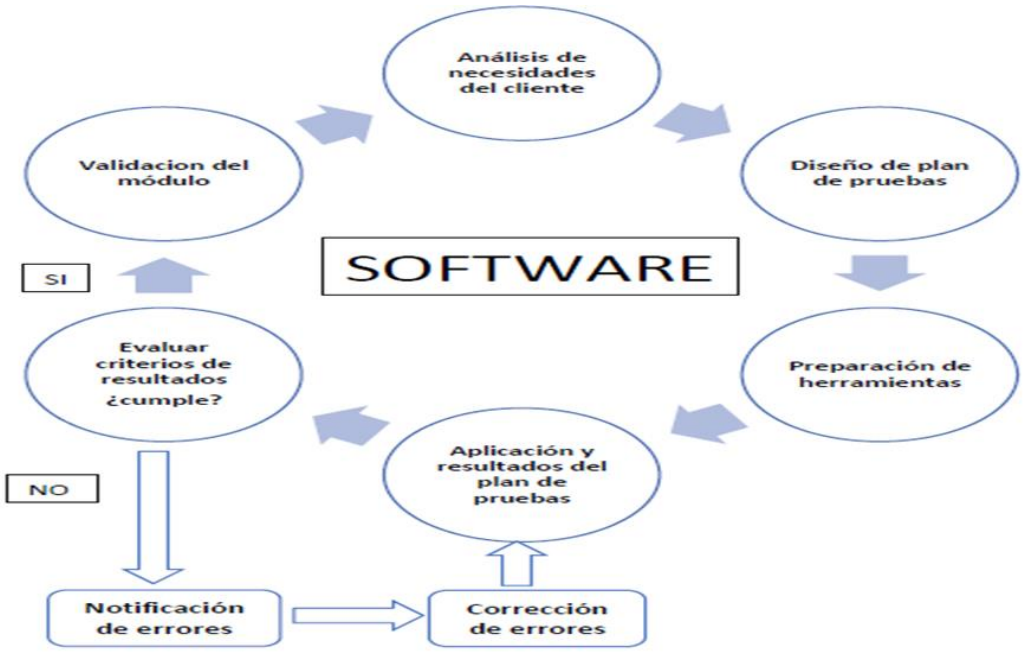


Figura 3.8.- Proceso de aplicación de pruebas a módulo (Elaboración propia).

3.3.3 Diseño del plan de pruebas de validación

El *tester* externo es el responsable de diseñar el plan de pruebas para validar el software terminado, por lo que debe tener el conocimiento suficiente sobre estándares de calidad, pero sobre todo de la estándar de calidad de software que se propone en esta metodología la *ISO/IEC 9126* y también debe conocer sobre las necesidades del cliente que debe resolver el software.

La elaboración del plan de pruebas de validación se basa de las características de calidad de la *ISO/IEC 9126*, el estándar ofrece 6 atributos de calidad de las cuales contiene 21 sub características para aplicar pruebas y 6 para revisar que cumpla con las normativas.

Con los datos anteriores significa que al menos se aplicaran 21 pruebas para comprobar la existencia de los 6 atributos de calidad implementados en el software.

En la figura 3.9 se observa un esquema de la dinámica que el *tester* debe llevar acabo para organizar sus pruebas, en él se muestra la clasificación de las características de calidad con sus respectivos sub características para asignar una prueba a cada uno de ellos y así sucesivamente hasta tener organizado todos los atributos de calidad.

Por ejemplo: en el atributo funcionalidad contiene cuatro sub características, de las cuales se está asignando cuatro pruebas A, B, C, y D, por lo que durante la evaluación serán las primeras pruebas en ejecutar y así sucesivamente hasta terminar de ejecutar todo el conjunto de pruebas de la A hasta la P.

Si existe algún otro tipo de normas que debe cumplir cada atributo entonces se debe revisar que los seis atributos cumplan con las normativas implementados por la empresa.

Como ya se ha mencionado con anterioridad, todo inicia desde la pila del producto donde se priorizan un conjunto de *Sprints* a cargo del dueño del producto, por lo que la lista de *Sprints* refleja lo que debe ser el software y por esa razón el *tester* externo analiza las necesidades del cliente y obtiene el plan de pruebas de validación mientras se encuentra el software en desarrollo.

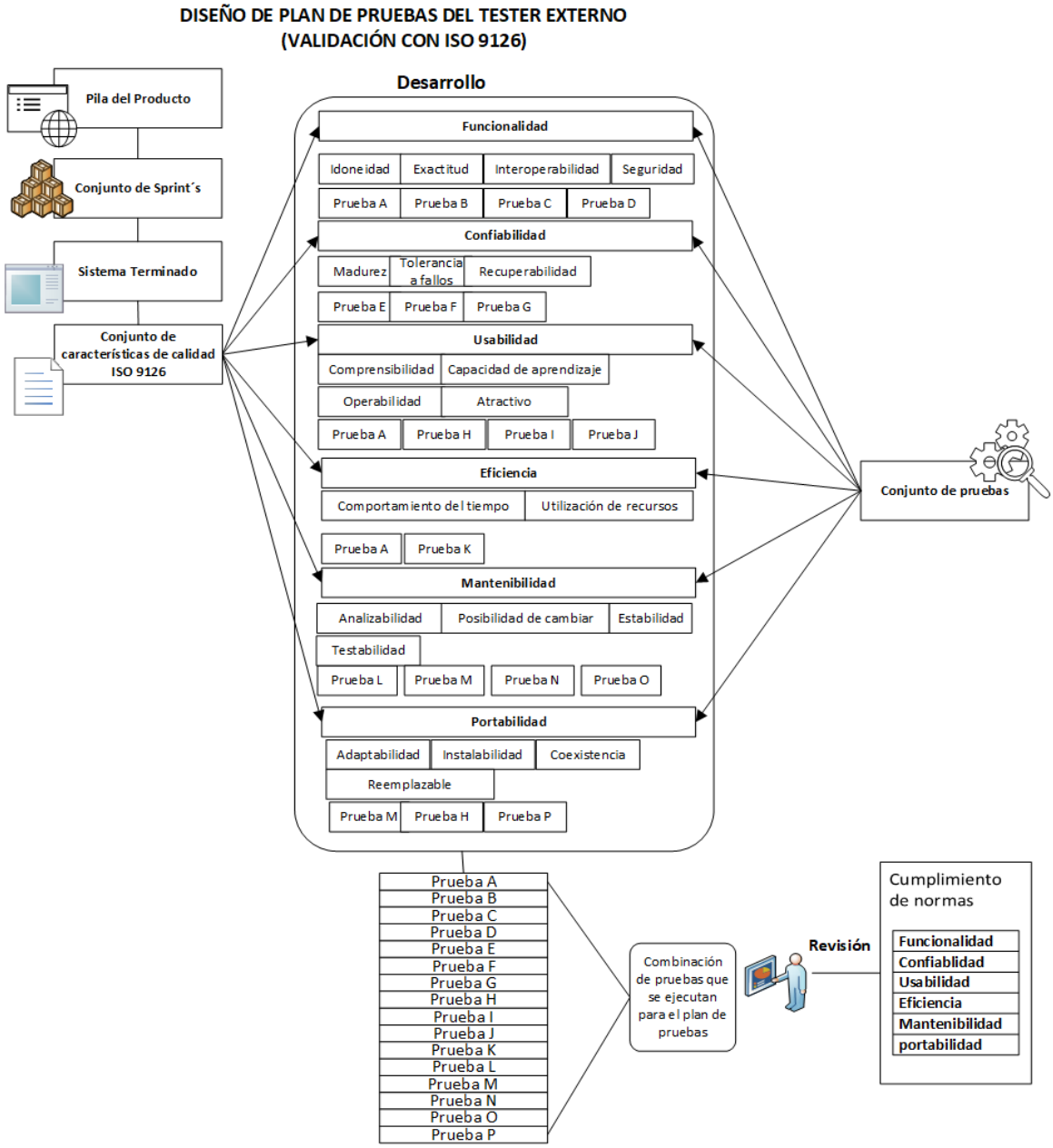


Figura 3.9.- Elaboración del plan de pruebas para la validación del software (Elaboración propia).

3.3.4 Pruebas de aceptación

Para llevar a cabo la evaluación de pruebas de aceptación que se aplica al cliente por parte del *tester* externo sobre el producto de software, primero debe diseñar los objetivos a cumplir del software de las cuales deben estar en común acuerdo con el equipo de Scrum, ver en la figura 3.10.

Los intereses y las necesidades del cliente son los objetivos que debe cumplir el software y esos objetivos a su vez se convierten en criterios en la prueba de aceptación.

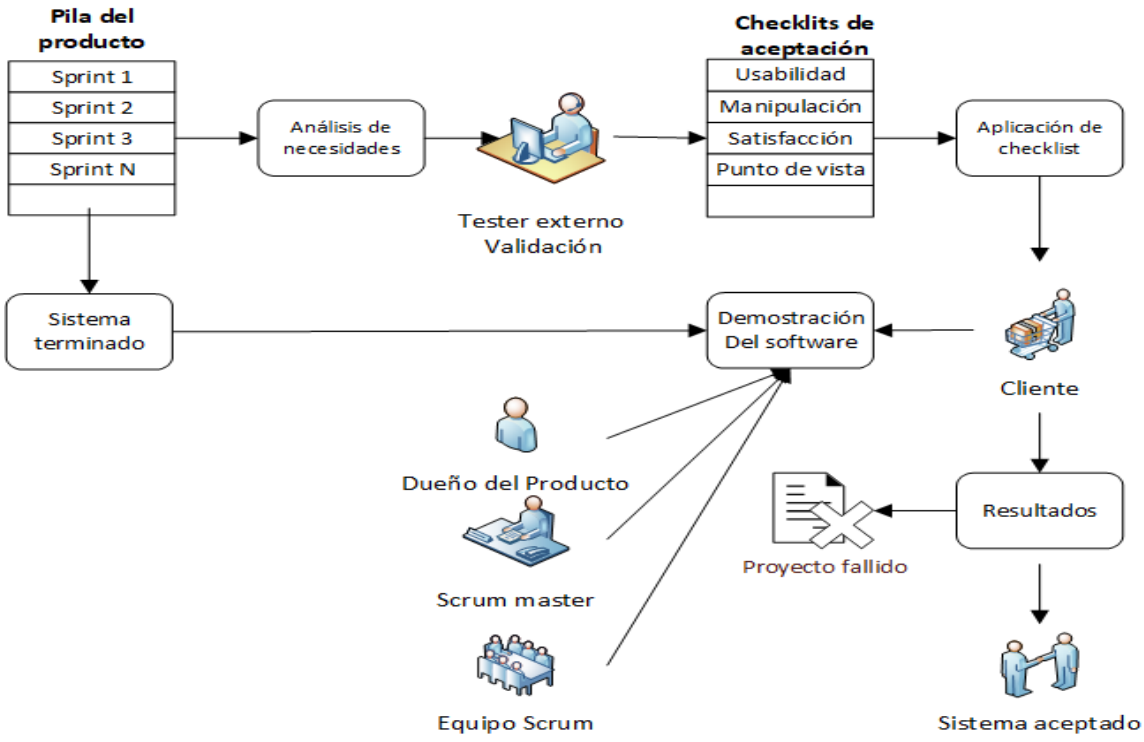


Figura 3.10.- Proceso de pruebas de aceptación del tester externo (Elaboración propia).

En la fase de revisión con el cliente, el *tester* externo actúa como un observador del cliente durante la demostración del producto de software, con la intención de evaluar al cliente en la manipulación del sistema y del grado de satisfacción ante el software.

En la fase de demostración de funcionamiento del software participan todos los que intervinieron en el desarrollo, para conocer de primera mano lo bueno y lo malo del trabajo realizado, pero sobre todo saber si se logró con el objetivo del proyecto.

En los resultados obtenidos de la prueba de aceptación, si los datos son positivos entonces se puede proceder a la entrega del software desarrollado al cliente, pero en caso contrario si los resultados son negativos entonces estamos ante un proyecto fallido.
Algunos casos de datos positivo pueden ser:

- software cumple con las necesidades del cliente.
- El cliente se siente satisfecho de la funcionalidad y comodidad del software.
- El cliente manipula fácilmente el software.
- El software contiene un grado de calidad aceptable.

Algunos casos de datos negativos pueden ser:

- El software no cumple con las necesidades del cliente.
- El cliente se siente insatisfecho e incómodo ante el software.
- Al cliente se le complica la manipulación del software.
- El software no contiene la suficiente calidad para operar.

3.3.5 Interacción del tester externo con los Stakeholders

La comunicación en el proceso de validación es importante para el *tester* externo para coordinar los objetivos de evaluación con los objetivos de desarrollo de scrum, también es importante en caso de identificar alguna falla pueda notificarlo de manera ágil para su pronta corrección.

La metodología propone en todos los procesos romper con las barreras que llegan a tener entre áreas y trabajar en conjunto para lograr objetivos en común.
El *tester* evalúa a favor del cliente, pero eso no significa que el equipo de *Scrum* lo tenga que ver como un enemigo a vencer sino al contrario tienen que aceptarlo como un valor agregado en la calidad que debe contener el software, además es un filtro calidad y de apoyo para que el proyecto en general tenga más posibilidades de tener éxito.

En la figura 3.11 se observa un esquema general de las interacciones que tiene el *tester* durante el proceso de evaluación para validar el software, es de vital importancia saber con quién dirigirse en caso de alguna duda o algún problema que pueda suscitarse.

También se observa que el primer autor con el que interactúa el *tester* externo es con el dueño del producto esto sucede durante la fase de generación de la Pila del producto (la lista priorizada) para realizar análisis de las necesidades del cliente.

Los siguientes autores de scrum con el que interactúa él *tester* es con los programadores durante la fase de evaluación del software, esto sucede cuando se detecta alguna falla y se notifica de manera ágil a los programadores para que realicen acciones correctivas lo más pronto posible.

Por último, el autor con el que interactúa el *tester* externo es con el cliente, esto sucede durante la fase de pruebas de aceptación y de demostración del software, aunque al inicio el *tester* solo observa el comportamiento del cliente, pero cuando es necesario saber su opinión sobre el software, es la única comunicación directa que tiene el *tester*.

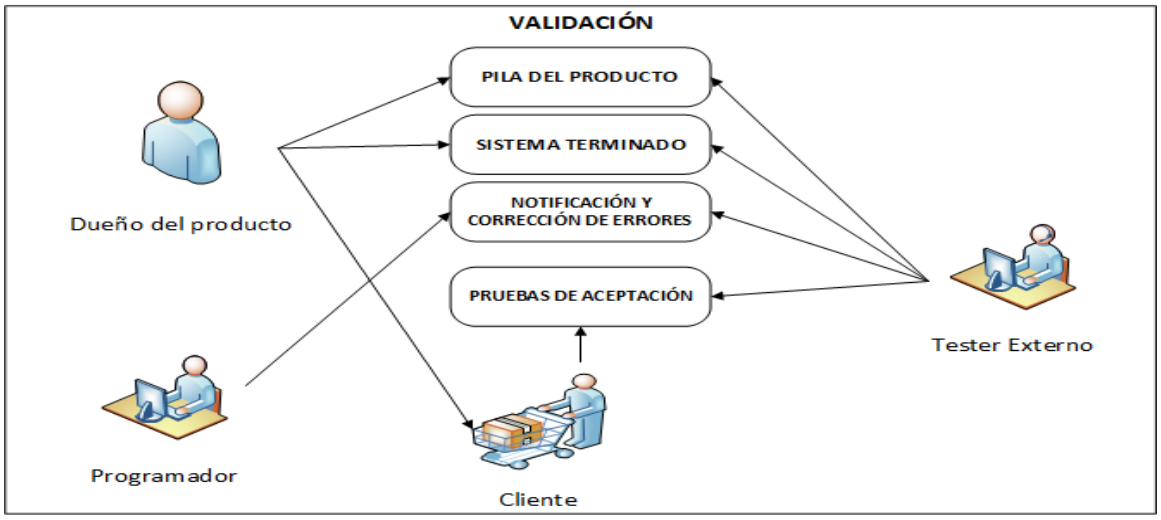


Figura 3.11.- Interacción de autores de Scrum con el tester externo (Elaboración propia).

3.4 Formatos

Para registrar los datos de análisis, del plan de pruebas y de los resultados obtenidos se necesita del apoyo de formatos o documentos que evidencien el trabajo realizado.

En un formato se presentan los datos generales del producto a evaluar y del responsable, también se presentan el registro de análisis o necesidades del cliente, también contiene el plan de pruebas las métricas, criterios de evaluación y las herramientas a utilizar.

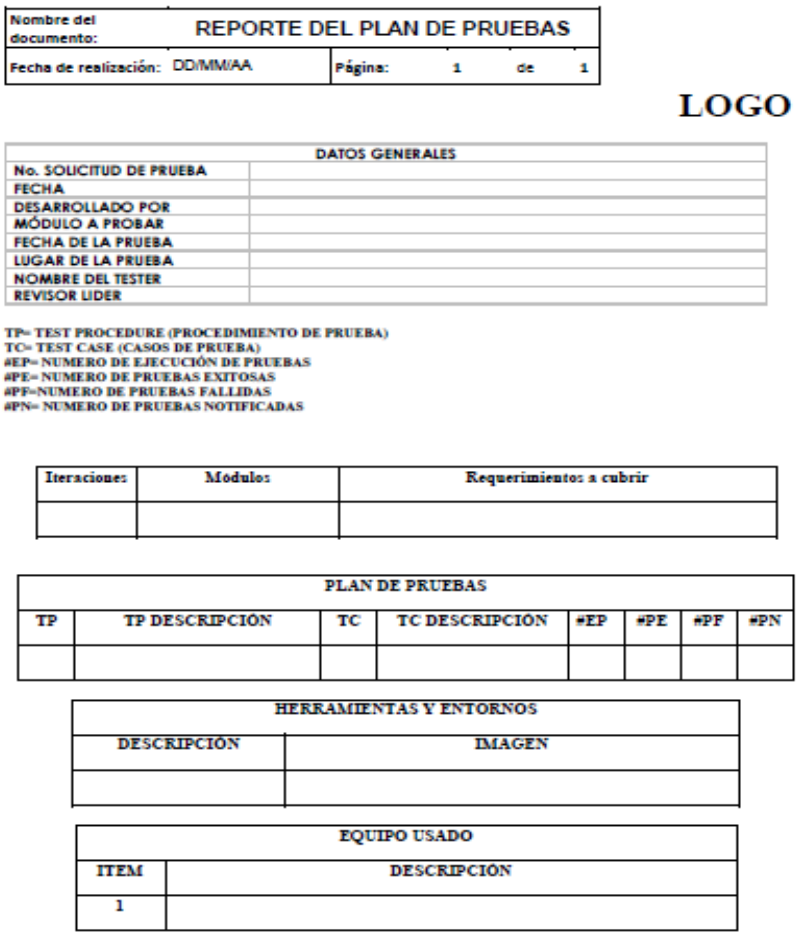
En otro formato se representa evidencias de las pruebas realizadas y los resultados obtenidos, los criterios cumplidos de verificación y de validación.

Una vez que se tiene planeado la aplicación de pruebas se deben instalar y configurar las herramientas que se van a utilizar para ejecutar cada uno de las pruebas.

3.4.1 Reporte de plan de pruebas

El primer formato contiene los datos generales del módulo o sistema a evaluar, con tablas para registrar datos de los procedimientos y casos de prueba a evaluar con sus respectivas descripciones de los requerimientos a cubrir.

También con tiene una tabla donde se registran y describen las herramientas o entornos a utilizar para la ejecución de pruebas y se agrega una imagen para fundamentar la utilización de equipo informático como se muestra en la figura 3.12.



Nombre del documento:	REPORTE DEL PLAN DE PRUEBAS		
Fecha de realización:	DD/MM/AA	Página:	1 de 1

LOGO

DATOS GENERALES	
No. SOLICITUD DE PRUEBA	
FECHA	
DESARROLLADO POR	
MÓDULO A PROBAR	
FECHA DE LA PRUEBA	
LUGAR DE LA PRUEBA	
NOMBRE DEL TESTER	
REVISOR LIDER	

TP= TEST PROCEDURE (PROCEDIMIENTO DE PRUEBA)
TC= TEST CASE (CASOS DE PRUEBA)
#EP= NUMERO DE EJECUCIÓN DE PRUEBAS
#PE= NUMERO DE PRUEBAS EXITOSAS
#PF=NUMERO DE PRUEBAS FALLIDAS
#PN= NUMERO DE PRUEBAS NOTIFICADAS

Iteraciones	Módulos	Requerimientos a cubrir

PLAN DE PRUEBAS							
TP	TP DESCRIPCIÓN	TC	TC DESCRIPCIÓN	#EP	#PE	#PF	#PN

HERRAMIENTAS Y ENTORNOS	
DESCRIPCIÓN	IMAGEN

EQUIPO USADO	
ITEM	DESCRIPCIÓN
1	

Figura 3.12.- Formato del plan de pruebas (Elaboración de pruebas).

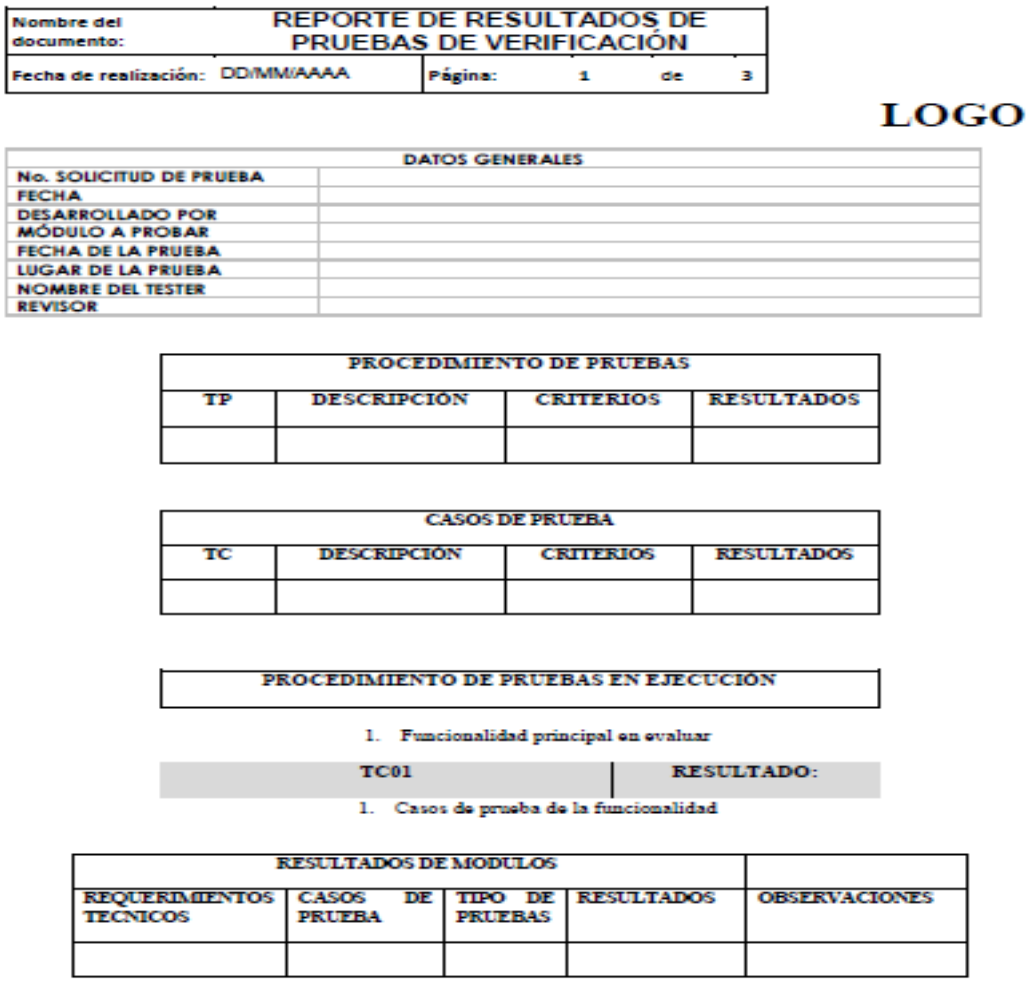
En el formato se muestra una tabla para registrar descripciones técnicas del equipo físico usado en la ejecución de las pruebas, esto nos permite saber qué tipo de tecnologías funcionan las pruebas.

3.4.2 Reporte de resultados de verificación

Cada procedimiento son funcionalidades de las cuales pueden ser más de uno dentro de un módulo a evaluar, por lo que es importante describir la funcionalidad y los criterios de evaluación para comprobar que el módulo está realizando lo deseado.

Los casos de prueba desglosan funcionalidades específicas de las cuales se deben describir y comprobar que cada caso cumple con los criterios de validación del módulo.

En el procedimiento de pruebas en ejecución se registran evidencia de las pruebas ejecutadas de cada procedimiento con sus respectivos casos de pruebas como se muestra en la figura 3.13.



Nombre del documento:	REPORTE DE RESULTADOS DE PRUEBAS DE VERIFICACIÓN		
Fecha de realización: DD/MM/AAAA	Página: 1 de 3		

LOGO

DATOS GENERALES	
No. SOLICITUD DE PRUEBA	
FECHA	
DESARROLLADO POR	
MÓDULO A PROBAR	
FECHA DE LA PRUEBA	
LUGAR DE LA PRUEBA	
NOMBRE DEL TESTER	
REVISOR	

PROCEDIMIENTO DE PRUEBAS			
TP	DESCRIPCIÓN	CRITERIOS	RESULTADOS

CASOS DE PRUEBA			
TC	DESCRIPCIÓN	CRITERIOS	RESULTADOS

PROCEDIMIENTO DE PRUEBAS EN EJECUCIÓN	
1. Funcionalidad principal en evaluar	
TC01	RESULTADO:

1. Casos de prueba de la funcionalidad

RESULTADOS DE MODULOS				
REQUERIMIENTOS TECNICOS	CASOS DE PRUEBA	TIPO DE PRUEBAS	RESULTADOS	OBSERVACIONES

Figura 3.13.- Formato de resultados de pruebas de verificación pagina 1 (Elaboración propia).

En el formato de resultados de los módulos, se registran los requerimientos que se debe cubrir, también los casos de pruebas que se desglosan en el módulo con los tipos de pruebas que se deben ejecutar para comprobar los requerimientos y para obtener resultados de la prueba, pero en caso de identificar algún error se realizan comentarios en el apartado de observaciones como se muestra en la figura 3.14.

3.4.3 Reporte de Resultados de validación

Nombre del documento:	REPORTE DE RESULTADOS DE PRUEBAS DE VERIFICACIÓN		
Fecha de realización: DD/MM/AAAA		Página: 2 de 3	

LOGO

NOTIFICACIONES DE FALLOS				
CASOS DE PRUEBA	NO. EJECUCIÓN DE PRUEBAS	NO. PRUEBAS EXITOASAS	NO. PRUEBAS FALLIDAS	NO. FALLOS NOTIFICADOS

CORRECCIONES DE FALLOS y VALIDACIÓN			
MÓDULO	NO. FALLOS NOTIFICADOS	NO. FALLOS RESUELTOS	NO. FALLOS VALIDADOS

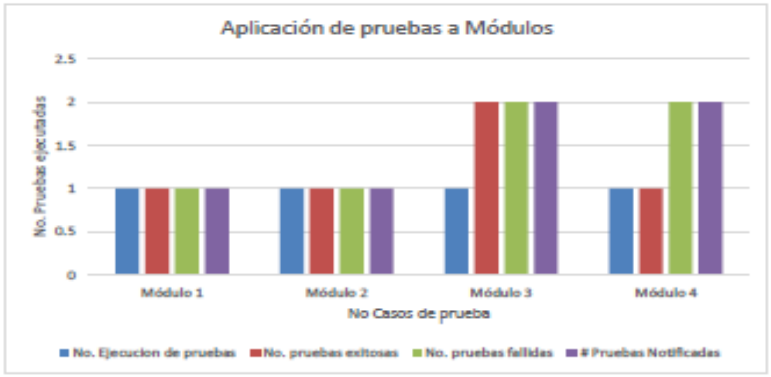


Figura 3.14.- Formato de resultados de pruebas de verificación pagina 2 (Elaboración propia).

Cuando los resultados se han obtenidos se realiza una comparativa con los criterios de validación de la prueba, en caso de que no cumpla con lo deseado o existe un error en las funciones de los módulos se debe notificar al programador para su pronta corrección.

Se grafican los datos obtenidos para conocer la calidad del módulo y tomar decisiones respecto a los resultados, como se muestra en la figura 3.15.

Nombre del documento:	REPORTE DE RESULTADOS DE PRUEBAS DE VERIFICACIÓN				
Fecha de realización:	DD/MM/AAAA	Página:	3	de	3

LOGO

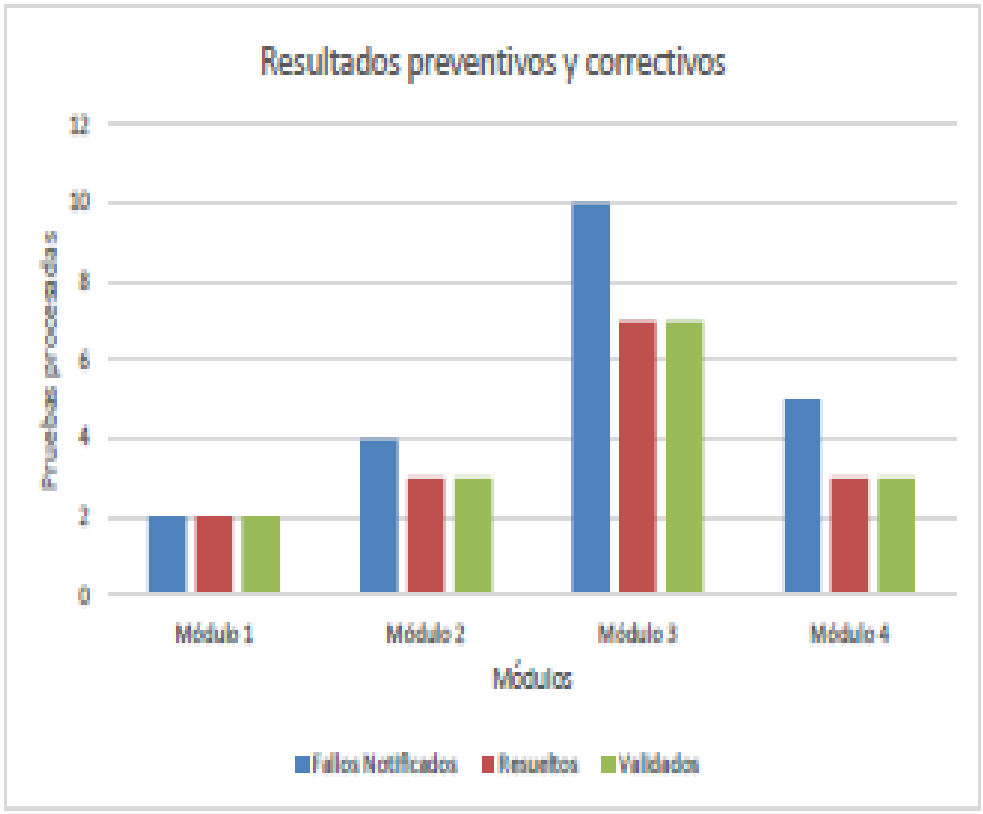
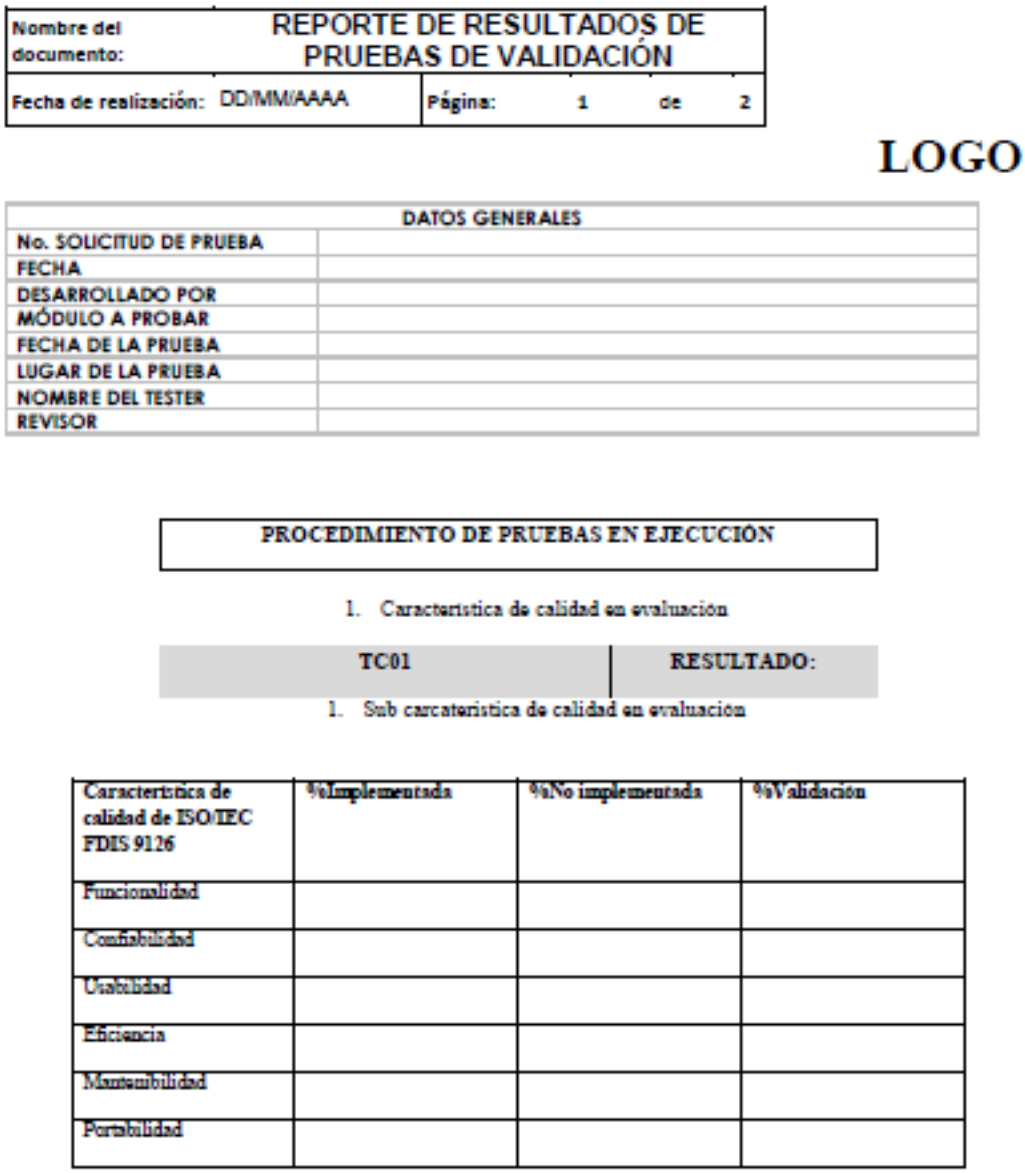


Figura 3.15.- Formato de resultados de pruebas de verificación pagina 3 (Elaboración propia).

En la figura 3.16 se muestra un formato para los resultados de pruebas de validación, del cual se fundamenta con las pruebas realizadas y se compara el grado de implementación de las características de calidad



Nombre del documento:	REPORTE DE RESULTADOS DE PRUEBAS DE VALIDACIÓN		
Fecha de realización: DD/MM/AAAA	Página: 1 de 2		

LOGO

DATOS GENERALES	
No. SOLICITUD DE PRUEBA	
FECHA	
DESARROLLADO POR	
MÓDULO A PROBAR	
FECHA DE LA PRUEBA	
LUGAR DE LA PRUEBA	
NOMBRE DEL TESTER	
REVISOR	

PROCEDIMIENTO DE PRUEBAS EN EJECUCIÓN

1. Característica de calidad en evaluación

TC01	RESULTADO:

1. Sub carcaterística de calidad en evaluación

Característica de calidad de ISO/IEC FDIS 9126	%Implementada	%No implementada	%Validación
Funcionalidad			
Confiabilidad			
Usabilidad			
Eficiencia			
Mantenibilidad			
Portabilidad			

Figura 3.16.- Formato de resultados de pruebas de validación Pagina 1(Elaboración propia).

En la figura 3.17 se muestra el nivel de porcentaje que se encuentra implementado en el sistema cada uno de las características de calidad de la *ISO/IEC FDIS 9126*, de esa forma los resultados reflejaran el nivel de calidad que contiene el sistema en evaluación.

Nombre del documento:	REPORTE DE RESULTADO$ DE PRUEBAS DE VALIDACIÓN	
Fecha de realización: DD/MM/AAAA	Página: 2 de 2	

LOGO

Figura 3.17.- Formato de resultados de pruebas de validación pagina 2 (Elaboración propia).

Cuando los fallos han sido notificados se debe volver aplicar el plan de pruebas diseñado para el módulo bajo prueba, esto con la intención de conocer si han sido correctamente resueltos los fallos y que no ha afectado otra funcionalidad del módulo.

Los siguientes *checklist* están diseñados para comprobar la existencia de las características de calidad en el sistema que se pueden observar en las tablas 3.7.

Tabla 3.7.- Checklist de Funcionalidad (Elaboración propia)

Atributo	**Métrica**	**Preguntas para evaluar la calidad del sistema**	**SI**	**NO**	**OBS.**
FUNCIONALIDAD Nota: El sistema debe tener la capacidad para proporcionar funciones que cumplan con las necesidades declaradas e implícitas cuando el software se utiliza bajo condiciones especificadas.	Idoneidad (conveniencia)	¿El software tiene la capacidad para proporcionar un conjunto adecuado de funciones para las tareas especificadas del cual cumpla con los objetivos del usuario final?			
		¿Al someter al programa a diversas tareas básicas, cumple este el/los requerimientos/ del/los usuario/s?			
		¿Puede el software desempeñar las tareas requeridas? ¿el resultado es el esperado?			
	Precisión	¿El software tiene la capacidad para proporcionar los resultados o efectos correctos o acordados con el grado de precisión necesario?			
	Interoperabilidad	¿El Software tiene la capacidad de interactuar con otros sistemas previamente especificados?			
	Seguridad	¿El software tiene la capacidad para proteger la información y los datos para que personas o sistemas no autorizados no puedan leerlos o modificarlos y las personas o sistemas autorizados no se les niega el acceso a ellos?			
		¿Están protegidos los datos que manipula el sistema, ya sea en su tiempo de proceso y tránsito, como así también en su estado de almacenamiento?			
		¿El acceso al sistema está protegida por un usuario y password?			
		¿Contiene nivel de acceso por medio de permisos para realizar actividades en el sistema (Administrador, consulta, etc.)?			
	Cumplimiento de funcionalidad	¿El Software se adhiere a estándares, convenciones o regulaciones en leyes y prescripciones similares relacionadas con la funcionalidad?			

Tabla 3.8.- Checklist de Confiabilidad (Elaboración propia)

Atributo	Métrica	Preguntas	SI	NO	OBS.
CONFIABILIDAD Nota: El sistema debe tener la capacidad del producto de software para mantener un nivel de rendimiento especificado cuando se usa	Madurez (vencimiento)	¿El software tiene la capacidad de restablecer el nivel de operación y recobrar los datos que hayan sido afectados directamente por una falla, así como al tiempo y el esfuerzo necesario para lograrlo?			
		¿Es confiable el software para el usuario final?			
	Tolerancia a fallas	¿El software tiene la capacidad para mantener un nivel de rendimiento especificado en casos de fallas de software o de infracción de su interfaz especificada?			
		¿Después de un buen periodo de uso: sucede a veces que el usuario desconfía porque en ocasiones anteriores ha perdido datos importantes que lo ha llevado tiempo cargar?			
		¿Cuándo falla, son fallas graves?			
	Recuperable	¿El software tiene la capacidad para restablecer un nivel especificado de rendimiento y recuperar los datos directamente afectados en el caso de un fallo?			
		¿El software tiene la capacidad de estar en un estado para realizar una función requerida en un momento dado, bajo condiciones de uso establecidas?			
		¿El sistema se “cae” muy a menudo?			
	Cumplimiento de fiabilidad	¿El software tiene la capacidad para cumplir con las normas, convenciones o reglamentos relacionados con la fiabilidad?			

Tabla 3.9.- Checklist de Usabilidad (Elaboración propia)

Atributo	Métrica	Preguntas	SI	NO	OBS.
USABILIDAD Nota: El sistema debe tener la capacidad del producto de software para ser entendido, aprendido, usado y atractivo para el usuario, cuando se usa bajo condiciones especificadas Nota: Los aspectos de idoneidad, capacidad de cambio, adaptabilidad e instalación pueden afectar la operatividad.	Comprensibilidad	¿El software tiene la capacidad para permitir al usuario comprender si el software es adecuado y cómo se puede utilizar para tareas y condiciones de uso específicas?			
		¿Se entregó manual de usuario al cliente?			
		¿Es amigable el software para los desarrolladores?			
		¿Pueden comprender su estructura lógica, sus funciones de ejecución y procesamiento, su código fuente es fácilmente legible y comprensible?			
		¿Son cómodos los menús, los botones, las ventanas de interfaces, los cuadros de diálogos, los formularios, etc.?			
		¿Las jerarquías visuales son correctas?			
	Aprendizaje	¿El software tiene la capacidad para permitir al usuario aprender su aplicación?			
		¿El aprendizaje del usuario para manipular el software ha sido rápido?			
		¿Es intuitivo, posee toda la información y ayudas adecuadas como para que el usuario no tenga que depender de alguien que explique cómo utilizar cada función?			
	Operatividad	¿El software tiene la capacidad para permitir al usuario operar y controlarlo?			
		¿Es sencillo de entender y manejar el software para los usuarios a los cuales está destinado su uso?			
		¿Es sencillo buscar y filtrar información dentro del programa?			
	Atractivo	¿El software es visualmente agradable para el usuario, como el uso del color y la naturaleza del diseño gráfico?			
		¿Las formas de los formularios, menús y botones son agradables, así como logos y graficas?			
		¿Las interfaces del software son responsivas?			
	Cumplimiento de usabilidad	¿El software tiene la capacidad para adherirse a normas, convenciones, guías de estilo o regulaciones relacionadas con la usabilidad?			

Tabla 3.10.- Checklist de Eficiencia. (Elaboración propia)

Atributo	**Métrica**	**Preguntas**	**SI**	**NO**	**OBS**
EFICIENCIA Nota: El sistema debe tener la capacidad del producto de software para proporcionar un rendimiento adecuado, en relación con la cantidad de recursos utilizados, bajo condiciones establecidas.	Comportamiento del tiempo	¿El software tiene La capacidad para proporcionar la respuesta apropiada y los tiempos de procesamiento y tasas de rendimiento al realizar su función, bajo condiciones establecidas?			
		¿Existe alguna funcionalidad que realice la operación de manera lenta?			
		¿Se ve afectada la productividad de los usuarios por alguna lentitud en sus funcionalidades?			
	Utilización de recursos	¿El software tiene la capacidad para usar cantidades apropiadas y tipos de recursos cuando el software realiza su función bajo condiciones establecidas?			
		¿Cuándo el volumen de datos crece dentro de lo contemplado ¿el software se vuelve lento?			
		¿Es capaz el software de procesar/almacenar datos de manera eficiente?			
		¿Comienza a consumir muchos recursos de hardware?			
	Cumplimiento de la eficiencia	¿El software tiene la capacidad para adherirse a las normas o convenciones relacionadas con la eficiencia?			

Tabla 3.11.- Checklist de mantenibilidad. (Elaboración propia)

Atributo	Métrica	Preguntas	SI	NO	OBS.
MANTENIBILIDAD Nota: El sistema debe tener la capacidad del producto de software de ser modificado. Las modificaciones pueden incluir correcciones, mejoras o adaptación del software a los cambios en el entorno, y en los requisitos y especificaciones funcionales.	Análisis	¿En el software se puede identificar la capacidad del producto de software para detectar deficiencias o causas de fallas en el software o para identificar las partes que se van a modificar?			
	Cambios	¿El software tiene la capacidad para permitir que se implemente una modificación especificada?			
		¿Se puede implementar nueva codificación, diseño y documentación de cambios?			
		¿Tiene características de escalabilidad para implementar alguna funcionalidad nueva?			
	Estabilidad	¿El software tiene la capacidad para evitar efectos inesperados de las modificaciones del software?			
	Testabilidad	¿El software tiene la capacidad para permitir que el software modificado sea validado?			
	cumplimiento de mantenibilidad	¿El software tiene la capacidad para adherirse a las normas o convenciones relacionadas con la mantenibilidad?			

Tabla 3.12.- Checklist de portabilidad. (Elaboración propia)

Atributo	Métrica	Preguntas	SI	NO	OBS.
PORTABILIDAD Nota: El sistema debe tener la capacidad del producto de software de ser transferido de un entorno a otro. El entorno puede incluir el entorno de organización, hardware o software.	Adaptabilidad	¿El software tiene la capacidad de ser adaptado para diferentes entornos especificados sin aplicar acciones o medios distintos de los proporcionados para este propósito?			
		¿El software tiene la característica de escalabilidad en caso de integrar nuevos componentes (por ejemplo, campos de pantalla, tablas, volúmenes de transacciones, formatos de informes, etc.)?			
	Instalación	¿El software tiene la capacidad de ser instalado en un entorno especificado?			
	coexistencia	¿El software tiene la capacidad de coexistir con otro software independiente en un entorno común que comparte recursos comunes?			
		¿El software puede ejecutarse en cualquier navegador?			
		¿El software se puede ejecutar en cualquier dispositivo?			
	Reemplazable	¿El software tiene La capacidad de ser utilizado en lugar de otro producto de software especificado para el mismo propósito en el mismo entorno (nueva versión)?			
	Cumplimiento de portabilidad	¿El software tiene La capacidad para cumplir con las normas o convenciones relacionadas con la portabilidad?			

Conclusión

Con los fundamentos de investigación del marco teórico, se logró diseñar una nueva metodología llamada *VEyVAA* que cubre las necesidades de evaluación en cada proceso de evaluación como son la verificación y la validación, con la evaluación de estos dos enfoques se podrá comprobar la calidad de los módulos y del sistema completo. Además de adaptar correctamente el funcionamiento de la metodología con scrum siendo eficiente en la detección de errores y en la notificación de esos errores detectados aumentando así la calidad del software. Los *checklist* anteriores contiene algunas preguntas que pueden comprobar la existencia de cada atributo de calidad, pero dependiendo de las necesidades se puede insertar más preguntas para profundizar en las características de calidad en el sistema.

Capítulo 4

Caso de prueba: aplicación de la metodología *VEyVAA* a *Head End System (HES)*

Introducción

En el presente capítulo se desarrolla la aplicación de la metodología VEyVAA (Verificación y validación ágil) para evaluar la calidad de un Head End System. El capítulo está conformado por tres partes claves que son:

1.- Análisis de los requerimientos y obtención de información sobre el Head End System a evaluar.
2.- La aplicación de los procesos de pruebas por módulos y del sistema terminado.
3.- Resultados de las pruebas y toma de decisiones.

Es de vital importancia que el tester tenga el conocimiento acerca del funcionamiento del Head End System, así como el entorno donde será aplicado, la arquitectura en la que se basara él sistema y las tecnologías que se ocuparan para su desarrollo, esto con la finalidad de enfocar las pruebas necesarias para corroborar el funcionamiento deseado, además de preparar las herramientas que permitan ejecutar dichas pruebas.

En la evaluación de los módulos se realiza una descripción del funcionamiento de los módulos, así como su utilidad, esto con la finalidad de comprender los casos de pruebas que serán ejecutados. También se obtiene los datos generales del módulo para tener la información necesaria en caso de detectar alguna falla y ser notificado al programador responsable del módulo.

Durante el desarrollo de aplicación de pruebas se mostrarán algunas figuras de la ejecución de las pruebas por cada módulo, esto con la finalidad de evidenciar la comprobación de la correcta funcionalidad o en caso contrario de la detección de algún error en el módulo.

El proceso está estructurado de acuerdo a la metodología VEyVAA que se propone en el capítulo 3 de esta tesis, por lo que la información de cada evaluación será organizada de la siguiente manera: análisis de los requerimientos, el plan de pruebas por cada módulo, la preparación de herramientas, la ejecución de pruebas, resultados y liberación del módulo.

En la evaluación del sistema se realizó con el apoyo de checklist que su diseño esta baso en el estándar ISO/IEC 9126 que también se propone en el capítulo 3 de esta tesis.

En la parte de resultados se muestran todos los casos y pruebas ejecutadas representadas en tablas y graficas para conocer de manera directa y general los datos que nos permiten tomar decisiones en las notificaciones de fallas y las correcciones de los programadores para cumplir con la verificación de los módulos hasta lograr la validación del sistema.

4.1 Análisis del *Head End System* y Análisis de requerimientos

Es importante revisar toda la información proporcionada por el cliente respecto al sistema que se va a desarrollar, la información pueden ser los requerimientos funcionales y no funcionales, esquemas, diseños, diagramas, maquetas, etc. Que permita visualizar el funcionamiento del sistema, esto con la finalidad de que el *tester* organice de manera correcta y eficiente su plan de pruebas.

Es importante mencionar que existe información confidencial y se ha tomado medidas necesarias para mostrar la suficiente información sin afectar a terceros, para comprender el trabajo realizado al implementar la metodología *VEyVAA.*

4.1.1. Head End System (HES)

La metodología será implementada en *Head End System*, el sistema se presenta como una alternativa de nivel tecnológico en *Smart cities*, para tener la comunicación y el control de los servicios prestados a la ciudadanía, facilitando el trabajo a las compañías responsables a la gestión de *Utilitys* (servicios públicos como el agua, el gas, la electricidad, etc.)

El *Head End System (HES)* es un sistema integral gestiona toda la información de forma centralizada, en el se pueden integrar más sistemas para realizar diversas funciones y procesamiento.

El *HES* sirve como un puente de comunicación entre la información generada por una red de medidores inteligentes y de un sistema informático que gestiona a la *Utility*, estos sistemas generan las tareas que deben ser ejecutadas por los medidores inteligentes. El *HES* gestiona todo el flujo de datos, para procesar y enviar las tareas correspondientes a la red de medidores inteligentes, la gestión de los datos funciona gracias a un conjunto de operaciones que realiza de manera interna, esas operaciones son claramente identificadas en un esquema que se observa en la figura 4.1.

El funcionamiento del *HES* inicia cuando la *utility* realiza una nueva petición, en ella contiene una o más tareas en un archivo *XML*, esta es recibida por la *web service* del *HES* que se encarga de realizar la conexión con la *utility*, realiza una comparativa del archivo con su esquema definido en *XML*, para validar que los datos no estén dañados o corrompidos.

Las peticiones validadas son almacenadas en la base de datos del sistema *HES*, para ser consultadas en el momento que se requiera por medio de una interfaz, pero sobre todo para priorizar tareas a ejecutar, ya que existe un módulo de servicio de *Windows* que monitorea continuamente a la base de datos en caso de existir una tarea, esta toma la tarea con más prioridad y lo envía a la red de medidores para ser ejecutada.

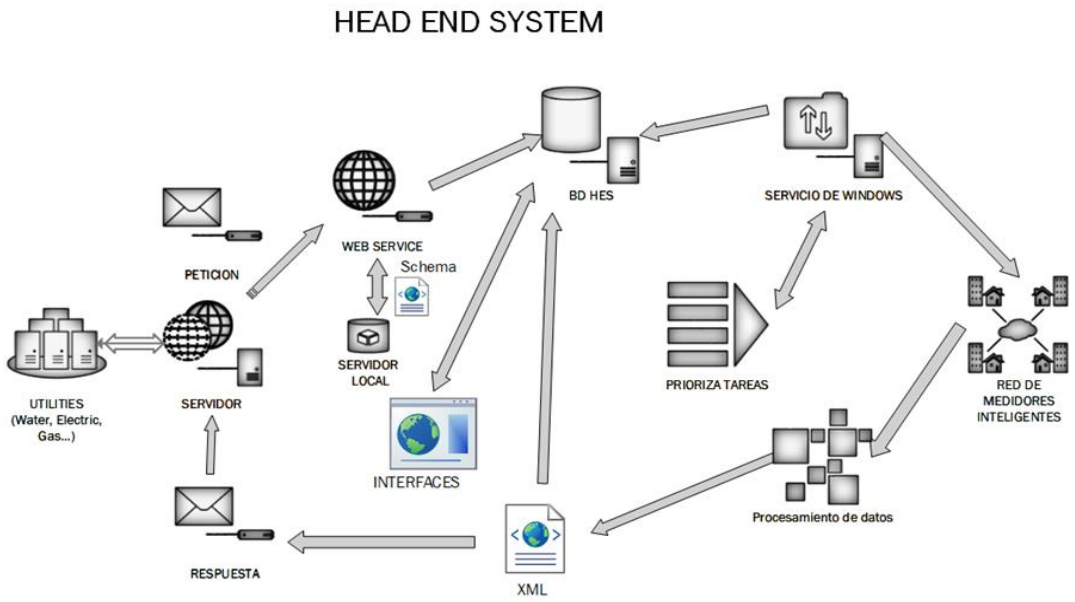


Figura 4.1.- Esquema del Head End System (Flores, Medina, Hernández y Sánchez, 2017).

La red de medidores inteligentes genera Paquetes de datos (*Frames*) como respuesta a las peticiones hechas por la *Utility*, la siguiente operación es procesar los datos de respuesta en un archivo *XML* de una manera legible para el usuario, este archivo toma dos rutas una para ser guardado en la base de datos del *HES* y pueda ser consultado por la interfaz y otro para ser enviado como respuesta a la *utility*. Este funcionamiento es gracias a la arquitectura *MVC*.

4.1.2 Modelo-Vista-Controlador *(MVC)* en *Head End System*

El *Head End System* está basado en la arquitectura modelo vista controlador *(MVC)*, en la figura 4.2, se puede observar la arquitectura dividido en tres capas, por lo que el sistema también se puede dividir en tres partes importantes, la primera capa que corresponde el modelo, en el contiene la estructura de datos (base de datos)y la lógica de negocios, la segunda capa que corresponde a las vistas, en el contiene la interfaz, y la tercera capa que corresponde al controlador, en el contiene la lógica de aplicación.

El funcionamiento de la arquitectura consiste en una petición del usuario (puede ser inicio de sesión, consulta, etc.), el controlador se encarga de gestionar las peticiones del usuario, realizando un enlace con las capas modelo y vista, ya que no tiene interacción directa con los datos.

El modelo es invocado por el controlador y gestiona los datos realizando diversos mecanismos en su estructura, para que puedan acceder a dichos datos. Una vez ejecutada la acción el modelo entrega los datos de respuesta al controlador y este selecciona una vista para representarlo o mostrarlo desde una interfaz de una manera legible para el usuario.

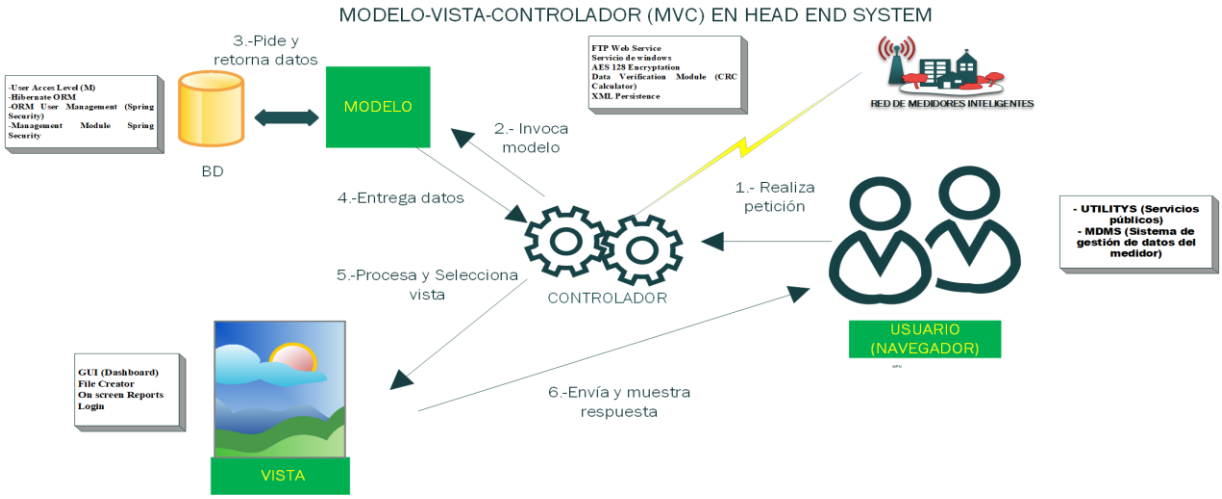


Figura 4.2.- MVC en funcionamiento con Head End System (Elaboración propia)

4.1.3. Tecnologías de desarrollo para el *Head End System*

Las tecnologías de desarrollo que se han propuesto para la realización del sistema, están clasificadas en las tres capas del sistema, cada uno tiene una función en específico que aportan en el desarrollo.

Las tecnologías en la capa del modelo son: *Oracle* que se encarga de estructurar los datos, *Maven* aporta la gestión de proyectos java, *JPA* como una Api de persistencia, *Hibernate* es un *framework* de persistencia de los datos, *spring MVC* proporciona un algoritmo compartido para el procesamiento de solicitudes. Por el lado de la vista las tecnologías son: *HTML 5*, *CSS* y *JSP* brinda la capacidad de construir interfaces web dinámicas, *BOOTSTRAP* es un *framework* para construir sistemas y sitios web, *JavaScript* y *Json* permite la programación orientado a objetos. En la capa del controlador: se selecciona las tecnologías *Ajax* para la transferencia de datos, *Java*, *Oracle Spring MVC* para tener el control de la base de datos y de las interfaces, ver en la figura 4.3.

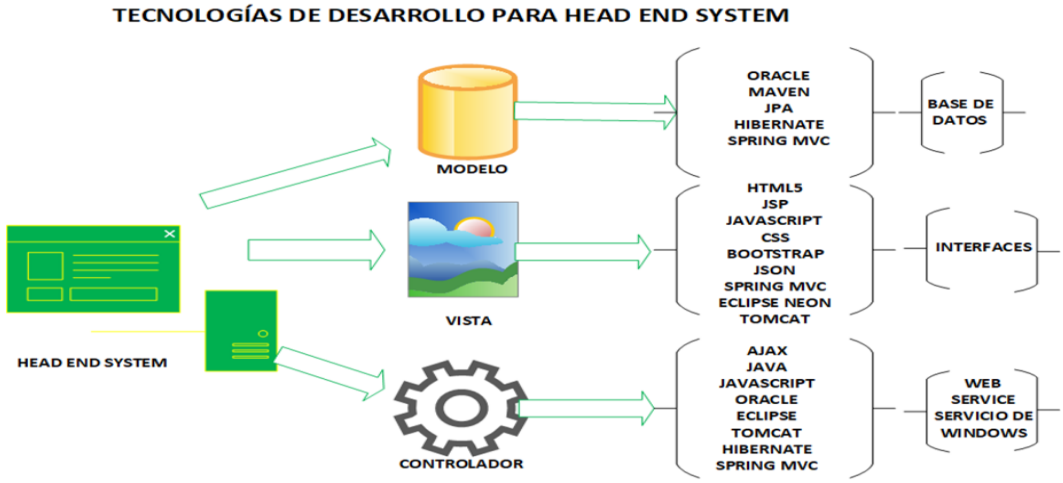


Figura 4.3.- Clasificación de tecnologías por capa MVC (Elaboración propia).

4.1.4 Análisis de requerimientos

En la tabla 4.1 refleja los módulos que componen cada iteración de desarrollo, siendo una solución que cubre los requerimientos del cliente, el tester organiza todos los módulos por capas MVC para verificar que tal requerimiento realmente se ha implementado en el módulo.

Tabla 4.1.- Sprintss de Scrum clasificados en módulos y requerimientos (Elaboración propia).

Sprints	**Módulos**	**Requerimientos a cubrir**
1ra iteración 8/04/2017 – 07/05/2017 (MVC)	*FTP Web Service (C)* *AES 128 Encryptation (C)* *User Acces Level (M)* *Data Verification Module (CRC Calculator) (C)*	*Cifrado de datos para una comunicación segura. *Verificación de la integridad de datos en las comunicaciones con los dispositivos remotos. *Configuración de roles y contraseñas. *Seguridad
2da iteración 8/05/2017 – 20/06/2017 (MVC)	*XML Persistence (C)* *GUI (Dashboard) (V)* *File Creator (V)* *CIM FTP Web Service (C)* *User Acces Level(M)* *Management Module Spring Security (M)*	*Garantizar el almacenamiento seguro de la información en la base de datos. *Contemplar mecanismos de redundancia y recuperación de desastres. *Capacidad para el almacenamiento de datos sin procesar los dispositivos automáticos o con duración seleccionable por el operador. *Interfaz de importación de la información de los dispositivos. La información del dispositivo puede incluir: Números de identificación (número de serie del fabricante / cliente, configuración, etc.), información del dispositivo (tipo de dispositivo, configuración, tipo de antena, etc.), información de ubicación (nombre de la ubicación, dirección, coordenadas, etc.) *Selección de los parámetros de las tareas a ser adquiridos de manera automática del dispositivo, los cuales pueden ser: -Las lecturas de registros. -Demanda. -Las alarmas. -La información de eventos de los medidores. -El perfil de carga. *Selección de las instrucciones de las tareas a ser enviadas de manera automática al dispositivo, las cuales pueden ser: -Sincronización de tiempo. -Configuración de eventos para conectar / desconectar. -Configuración de parámetros en los dispositivos. -Descarga de tareas *(jobs)* en los nodos de comunicación y/o dispositivos. -Descarga de programas de tareas *(schedule)* en los nodos de comunicación y/o dispositivos.
3ra iteración 21/06/2017 – 30/07/2017 (MVC)	*Char Validation Middleware* *Security Middleware (Code Injection) (M Y V)*	*Comunicación bidireccional con dispositivos remotos. *Ejecución automática de tareas *(Jobs)* periódicas de adquisición de datos y/o envío de instrucciones de un dispositivo o de un grupo de dispositivos o de la totalidad

	ORM User Management (Spring Security) (M) *On screen Reports (V)* *UDP Socket (C)* *Hibernate ORM (M)* *Task Manager (C)* *User Acces Level Management Module (M y V)* *CIM FTP Web Service (C)* *GUI (Dashboard) (V)*	de dispositivos y con periodicidad seleccionable por el operador u otro componente del Sistema AMI. *Ejecución automática de programas de tareas *(Schedule).* *Adquisición de datos o envío de instrucciones a-petición del operador de un dispositivo o de un grupo de dispositivos o de la totalidad de dispositivos. *Consulta de datos a-petición del operador mediante un filtrado seleccionable. *Uso de prioridades para el manejo de adquisición de datos y envío de instrucciones. *Recibir y registrar de manera automática las alarmas de un dispositivo o de un grupo de dispositivos o de la totalidad de dispositivos (Alarmas). *Registro del estado e historial de la comunicación con los dispositivos. *Generación de informes de manera automática o a-petición. -Visualización de parámetros, atributos, estadísticas, estado, etc. de todos los nodos de comunicación y dispositivos. -Informes de resumen de nodos de comunicación y dispositivos por su estado como nuevo, activo, fallido, perdido, etc.

El principal objetivo del proyecto es realizar la comunicación entre los medidores inteligentes (SON) y el sistema MDMS, es así que para poder lograr esa comunicación se identificaron módulos funcionales que ayudarían a realizar las operaciones necesarias, por lo que la metodología de *SQA* se enfoca a los módulos a probar asignando técnicas de pruebas para verificar el correcto funcionamiento que a continuación en breve se explica:

- módulo Encriptación *AES128*: protección de los datos al momento de ser enviados por el *Head End System* y al ser recibidos por los sistemas de servicio público.
- Debe ser capaz de encriptar y desencriptar cualquier carácter permitido por el sistema
- módulo de verificación de redundancia cíclica *(CRC):* permite validar que los datos o *frames* enviados por los medidores no sean corrompidos.
- Los datos enviados por los medidores inteligentes son en hexadecimal por lo que por medio de estándares de comunicación el sistema deberá identificar y validar los datos
- módulo de *Web Services*: Mantiene la comunicación entre el administrador de dispositivos *(MDMS),* recibe peticiones para ser enviadas a la red distribuida y regresa la respuesta recibida de la red.
- Levantar Servicio Web

- Conexión y acceso con servidor remoto para obtener archivo *XML*.
- Validación de archivos *XML*.
- Servicio de ejecución de tareas: Mantiene la comunicación entre la red distribuida y las tareas que se deben ejecutar.
- Mantener en ejecución servicio Windows
- Enviar tarea a red distribuida para ser ejecutada
- Manejador de tareas: Administra las tareas que deben ser ejecutadas, controlando las excepciones que se generen para continuar con las tareas sin perder la constancia y coherencia en la ejecución.
- Obtener tareas priorizadas
- Control de excepciones para el manejo de interrupción de tareas
- Generación de archivo *XML* de respuesta de la red distribuida
- módulo de Base de Datos: en él se almacenan los datos que recibe el *Head End System* de los medidores inteligentes, para ser consultados cuando sea necesario.
- Debe contener las tablas y campos definidos en los requerimientos
- Debe ser protegida por niveles de acceso
- modulo Interfaces: son vistas accesibles para el usuario que manipulara el sistema para tener acceso a los datos almacenados en la base de datos y para ejecutar tareas asignados a los medidores inteligentes.
- Las interfaces deben realizar las funciones correctamente
- Las interfaces deben ser compatibles con los navegadores básicos
- Las interfaces deben contener un buen nivel de usabilidad
- Las interfaces deben contener nivel acceso a las interfaces y seguridad (*login*, validación de datos insertados)
- Las interfaces deben ser escalables
- Las interfaces deben ser eficientes (fácil recuperación ante cualquier fallo interno o externo)
- Las interfaces deben ser responsivas para laptop, Tablet, móvil y computadoras de escritorio.

4.2 Evaluación enfocado a verificación

La información que se obtiene en el análisis de requerimientos es fundamental para poder diseñar el plan de pruebas por cada módulo. La información que contiene el plan de pruebas es el procedimiento de prueba que se debe llevar a cabo, en él se desglosan los casos de prueba además se obtienen las pruebas a ejecutar, las pruebas pasadas, las pruebas fallidas, las pruebas notificadas y resultados.

Durante el procedimiento de prueba se mostrará la informacion sobre los casos de prueba que se aplicó para evaluar cada módulo, las herramientas que utilizan para ejecutar la prueba y para evidenciar la ejecución se muestran algunas figras de las pruebas en ejecución.

Para organizar e identificar cada prueba se utilizan dos tipos de identificación el TP *(Test Procedure)* Y TC *(Test Case).* El TP identifica el procedimiento y el funcionamiento principal del módulo del cual se desglosan un conjunto de casos de prueba (TC) que especifica condiciones o variables que satisfacen la funcionalidad del módulo.
La numeración de los identificadores debe ser en números arábigos del cual debe tener una secuencia para su fácil identificación.

4.2.1 Verificacion del módulo de encriptación y desencriptación *AES128*

Las pruebas fueron realizadas en 3 entornos diferentes (*JGrasp, NetBeans* y en consola de *Windows*) esto con la intención de conocer cómo se comporta el módulo en esos entornos, además de conocer los limitantes de estos entornos en sus resultados.

En la tabla 4.2, se muestra los datos generales del módulo *AES* para que en caso de alguna duda o detección de fallo se reporte de manera inmediata al programador que desarrollo el módulo.
Se diseñó un plan de pruebas enfocado a un módulo de encriptación *Advanced Encryption Standard AES128*, se puede ver en la tabla 4.3, donde permitirá evaluar el ingreso de cadenas de diferentes tipos y el módulo debe ser capaz de encriptar y desencriptar la cadena ingresada.

Las figuras que se muestran en este reporte la mayoría son pruebas que se ejecutaron en *NetBeans*, en casos especiales en la consola de *Windows* (para asegurar la ejecución del programa en ciertos criterios) y se ejecutaron algunos ejemplos en *JGrasp* para mostrar como un entorno puede limitarse en la parte de configuración del *charset* (la configuración del *charset* se realizó a *UTF-8*, porque la configuración por default del entorno no daba resultados esperados pero a pesar de la configuración no se logró el objetivo ya que en algunos signos el entorno no los pudo reconocer.

Tabla 4.2.- Datos generales de evaluación del módulo AES128 (Elaboración propia).

DATOS GENERALES	
No Solicitud	2
Fecha	28/03/2017
Desarrollado por	Mario Alberto Méndez Carmona
Producto	Módulo de encriptación AES128
Versión	1.0
Fecha de prueba	31/03/2017
Lugar de prueba	Área de Ingeniería de software
Nombre del tester	Manuel Flores Nava
Revisor	Agustín Sánchez Atonal

Tabla 4.3.- Plan de pruebas del módulo AES 128 (elaboración propia).

PLAN DE PRUEBAS AES 128							
TP	**TP DESCRIPCIÓN**	**TC**	**TC DESCRIPCIÓN**	**#EP**	**#PE**	**#PF**	**#PN**
TP01	encriptación y des encriptación de una cadena	TC 01	Se ingresa una Cadena con caracteres combinados, letras mayúsculas/minúscula, números, signos y espacios vacíos				
TP02	Ingresar diferentes tipos de datos en una cadena	TC 01	Validar cadena de letras Mayúsculas y minúsculas				
		TC 02	Validar una cadena de números.				
		TC 03	Validar una cadena de signos.				
		TC 04	Validar cadena de campo vacío.				
		TC 05	Validar cadena sin insertar dato				
		TC 06	Validar cadena con valor cero				
TP03		TC 01	Encriptar y des encriptar una cadena grande con letras Mayúscula y minúscula, en un mínimo de tiempo				

	Prueba de Estrés, rendimiento y carga	TC 02	Encriptar y des encriptar una cadena grande con letras mayúsculas, minúsculas, y números, en un mínimo de tiempo.				
		TC 03	Encriptar y des encriptar una cadena grande con letras mayúsculas, minúsculas, números signos, en un mínimo de tiempo.				
		TC 04	Encriptar y des encriptar una cadena grande con letras mayúsculas, minúsculas, números signos y espacios, en un mínimo de tiempo.				
TP04	Caracteres especiales (secuencias de escape)	TC 01	Reconocimiento de doble comillas \”				
		TC 02	Reconocimiento de la diagonal inversa \\				
		TC 03	Reconocimiento de salto de línea \n				
		TC 04	Reconocimiento de retorno de carro \r				
		TC 05	Reconocimiento de tabulador \t				
		TC 06	Reconocimiento de *BackSpace* \b				
		TC 07	Reconocimiento de *Form Feed* \f				
		TC 08	Reconocimiento de comilla simple \’				
		TC 09	Reconocimiento de un combinado de secuencias de escape (\”. \’, \n, \t, \b, \\, \r).				

Herramientas y entornos

Ver en la figura 4.4 las diferentes herramientas que se utilizan para evaluar los siguientes puntos

- Preparación de la Laptop *HP650* para poder utilizar los entornos para compilar *Java.*
- Preparación de los entornos de *Java* como es el *NetBeans IDE 8.0.2* para ejecución de las pruebas.
- Preparación de herramienta *Beyond compare* que funciona para comparar que los caracteres de entradas sean igual a la de salida y también para comparar la cantidad de caracteres.

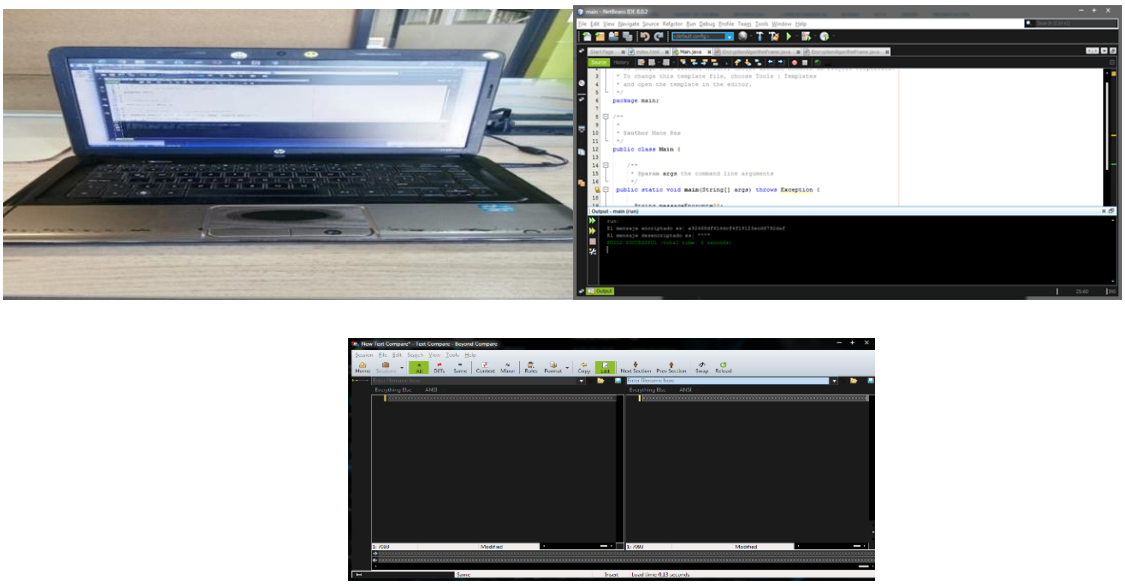


Figura 4.4.- Herramienta y entorno a utilizar en el módulo AES128 (Elaboración propia).

TP01 Encriptación y des encriptación

El módulo *AES 128* debe ser capaz de encriptar y des encriptar.

Casos de pruebas

TC01 Se ingresa una cadena con caracteres combinados, letras mayúsculas/minúsculas, números, signos y espacios vacíos.

El módulo debe reconocer la cadena y ser capaz de encriptar y des encriptar. En la figura 4.5 se muestra la Compilación de la cadena con diferentes caracteres

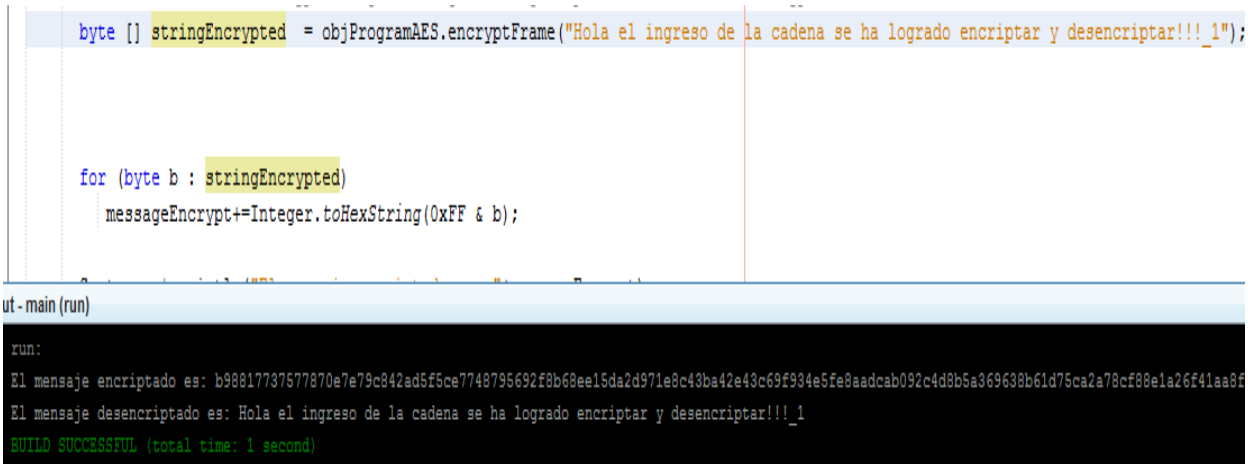


Figura 4.5.- Cadena con caracteres (Elaboración propia).

TP02 Ingresar diferentes tipos de datos en una cadena

Validar los diferentes tipos de datos ingresados en la cadena y encriptarlo/des encriptarlo.

Casos de pruebas

TC01 Validar cadena de letras Mayúsculas y minúsculas.

Reconocer la cadena de letras, encriptarlo y des encriptarlo, en la figura 4.6 se muestra la Compilación de una cadena con letra mayúscula y minúscula como se observa en la figura 42.

1. Aa. 2. Bb 3. Xx 4. Zz

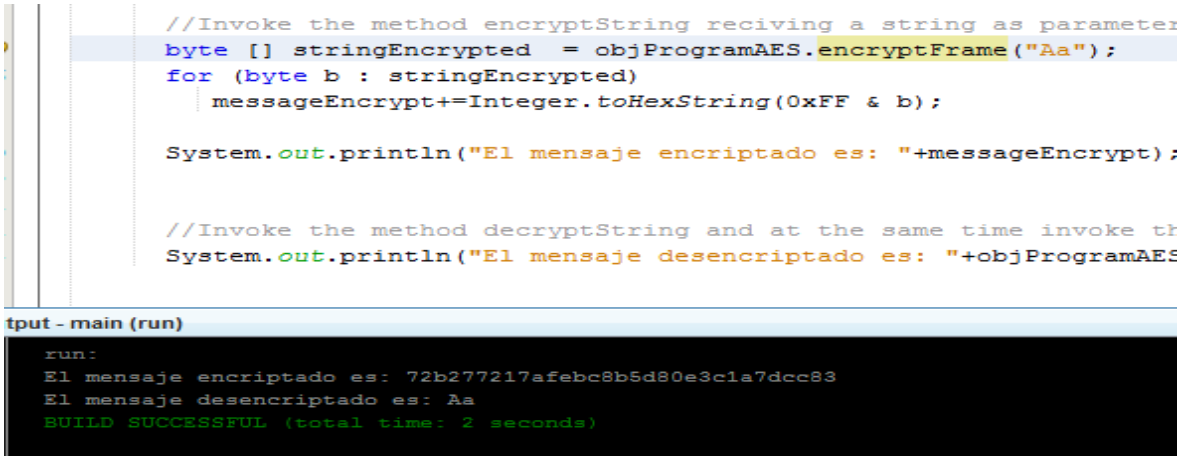


Figura 4.6.- Cadena Mayúscula y minúscula. (Elaboración propia)

TC02 Validar una cadena de números.

Reconocer la cadena de números, encriptarlo y des encriptarlo. Compilación de una cadena con números en figura 4.7. 1.- 012345678909876543210, 2. - 1010101010101, 3.- 54321, 4. - 98765

```
//Invoke the method encryptString reciving a string as parameter the to be encr
byte [] stringEncrypted  = objProgramAES.encryptFrame("01234567899876543210");
for (byte b : stringEncrypted)
   messageEncrypt+=Integer.toHexString(0xFF & b);

System.out.println("El mensaje encriptado es: "+messageEncrypt);


//Invoke the method decryptString and at the same time invoke the method printS
System.out.println("El mensaje desencriptado es: "+objProgramAES.decryptFrame(s
```

```
tput - main (run)
run:
El mensaje encriptado es: cb76dd4268f0dd3a1bf4a1198743ff05e698651a9f7f4466233022148dcb
El mensaje desencriptado es: 01234567899876543210
BUILD SUCCESSFUL (total time: 2 seconds)
```

Figura 4.7.- Cadena con números (Elaboración propia).

TC03 Validar una cadena de signos.

Reconocer la cadena de signos, encriptarlo y des encriptarlo. Compilación de una cadena con signos en figura 4.8.

1. -.,-_<|¡°¬*+!$%&'^ , 2. #(){}:; , 3. $%&/()=, 4.¿¿¿¿

```
//Invoke the method encryptString reciving a string as parameter the to be encry
byte [] stringEncrypted  = objProgramAES.encryptFrame(" -.,-_<>|¡°¬*+!%&$' ^");
for (byte b : stringEncrypted)
   messageEncrypt+=Integer.toHexString(0xFF & b);

System.out.println("El mensaje encriptado es: "+messageEncrypt);


//Invoke the method decryptString and at the same time invoke the method printSt
System.out.println("El mensaje desencriptado es: "+objProgramAES.decryptFrame(st
```

```
tput - main (run)
run:
El mensaje encriptado es: 5ed82bbc46275187dba83e823f4befb2dff5e39f1151825f8af484dc747c
El mensaje desencriptado es:  -.,-_<>|¡°¬*+!%&$' ^
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 4.8.- Cadena con signos (Elaboración propia).

TC04 Validar cadena de campo vacío.

Reconocer la cadena de campo vacíos, encriptarlo y des encriptarlo, ver compilación en la figura 4.9

1. Se hace clic en la Tecla espacio y se compila

```
//Invoke the method encryptString reciving a string as para
byte [] stringEncrypted  = objProgramAES.encryptFrame(" ");
for (byte b : stringEncrypted)
   messageEncrypt+=Integer.toHexString(0xFF & b);

System.out.println("El mensaje encriptado es: "+messageEncr

//Invoke the method decryptString and at the same time invo
System.out.println("El mensaje desencriptado es: "+objProgr
```

```
ut - main (run)
run:
El mensaje encriptado es: 123f1f3a820c7512f1c99355b8797f7
El mensaje desencriptado es:
BUILD SUCCESSFUL (total time: 2 seconds)
```

Figura 4.9.- Cadena con campo vacío (Elaboración propia).

TC05 Validar cadena sin insertar dato

Reconocer la cadena sin insertar dato, encriptarlo y des encriptarlo, ver compilación en la figura 4.10

1. No se ingresa ningún dato, ni siquiera un espacio. |

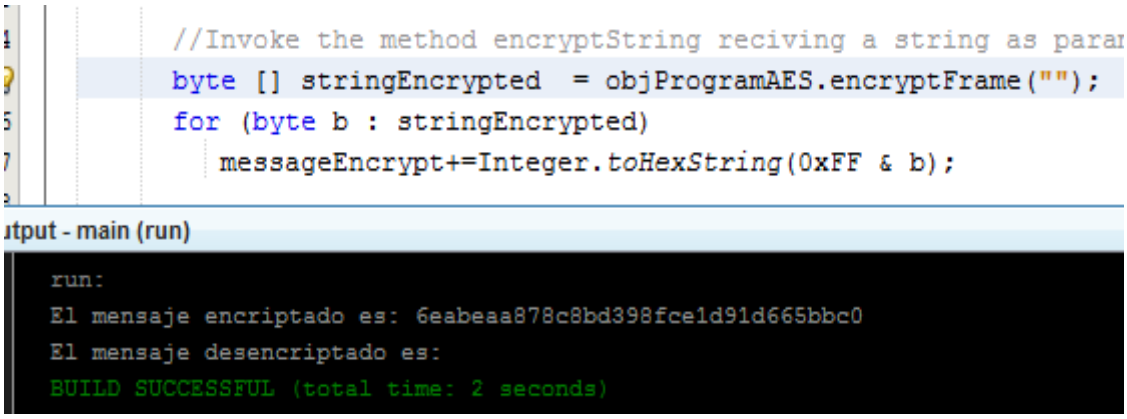


Figura 4.10.- Cadena sin elementos (Elaboración propia).

TC06 Validar cadena con valor cero.

Reconocer la cadena de valor cero, encriptarlo y des encriptarlo. Validar cadena con valor cero, ver compilación en la figura 4.11

1. Se ingresa un valor cero y se compila.

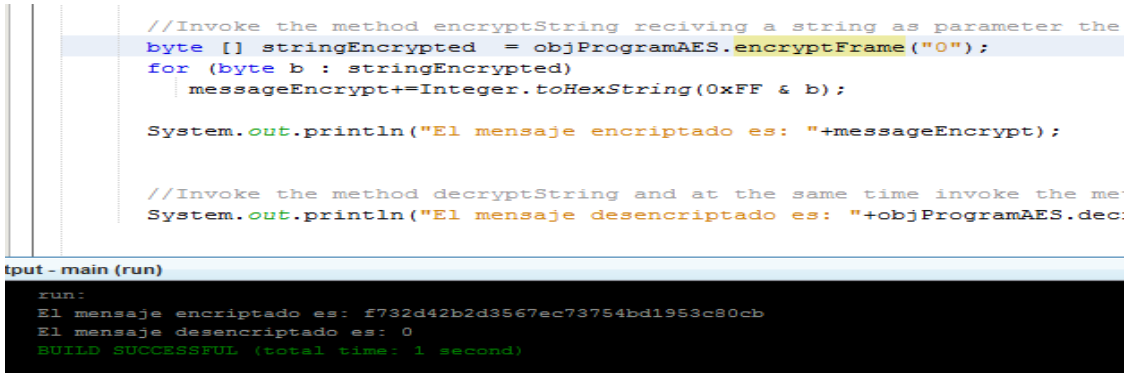


Figura 4.11.- Cadena con valor cero (Elaboración propia).

TP03 Prueba de estrés, rendimiento y carga

Se ingresan caracteres distintos (letras mayúsculas/minúscula, números, símbolos, espacios) en más de mil caracteres y el módulo debe ser capaz de encriptarlo y des encriptarlo en un mínimo de tiempo.

Casos de pruebas

TC01 Encriptar y des encriptar una cadena larga con letras Mayúscula y minúscula.

Reconocer la cadena de letras, encriptarlo y des encriptarlo. Compilación de una cadena larga con letras Mayúscula y minúscula, en un mínimo de tiempo. Se ingresa a la cadena una cantidad de 7093 caracteres solo de letras. Ver compilación en la figura 4.12.

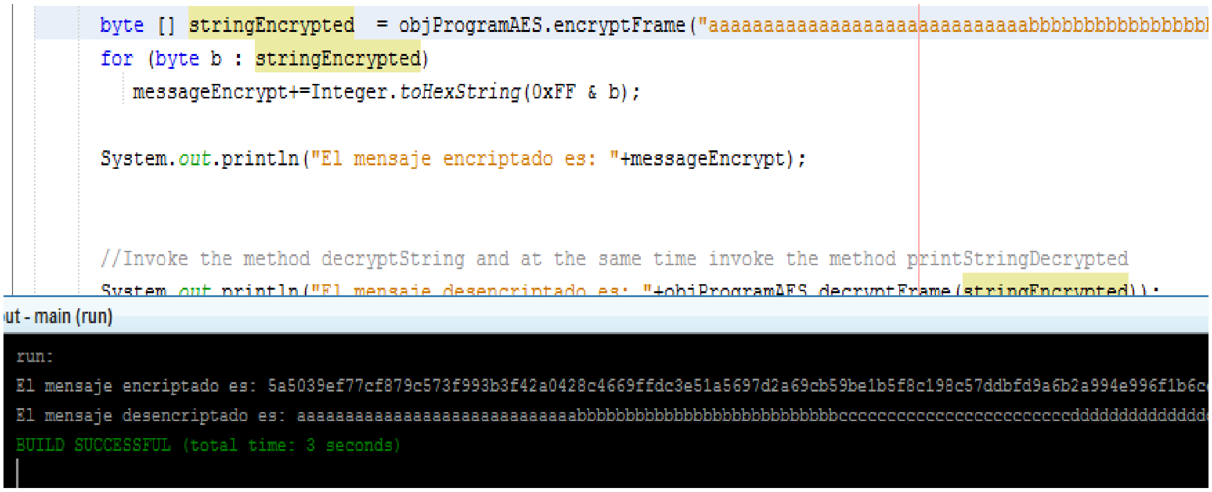


Figura 4.12.- Cadena con 7093 caracteres (Elaboración propia).

La comparativa de las cadenas para saber si la desencriptación coincide con la cadena encriptada se observa en la figura 4.13

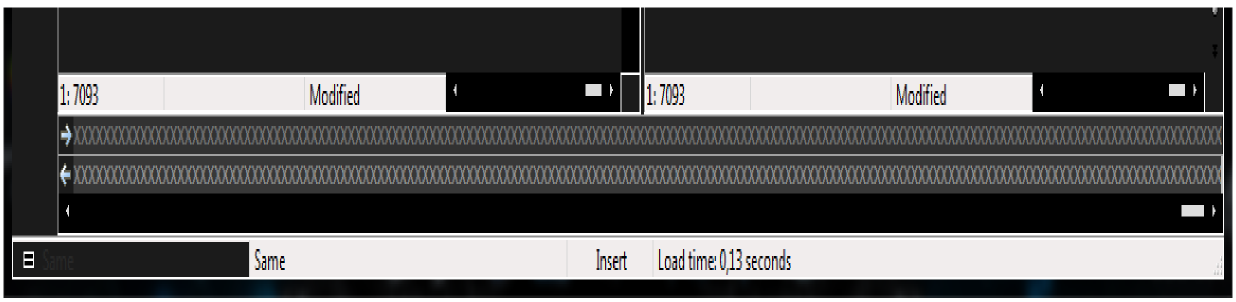


Figura 4.13.- Comparación de cadenas (Elaboración propia).

TC02 Encriptar y des encriptar una cadena grande con letras mayúsculas, minúsculas, y número. Compilación de una cadena grande con letras mayúsculas, minúsculas, y números, en un mínimo de tiempo. Se ingresa a la cadena una cantidad de 10512 caracteres de un combinado de letras y números. Reconocer la cadena de letras, números, encriptarlo y des encriptarlo, ver en la figura 4.14.

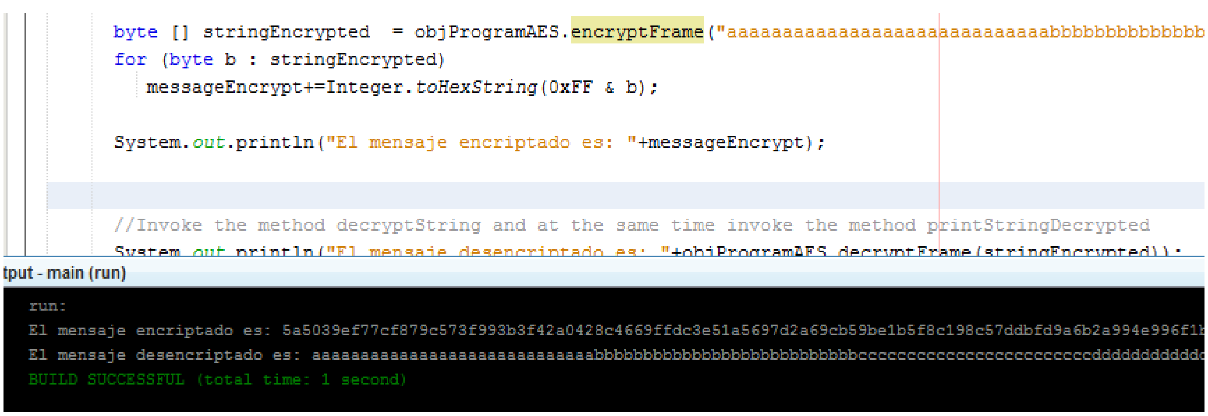


Figura 4.14.- Cadena con combinación de caracteres (Elaboración propia).

Comparativa de cadenas de la ejecución anterior que se pueden observar en la figura 4.15.

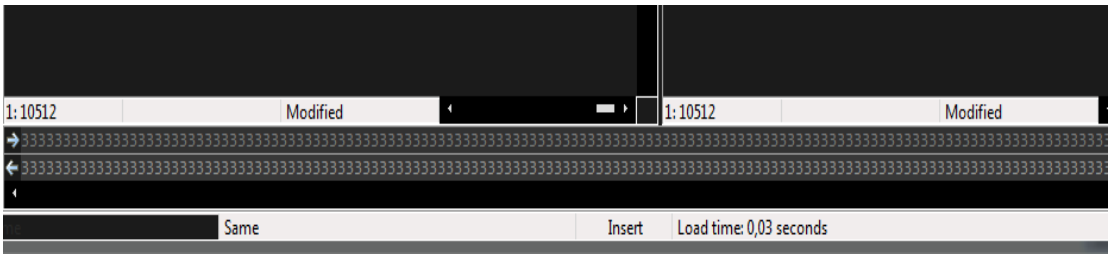


Figura 4.15.- Comparativa de cadenas (Elaboración propia).

TC03 Encriptar y des encriptar una cadena grande con letras mayúsculas, minúsculas, números signos. Compilación de una cadena grande con letras mayúsculas/minúsculas, números y signos, en un mínimo de tiempo. Se ingresa a la cadena una cantidad de 12221 caracteres en un combinado de letras mayúsculas/minúsculas, números y signos. Reconocer la cadena de letras, números y signos, encriptarlo y des encriptarlo. Ver en la figura 4.16 y la comparativa de cadenas en la figura 4.17.

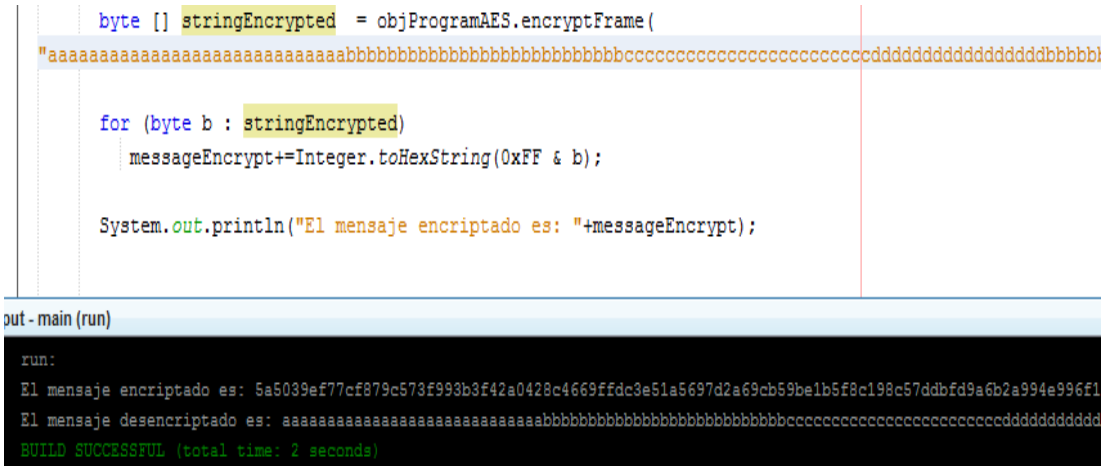


Figura 4.16.- Cadena de 12221 caracteres combinados (Elaboración propia).

Figura 4.17.- Comparativa de grandes cadenas (Elaboración propia).

TC04 Encriptar y des encriptar una cadena grande con letras mayúsculas, minúsculas, números signos y espacios.

Compilación de una cadena grande con letras mayúsculas/minúsculas, números, signos y espacios vacíos, en un mínimo de tiempo. Compilación de la prueba se puede observar en la figura 4.18 y comparativa de cadenas en la figura 4.19.

1. Se ingresa a la cadena una cantidad de 12286 caracteres en un combinado de letras mayúsculas/minúsculas, números, signos y espacios vacíos. Reconocer la cadena de letras, números, signos y espacios, encriptarlo y des encriptarlo.

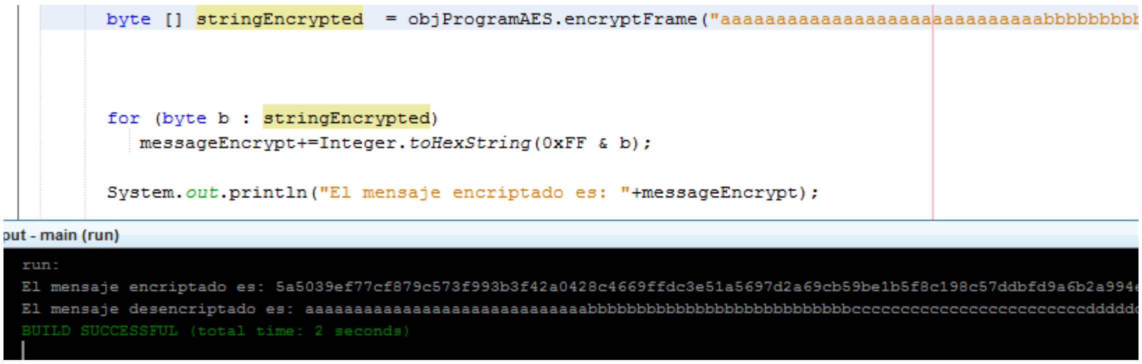


Figura 4.18.- Cadena de 12286 caracteres incluyendo espacios vacíos (Elaboración propia).

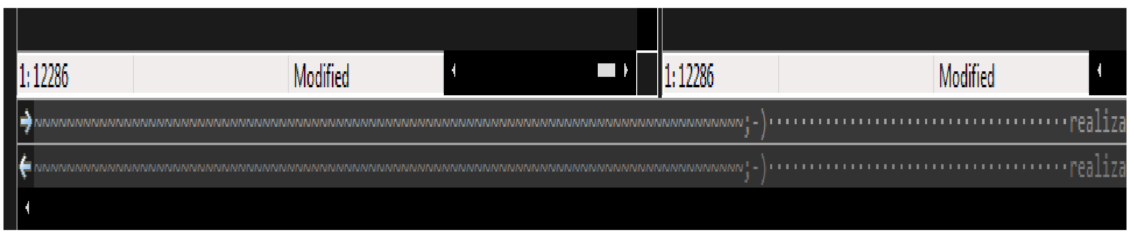


Figura 4.19.- Comparativa de dos cadenas grandes combinadas (Elaboración propia).

TP04 Caracteres especiales (secuencias de escape)

El módulo debe ser capaz de interpretar cualquier carácter de secuencia de escape y debe encriptar y desencriptar

Casos de pruebas

TC01 Reconocimiento de doble comillas \" Reconocer la cadena con doble comillas, encriptarlo y des encriptarlo.

1 Se ingresa a la cadena dobles comillas, pero con una sintaxis especial y es de la siguiente forma: \", Se ejecuta el módulo y el resultado es una encriptación y la desencriptación de la cadena. La compilación de la prueba se observa en la figura 4.20

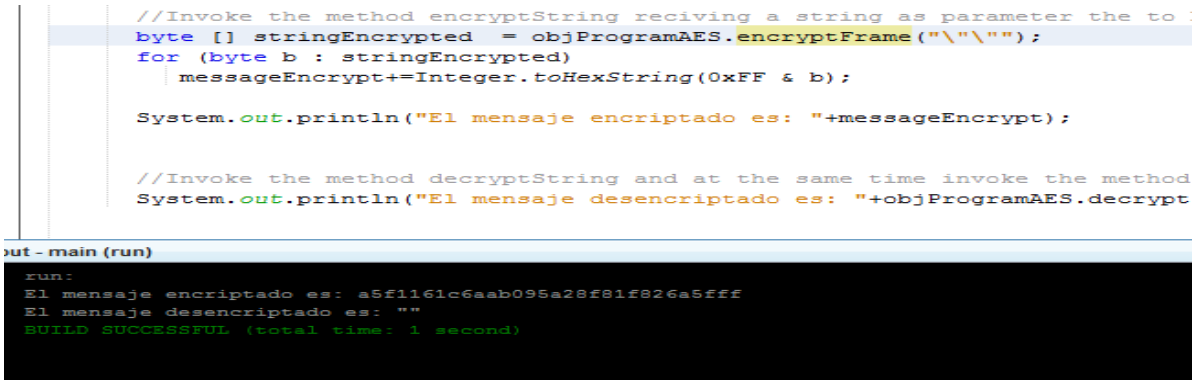


Figura 4.20.- Cadena con doble comilla (Elaboración propia).

TC02 Reconocimiento de la diagonal inversa \\

Reconocer la cadena con diagonal inversa, números, encriptarlo y des encriptarlo.

Compilación de una cadena con diagonal inversa que se puede observar en la figura 4.21

1. Se ingresa a la cadena una diagonal, pero con una sintaxis especial y es de la siguiente forma: \\, Se ejecuta el módulo y el resultado es una encriptación y la desencriptación de la cadena

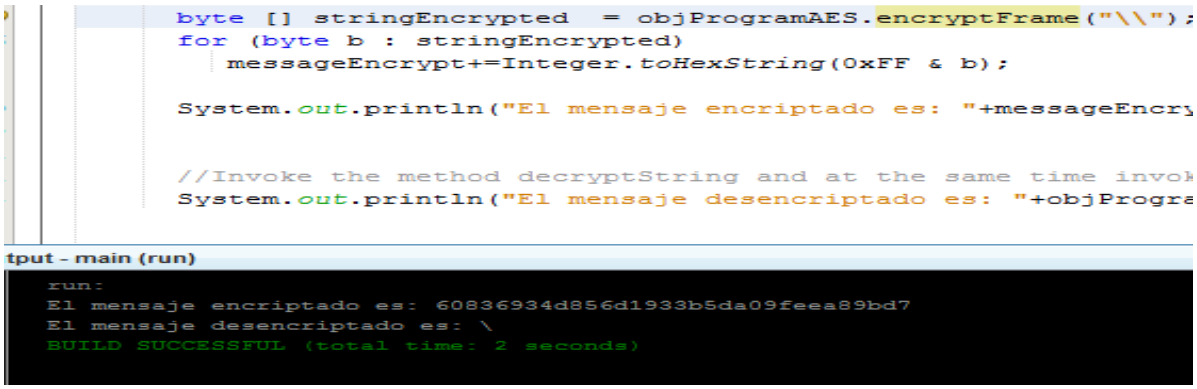


Figura 4.21.- Cadena con diagonal inversa (Elaboración propia).

TC03 Reconocimiento de salto de línea \n Reconocer la cadena con salto de línea (pasa a la siguiente línea), encriptarlo y des encriptarlo.

Compilación de una cadena con salto de línea que se observa en la figura 4.22

1. Se ingresa a la cadena saltos de línea, pero con una sintaxis especial y es de la siguiente forma: \n. Se ejecuta el módulo y el resultado es un salto de renglón además de la encriptación y la desencriptación de la cadena.

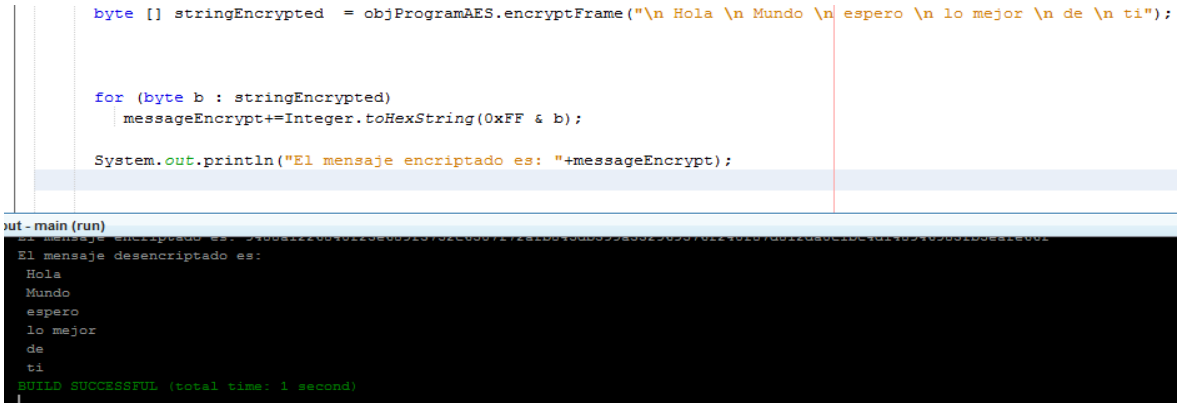


Figura 4.22.- Cadena con salto de línea (Elaboración propia).

TC04 Reconocimiento de retorno de carro \r

Reconocer la cadena con retorno de carro (elimina la línea en donde está ubicado) encriptarlo y des encriptarlo. Compilación de una cadena con retorno de carro que se observa en la figura 4.23.

1. Se ingresa a la cadena con un retorno de carro, pero con una sintaxis especial y es de la siguiente forma: \r. Se ejecuta el módulo y el resultado es que elimina la línea de donde se encuentra ubicado el retorno de carro, además de la encriptación y le des encriptación de la cadena.

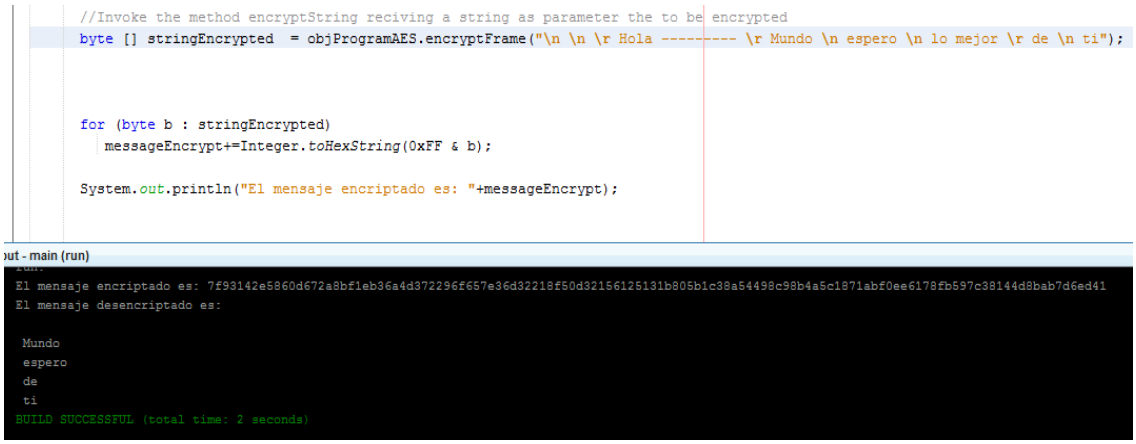


Figura 4.23.- Cadena con retorno de carro (Elaboración propia).

TC05 Reconocimiento de tabulador \t Reconocer la cadena con tabulador (crea espacios) encriptarlo y des encriptarlo.

Compilación de una cadena con Tabulador se observa en la figura 4.24.

1. Se ingresa a la cadena con tabulador, pero con una sintaxis especial y es de la siguiente forma: \t. Se ejecuta el módulo y el resultado es que da un espacio entre los caracteres donde esté ubicado el tabulador, además de la encriptación y la desencriptación de la cadena.

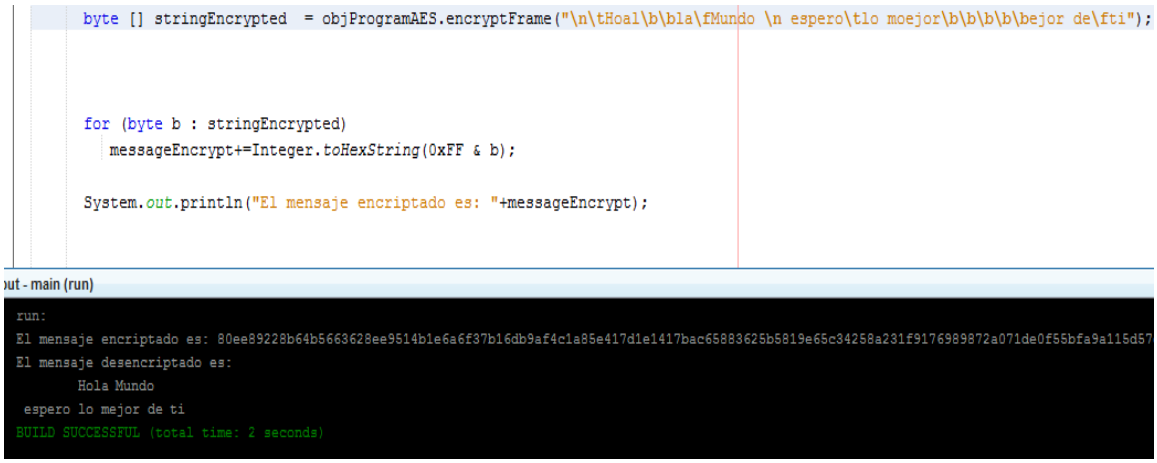


Figura 4.24.- Cadena con tabulador (Elaboración propia).

TC06 Reconocimiento de *BackSpace* \b.

Reconocer la cadena con *backspace* (borra a la izquierda y corrige algún carácter) encriptarlo y des encriptarlo. Compilación de una cadena con *BackSpace* que se observa en la figura 4.25

1. Se ingresa a la cadena con un *BackSpace* pero con una sintaxis especial y es de la siguiente forma: \b. Se ejecuta el módulo y el resultado es que borra a la izquierda y corrige algún carácter de donde se encuentra ubicado, además de la encriptación y la desencriptación de la cadena.

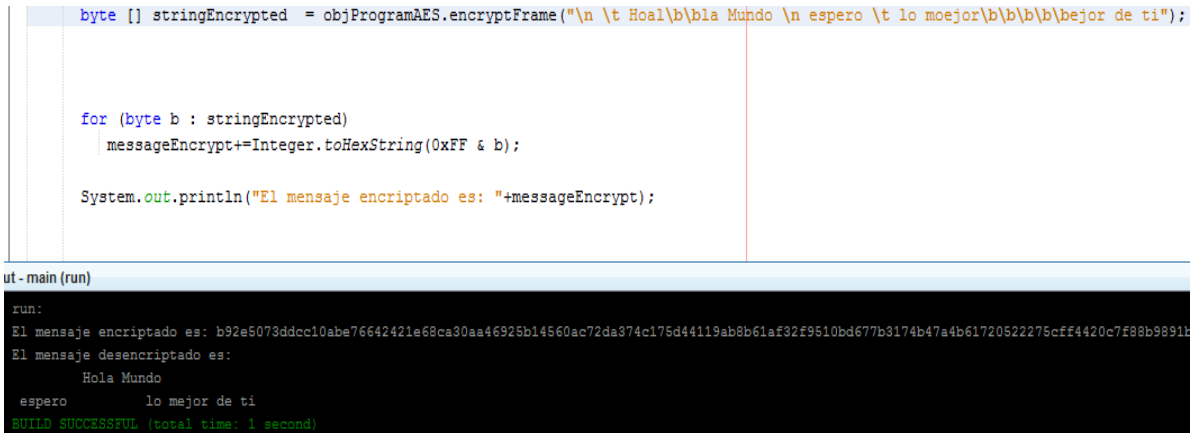


Figura 4.25.- Cadena con BackSpace (Elaboración propia).

TC07 Reconocimiento de *Form Feed* \f

Reconocer la cadena con *Form feed* (identifica el inicio de página o de sección, y los separa) encriptarlo y des encriptarlo. Compilación de una cadena con *Form feed* que se observa en la figura 4.26.

1. Se ingresa a la cadena con *Form feed* pero con una sintaxis especial y es de la siguiente forma: \f. Se ejecuta el módulo y el resultado es que identifica el inicio de página o sección y los separa, además de la encriptación y la desencriptación de la cadena.

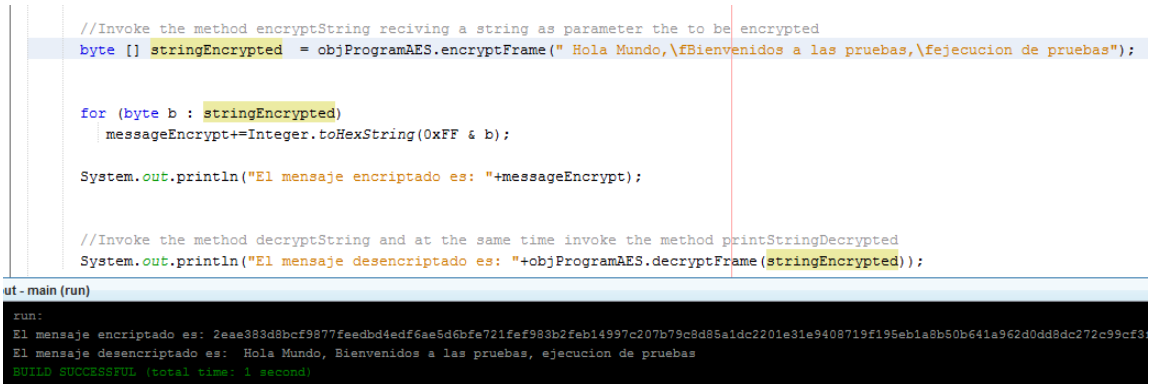


Figura 4.26.- Cadena con form feed (Elaboración propia).

TC08 Reconocimiento de comilla simple \’

Reconocer la cadena con comilla simple encriptarlo y des encriptarlo.

Compilación de una cadena con comilla simple que se observa en la figura 4.27.

1. Se ingresa a la cadena con una comilla simple, pero con una sintaxis especial y es de la siguiente forma: \’. Se ejecuta el módulo y las comillas simples son interpretados, además de la encriptación y la desencriptación de la cadena.

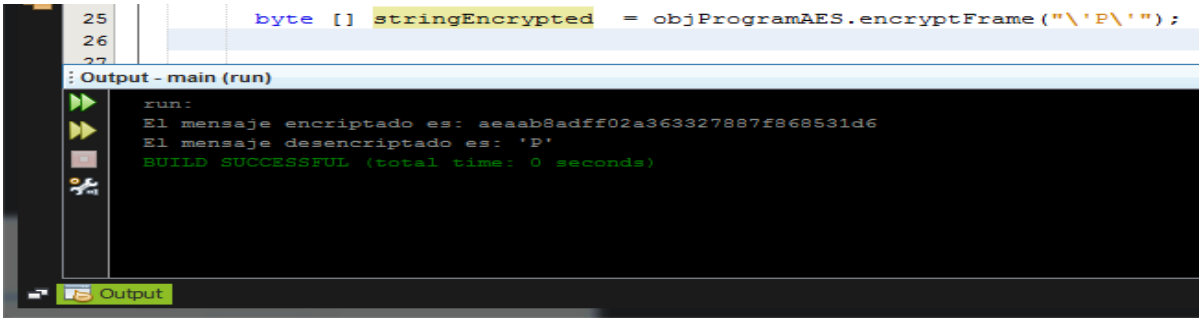


Figura 4.27.- Cadena con comilla simple (Elaboración propia).

TC09 Reconocimiento de un combinado de secuencias de escape (\”. \’, \n, \t, \b, \\, \r).

Reconocer la cadena con un combinado de secuencias de escape (esta se ingresa a la cadena en una frase, dentro de ella se combina todas secuencias de escape para visualizar su funcionalidad) encriptarlo y des encriptarlo. Compilación de una cadena con una combinación de secuencias de escape que se observa en la figura 4.28.

1. Se ingresa a la cadena con un retorno de carro, pero con una sintaxis especial y es de la siguiente forma: \”. \’, \n, \t, \b, \\, \r. Se ejecuta el módulo y el resultado es la interpretación de las secuencias de escape, además de la encriptación y la desencriptación de la cadena.

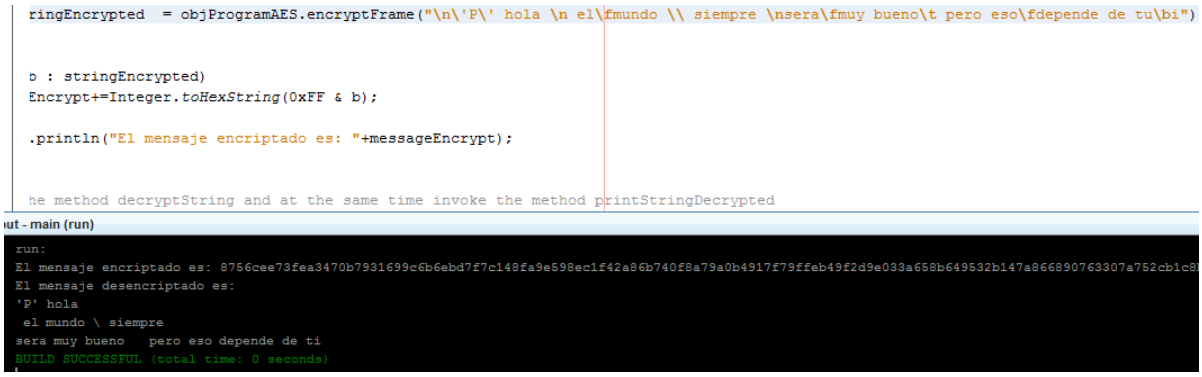


Figura 4.28.- Cadena con secuencia de escape (Elaboración propia).

4.2.2 Verificación al módulo de *CRC* (Calculo de Redundancia Cíclica)

Se diseñó un plan de pruebas enfocado a un módulo de cálculo *CRC* (Calculo de Redundancia Cíclica), de la cual se Aplica un tren de pruebas donde valide que realmente está cumpliendo con los requerimientos necesarios para el *Head End System*, este módulo recibirá *frames* o paquetes que se envían desde un medidor inteligente en forma de Hexadecimal, los *frames* están estructurados con base al estándar de comunicación *ANSI C12.18.*

El *frame* se encarga de transportar la información en bytes, por ello que para asegurar que no se han corrompido esos datos y validar la estructura del *frame* se calcula el *CRC*, este es un código de detección de errores usado frecuentemente en redes digitales y en dispositivos de almacenamiento para detectar cambios accidentales en los datos.

Los bloques de datos ingresados en estos sistemas contienen un valor de verificación adjunto, basado en el residuo de una división de polinomios; el cálculo es repetido, y la acción de corrección puede tomarse en contra de los datos presuntamente corruptos en caso de que el valor de verificación no concuerde.

Este módulo se encontrará integrado en el *Head End System*, que permitirá recibir toda la información que la *CCG* (centro de control de gabinete) envié de los medidores y además toda la funcionalidad del módulo lo hará de manera automática y oculta por lo tanto no será visible por el usuario.

Los paquetes que enviara la CCG deben tener la estructura siguiente:

<packet>:: = <STP><Identity><ctrl><Seq-nrb><length><data><CRC>

EE = Carácter de comienzo de paquete

En base a esta estructura se ingresarán los *frames* en el módulo de cálculo *CRC*, Cabe recalcar que el *frame* que llegue a este módulo aun no contiene el *CRC*, por lo tanto, el *frame* que se ingresara solo será hasta *<data>*. Antes de comenzar se debe tener los datos generales del módulo a evaluar cómo se observa en la tabla 4.4.

Para aplicar el plan de pruebas que se puede ver en la tabla 4.5 el programador implemento una interfaz temporal para ingresar *frames* en un campo y realizar el cálculo mediante un botón que contiene el evento de calcular un polinomio CRC-16: x16 + x12 + x5 + 1. Para flujos de 8 bits, con 16 de redundancia.

Tabla 4.4.- Datos Generales del módulo CRC (Elaboración propia).

DATOS GENERALES	
No Solicitud	3
Fecha	05/04/2017
Desarrollado por	Moisés Morales Guzmán
Producto	Módulo de cálculo de redundancia cíclica CRC
Versión	1.0
Fecha de prueba	11/04/2017
Lugar de prueba	Área de Ingeniería de software
Nombre del tester	Manuel Flores Nava
Revisor	Agustín Sánchez Atonal

Tabla 4.5.- Plan de pruebas del módulo CRC (Elaboración propia).

PLAN DE PRUEBAS CRC							
TP	**TP DESCRIPCIÓN**	**TC**	**TC DESCRIPCIÓN**	**#EP**	**#PE**	**#P F**	**#PN**
TP01	Cálculo de CRC	TC01	Verificación del cálculo del CRC				
TP02	Frames desordenados	TC01	Frames sin espacios				
		TC02	Frames con espacios al azar				
		TC03	Carácter de comienzo (EE) en mayúscula, minúscula y combinado				
TP03	Tipos de frames según cada dispositivo	TC01	Frame para el dispositivo CCG (20)				
		TC02	Frame para el dispositivo medidor (00)				
		TC03	para el dispositivo IR´S (80 a 85)				
TP04	servicios de dispositivos en frames	TC01	servicio identification				
		TC02	Servicio Log On				
		TC03	Servicio security				
		TC04	Servicio Full write				

		TC05	Servicio Pwrite Offset				
		TC06	Servicio Full Read				
		TC07	Servicio Pread Offset				
		TC08	Servicio Log Off				
		TC09	Servicio Terminate				
TP05	Advertencias de frame's inválidos y corrompidos	TC01	Frame invalido				
		TC02	Frame corrompido				
		TC03	Manejo de excepciones ante frames inválidos y corrompidos				
TP06	Pruebas de regresión después de la corrección.	TC01	Manejo de excepciones ante frame's inválidos y corrompidos (Corregidos)				
		TC02	Prueba de regresión en frames de manera general (ordenados, desordenados, mayúscula y minúscula)				

Herramientas y entornos

- Preparación de la Laptop HP650 para poder utilizar los entornos para compilar *Java*.
- *C*onsola de *Windows 7 pro*
- Preparación de *EOLO* (herramienta proporcionado por pruebas y validación de la empresa).

Las herramientas se visualizan en la figura 4.29.

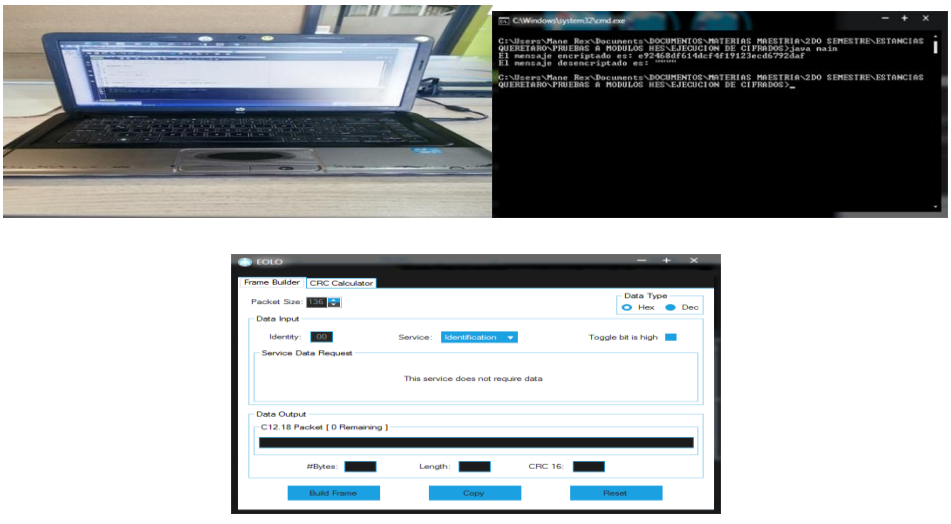


Figura 4.29.- Herramientas y entornos para ejecutar pruebas para CRC (Elaboración propia).

TP01 Cálculo de CRC

Como primera importancia es que el módulo debe calcular el *CRC* de cualquier *frame* que este correctamente estructurado al estándar *ANSI C12.18*.

Casos de pruebas

TC01 Verificación del cálculo del *CRC*

1. Se ingresa un *frame* correctamente estructurado al estándar *ANSI C12.18*. Se ejecuta el módulo de cálculo de *CRC* con un *frame* que está correctamente estructurado al estándar *C12.18*, y como lo importante de este módulo es que calculo el CRC de cualquier *frame* entonces se espera como resultado el *CRC* correcto del *frame*.
2. Se genera un *frame* desde *EOLO Frame*: EE 00 00 00 00 15 51 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 CRC: D6 2A
3. El *frame* generado se ingresa al módulo de cálculo de *CRC* sin el numero *CRC*. Se ejecuta el módulo y este debe reconocer que el *frame* ingresado es correcto y debe generar el *CRC*, este resultado se compara con la herramienta *EOLO* y sí coinciden en el *CRC* entonces la prueba ha sido pasada. Ver la figura 4.30 y 4.31.

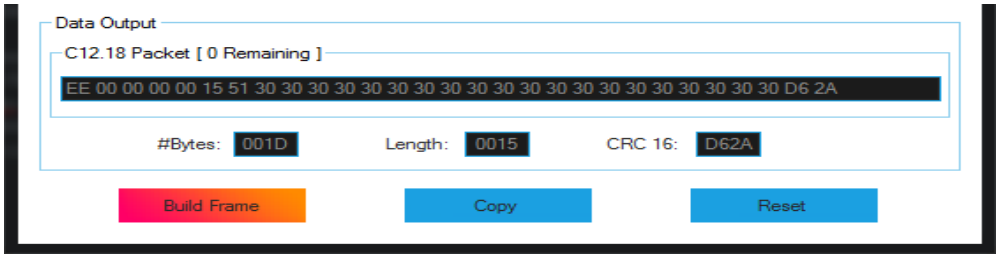


Figura 4.30.- Generación de un frame desde EOLO (Elaboración propia).

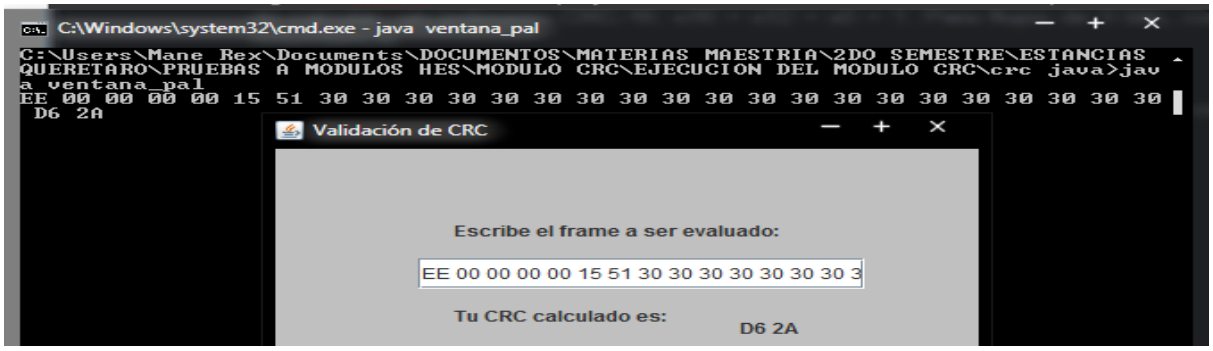


Figura 4.31.- Ejecución del CRC en consola (Elaboración propia).

TP02 *Frames* aleatorias

El módulo debe ser capaz de reconocer *frames* desordenados (sin espacios, con espacios al azar, mayúsculas, minúsculas o combinado) y calcular su CRC, esta se observa en la figura 4.32 para generar *frame* y ejecutar código se observa en la figura 4.33.

casos de pruebas

TC01 *Frames* sin espacios

Se ingresan *frames* sin espacios (todos los caracteres están juntos) respetando la estructura *C12.18* Se ejecuta el módulo para calcular *CRC* del *frame* y su resultado de *CRC* debe ser igual a la de *EOLO*. Esto por la razón de que existe la posibilidad que los dispositivos envíen paquetes desordenados por cualquier situación, la cual no afecta mientras respete la estructura del paquete.

2. Se genera un *frame* desde *EOLO*, pero sin espacios (los espacios se quitan de manera manual)
3. EE000000001551303030303030303030303030303030303030303030 CRC: D6 2A

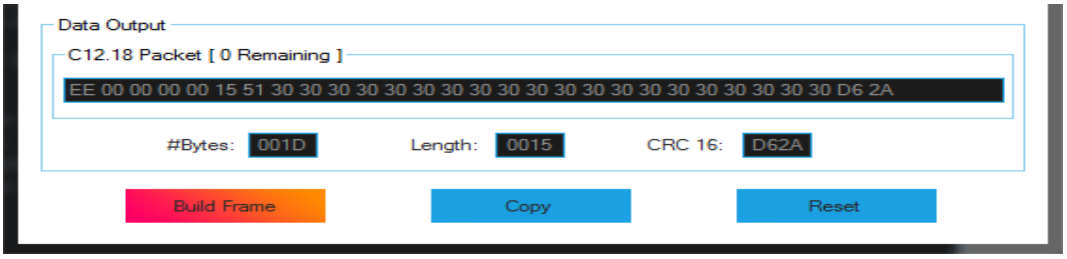


Figura 4.32.- Se genera un frame desde EOLO sin espacios (Elaboración propia).

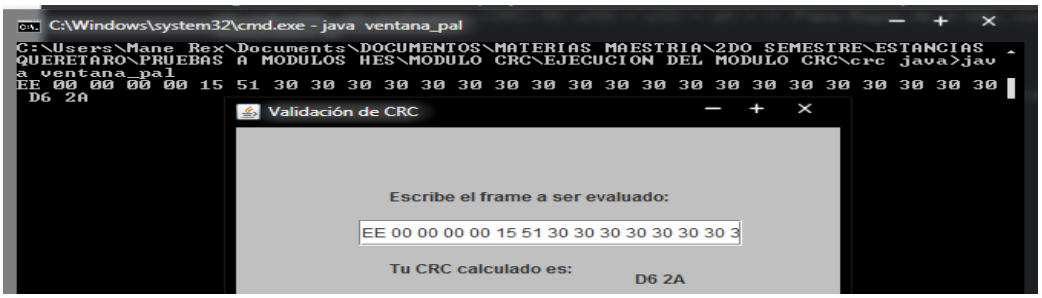


Figura 4.33.- Se genera frame sin espacios en CRC y coincide con EOLO (Elaboración propia).

TC02 *Frames* con espacios al azar.

Se ingresan *frames* con espacios al azar (todos los caracteres están separados con un espacio o cuatros espacios entre caracteres) respetando la estructura C12.18 al módulo, este debe reconocerlo y calcular el CRC. Mientras el frame respete la estructura *C12.18*, el módulo debe aceptarlo, Se ejecuta el módulo para calcular *CRC* del *frame* y su resultado de *CRC* debe ser igual a la de *EOLO*.

Se genera un *frame* desde *EOLO*, pero con espacios múltiples (los espacios se agregan de manera manual) E E 0 000 000 0155 1303 03030 303 030303 0303030 3030 3030 30303030 CRC: D6 2A.

La ejecución se puede observar en la figura 4.34 la generación de los *Frames* y la ejecución en la figura 4.35.

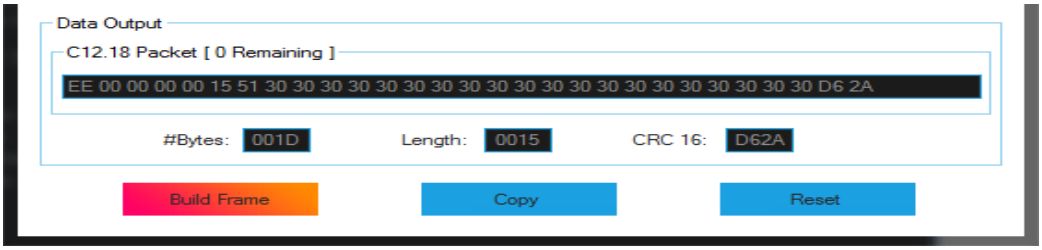


Figura 4.34.- Se genera un frame desde EOLO con espacios múltiples (Elaboración propia).

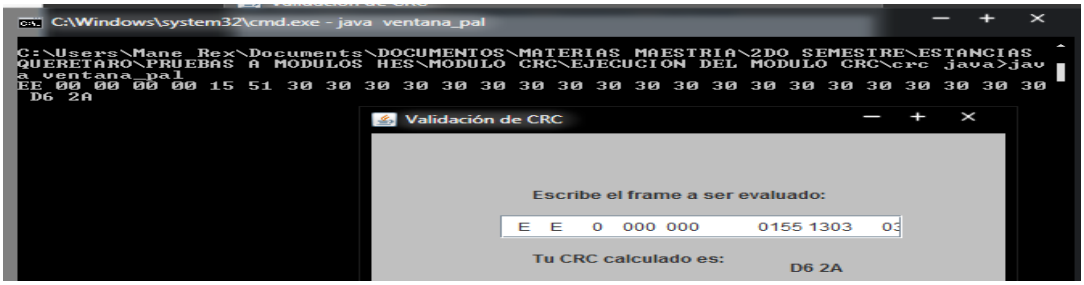


Figura 4.35.- Se genera frame con espacios en CRC y coincide con EOLO (Elaboración propia).

TC03 Carácter de comienzo (EE) en mayúscula, minúscula y combinado

Se ingresan *Frames* donde el carácter de comienzo de paquete sea en mayúscula, minúscula o combinado de las dos maneras respetando la estructura *C12.18* al módulo, este debe reconocerlo y calcular el *CRC.* Mientras el *frame* respete la estructura *C12.18*, el módulo debe aceptarlo y calcular su *CRC.* Se ejecuta el módulo para calcular CRC del *frame* y su resultado de CRC debe ser igual a la de EOLO. Se pueden observar en la figura 4.36 la generación de los *Frame* y 4.37 para observar la ejecución respectivamente.

1. EE 00 00 00 00 15 51 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
2. ee 00 00 00 00 15 51 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
3. Ee 00 00 00 00 15 51 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
4. eE 00 00 00 00 15 51 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
5. CRC: D6 2A

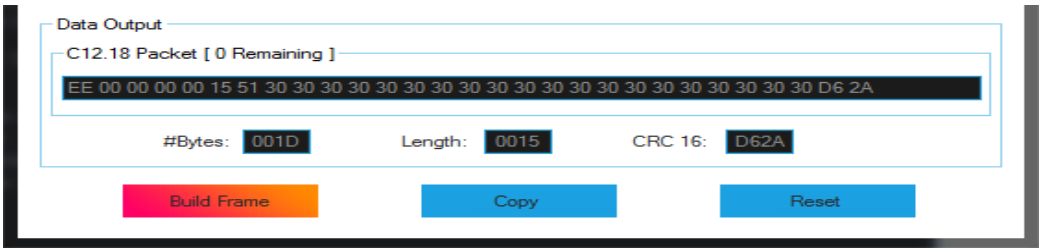


Figura 4.36.- Se genera un frame desde EOLO de comienzo de paquete (Elaboración propia).

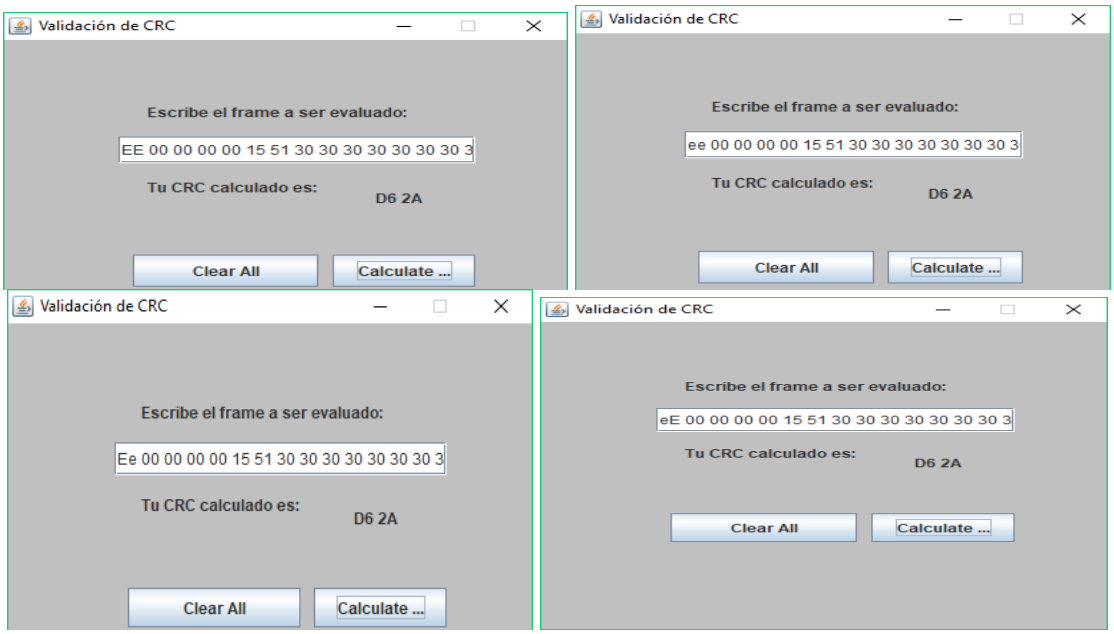


Figura 4.37.- Ejecución de cadena mayúscula, minúscula y combinada (Elaboración propia).

TP05 Advertencias de *frames* inválidos y corrompidos

El módulo debe ser capaz de reconocer cuando un *frame* es invalido por caracteres fuera de un hexadecimal o por que han sido corrompido por alguna modificación, pero lo más importante es que ante estos casos el módulo lance excepciones controladas. se puede observar en la figura 4.38.

casos de pruebas

TC01 *Frame* invalido,

El módulo debe reconocer cuando se ha ingresado un *frame* con caracteres inválidos, estos pueden ser cualquier letra del abecedario, que no forma parte de un hexadecimal (0 al 9 y A-F) por ejemplo de la G a la Z o cualquier símbolo existente.

1.- Se genera un *frame* desde *EOLO*

Paquete original: EE 00 00 00 00 0B 40 00 00 00 05 00 00 00 00 00 00 CRC: AC ED

Paquete con carácter invalido: EE 00 00 00 00 0G 40 00 00 00 05 00 00 00 00 00 00

Invalido

2.-El *frame* generado se ingresa al módulo de cálculo de *CRC* sin el *CRC.* Se ejecuta el módulo y este debe reconocer que el *frame* ingresado es invalido por un carácter que esta fuera de un hexadecimal y debe arrojar una advertencia.

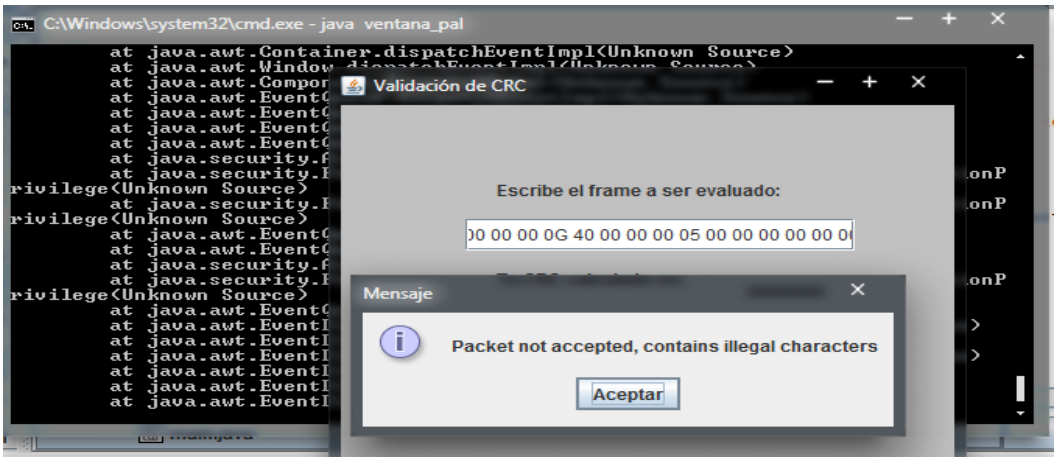


Figura 4.38.- Paquete con carácter invalido (Elaboración propia).

TC02 *Frame* corrompido.

El *frame* es modificado en la longitud del paquete o se elimina una parte de la estructura y por ese cambio el *frame* es corrompido, el módulo debe reconocerlo y arrojar una advertencia de que el *frame* ha sido corrompido. Se puede observar en la figura 4.39.

1.- Se genera un *frame* desde *EOLO*. El *frame* generado se ingresa al módulo sin el *CRC*

Paquete original: Paquete original: EE 00 00 00 00 0B 40 00 00 00 05 00 00 00 00 00 00 CRC: AC ED

Paquete modificado: EE 00 00 00 00 0A 40 00 00 00 05 00 00 00 00 00 00

2.- Se ejecuta el módulo y este debe reconocer que el *frame* ingresado esta corrompido y debe arrojar una advertencia.

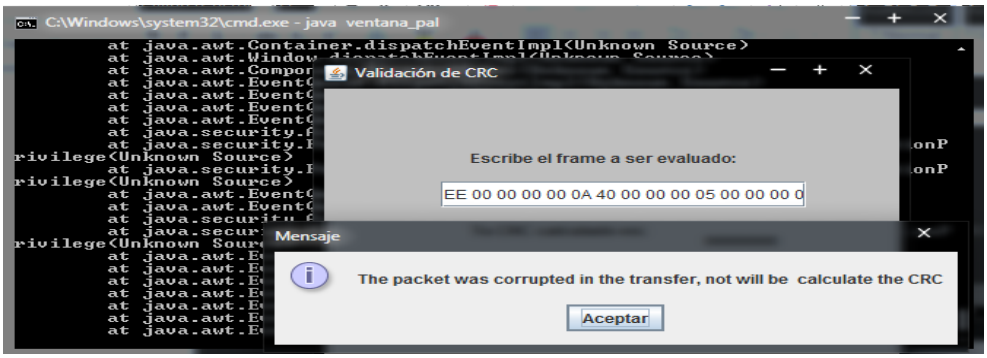


Figura 4.39.- Paquete corrompido (Elaboración propia).

TC03 Manejo de excepciones ante *frames* inválidos y corrompidos.

Se ingresa un *frame* con los mismos casos anteriores, pero ahora es observar si las excepciones que realiza el módulo los puede controlar por lo que no solo lanza advertencias sino excepciones de manera interna de la cual deben ser controladas. Se puede observar la compilación en la figura 4.40.

1.- Se genera un *frame* desde *EOLO*

EE 00 00 00 00 0G 40 00 00 00 05 00 00 00 00 00 00

EE 00 00 00 00 0A 40 00 00 00 05 00 00 00 00 00 00

2.-El *frame* generado se ingresa al módulo de cálculo de *CRC* sin el *CRC. S*e ejecuta el módulo y este debe reconocer que el *frame* ingresado esta

Se ingresa el siguiente *frame* con carácter invalido (se agrega un carácter G)

EE 00 00 00 00 0G 40 00 00 00 05 00 00 00 00 00 00

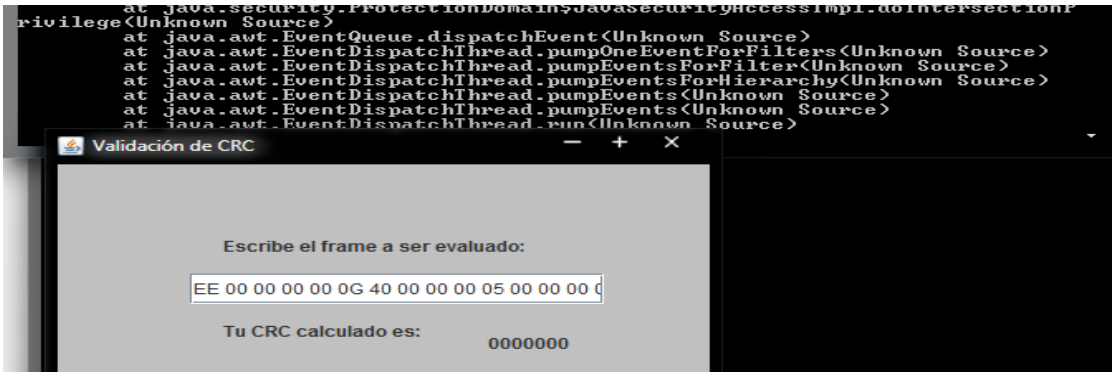


Figura 4.40.- Excepciones de CRC ante alguna falla (Elaboración propia).

Se ingresa un *frame* corrompido EE 00 00 00 00 0A 40 00 00 00 05 00 00 00 00 00 00

La compilación de la prueba se observa en la figura 4.41.

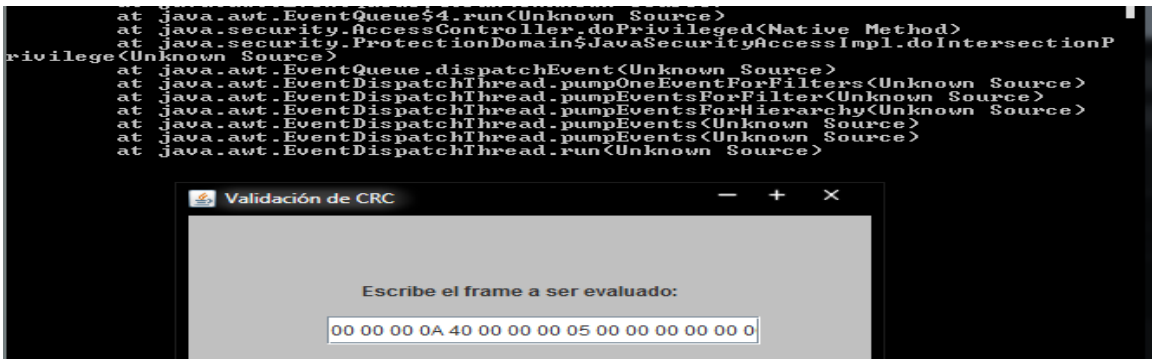


Figura 4.41.- Excepción de CRC ante un frame corrompido (Elaboración propia).

Prueba fallida en excepciones y se realiza corrección por parte del programador

TC01 Manejo de excepciones ante *frames* inválidos y corrompidos (Corregidos)

El módulo debe ser capaz de reconocer cuando un *frame* es invalido por caracteres fuera de un hexadecimal o por que han sido corrompido por alguna modificación, pero lo más importante es que ante estos casos el módulo lance excepciones controladas. En el caso de prueba anterior fueron fallidas y el problema se notificó para que sea corregido.

El módulo debe reconocer cuando se ha ingresado un *frame* con caracteres inválidos, estos pueden ser cualquier letra del abecedario, que no forma parte de un hexadecimal (0 al 9 y A-F) por ejemplo de la G a la Z o cualquier símbolo existente. Como se observa en la figura 4.42

1. Paquete original: EE 00 00 00 00 0B 40 00 00 00 05 00 00 00 00 00 00 CRC: AC ED
2. Paquete con carácter invalido: EE 00 00 00 00 0G 40 00 00 00 05 00 00 00 00 00 00

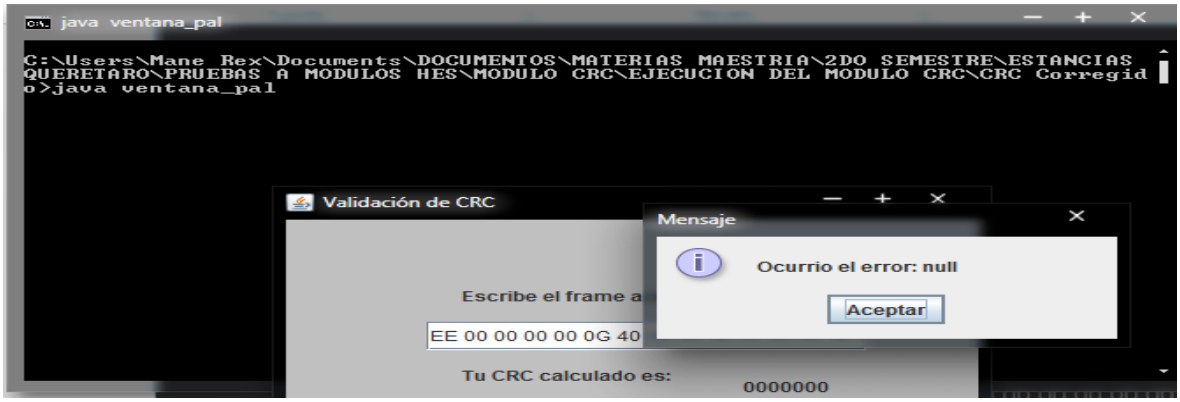


Figura 4.42.- Alertas de CRC ante algún fallo (Elaboración propia).

Pero también el módulo debe reconocer cuando se ha ingresado un *frame* corrompido esto puede suceder cuando se ha modificado alguna parte de la estructura o tamaño del paquete. Ver la figura 4.43

1. Paquete original: Paquete original: EE 00 00 00 00 0B 40 00 00 00 05 00 00 00 00 00 00 CRC: AC ED
2. Paquete modificado: EE 00 00 00 00 0A 40 00 00 00 05 00 00 00 00 00 00

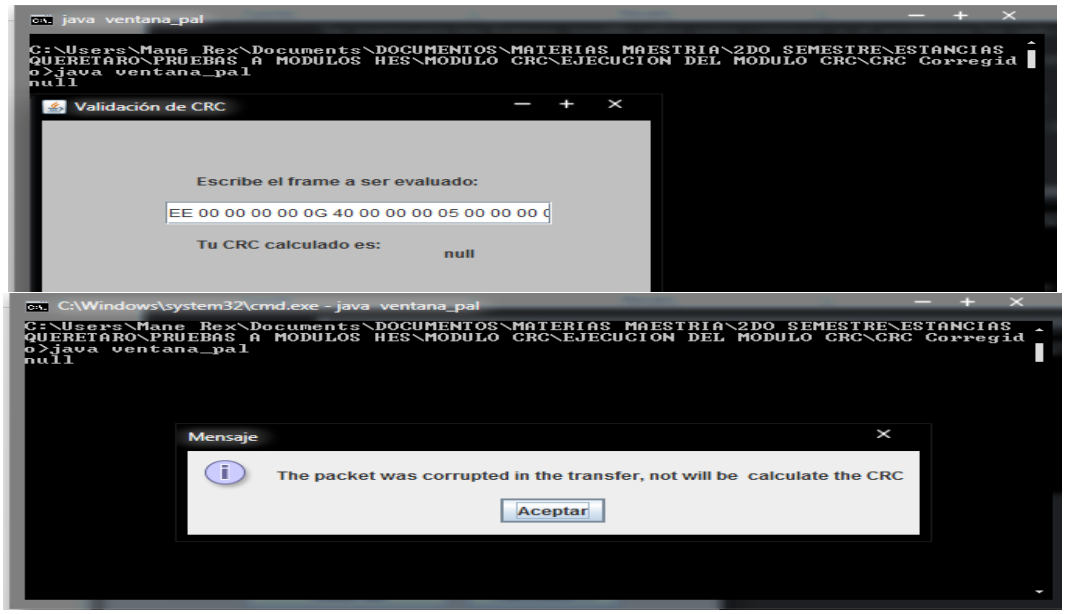


Figura 4.43.- Alertas de CRC ante algún paquete corrompido (Elaboración propia).

4.2.3 Verificacion al módulo de web service

Antes de aplicar alguna prueba se obtienen los datos generales sobre el módulo como se observa en la tabla 4.6 y Se diseñó un plan de pruebas que se puede ver en la tabla 4.7 para el módulo de *web service* que se implementara dentro del proyecto *HES.*

Un *web service* es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como internet.

De una manera más clara se podría decir que un *web service* es una función que diferentes servicios o equipos utilizan; es decir, solo se envían parámetros al servidor (lugar donde está alojado el *web service*) y éste responderá la petición.

El módulo *web service* que se ha desarrollado para el proyecto *Head End System*, el módulo se pondrá bajo prueba, su funcionamiento es limitada en solo recibir peticiones, entonces las pruebas que se van aplicar son muy específicas, la solicitud para aplicar las pruebas especifica lo siguiente:

- Comprobar la ejecución de la *web service.*
- Conexión de la *web service* con el servidor del cliente.
- Recibir petición del servidor por medio de un archivo *XML*, la *web service* debe ir por el archivo de petición.
- La *web service* realiza una comparación con el esquema que tiene por default para validar que el archivo respeta el orden de los datos y en caso de que no sea así la *web service* debe reportar el error.
- Guardar en la ruta local asignado para la *web service* para visualizar archivos recibidos.
- Guardar en la base de datos de *Oracle* lo que contiene el archivo de petición *XML*.
- Consultar tablas para verificar que los datos fueron guardados en la Base de Datos *Oracle*.

El trabajo que debe realizar al módulo de *web service*, es que mediante una petición en un archivo *XML* que le enviara al *Management Data Meter System (MDMS)* al *web service* que será integrado en el *Head End system*, comenzara un proceso de operaciones para asignar tareas que debe realizar los dispositivos o medidores inteligentes.

Los archivos que recibirá el módulo de *web services* serán en formato *XML* que se comparan contra un archivo que define la estructura o esquema que debe respetar todos los archivos enviados por el medidor.

Tabla 4.6.- Datos generales de evaluación del módulo Web Service (Elaboración propia)

DATOS GENERALES	
No Solicitud	2
Fecha	29/05/2017
Desarrollado por	Doris Castro
Producto	Módulo Web Service
Versión	1.0
Fecha de prueba	31/05/2017
Lugar de prueba	Área de Ingeniería de software
Nombre del tester	Manuel Flores Nava
Revisor	Agustín Sánchez Atonal

Tabla 4.7.- Plan de pruebas para el módulo Web Service (Elaboración propia).

PLAN DE PRUEBAS DE LA WEB SERVICE							
TP	**TP DESCRIPCIÓN**	**TC**	**TC DESCRIPCIÓN**	**#EP**	**#PE**	**#PF**	**#PN**
TP01	Ejecución de la web service	TC01	Ejecución de web service montado en el servidor apache Tomcat en eclipse				
TP02	Recepción de petición de tareas	TC01	Recibir petición en archivo xml				
		TC02	Guardar el archivo de petición *XML*				
		TC03	Validación de archivos *XML*				
		TC04	Guarda a la base de datos de hibernate (*Oracle 11g)*				

TP01 Ejecución de la *Web service*

TC01 ejecución de *web service* montado en el servidor apache *Tomcat*.

En este caso de prueba se comprueba que el módulo de *web service* puede ser ejecutada dentro del entorno simulado (eclipse con *Tomcat*) y observar si se ejecuta de manera rápida y sin algún error. Las herramientas han sido instaladas previamente por el *tester*.

En la figura 4.44 se observa que el módulo de *web services* se está ejecutando esto se puede comprobar con el mensaje *Server startup* in 8046 ms por lo tanto esta prueba ha sido pasada.

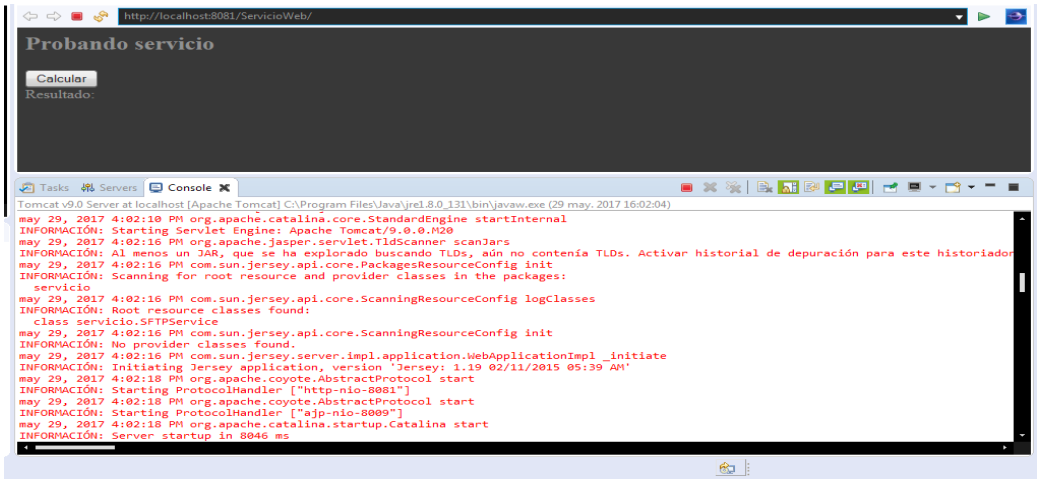


Figura 4.44.- Módulo de web service en ejecución en eclipse (Elaboración propia).

TP02 Recepción de petición de tareas.

TC01 Recibir petición en archivo *XML*.

En este caso de pruebas se simulará la comunicación entre la *web service* con el *MDMS* en la cual este último mandará una petición en archivo *XML*.

El entorno simulado es configurado por medio de 2 laptops de diferentes sistemas operativos (*Windows 7* en la *web services* y *Ubuntu* como un servidor del cliente), la conexión se realiza por medio de una red local tomando como referencia las ip.

El servidor del cliente debe colocar un archivo de petición (tareas) en formato *XML*, donde la *web service* va por el archivo de petición, todo este proceso lo realiza de manera remota para simular la operación de transferencia de archivos y comunicación del *Head End System* con un *MDMS*, además de probar la funcionalidad de la *web service*. En la figura 4.45 se muestra como el servidor coloca la petición en un archivo *XML* (enviounaTarea.xml) por lo tanto la *web service* reconoce el archivo y lo valida.

Figura 4.45.- Colocación de archivo de petición de algún servidor externo (Elaboración propia).

Como servidor del cliente el *MDMS* de manera remota (para las pruebas se realiza en una red *LAN*) se envía la petición a la *web service* y al ejecutar el siguiente link se carga el servicio de manera local (solo en pruebas). En la figura 4.46 se puede observar cuando se carga el servicio

1. http://localhost:8081/ServicioWeb/services/ReadValidXML/ReadValid?strFile=enviounaTarea.xml
2. Se carga el servicio de manera local y toma el archivo de petición del servidor conectado

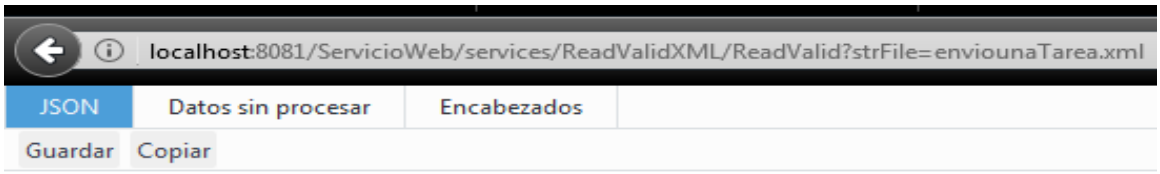


Figura 4.46.- Carga del servicio de la web service (Elaboración propia).

TC02 Guardar el archivo de petición *XML*

En este caso de prueba la *web service* debe Guardar el archivo de petición a la ruta especificada de la *web service*, en este caso se especifica en una carpeta local llamada ArchivosD.

En la figura 4.47 la *web service* ha tomado el archivo enviounaTarea.xml y lo guarda de manera local en una carpeta donde se le asigno de manera especial para la *web service*, donde recibirá todos los archivos que tome.

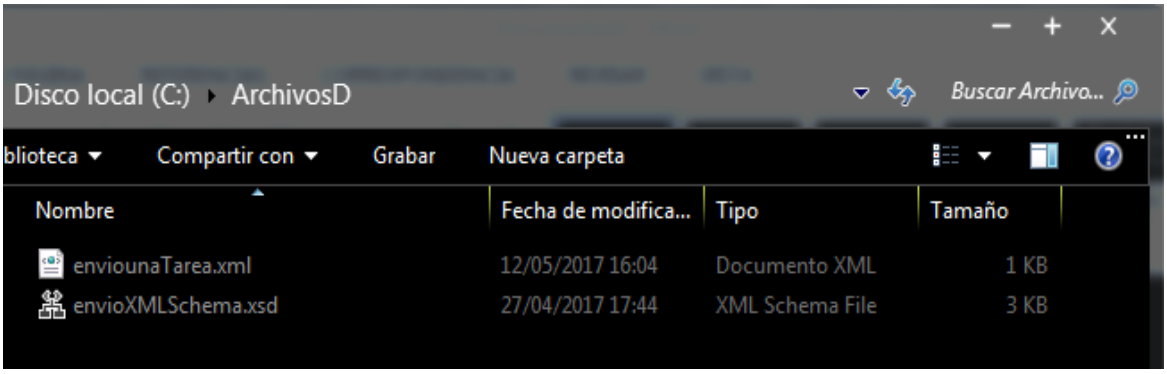


Figura 4.47.- Petición guardado en carpeta asignada para web service (Elaboración propia).

TC03 Validación de archivos *XML*

En este caso de prueba se probará el comportamiento de la *web service* cuando un archivo XML cumple o no con su esquema de *XSD*, el utilizar esquemas *XSD* facilita la adaptación para que se configure la comunicación con otros MDMS. También se analiza cuando no se encuentra ninguna petición en la *MDMS*.

Se evaluará la buena respuesta que da la *web service* ante esas situaciones que permitan reconocer que los archivos si son validados conforme al esquema base del mismo. Se puede observar la ejecución de la prueba en la figura 4.48 el archivo de petición de MDMS y 4.49 el esquema XSD respectivamente

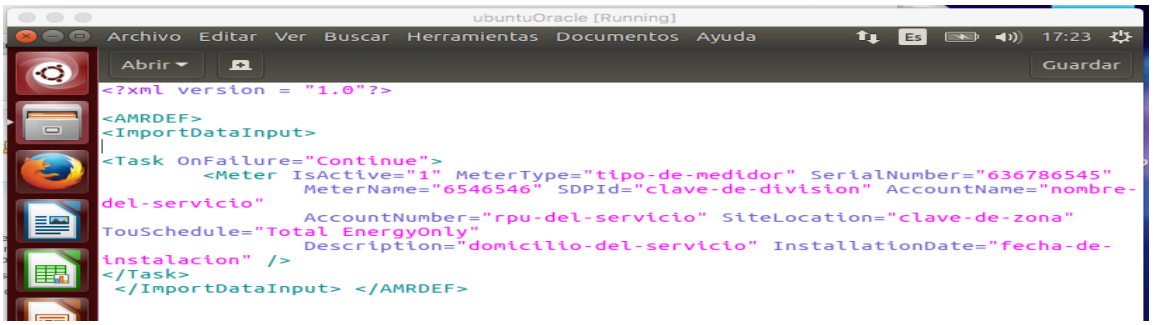


Figura 4.48.- Archivo de tareas (petición) con la estructura correcta (Elaboración propia).

Figura 4.49.- Esquema para validar archivos XSD por web service (Elaboración propia).

Cuando el archivo es validado la *web service* lo recibe y lo guarda de manera local en la ruta especificada para los archivos que recibe, en la figura 4.50 se muestra como la operación de validación fue cumplida ya que guardo en su carpeta el archivo *XML* (enviounaTarea.xml)

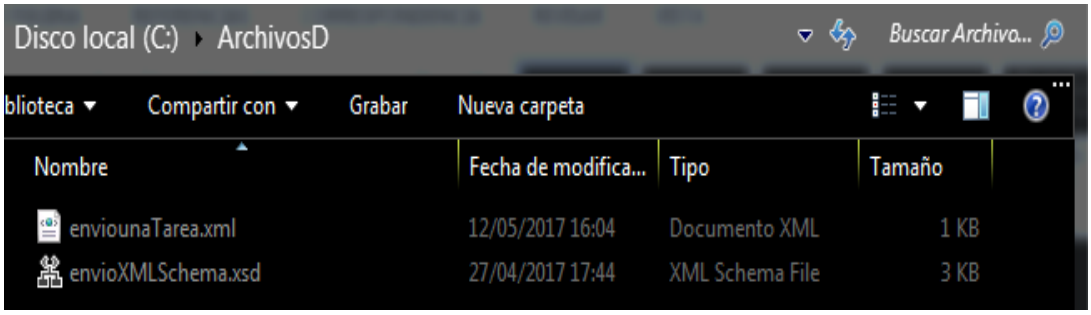


Figura 4.50.- Validación y guardado de archivo XML (Elaboración propia).

En la figura 4.51 se muestra que la *web service* ha detectado un error controlado al momento de ejecutar la página web esto debido a que la *web service* no reconoce el archivo *XML* de petición. Ante ello sugiere revisar el log de errores la cual debe encontrarse en la carpeta que asignamos a la *web service*.

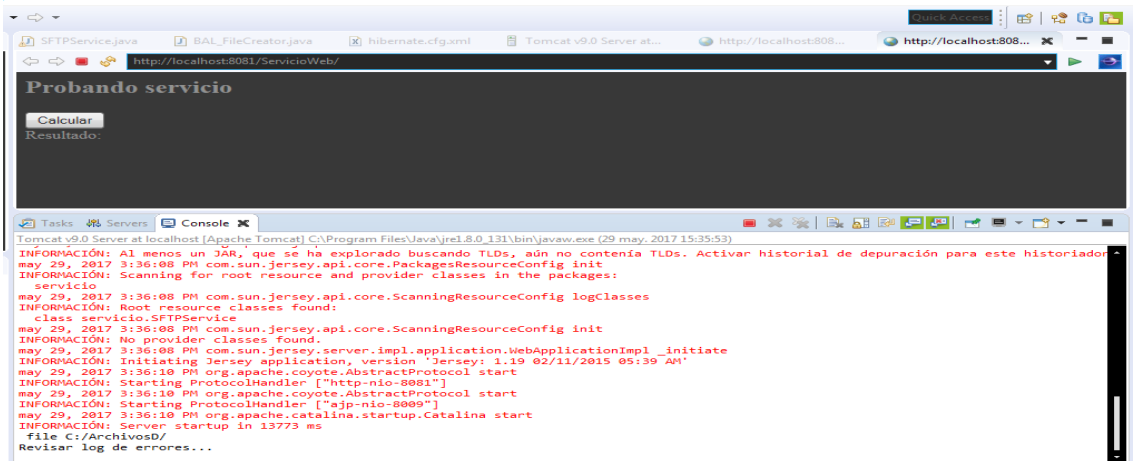


Figura 4.51.- Sugerencia de la web service para revisar el registro de errores (Elaboración propia).

En las figuras 4.52 y 4.53 se revisa el registro de errores que se guardó como un archivo *txt* (HesActivityLog.txt) en la ruta especificada (ArchivosD) con el mensaje de que el archivo no existe (*No such file*) o que el archivo fallo en la autenticación (*auth fail)*, como el módulo de *web service* no será visible por el usuario por lo que toda operación se realizara de manera interna por lo que el control de los errores es lo más apropiado por medio de archivos .txt, por lo tanto esta prueba ha sido pasada.

Observación: en el archivo log se debe ser más específico el tipo de errores que arroja la *web service*, por ejemplo: que el archivo no cumple con el esquema (envioXMLSchema.xsd) o el tipo de formato del archivo no es el correcto, para que así se conozca la razón del error y se corrija lo más pronto posible.

Figura 4.52.- Registro de errores (HesActivityLog.txt) en la carpeta ArchivosD (Elaboración propia).

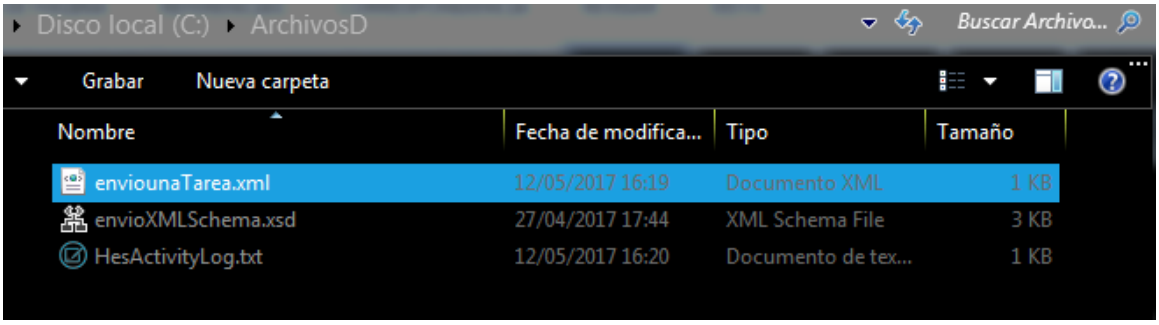


Figura 4.53.- Visualización de un registro de errores (Elaboración propia).

TC04 guarda a la base de datos de *Hibernate (Oracle 11g)*

En este caso de prueba se comprobará si la *web service* tiene conexión a *hibernate* donde recibirá y guardará todas las tareas del archivo de petición

Una vez validada el archivo *XML* (enviounaTarea.xml) donde cumple con el esquema de la *web service*, el archivo es guardado en la base de datos de *hibernate (Oracle 11g).*

Se recibe el archivo en *tomcat* y arroja estos resultados mostrado en la figura 4.54, por lo que se puede comprobar que está realizando la conexión y guardado a la base de datos de *hibernate.*

```
Hibernate: update attribute set attributename=?, attributevalue=? where lngAttributeId=?
Hibernate: insert into dotask_task (dotask_lngDoTaskId, lstTask_lngTaskID) values (?, ?)
1 tareas

0 Task: Meter
Task Attributes:
 / OnFailure / Continue
7 - Device: 6546546
Device Attributes:
3 / AccountName / nombre-del-servicio
4 / AccountNumber / rpu-del-servicio
5 / Description / domicilio-del-servicio
6 / InstallationDate / fecha-de-instalacion
7 / IsActive / 1
8 / MeterType / tipo-de-medidor
9 / SDPId / clave-de-division
10 / SerialNumber / 636786545
11 / SiteLocation / clave-de-zona
12 / TouSchedule / Total EnergyOnly
Success.
```

Figura 4.54.- Conexión y guardado de archivo de petición en Hibernate (Elaboración propia).

Lo anterior son resultados que arroja *hibernate* al guardar todos los datos contenidos en el archivo de petición *XML* que la *web service* recibió, el proceso se realizó sin ningún error y con la operación completada.

4.2.4 Verificación al modulo de Servicio de Windows

Antes de aplicar pruebas se obtiene los datos generales del módulo como se observa en la tabla 4.8 y Se diseñó un plan de pruebas que se puede ver en la tabla 4.9 para el módulo de servicio *Windows* y administrador de tareas que se implementara dentro del proyecto *HES*, el Servicio de ejecución de tareas: Mantiene la comunicación entre la red distribuida y las tareas que se deben ejecutar. Mantener en ejecución el servicio *Windows* y enviar tarea a red distribuida para ser ejecutada.

El administrador de tareas: Administra las tareas que deben ser ejecutadas, controlando las excepciones que se generen para continuar con las tareas sin perder la constancia y coherencia en la ejecución. Se debe Obtener tareas priorizadas, Control de excepciones para el manejo de interrupción de tareas y Generación de archivo *XML* de respuesta de la red distribuida.

Tabla 4.8.- Datos generales de evaluación del módulo Servicio Windows (Elaboración propia).

DATOS GENERALES	
No Solicitud	2
Fecha	20/05/2017
Desarrollado por	Doris Castro
Producto	Módulo de Servicio de Windows
Versión	1.0
Fecha de prueba	20/06/2017
Lugar de prueba	Área de Ingeniería de software
Nombre del tester	Manuel Flores Nava
Revisor	Agustín Sánchez Atonal

Tabla 4.9.- Resultados del servicio de Windows (Elaboración propia).

PLAN DE PRUEBAS DE SERVICIO DE WINDOWS							
TP	**TP DESCRIPCIÓN**	**TC**	**TC DESCRIPCIÓN**	**#EP**	**#PE**	**#PF**	**#PN**
TP01	Activación de tiempos para la creación de *thread pool* en el periodo establecido	TC01	funcionalidad de la ejecución de consulta a la base de datos en el tiempo especificado.				
TP02	Extracción de tareas priorizadas	TC01	validar que los registros obtenidos fueran los correctos se comparó con los registros en base de datos.				
TP03	Distribución de procesos en los hilos configurados	TC01	Conexión a la base un *thread pool* con 6 hilos de ejecución dando prioridad de datos				
		TC02	combinaciones de consulta según su priorización para comprobar que no afecta el proceso de administración de procesos.				
TP04	Generación de excepciones sin que se interrumpan los procesos	TC01	validando que los errores se registren en la excepción correspondiente en un log de errores.				

En la tabla 4.10 se observan los casos de prueba en multihilos y la Activación de tiempos para la creación de *thread pool* en el periodo establecido:

- Se utilizó el método *java.util.Timer.schedule(TimerTask task, long delay, long period)* en combinación con el método *java.util.TimerTask.TimerTask()* para obtener la funcionalidad de la ejecución de consulta a la base de datos en el tiempo especificado.

- Se imprimió en la consola el resultado de la información obtenida de la base de datos, validando que se realice en el tiempo establecido, obteniendo los tiempos esperados para configurar cada prueba.

  Extracción de tareas priorizadas y distribución de procesos en los hilos configurados.
- Validación de la consulta a la base de datos fueran las tareas correspondientes a la priorización, se desplegó en terminal el resultado para validar que los registros obtenidos fueran los correctos se comparó con los registros en base de datos.
- La configuración seleccionada fue de un *thread pool* con 6 hilos de ejecución dando prioridad a las tareas de la siguiente manera: tres hilos alta, dos hilos para media y un hilo para baja.
- Para validar la configuración se realizaron varias combinaciones de consulta según su priorización para comprobar que no afecta el proceso de administración de procesos. Se crearon registros para generar selección de registros con las siguientes combinaciones de prioridad.

Tabla 4.10.- Casos de prueba para el Thread pool (Elaboración propia).

	Tareas			**Cantidad de hilos para priorizar**		
	Alta	Media	Baja	3	2	1
Caso 1	3	2	0	2 altas	1 altas	2 media
Caso 2	5	2	1	3 altas	2 altas, 2 media	1 baja
Caso 3	0	2	3	2 medias	2 bajas	1 baja
Caso 4	4	1	2	2 altas	2 altas, 1 media	2 baja
Caso 5	2	3	4	2 altas, 2 medias	1 media, 2 bajas	2 bajas
Caso 6	5	0	0	3 altas	1 alta	1 alta
Caso 7	3	1	2	2 altas	1 alta, 1 media	2 bajas
Caso 8	5	2	1	3 altas	2 altas, 2 medias	1 baja
Caso 9	3	1	4	3 altas,	1 media, 2 bajas	1 baja
Caso 10	0	5	2	3 medias	2 medias, 1 baja	1 baja
Caso 11	1	3	1	1 alta	3 medias	1 baja
Caso 12	2	4	1	2 altas, 2 medias	2 media	1 baja
Caso 13	1	1	1	1 alta	1 media	1 baja
Caso 14	3	1	1	2 altas,	1 alta, 1 baja	1 baja
Caso 15	2	2	2	2 altas	2 medias, 1 baja	1 baja

Generación de excepciones sin que se interrumpan los procesos:

- Se generaron diferentes tipos de fallos que pudieran ocurrir al estar ejecutando la aplicación, validando que los errores se registraran en la excepción correspondiente a través de un log de errores.
- Se validó que tuviera un catch para excepciones no controladas y que se registrara el error que se produce.

Las pruebas que realizaron partiendo de los requisitos funcionales donde se especifica que se debe enviar a ejecución las tareas de acuerdo a su priorización sin generar duplicidad y cachando las excepciones que se generen para no detener la ejecución de todos los procesos.

Se integró el sistema en un ambiente simulado para validar la ejecución de las tareas encendido y apagado en un gabinete prueba con tres medidores. Las tareas se fueron ejecutando de acuerdo a su priorización y en el tiempo configurado. Los dispositivos se apagaban y encendían según lo que indicó cada petición en las tareas.

4.2.5 Verificación al módulo de Base de datos

Antes de aplicar pruebas se obtiene los datos generales del módulo como se observa en la tabla 4.11 y se diseñó un plan de pruebas que se puede ver en la tabla 4.12 enfocado a un módulo de base de datos, de la cual se Aplica un tren de pruebas donde valide que realmente está cumpliendo con los requerimientos necesarios para el *Head End System*, este módulo debe contener las tablas correspondientes a los requerimientos con sus respectivos campos. Por medio de consultas se busca verificar la existencia de dichas tablas y la correcta implementación.

Tabla 4.11.- Datos generales de evaluación del módulo de Base de Datos (Elaboración propia)

DATOS GENERALES	
No Solicitud	2
Fecha	20/06/2017
Desarrollado por	Luis Ángel Espina Hernández
Producto	Módulo de base de datos
Versión	1.0
Fecha de prueba	20/06/2017
Lugar de prueba	Área de Ingeniería de software
Nombre del tester	Manuel Flores Nava
Revisor	Agustín Sánchez Atonal

Tabla 4.12.- Plan de pruebas de la base de datos (Elaboración propia).

PLAN DE PRUEBAS DE LA BASE DE DATOS							
TP	**TP DESCRIPCIÓN**	**TC**	**TC DESCRIPCIÓN**	**#EP**	**#PE**	**#PF**	**#PN**
TP01	Conexión a la base de datos	TC01	Se debe tener acceso a la base de datos				
TP02	Consulta de nombre tablas	TC01	Nombre de tablas y campos correspondientes				
TP03	Nivel de acceso a la base de datos (*Spring security*)	TC01 TC02 TC03	Acceso de Administrador Acceso de solo consulta Acceso de superusuario				

A continuación, se muestran algunas consultas de pruebas hechas para verificar la existencia de las tablas y campos correspondientes.

TP 01 verificación de datos en *Oracle*

- Comprobación de tablas y atributos del archivo de petición (tareas) en la base de datos *Oracle 11g (Hibernate)*
- Es importante verificar que los datos del archivo *XML* haya sido guardado completamente en la base de datos de *Oracle*

TC 01 Conexión a la base de datos

Antes de todo se debe comprobar la conexión a la base de datos de *Oracle* para poder realizar las consultas.

En la figura 4.55 se muestra que efectivamente se tiene conexión con la base de datos por lo tanto está listo para realizar las consultas.

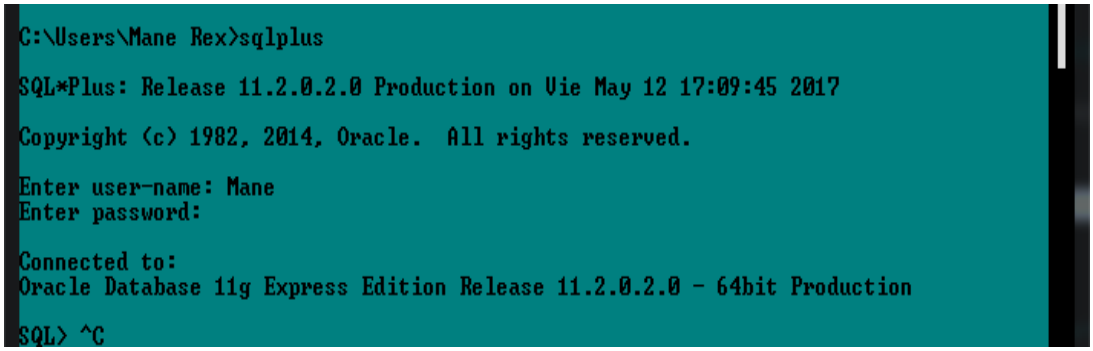


Figura 4.55.- Conexión a la base de datos de oracle (hibernate) (Elaboración propia).

TP 02. Consultas a la base de datos

En este caso de prueba se debe comprobar que los datos del archivo de petición *XML* se guardó de manera correcta en la base de datos de *Oracle* para ello se verifica en la base de datos la existencia de las tablas siguientes:

1.-*task:* tabla de tareas.

2.-*device:* tabla de dispositivos.

3.-*attribute:* tabla de atributos.

4.-*do task:* tareas por hacer.

5.-*device_attribute:* tablas relacionadas atributos de dispositivos.

6.-*task attribute:* Tablas relacionadas, atributos de tareas.

7.-*dotask_task:* Tablas relacionadas, tareas por hacer-tareas.

8.-*task_device:* Tablas relacionadas, tareas por dispositivos.

TC01 Consulta de tablas de la base de datos

Por medio de una consulta *SQL* se verifica la existencia de la tabla *task* con el siguiente comando *select * from task;* En la figura 4.56 se muestra la operación arrojando el resultado positivo.

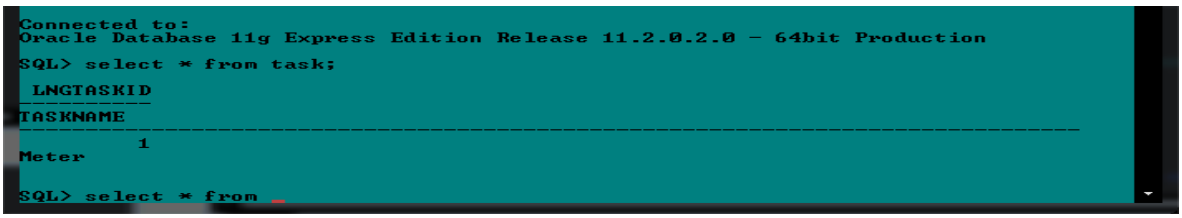


Figura 4.56.- Consulta a la tabla task. (Elaboración propia).

Por medio de una consulta *SQL* se verifica la existencia de la tabla *device* con el siguiente comando *select * from device;* En la figura 4.57 se muestra la operación arrojando el resultado positivo.

Figura 4.57.- Consulta a la tabla device (Elaboración propia).

Por medio de una consulta *SQL* se verifica la existencia de la tabla *attribute* con el siguiente comando *select * from attribute;* En la figura 4.58 y 4.59 se muestra la operación arrojando el resultado positivo.

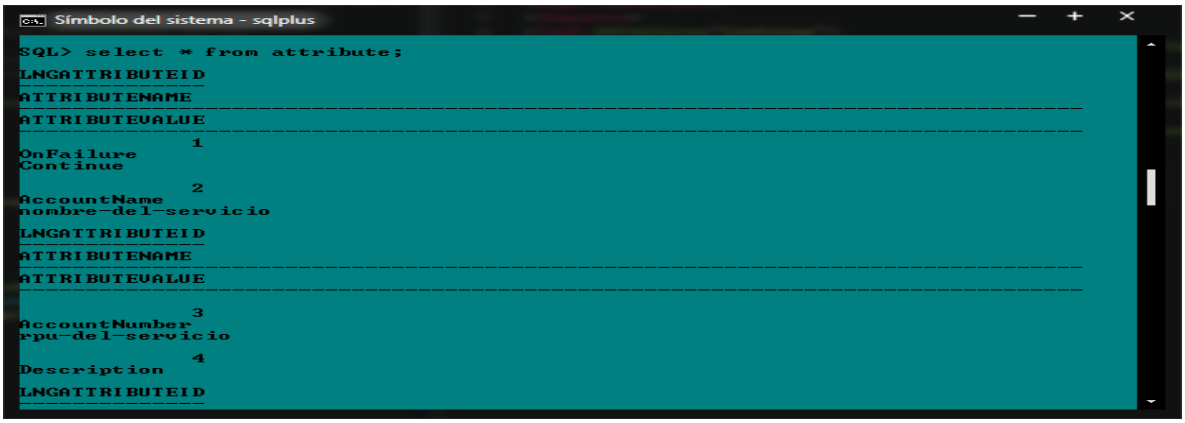


Figura 4.58.- Consulta a la tabla Attribute (Elaboración propia).

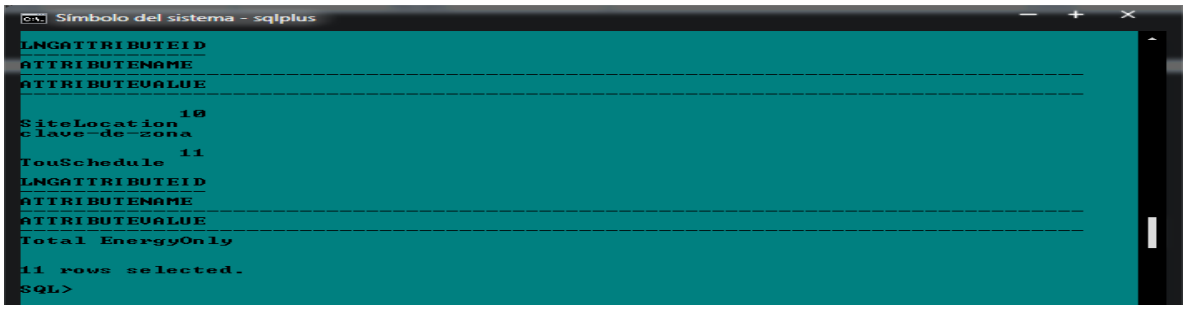


Figura 4.59.- Consulta a la tabla attribute (Elaboración propia).

Por medio de una consulta *SQL* se verifica la existencia de la tabla *dotask* con el siguiente comando *select * from dotask;* En la figura 4.60 se muestra la operación arrojando el resultado positivo.

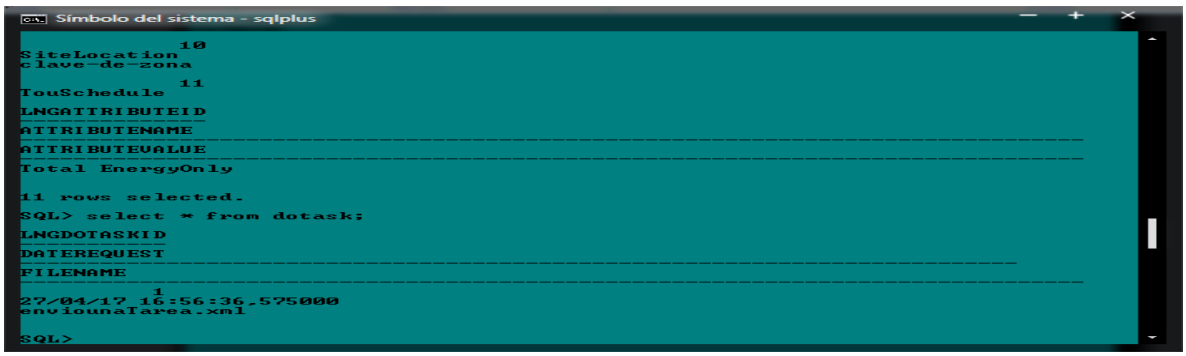


Figura 4.60.- Consulta a la tabla dotask (Elaboración propia).

Por medio de una consulta *SQL* se verifica la existencia de la tabla *device_attribute* con el siguiente comando *select * from device_attribute;* En la figura 4.61 y 4.62 se muestra la operación arrojando el resultado positivo.

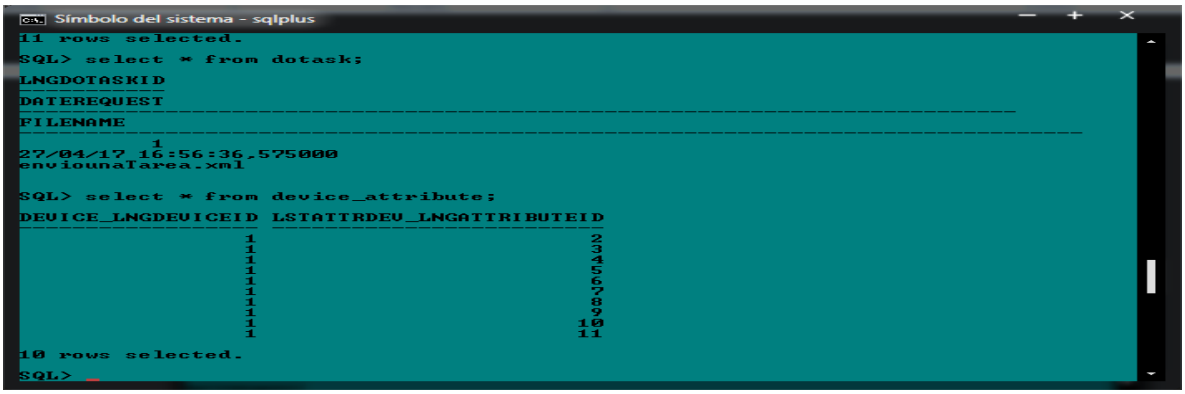


Figura 4.61.- Consulta a la tabla device_attribute (Elaboración propia).

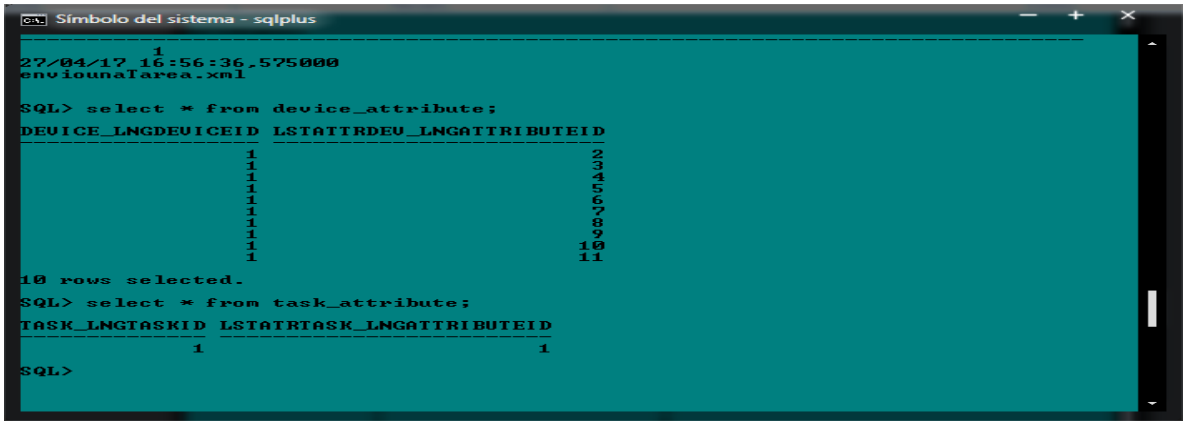


Figura 4.62.- Consulta a la tabla Device_attribute (Elaboración propia).

Consulta a la tabla *task_attribute*

Por medio de una consulta *SQL* se verifica la existencia de la tabla *task_attribute* con el siguiente comando *select * from task_attribute;* En la figura 4.63 se muestra la operación, arrojando el resultado positivo.

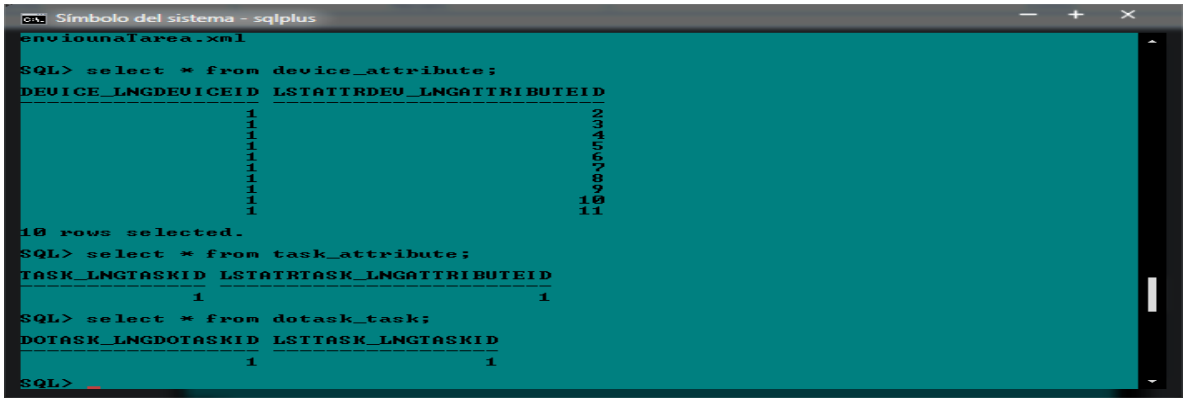


Figura 4.63.- Consulta a la tabla task_attribute (Elaboración propia).

Por medio de una consulta *SQL* se verifica la existencia de la tabla *dotask_task* con el siguiente comando *select * from dotask_task;* Y de la tabla *task_device*. *select * from task_device;*

En la figura 4.64 se muestra la operación arrojando el resultado positivo

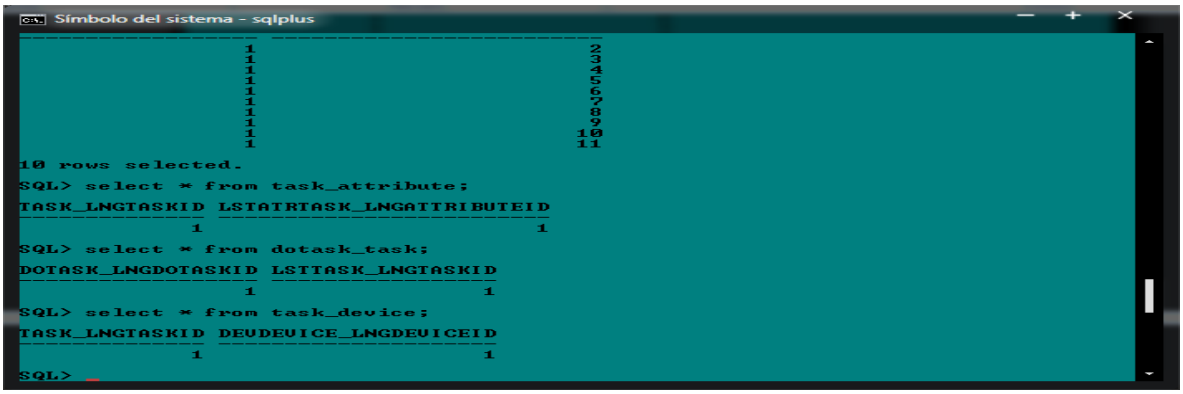


Figura 4.64.- Consulta a la tabla Task_device (Elaboración propia).

TP03 Nivel de acceso con *Spring security*

El privilegio representa el nivel de acceso que cada usuario tiene sobre una funcionalidad o sección funcional correspondiente al *Head End System*. Los roles representan el conjunto de funciones a las que tiene acceso un usuario determinado. Bajo este esquema de seguridad, cada usuario tiene acceso a solo un conjunto de secciones del sistema bajo ciertos privilegios, restringiendo de esta manera el acceso del usuario según el nivel asignado a su cuenta.

Spring ya ha desarrollado la funcionalidad para el manejo de acceso de usuario bajo este esquema, por lo que esta parte del sistema se codificará según las pautas de *Spring Security*.

Debido a la curva de aprendizaje de la tecnología y por el tiempo limitado el desarrollador ya no se logró implementar dicho modulo por lo que se tomaron otras medidas de seguridad de la misma forma, pero sin *Spring security.*

Para visualizar el nivel de acceso se puede observar en la parte de evaluación del sistema o de las interfaces.

4.2.6 Verificacion al módulo de Interfaces

Antes de aplicar pruebas se obtienen los datos generales del módulo a evaluar como se muestra en la tabla 4.13 y se diseñó un plan de pruebas que se puede observar en la tabla 4.14 enfocado a un módulo de interfaz, de la cual se Aplica un tren de pruebas donde valide que realmente está cumpliendo con los requerimientos necesarios para el *Head End System*, este módulo recibirá *frames* o paquetes que se envían desde un medidor inteligente del cual tendrá que mostrar los datos correspondientes.

Se aplicaron pruebas que verifica el cumplimiento ciertas características de calidad: por ejemplo, seguridad, eficiencia, adaptabilidad, escalable, compatibilidad, ergonómico, responsivo, navegables, accesible.

Tabla 4.13.- Datos generales de evaluación del módulo de interfaces (Elaboración propia)

DATOS GENERALES	
No Solicitud	2
Fecha	10/06/2017
Desarrollado por	Moisés Morales Guzmán
Producto	Módulo de interfaces
Versión	1.0
Fecha de prueba	10/06/2017
Lugar de prueba	Área de Ingeniería de software
Nombre del tester	Manuel Flores Nava
Revisor	Agustín Sánchez Atonal

Tabla 4.14.- Plan de Pruebas de Interfaces (Elaboración propia).

PLAN DE PRUEBAS DE INTERFACES							
TP	**TP DESCRIPCIÓN**	**TC**	**TC DESCRIPCIÓN**	**#EP**	**#PE**	**#PF**	**#PN**
TP001	Las interfaces deben realizar las funciones correctamente	TC001	Se debe tener acceso a la base de datos				
TP002	Las interfaces deben contener un buen nivel de usabilidad	TC001	Nombre de tablas y campos correspondientes				

TP03	Las interfaces deben contener el nivel de acceso a las interfaces y seguridad	TC001	Inicio de sesión: -Administrador -consulta -super usuario				
		TC002	validación de datos insertados (permitidos y no permitidos)				
TPO4	Las interfaces deben ser escalables	TC01	La interfaz debe contener posibilidad de modificar sin afectar el sistema				
TP05	Las interfaces deben ser compatibles con los navegadores básico.	TC01	Chrome				
		TC02	Firefox				
		TC03	Safari				
		TC04	Opera				
		TC05	Edge				
TP06	Las interfaces deben ser eficientes (fácil recuperación ante cualquier fallo interno o externo)	TC01	Recuperación con algún fallo intencional				
TP07	Las interfaces deben ser responsivas para laptop, Tablet, móvil y computadoras de escritorio.	TC01	Laptop				
		TC02	Desktop				
		TC03	Tablet				
		TC04	Smartphone				

TP01 Las interfaces deben realizar las funciones correctamente

En la figura 4.65 se puede observar que se han registrado algunos datos de ejemplo sobre las alarmas.

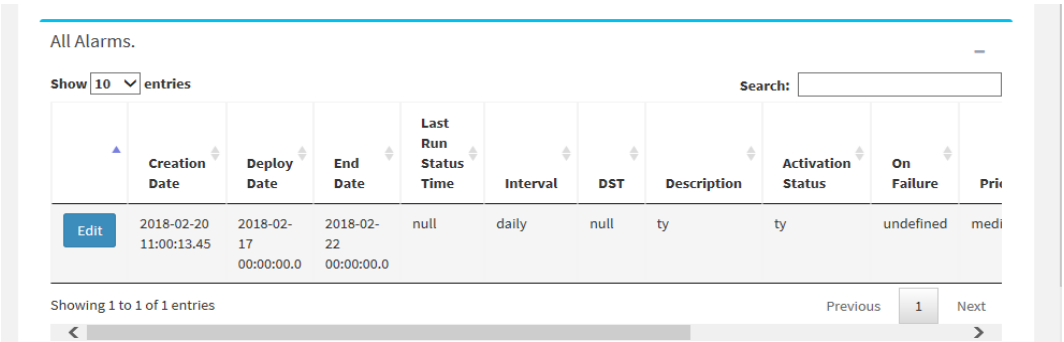


Figura 4.65.- Interfaz de alarmas (Elaboración propia).

TP02 Las interfaces deben contener un buen nivel de usabilidad

En la figura 4.66 se puede observar la información clara y con el menú accesible sin afectar la manipulación del usuario.

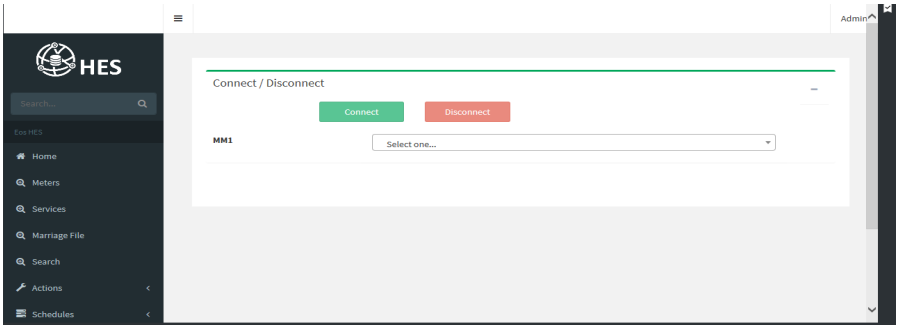


Figura 4.66.- Interfaz de conexión y desconexión de medidores (Elaboración propia).

TP03 Las interfaces deben contener el nivel de acceso a ciertos menús, pantallas e información

En la figura 4.67 se puede observar que se las interfaces contienen el acceso al sistema como administrador.

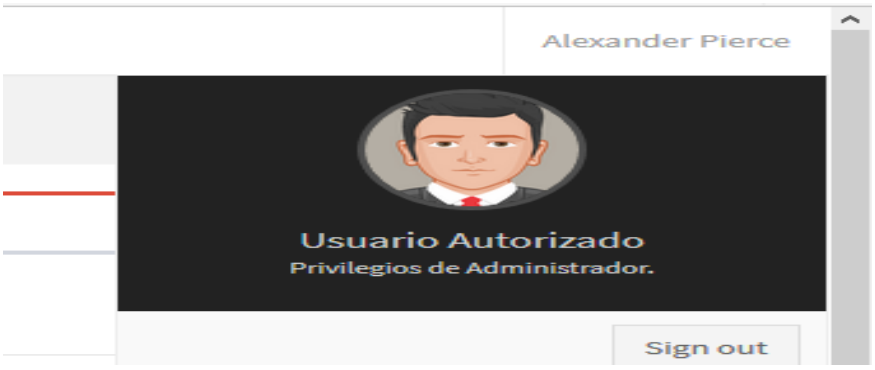


Figura 4.67.- Usuario Administrador (Elaboración propia).

TP04 Las interfaces deben ser escalables

En la figura 4.68 se puede observar que se tiene contemplado interfaces para utilizarlo en el manejo de datos de nuevos servicios como el gas y el agua

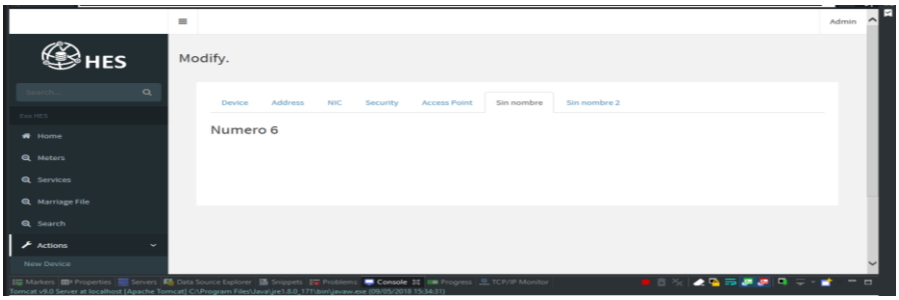


Figura 4.68.- Interfaz de escalabilidad (tiene interfaz disponible) (Elaboración propia).

TP05 Las interfaces deben ser compatibles con los navegadores básico.

En la figura 4.69 se puede observar que se ejecuta el sistema en un navegador.

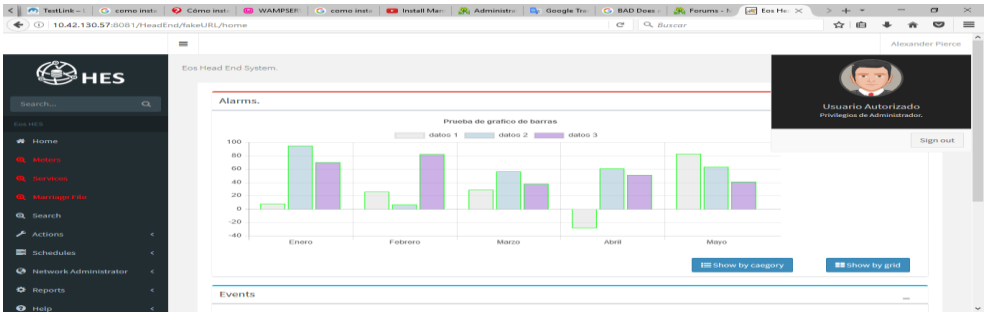


Figura 4.69.-Menú principal (elaboración propia)

TP06 Las interfaces deben ser eficientes (fácil recuperación ante cualquier fallo interno o externo)

En la figura 4.70 se puede observar un mensaje de advertencia para cancelar o aceptar una operación.

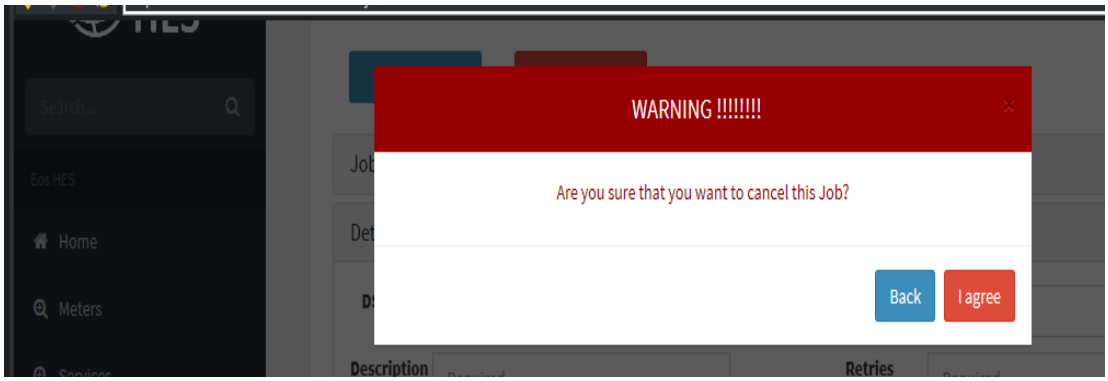


Figura 4.70.- Mensaje de alerta cuando se intenta cancelar una operación (Elaboración propia).

TP07 Las interfaces deben ser responsivas para laptop, Tablet, móvil y computadoras de escritorio.

En la figura 4.71 se muestra un menú responsivo, ya que al utilizar los campos el menú se minimiza para mostrar la información en la mayor parte del espacio.

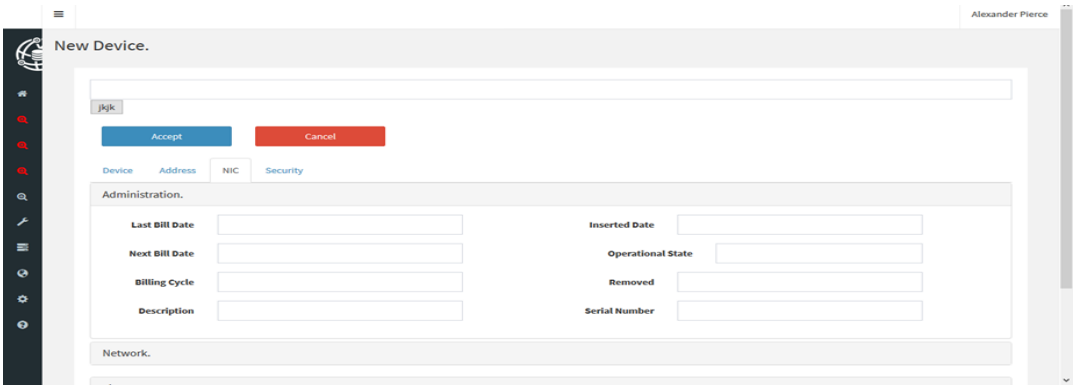


Figura 4.71.- Interfaces responsivas (elaboración propia).

4.3 Evaluación de validación del sistema

Una vez realizado el proceso de verificación se prosigue con el proceso de validación del sistema completo, para llevar acabo la evaluación se preparan los *checklist* de la metodología y se instalan y configuran las herramientas necesarias para ejecutar el *Head End System* en un servidor, debido a que los sistemas que tendrán interacción con el HES aún no se encuentran disponibles para realizar pruebas más reales.

La mejor opción para ejecutar el sistema será mediante la arquitectura cliente servidor donde realizara todas las funciones necesarias con el apoyo de laptops que se conectaran a la red del servidor simulando a un usuario que accede al sistema y la rede de medidores inteligentes es simulado por gabinete de medidores inteligentes proporcionado por la empresa *Eos tech* que por medio de un serial se conectara al *HES*.

Nota: Para conocer más sobre las actividades realizadas durante las evaluaciones al HES, se puede visualizar en el ANEXO 1 de esta tesis, documentos oficiales, planeación y bitácoras que fundamentan el trabajo aquí presentado.

4.3.1 Creación del servidor (cliente-servidor)

El servidor se monta en un ordenador con los suficientes recursos para soportar las ejecuciones del *Head End System* bajo la arquitectura cliente servidor, debido a que los sistemas que van a interactuar con el *HES* aún no se encuentran totalmente desarrollado, por lo que se tiene simular el funcionamiento del *HES.*

Se instala el servidor *tomcat* además de *Oracle 11g* para la ejecución de la base de datos, cuando *tomcat* manda el mensaje en un navegador de servicio disponible como se observa en la figura 4.72 significa que está listo para montar cualquier sistema.

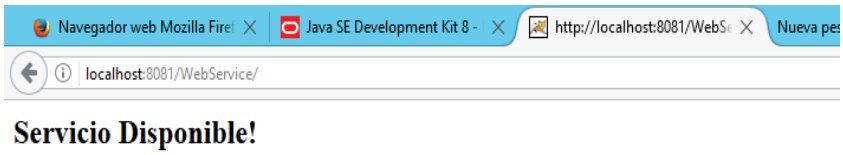


Figura 4.72.- Servicio disponible para consumir (Elaboración propia).

A continuación, se ejecuta el *Head End System* dentro del servidor configurado, el servidor se encuentra en un *Workstation* que soportara la ejecución de mayor recurso. El servidor se observa en la figura 4.73 donde ya se encuentra disponible.

Figura 4.73.- Sistema montado en el servidor (Elaboración propia)

Cuando el *Head End System* se encuentra en el servidor en ejecución se puede tener acceso de forma remota desde cualquier dispositivo con el apoyo de una IP.

4.3.2 Evaluacion del sistema

El sistema de seguridad del sistema está protegido mediante sesiones con usuario y *password*, cuando las credenciales de usuario no son válidas el sistema redirecciona a un mensaje de advertencia de un usuario no autorizado para ingresar al sistema, como se puede ver en la figura 4.74, esto es debido a que no se encuentra registrado en el sistema.

Figura 4.74.- Advertencia de usuario no autorizado (Elaboración propia).

En la manipulacion de *login* se pudo constatar que si cumple con la validacion del usuario y *password* las ejecuciones del *login* se observan en la figura 4.75.

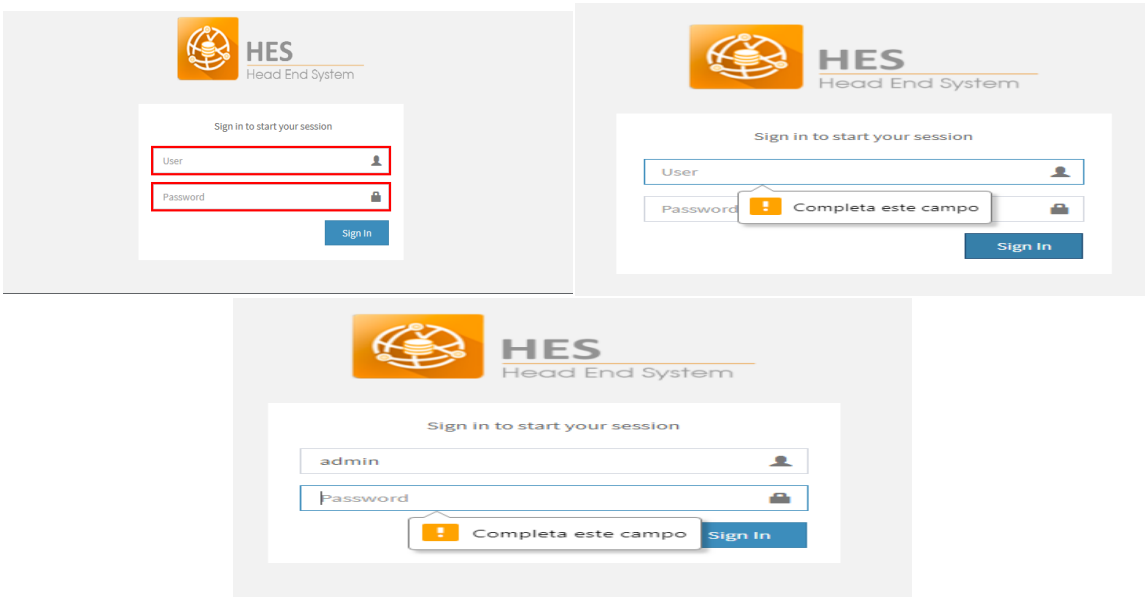


Figura 4.75.- Validacion de login (Elaboración propia).

En la figura 4.76 se muestra la pantalla principal del sistema donde se puede observar un menu lateral de facil acceso, con las graficas de las alarmas de los medidores inteligentes, ademas se tomo en cuenta los colores propuestos por el cliente y un logotipo tambien seleccionado por el cliente, por lo que satisface los requerimeintos del cliente.

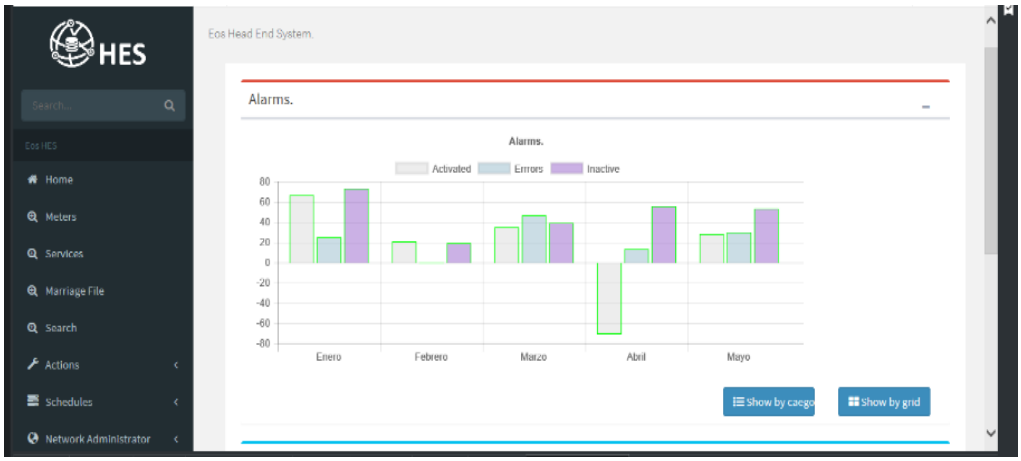


Figura 4.76.- Menú principal (Elaboración propia).

En la figura 4.77 se observa el apartado de alarmas con los campos planteados en los requerimientos

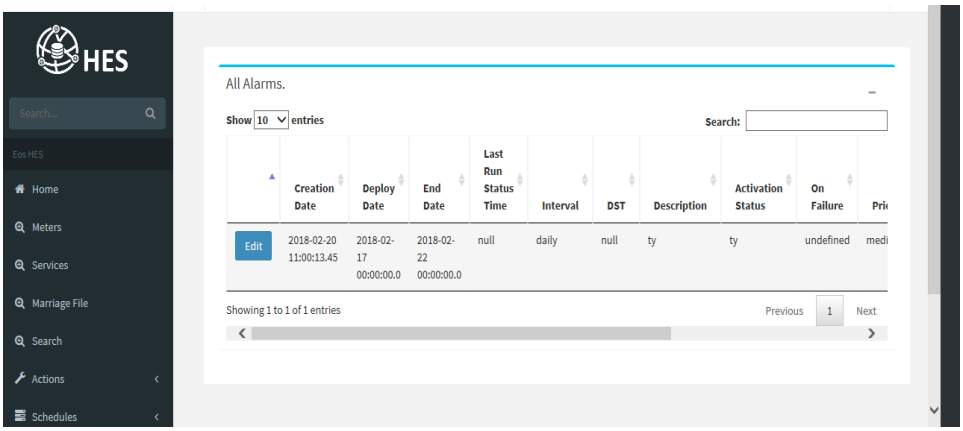


Figura 4.77.- Interfaz de alarmas (Elaboración propia).

Se solicito un apartado donde se pueda importar los archivos *XML* para migrar datos generados por los medidores y vaciarlo a la base de datos esa operación se observa en la figura 4.78 y 4.79.

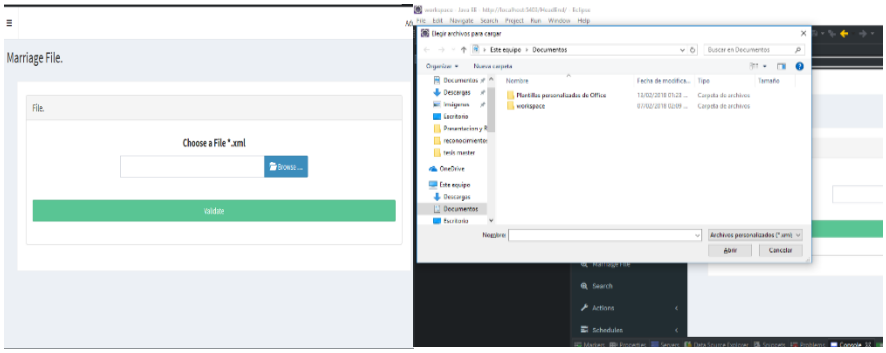


Figura 4.78.- Interfaz del archivo XML a seleccionar en una ruta local (Elaboración propia).

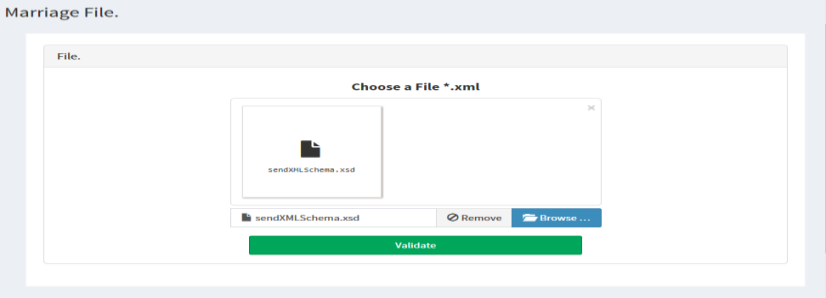


Figura 4.79.- Interfaz del archivo XML seleccionado es agregado al sistema (Elaboración propia).

Se solicito tener un apartado para agregar, modificar y eliminar información sobre la *NIC (Network Interface Comunication)*, la interface se muestra en la figura 4.80.

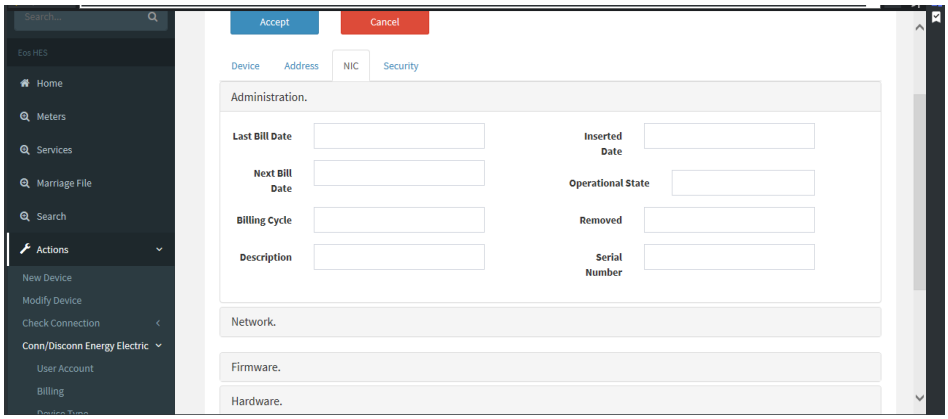


Figura 4.80.- Interfaz para configurar NIC´S (Elaboración propia).

El cliente solicito tener un apartado para agregar, modificar y eliminar información sobre los *Jobs* la interface se muestra en la figura 4.81.

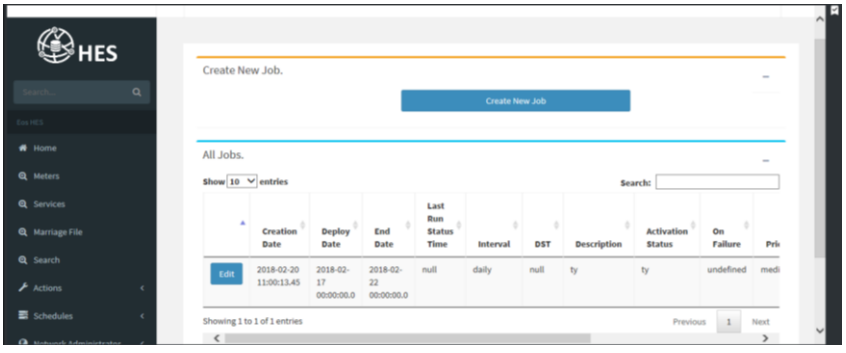


Figura 4.81.- Interfaz para crear nuevos Jobs (Elaboración propia).

Utilización de herramientas útiles como el calendario cuando se necesite asignar una fecha facilita el ingreso de los datos correctos como se muestra en la figura 4.82.

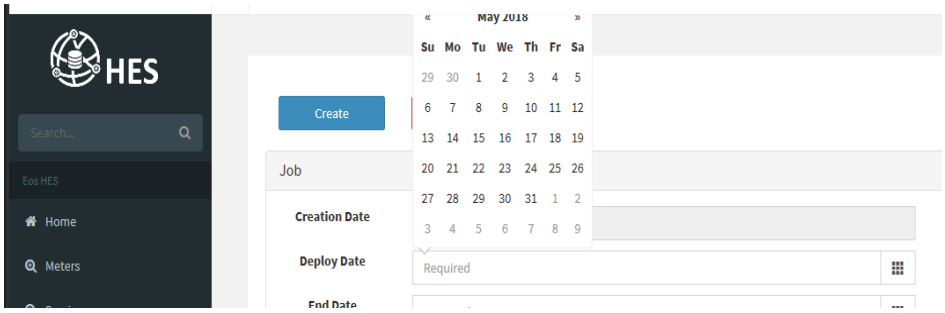


Figura 4.82.- Uso de calendario eficiente (Elaboración propia).

En la figura 4.83 se observa una advertencia cuando en el llenado de información del formulario no se ha completado hasta que sea ingresado en el sistema.

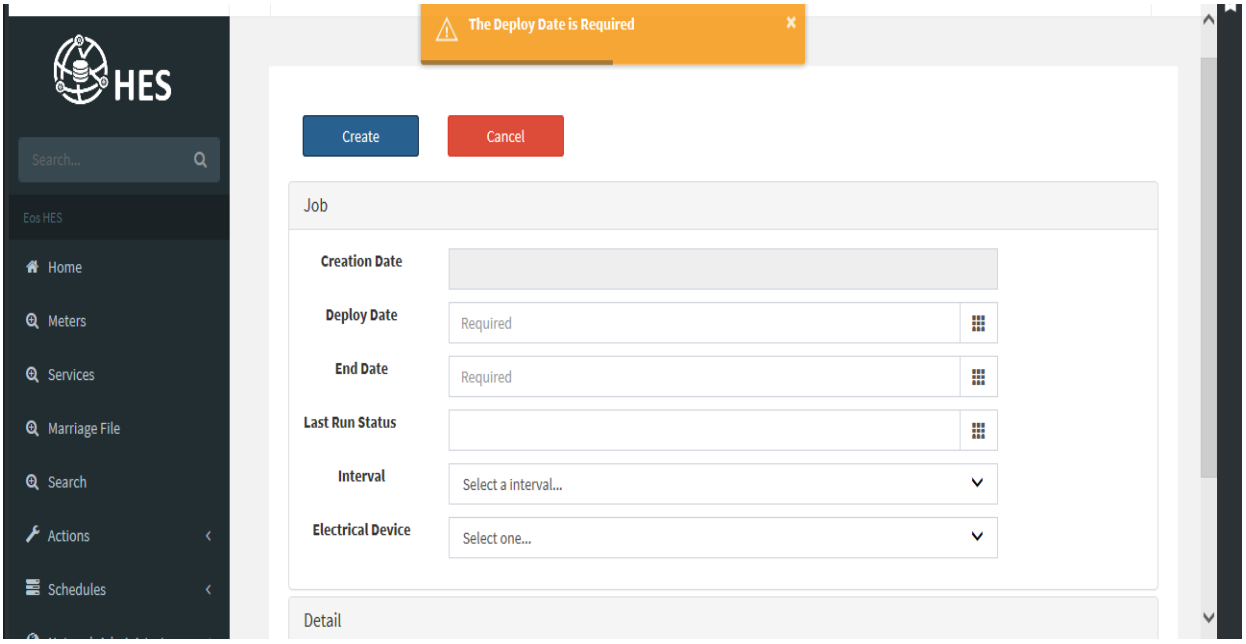


Figura 4.83.- Mensaje de alerta cuando un dato es requerido (Elaboración propia).

Cuando se realiza una asociación de medidores incorrecta se lanza una advertencia para volver asociar con un medidor valido. La advertencia se puede observar en la figura 4.84.

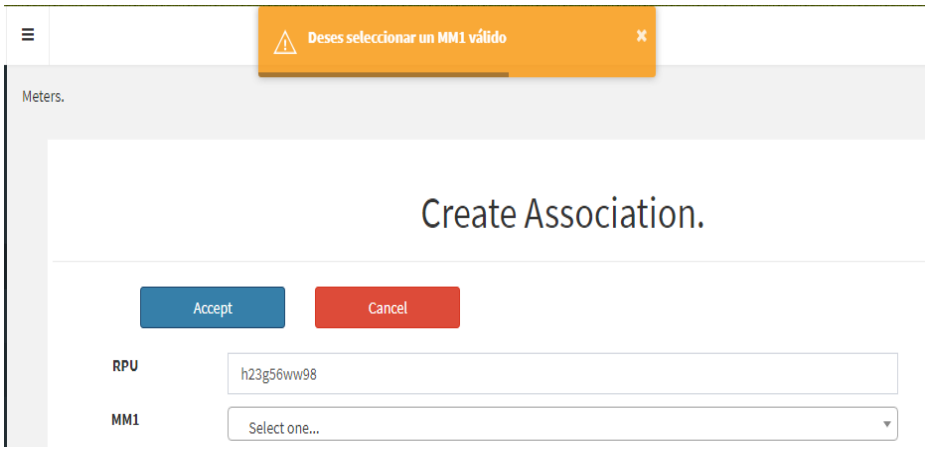


Figura 4.84.- Mensaje de alerta cuando un medidor es invalido (Elaboración propia).

En la interfaz de registro de direcciones (*Address*) de la figura 4.85, se muestran algunos ejemplos de registros ingresados para validar cada uno de sus campos y el buen funcionamiento de los botones para modificar si es necesario algún registro.

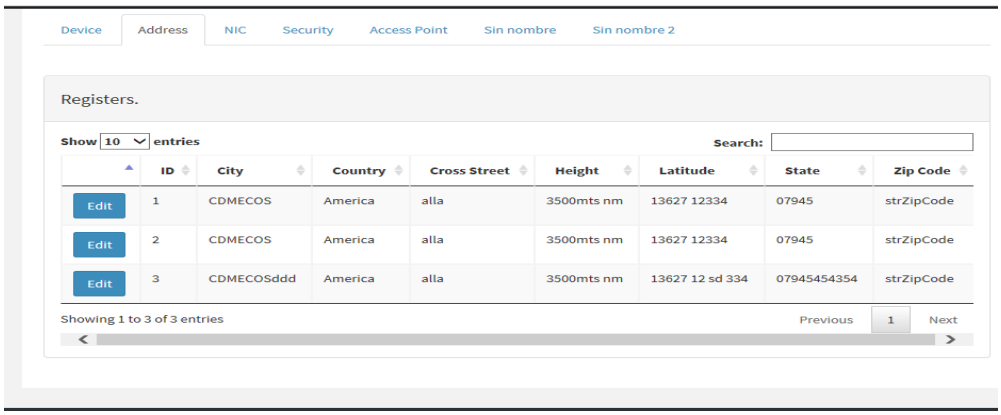


Figura 4.85.- Interfaz de registro y de modificación de NIC (Elaboración propia).

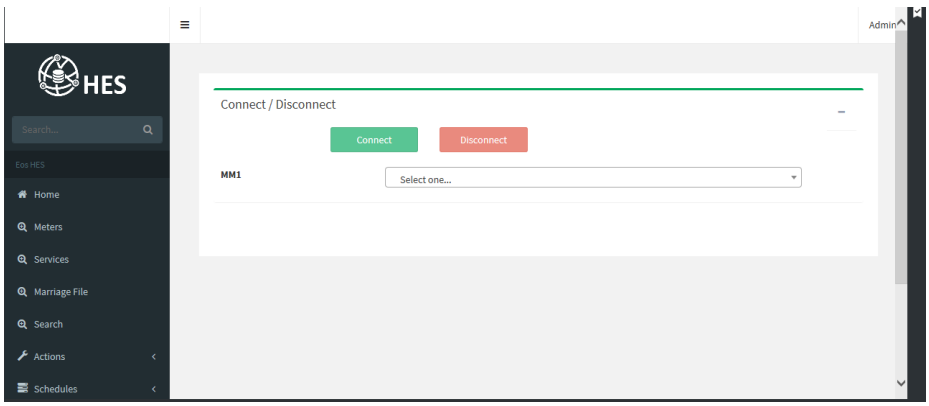


Figura 4.86.- Interfaz de conexión y desconexión de medidores (Elaboración propia)

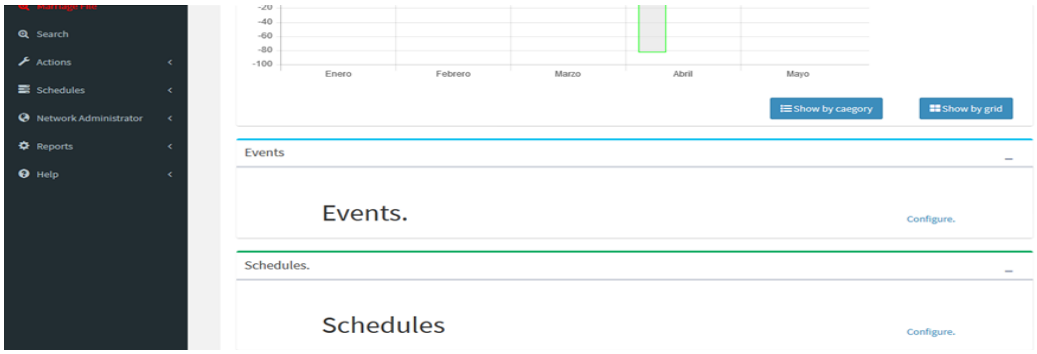


Figura 4.87.- Interfaz sobre eventos y horarios (elaboración propia)

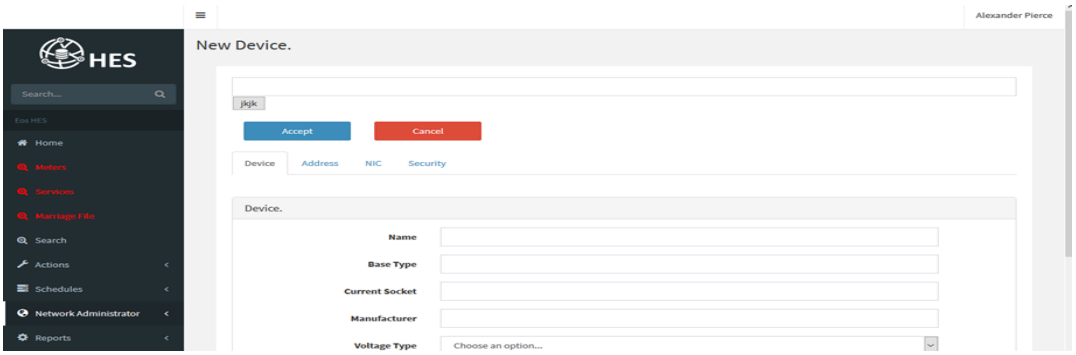


Figura 4.88.- Interfaz para agregar nuevos dispositivos (elaboración propia)

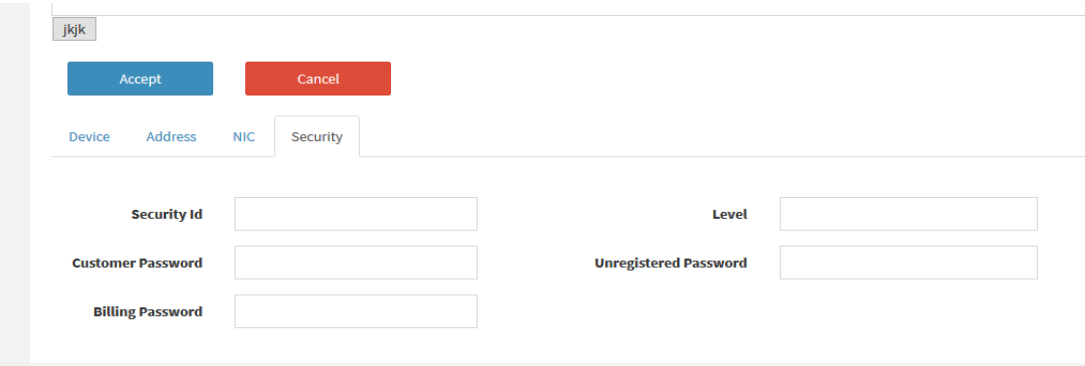


Figura 4.89.-Interfaz para agregar el acceso de un nuevo cliente (Elaboración propia)

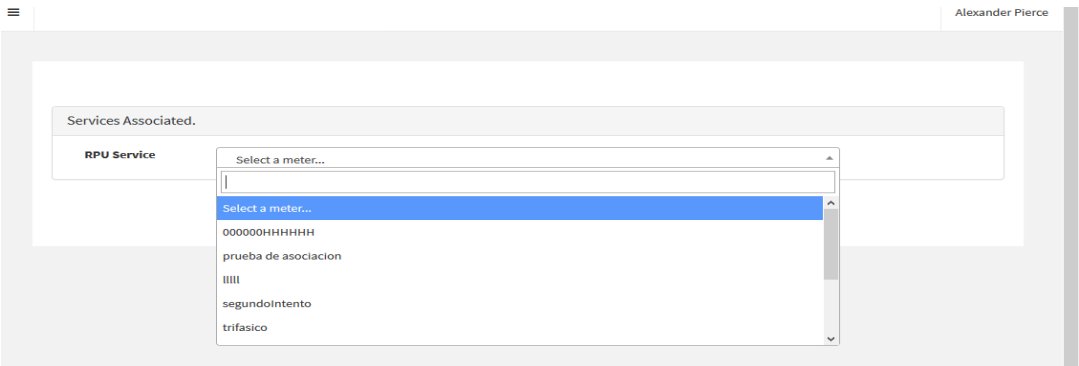


Figura 4.90.- Interfaz del número de serie del cliente (RPU) (Elaboración propia)

4.4 Resultados generales del *Head End System*

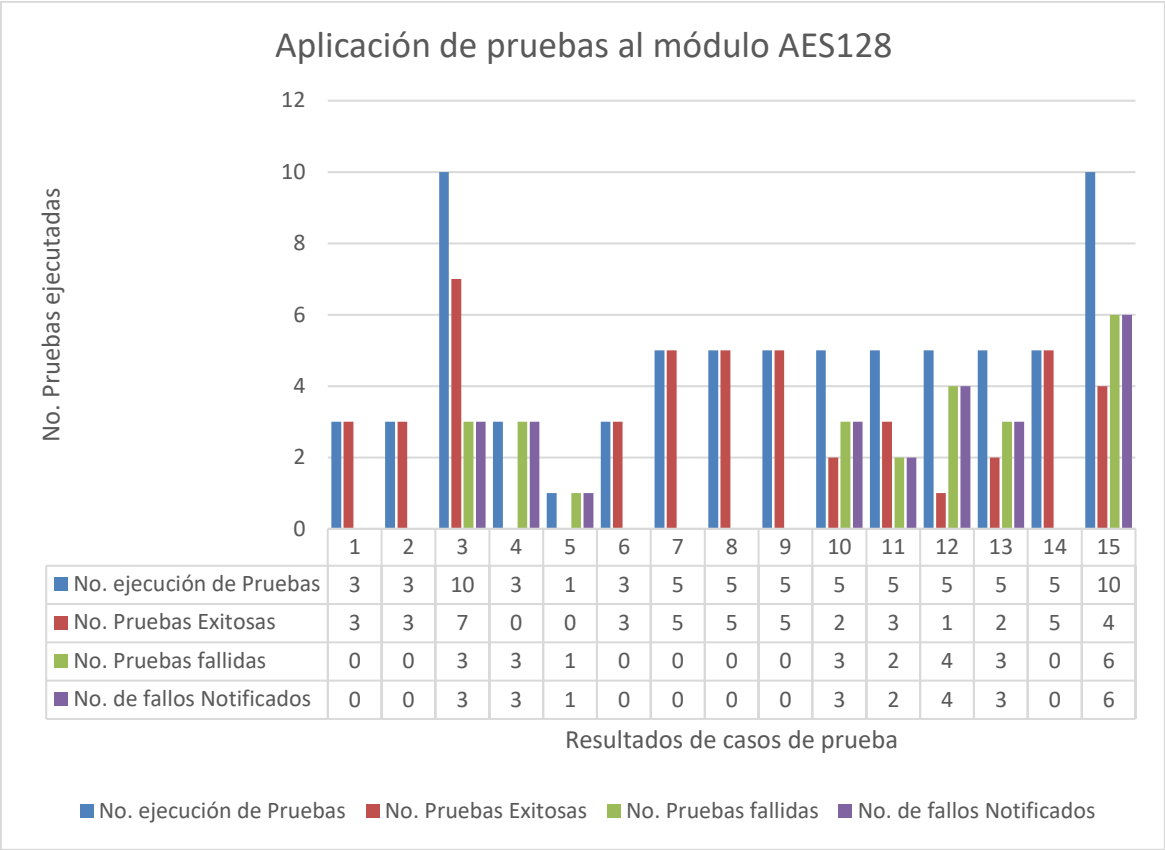
En esta fase se exhiben los resultados de las pruebas ejecutadas en los módulos y los resultados de calidad del HES, los datos se presentan en tablas y graficas para el fácil acceso a la información.

4.4.1 Resultados del módulo de *Advanced Encryption Standard AES 128*

En el módulo *Advanced Encryption Standard AES 128* se aplicaron pruebas de caja negra, pruebas de seguridad, pruebas de carga y de rendimiento, para obtener resultados que se observan en la figura 4.91 y tabla 4.15, de la eficiente encriptación y desencriptación a pesar de la cantidad de la información obtenida por los medidores.

Tabla 4.15.- Resultados al módulo de encriptación AES 12 (Elaboración propia).

RESULTADOS AES 128				
Requerimientos Tecnicos	**Casos de prueba**	**TIPO DE PRUEBAS**	**RESULTADOS**	**Observaciones**
encriptación y des encriptación de una cadena de datos	Validar cadena de letras Mayúsculas y minúsculas	Caja negra	si	
	Validar una cadena de números.	Caja negra	si	
	Validar una cadena de signos.	Caja negra	si	
	Validar cadena de campo vacío.	Caja negra	no	Se notifica a desarrollador
	Validar cadena sin insertar dato	Caja negra	no	Se notifica a desarrollador
	Validar cadena con valor cero	Caja negra	si	
	Reconocimiento de doble comillas \”	Caja negra	no	
	Reconocimiento de la diagonal inversa \\	Caja negra	si	
	Reconocimiento de salto de línea \n	Caja negra	si	
	Reconocimiento de retorno de carro \r	Caja negra	no	Se notifica a desarrollador
	Reconocimiento de tabulador \t	Caja negra	no	Se notifica a desarrollador
	Reconocimiento de BackSpace \b	caja negra	no	Se notifica a desarrollador
	Reconocimiento de Form Feed \f	Caja negra	no	Se notifica a desarrollador
	Reconocimiento de comilla simple \’	Caja negra	si	
	Reconocimiento de un combinado de secuencias de escape (\”. \’, \n, \t, \b, \\, \r).	Caja negra	no	Se notifica a desarrollador



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
No. ejecución de Pruebas	3	3	10	3	1	3	5	5	5	5	5	5	5	5	10
No. Pruebas Exitosas	3	3	7	0	0	3	5	5	5	2	3	1	2	5	4
No. Pruebas fallidas	0	0	3	3	1	0	0	0	0	3	2	4	3	0	6
No. de fallos Notificados	0	0	3	3	1	0	0	0	0	3	2	4	3	0	6

Figura 4.91.- Resultado de pruebas a módulo AES128 (Elaboración propia).

4.4.2 Resultados del módulo del *CRC*

Este módulo recibe *frames* o paquetes que se envían desde un medidor inteligente en forma de Hexadecimal, los *frames* están estructurados con base al estándar de comunicación.

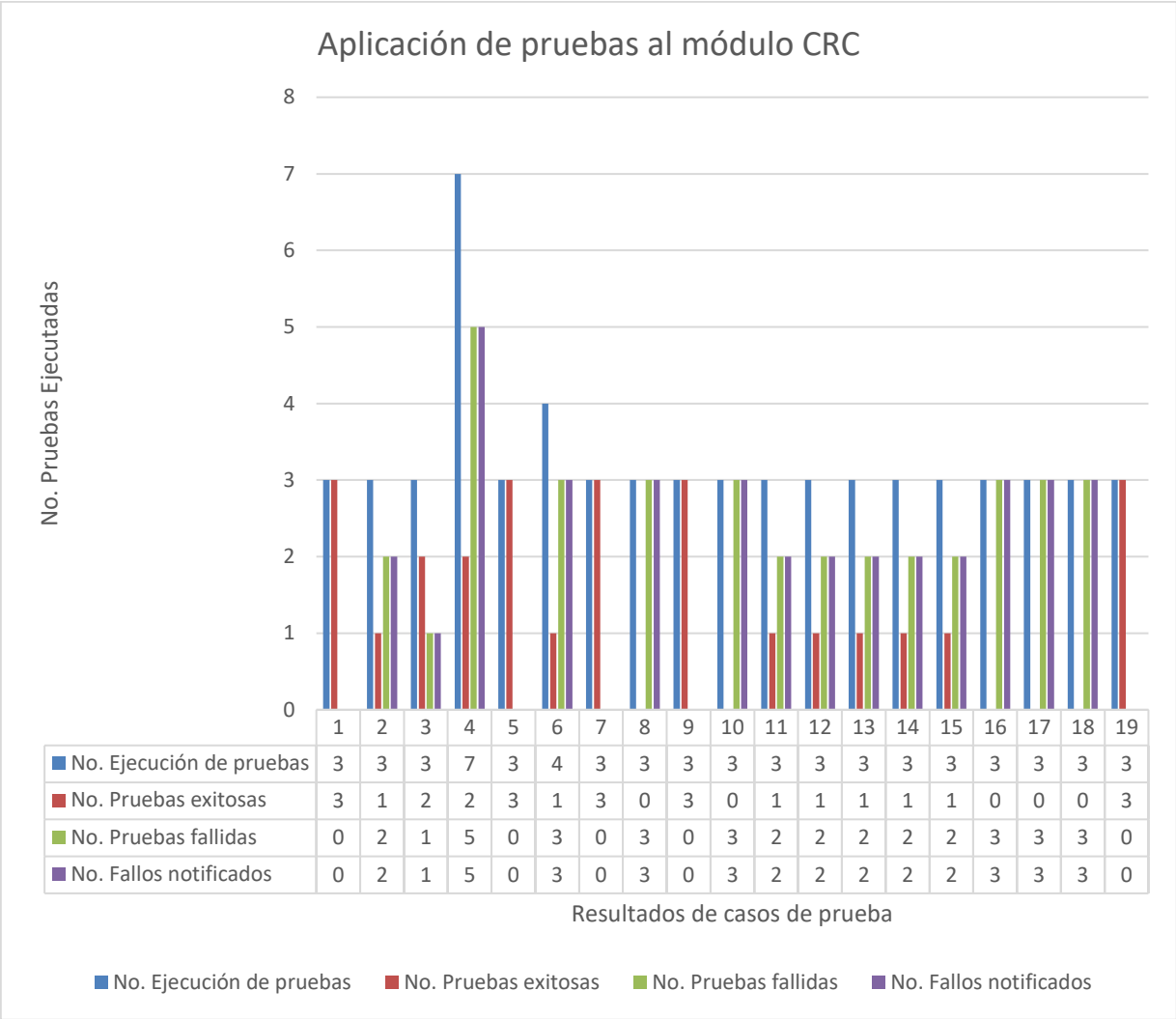
El *frame* se encarga de transportar la información en bytes, por ello que para asegurar que no se han corrompido esos datos y validar la estructura del *frame* se calcula el *CRC*, este es un código de detección de errores usado frecuentemente en redes digitales y en dispositivos de almacenamiento para detectar cambios accidentales en los datos.

Los bloques de datos ingresados en estos sistemas contienen un valor de verificación adjunto, basado en el residuo de una división de polinomios; el cálculo es repetido, y la acción de corrección puede tomarse en contra de los datos presuntamente corruptos en caso de que el valor de verificación no concuerde.

En la figura 4.92 y tabla 4.16, se observa los resultados del módulo *CRC*, el módulo permite validar la estructura de un *frame* enviado por los medidores inteligentes, con la intención de asegurar que la información no ha sido corrompida. En él se aplicaron pruebas caja negra, pruebas de rendimiento y carga, en el manejo de excepciones se utilizó las pruebas de caja blanca.

Tabla 4.16.- Resultados del módulo CRC (Elaboración propia).

RESULTADOS CRC				
Requerimientos Tecnicos	**Casos de prueba**	**TIPO DE PRUEBAS**	**RESULTADOS**	**Observaciones**
Seguridad en los *frames* por medio del Cálculo de CRC	Verificación del cálculo del CRC	Caja negra Rendimiento carga	SI	
	Frame invalido	Caja negra	SI	
	Frame corrompido	Caja negra	SI	
	Manejo de excepciones	Caja Blanca/caja negra	No	Se notifica a desarrollador
	Frames sin espacios	Caja negra	SI	
	Frames con espacios al azar	Caja negra	No	Se notifica a desarrollador
	Carácter de comienzo (EE) en mayúscula, minúscula y combinado	Caja negra	SI	
Tipos de frame's según cada dispositivo	*Frame* para el dispositivo CCG (20)	Caja negra	No	Se notifica a desarrollador
	Frame para el dispositivo medidor (00)	Caja negra	SI	
	para el dispositivo IR´S (80 a 85)	Caja negra	No	Se notifica a desarrollador
servicios de dispositivos en *frames*	servicio identification	Caja negra	SI	
	Servicio Log On	Caja negra	SI	
	Servicio security	Caja negra	SI	
	Servicio Full write	Caja negra	SI	
	Servicio Pwrite Offset	Caja negra	SI	
	Servicio Full Read	Caja negra	No	Se notifica a desarrollador
	Servicio Pread Offset	Caja negra	No	Se notifica a desarrollador
	Servicio Log Off	Caja negra	No	Se notifica a desarrollador
	Servicio Terminate	Caja negra	Si	



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
No. Ejecución de pruebas	3	3	3	7	3	4	3	3	3	3	3	3	3	3	3	3	3	3	3
No. Pruebas exitosas	3	1	2	2	3	1	3	0	3	0	1	1	1	1	1	0	0	0	3
No. Pruebas fallidas	0	2	1	5	0	3	0	3	0	3	2	2	2	2	2	3	3	3	0
No. Fallos notificados	0	2	1	5	0	3	0	3	0	3	2	2	2	2	2	3	3	3	0

Figura 4.92.- Resultados de pruebas a modulo CRC (Elaboración propia).

4.4.3 Resultados del módulo de la *Web Service*

Los resultados de la *web service* observa en la figura 4.93 y tabla 4.17, el módulo es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones, el módulo será montado en un servidor para recibir múltiples peticiones de diferentes sistemas por lo que se tuvieron que generar scripts que simulen las peticiones y scripts que responden a las peticiones (pruebas automáticas), debido a que la *web service* contiene un esquema *XML* que las peticiones deben respetar, se tuvieron que aplicar pruebas de caja negra y caja blanca.

Tabla 4.17.- Resultados de la web service (Elaboración propia).

RESULTADOS DE WEB SERVICE				
Requerimientos Tecnicos	**Casos de prueba**	**TIPO DE PRUEBAS**	**RESULTADOS**	**Observaciones**
ejecución de la *web service* para la Recepción de peticiones de tareas	Ejecución de *web service*	Automatizado en entorno cliente-servidor Rendimiento Carga	SI	
	Recibir petición en archivo XML	Automatizado (*scripts* de ejecución) Rendimiento carga	SI	
	Guardar el archivo de petición *XML*	Automatizado (*scripts* de ejecución)	SI	
	Validación de archivos *XML*	Automatizado (*scripts* de ejecución)	SI	
	Guarda a la base de datos de *Hibernate (Oracle 11g)*	Automatizado	No	Se notifica a desarrollador
verificación de datos en *oracle*	Conexión a la base de datos	*Consultas*	Si	
	Consultas a la base de datos para corroborar la existencia de las tablas guardadas	Caja negra	No	Se notifica a desarrollador

Nota: si se desea conocer más a fondo sobre el desarrollo de los módulos de web service y de servicio de Windows puedes consultar los artículos siguientes:

Castro, D., Hernández, J. J. Medina, M. G., Sánchez, A. (2017) Interoperabilidad en sistemas informáticos de ciudades inteligentes a través de servicios web. Universidad Veracruzana Región Poza Rica-Tuxpan, Congreso Internacional de investigación academia journals Tuxpan. ISSN 1946-5351, ISBN 978-1-939982-30-8

Castro, D., Hernández, J. J. Medina, M. G., Sánchez, A. (2017) Desarrollo del Módulo de administración de tareas para un sistema Head End. Tecnológico Nacional de México en Celaya, Congreso Internacional de investigación academia journals Celaya. ISSN 1946-5351 Vol.9, No. ISBN 978-1-939982-32-2

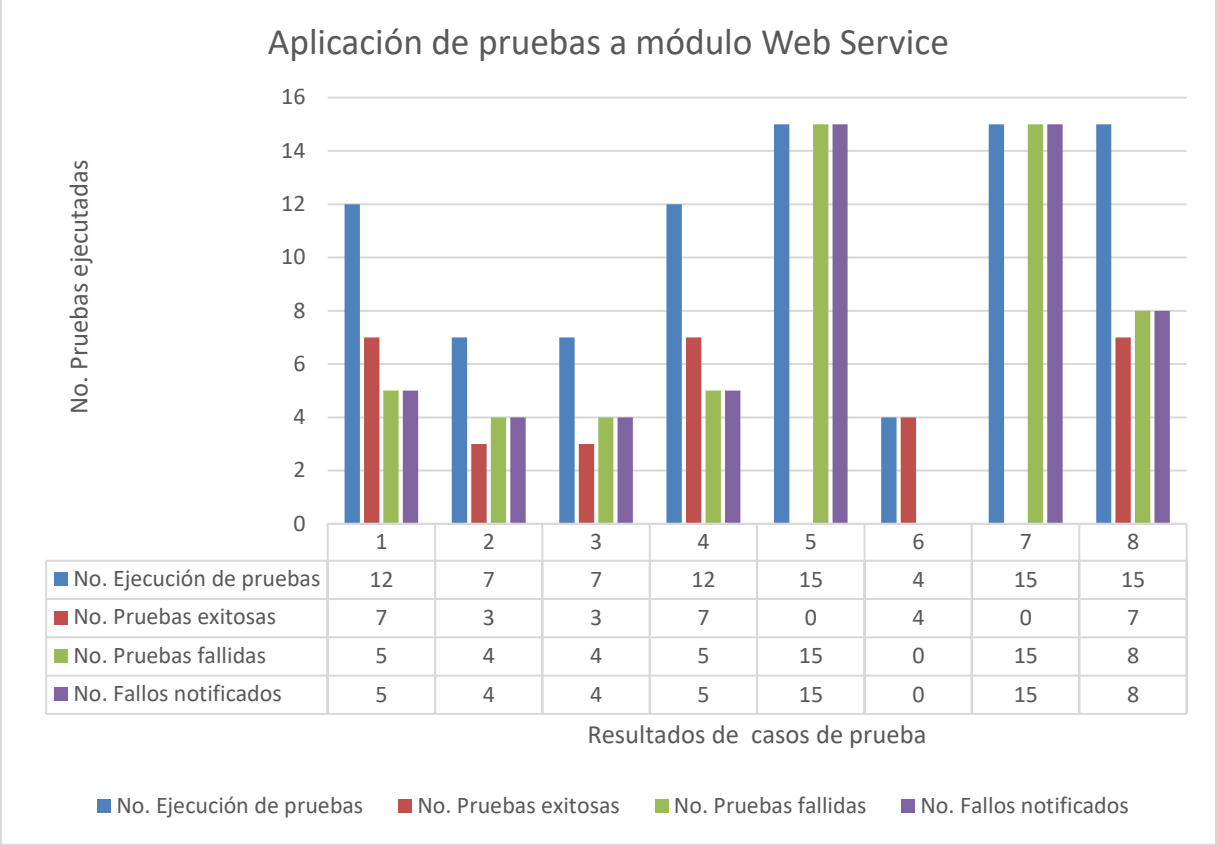


Figura 4.93.- Resultados de pruebas a web service (Elaboración propia).

4.4.4 Resultados del módulo de servicio de Windows.

El Servicio de ejecución de tareas: Mantiene la comunicación entre la red distribuida y las tareas que se deben ejecutar. Mantener en ejecución servicio *Windows* y Enviar tarea a red distribuida para ser ejecutada.

El administrador de tareas: Administra las tareas que deben ser ejecutadas, controlando las excepciones que se generen para continuar con las tareas sin perder la constancia y coherencia en la ejecución. Se debe Obtener tareas priorizadas, Control de excepciones para el manejo de interrupción de tareas y Generación de archivo *XML* de respuesta de la red distribuida.

Los resultados del módulo de servicio de *Windows* se observan en la figura 4.94 y la tabla 4.18, en él se realizaron pruebas automatizados y pruebas de caja blanca, para conocer el comportamiento de múltiples hilos (*thread pool*), para la gestión de tareas priorizadas en el sistema y para controlar las excepciones generadas.

Tabla 4.18.- Resultados del módulo servicio Windows (Elaboración propia).

RESULTADOS DE SERVICIO DE WINDOWS				
Requerimientos Tecnicos	**Casos de prueba**	**TIPO DE PRUEBAS**	**R**	**Observaciones**
Extracción de tareas priorizadas y Distribución de procesos en los hilos configurados	La creación de *thread pool* en el periodo establecido	Automatizado en entorno cliente-servidor Rendimiento Carga	SI	
	Validar los registros obtenidos con los registros en base de datos.	Caja negra	No	Se notifica a desarrollador
	Conexión a la base un *thread pool* con 6 hilos de ejecución dando prioridad de datos	Automatizado Rendimiento carga	SI	
	Combinaciones de consulta según su priorización y comprobar que no afecta el proceso de administración de procesos.	Automatizado Rendimiento carga	No	Se notifica a desarrollador
Generación de excepciones sin que se interrumpan los procesos	Validando que los errores se registren en la excepción correspondiente en un log de errores.	Caja blanca/caja negra	SI	

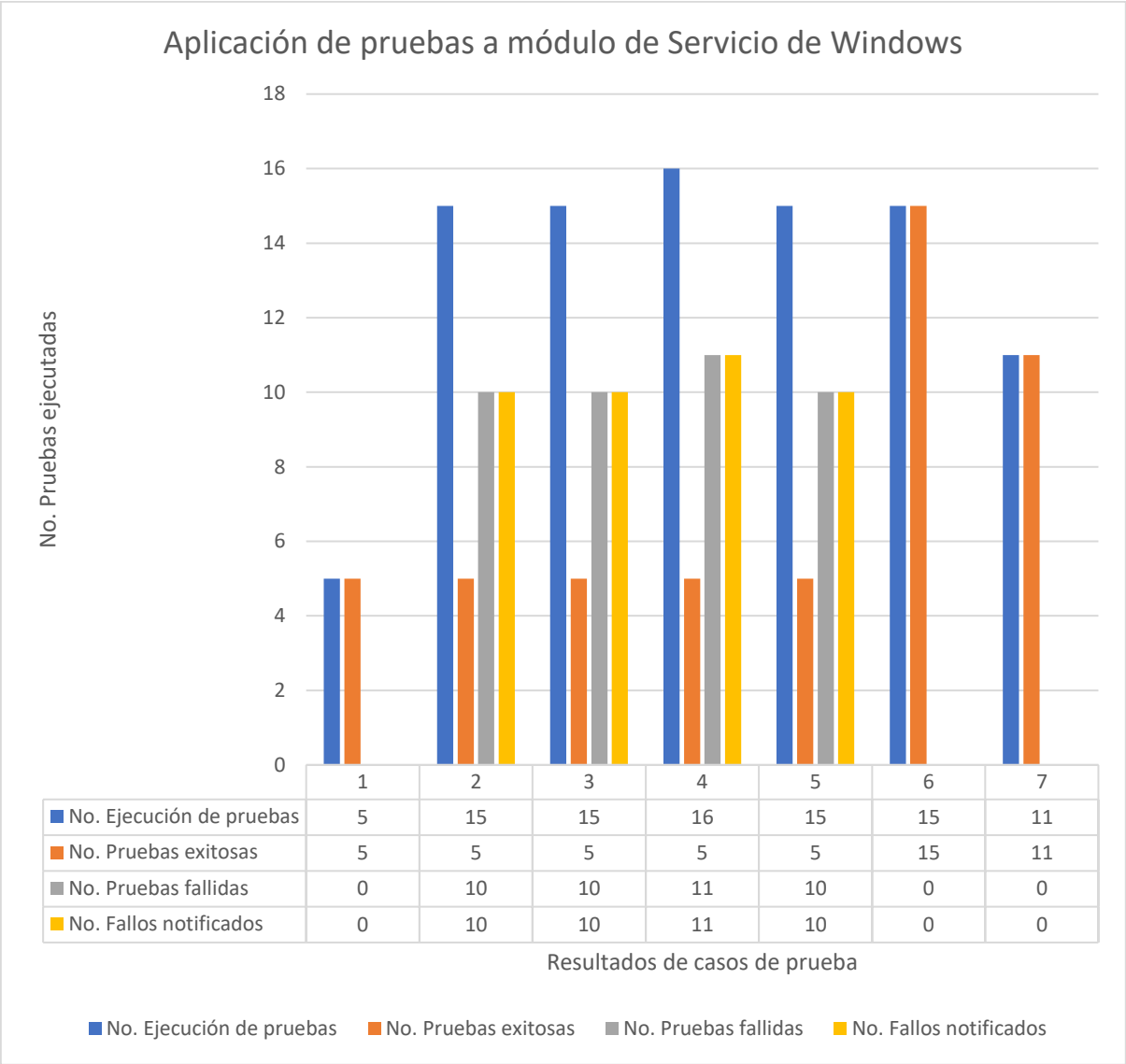


Figura 4.94.-. Resultado de pruebas a servicio Windows (Elaboración propia).

4.4.5 Resultados del módulo de Base de datos

En la figura 4.95 y tabla 4.19, se observan los resultados de la base de datos, en este módulo se guardarán los datos generados por los medidores inteligentes, por lo que debe contener una estructura de tablas con sus respectivos campos que debe coincidir con la estructura que contiene los medidores, por lo que se aplicaron pruebas de caja negra en base a consultas de tablas y campos, además de pruebas de seguridad e inyección de código.

Tabla 4.19.- Resultados del módulo de base de datos (Elaboración propia).

RESULTADOS DE LA BASE DE DATOS				
Requerimientos Tecnicos	**Casos de prueba**	**TIPO DE PRUEBAS**	**RESULTADOS**	**Observaciones**
Conexión a la base de datos	Acceso a la base de datos	CAJA NEGRA (Entorno cliente-servidor)	SI	
Consulta de nombre tablas	Nombre de tablas y campos correspondientes	Consultas Análisis estático	No	Se notifica a desarrollador
Nivel de acceso a la base de datos	Acceso como supe usuario	Caja negra Inyección de código	SI	
	Acceso como administrador	Caja negra Inyección de código	SI	
	Acceso como consulta	Caja negra Inyección de código	No	Se notifica a desarrollador
	Implementación de *Spring security*	Caja negra Inyección de código	No	Se notifica a desarrollador

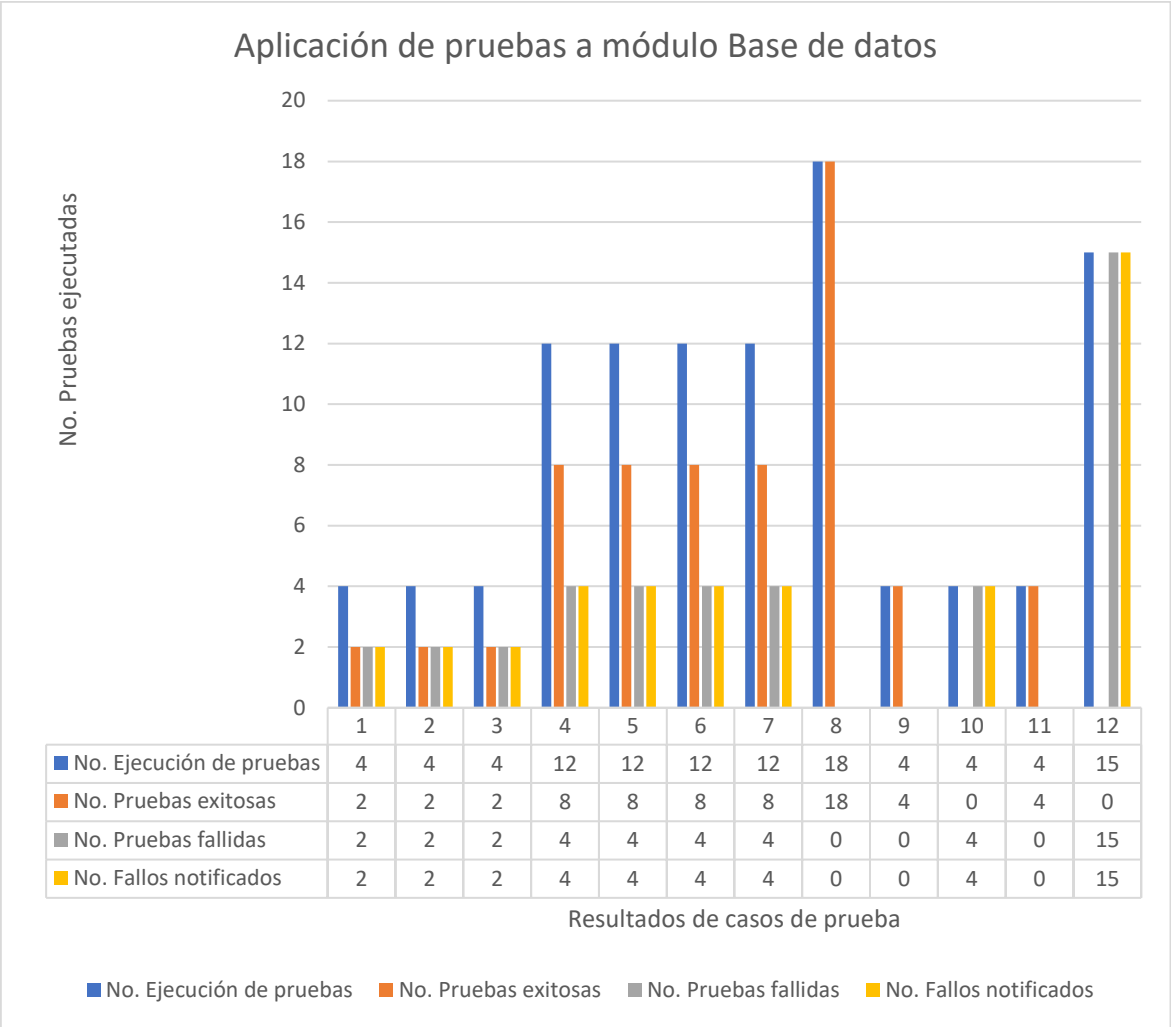


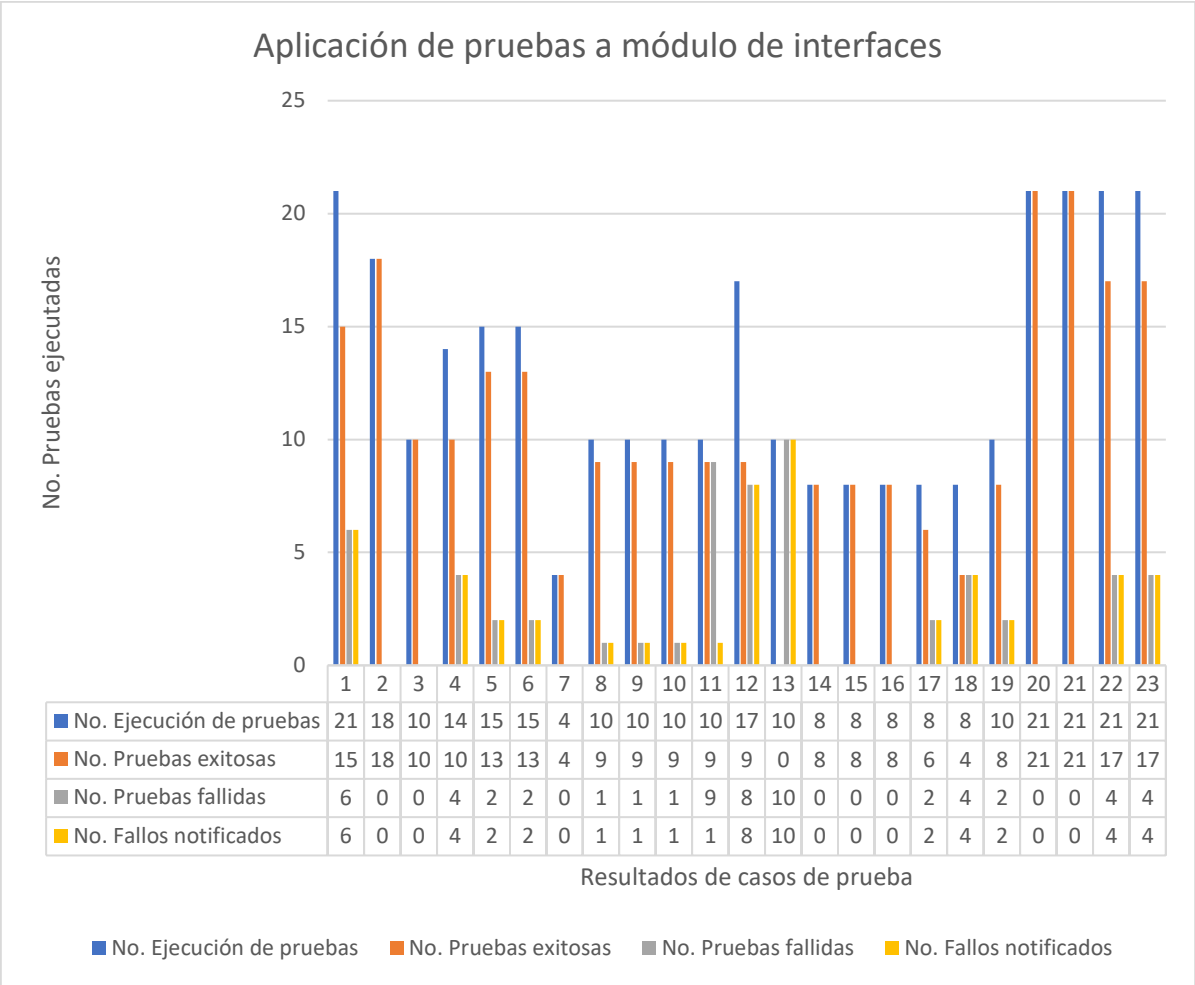
Figura 4.95.- Resultado de pruebas a Base de datos (Elaboración propia).

4.4.6 Resultados del módulo de interfaces

En el módulo de interfaces se aplicaron pruebas de caja negra para conocer el funcionamiento de las interfaces y la validación de los datos al ingresar en los campos, también se aplicaron pruebas de seguridad como permisos de acceso a ciertas interfaces según sea el usuario también se realizaron la inyección de código en los campos de cada formulario, los resultados se observan en la figura 4.96 y tabla 4.20.

Tabla 4.20.- Resultados del módulo de interfaces (Elaboración propia).

RESULTADOS DE LAS INTERFACES				
Requerimientos Tecnicos	**Casos de prueba**	**Tipo de prueba**	**resultados**	**Observaciones**
Las interfaces deben realizar las funciones correctamente	Debe realizar las funciones y operaciones correctamente con resultados deseados	Caja negra	No	Se notifica a desarrollador
Las interfaces deben contener un buen nivel de usabilidad	Debe contener un nivel de usabilidad que facilite el trabajo del usuario.	Usabilidad/Caja negra	No	Se notifica a desarrollador
Seguridad y nivel de acceso a las interfaces	Super usuario	Caja negra/ inyección de código	SI	
	Administrador	Caja negra/ inyección de código	SI	
	Consulta	Caja negra/ inyección de código	SI	
	validación de datos	Caja negra/ inyección de código	SI	
Interfaces con escalabilidad	La interfaz debe contener posibilidad de modificar sin afectar el sistema	Caja negra/caja blanca	SI	
Compatibilidad con los navegadores básico.	Chrome	Navegación/caja negra	SI	
	Firefox	Navegación/caja negra	SI	
	Safari	Navegación/caja negra	SI	
	Opera	Navegación/caja negra	SI	
	Edge	Navegación/caja negra	SI	
Eficientes con fácil recuperación ante cualquier fallo interno o externo	Recuperación con algún fallo intencional	Caja negra Rendimiento Carga	No	Se notifica a desarrollador
Interfaces responsivas para laptop, Tablet, móvil y desktop.	Laptop	Caja negra/navegación	SI	
	Desktop	Caja negra/navegación	SI	
	Tablet	Caja negra/navegación	SI	
	Smartphone	Caja negra/navegación	SI	



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
No. Ejecución de pruebas	21	18	10	14	15	15	4	10	10	10	10	17	10	8	8	8	8	8	10	21	21	21	21
No. Pruebas exitosas	15	18	10	10	13	13	4	9	9	9	9	9	0	8	8	8	6	4	8	21	21	17	17
No. Pruebas fallidas	6	0	0	4	2	2	0	1	1	1	9	8	10	0	0	0	2	4	2	0	0	4	4
No. Fallos notificados	6	0	0	4	2	2	0	1	1	1	1	8	10	0	0	0	2	4	2	0	0	4	4

Figura 4.96.- Grafica de resultados del módulo interfaces (Elaboración propia).

Nota: si se desea conocer más a fondo sobre el desarrol*lo de los módulos de CRC y de Interfaces* puedes consultar a los artículos siguientes:

Morales, M.., Sánchez, M. J., Hernández, J. J., Sánchez, A. (2017) Diseño de una Interfaz Gráfica de usuario para una Red de Objetos Inteligentes Usando la Arquitectura MVC y la Metodología Scrum. Tecnológico Nacional de México en Celaya, Congreso Internacional de Investigación de Academia Journals Celaya. ISSN 1946-5351 VOL.9 No. 6, ISBN 978-1-939982-32-2

Morales, M.., Sánchez, M. J., Hernández, J. J., Sánchez, A. (2018) Implementación de una Interfaz Gráfica de Usuario Utilizando Framework´s y la Arquitectura MVC para un Sistema Head End. Universidad Nova Spania, Congreso Internacional de Investigación de Academia Journals Morelia. ISSN 1946-5351 VOL. 10#3, ISBN 978-1-939982-36-0.

4.4.7 Resultados preventivos y correctivos de los módulos

Cada uno de los fallos detectados en cada uno de los módulos, fueron notificados al desarrollador para su pronta solución, en la mayoría de los casos se lograron resolver excepto en la implementación de *Spring Security* en la base de datos que contenía 15 pruebas fallidas, por lo que afecto en la integración de los módulos, por ende, los desarrolladores tuvieron que recurrir a soluciones prontas debido al corto tiempo de entrega y se decidió implementar dicha funcionalidad en próximas actualizaciones. Los resultados de cada módulo se ven reflejados en la figura 4.97 y tabla 4.21.

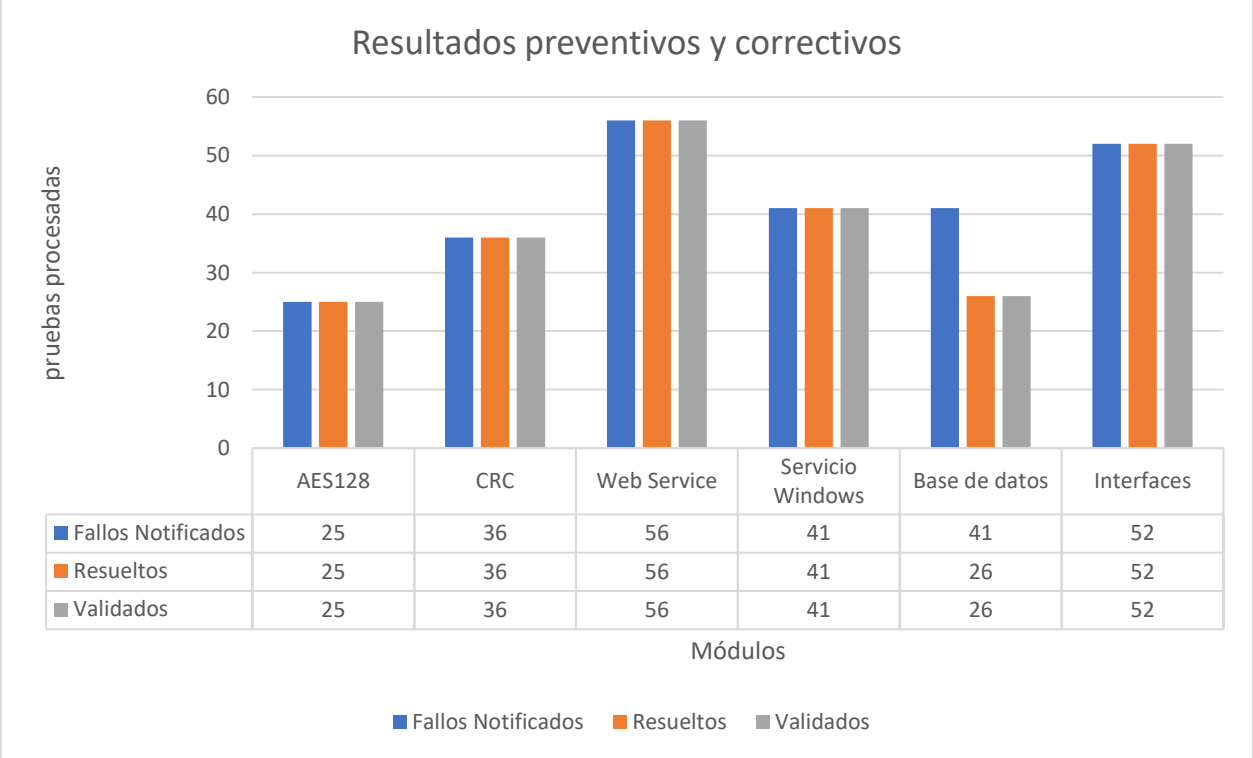


Figura 4.97.- Resultado de total de casos de fallos notificados, resueltos y validados (Elaboración propia).

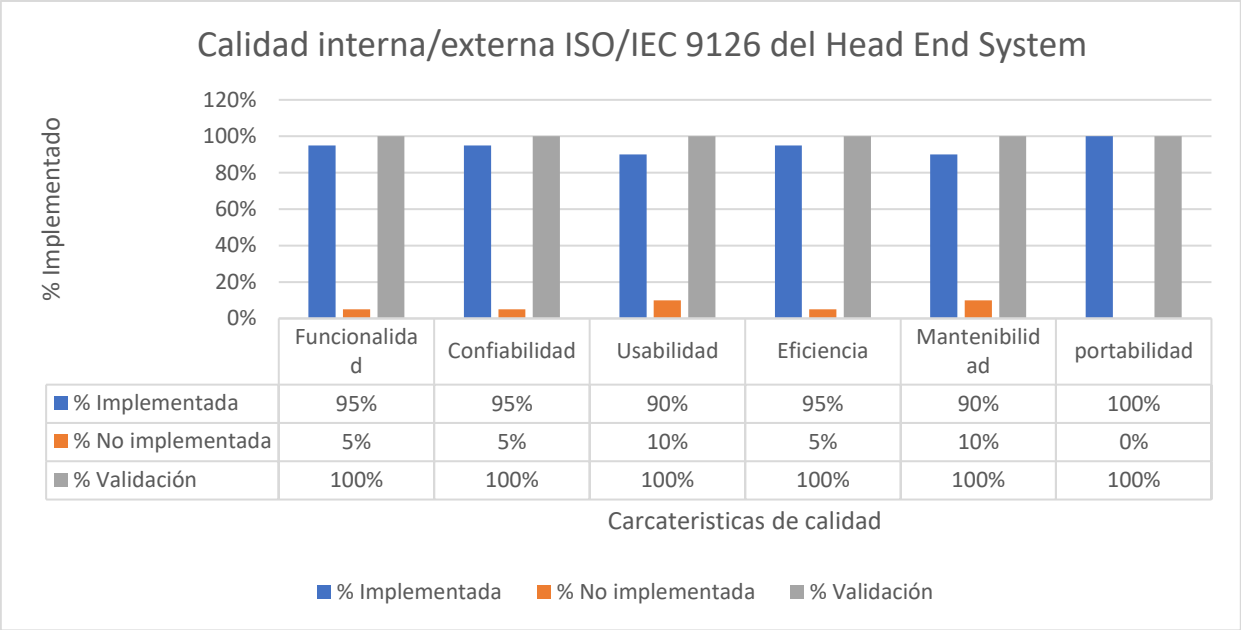
Tabla 4.21.- Total de casos de pruebas y pruebas ejecutadas en módulos (Elaboración propia).

Módulos	Casos de prueba	Pruebas ejecutadas
AES128	15	73
CRC	19	62
WEB SERVICE	8	87
SERVICIO WINDOWS	7	92
BASE DE DATOS	12	105
INTERFACES	23	298
TOTAL=	84	717

4.4.8 Resultados de la evaluación del sistema

El *Head End System* se ha desarrollado cumpliendo con los requerimientos iniciales, la evaluación del sistema es enfocado a comprobar la implementación de las principales funcionalidades deseadas, por lo que la mayor parte de ellos se ven reflejados en los módulos anteriormente evaluados. Y las otras funcionalidades serán comprobadas por medio de *checklist* diseñados para identificar las características de calidad en el sistema.

Las características y sub características del estándar *ISO/IEC FDIS 9126*, facilitaron la evaluación del sistema identificando los atributos necesarios, enfocando los esfuerzos en la comprobación del grado de existencia de cada característica de calidad en el sistema. En la figura 4.98 se muestran los porcentajes generales de cada característica de calidad, fundamentado en los resultados de los *checklist* aplicados durante la manipulación del sistema.



	Funcionalidad	Confiabilidad	Usabilidad	Eficiencia	Mantenibilidad	portabilidad
% Implementada	95%	95%	90%	95%	90%	100%
% No implementada	5%	5%	10%	5%	10%	0%
% Validación	100%	100%	100%	100%	100%	100%

Figura 4.98.- Resultados de evaluación de calidad ISO/IEC FDIS 9126 (Elaboración propia).

Conclusión

La ejecución del plan de pruebas de cada uno de los módulos y del sistema completo han sido suficientes para poder evaluar la calidad del sistema mediante la metodología *VEyVAA* que ha permitido detectar y notificar errores de manera inmediata a los programadores para su pronta corrección. La forma de presentar los datos cumple con el objetivo de dar accesibilidad a la información obtenida en las pruebas, con la finalidad de tomar decisiones correctas y fundamentadas al momento de validar el producto de software.

Capítulo 5

Conclusiones y trabajos a futuro

5.1 Conclusiones

La metodología *VEyVAA* se pudo implementar de manera exitosa en las fases de Scrum logrando una excelente comunicación entre el equipo de *testers* y el equipo *Scrum*, facilitando las notificaciones de fallos y las correcciones a tiempo, rompiendo las barreras mediante objetivos en común que es enfocar los esfuerzos hacia las vulnerabilidades del sistema y hacia lo que el cliente requiere, pero sobre todo entregar un producto de software de excelente calidad. Estas acciones dan como consecuencia la confianza y satisfacción de los usuarios finales, representando un éxito en el proyecto y buena reputación para cualquier empresa.

La implementación de la metodología *VEyVAA* en un desarrollo *Scrum*, permitió modularizar un sistema integral en pequeños componentes de software (*Sprint*), esta posibilidad garantizo la calidad de un sistema, además de estructurarlo bajo la arquitectura Modelo-Vista-Controlador, al *tester* permitió identificar y organizar su plan de pruebas dirigidas a una base de datos, a las interfaces y a la lógica de aplicación.

La integración de las pruebas y del *tester* dentro de las actividades de Scrum en el desarrollo del software proporciona beneficios para el proyecto, por ello los *tester* deben ser vistos como aliados para la mejora de los sistemas con la finalidad de enfocar los esfuerzos hacia las vulnerabilidades del sistema y hacia lo que el cliente requiere. Entregar un producto de software de buena calidad que cumpla con los objetivos iniciales, pero sobre todo que logre la confianza y satisfacción del cliente o de los usuarios finales, representa un éxito y buena reputación para cualquier empresa.

La metodología permitió analizar los requerimientos del cliente para crear los criterios de evaluación conforme a las necesidades del cliente y para comprobar la implementación de los requerimientos en el sistema.

Los procesos de evaluación implementadas en la metodología, permitieron evaluar en fases tempranas a los módulos funcionales previniendo y detectando errores de manera eficiente sin afectar los tiempos y costos del proyecto.

La verificación y validación dentro de la metodología permitió a los *testers* evaluar el sistema mediante dos enfoques diferentes:
El primer enfoque pudo evaluar de manera objetiva vigilando que los módulos desarrollados realicen de manera correcta sus funciones de acuerdo a los requerimientos técnico.
El segundo enfoque pudo evaluar de manera subjetiva al sistema completo, de una forma "destructiva" simulando a un usuario final o inexperto, realizando maniobras intencionales para provocar fallos en el sistema y así conocer la capacidad de respuesta ante situaciones complejas. Además de evaluar la calidad del sistema de acuerdo al estándar.

Las acciones de la metodología en la identificación oportuna de los errores en los módulos funcionales, han permitido disminuir los fallos en el sistema terminado. Es importante mencionar que en las acciones de corrección de errores nuevamente se ejecuta el plan de pruebas, para comprobar que se ha corregido correctamente y para descartar la existencia de nuevos errores por las modificaciones hechas durante la corrección.

La implementación del estándar *ISO 9126* en la metodología *VEyVAA* fortaleció de manera importante la forma de evaluar la calidad del software, gracias a las características de calidad interna y externa que permite profundizar en diferentes aspectos del sistema ofreciendo una garantía de aseguramiento de calidad.

La metodología pudo guiar al *tester* para diseñar un plan de pruebas eficiente en base a los requerimientos técnicos y del cliente, así como un plan a base de *checklist* basados en el estándar *ISO 9126* mediante pruebas de caja negra.

En el proceso de aplicación de pruebas la metodología permite dedicar tiempo en la selección y preparación de herramientas para ejecutar cada una de las pruebas planeadas antes de que los módulos estén desarrollados esto con la finalidad de ahorrar tiempo ya que el proceso debe ser ágil. Es por ello que es muy importante que el *tester* interno se involucre en las reuniones diarias de scrum donde se comparten toda la información de sus avances, así como las tecnologías que se utilizaran.

La metodología permitió organizar las pruebas con la ayuda de un identificador para pruebas de procedimiento TP y un identificador para pruebas de caso de uso TC, cada uno de los identificadores contienen un numero secuencial que permitió facilitar la búsqueda de cualquier prueba.

La selección de las técnicas de pruebas ha sido fundamental para la identificación de los fallos, la más compleja de las pruebas realizadas fueron las de rendimiento y carga, debido a la limitación de hardware para ejecutar miles de peticiones y de las múltiples funcionalidades.

El *Head End System* aún se encuentra limitado para comprobar funcionalidades de comunicación con otros sistemas, esto es debido a que los sistemas aún se encuentran en desarrollo, por lo que se tuvieron que realizar simulaciones por medio de un entorno cliente servidor, del cual facilito la manipulación del sistema y de la ejecución de las pruebas.

La mantenibilidad es el mayor porcentaje 10% que no se encuentra implementado en el sistema, esto debido a que no se esperan cambios de gran magnitud en la estructura del sistema, pero si se tomaron algunas acciones como la escalabilidad para agregar módulos funcionales.

El sistema fue presentado ante el cliente con una demostración de funcionamiento del sistema, para que dé su punto de vista, como resultado de ello en esta primera entrega del sistema, ha cumplido con los requerimientos que fueron planteados al equipo encargado para el desarrollo del sistema, logrando así la satisfacción del cliente.

La curva de aprendizaje en la manipulación y comprensión de las tecnologías y herramientas para ejecutar las pruebas fue basto, estas son algunas de las tecnologías que se ocuparon para poder ejecutar los módulos: *Oracle, Maven, JPA, Hibernate, spring MVC, HTML* 5, *CSS, JSP, BOOTSTRAP, JavaScript, json, eclipse, tomcat, Linux, Windows, mac, Android, Ajax, java, Oracle Spring mvc*, entre otros.

Cada proceso para aplicar pruebas a cada módulo, se llevó acabo en un promedio de 5 a 15 días dependiendo de la cantidad de casos de prueba y de la complejidad de configuración de las herramientas a utilizar, pero en las fases de ejecución de pruebas, obtención de resultados y pruebas de regresión se realizaron entre 1 y 5 días.

El proyecto *HES* tuvo una duración de 6 meses de las cuales 2 meses fueron dedicados en la investigación y en cursos de aprendizaje de estándares de comunicación, manipulación de dispositivos entre otras cosas y 4 meses se dedicaron para la construcción de módulos del sistema terminado. de las cuales fueron verificados y validado.

Es preciso mencionar que el éxito de la metodología depende de la agilidad de los *tester*, ya que en este tipo de desarrollo se exige una evaluación concreta, rápida y eficiente. Por lo que el *tester* interno y t*ester* externo deben tener la habilidad de implementar correctamente la metodología y diseñar el plan de pruebas eficiente que no afecte los tiempos y costos en el proyecto.

Para la aplicación de pruebas de cada módulo se realizó un análisis de herramientas que permitían ejecutarlos, en algunos casos se crearon pequeñas interfaces o scripts para que la prueba cumpla el objetivo. Por lo que la curva de aprendizaje fue muy buena ya que se tuvo que conocer y aprender a manipular las herramientas que el equipo de desarrollo estaba utilizando para construir sus módulos.

En experiencia dentro del proyecto en la empresa *Eos tech* fue de gran ayuda los cursos de inducción para conocer y adaptarse a todos sus procedimientos, la finalidad de cada área de la empresa y los procesos que realizaban para cumplir con sus objetivos. También se tomaron cursos para reforzar nuestros conocimientos y realizar propuestas de mejoras que en mi caso fueron muy bien recibidas por los jefes de área de pruebas de validación y área de ingeniería.

Gracias a los esfuerzos dedicados al proyecto *HES*, se lograron publicar dos artículos científicos del cual se pueden visualizar en el ANEXO 2 de esta tesis. De manera profesional me siento satisfecho por los logros alcanzados en tiempo y forma, ha resultado ser una experiencia de vida tanto profesional y como ser humano.

Gracias a las investigaciones realizadas, se logró diseñar e integrar una nueva metodología llamada *VEyVAA* en *Scrum* con evaluaciones de calidad mediante la verificación y validación ágil, siendo aplicado en un proceso real de evaluación de calidad de un sistema *Head End*, obteniendo excelentes resultados que demostraron la agilidad de las notificaciones de fallas detectadas del cual fueron atendidas de manera ágil y como consecuencia mejoraba las funcionalidades del producto hasta obtener un software de alta calidad, lo anterior fundamenta el cumpliendo de los objetivos específicos de la tesis, pero sobre todo respondiendo de manera positiva a la pregunta de investigación de esta tesis, ya que si es posible asegurar la calidad debido a que se generó un producto de software confiable logrando así la satisfacción del usuario final

5.2 Trabajos futuros

Implementación de inspección de calidad en las fases de requerimiento y diseño del software, que permita evaluar en base estándares de calidad sobre las fases mencionadas para prevenir algún error en la adquisición e implementación de los requerimientos del cliente (esta propuesta ya se encuentra diseñado, pero aún no ha sido implementado).

La implementación de un proceso de evaluación de fallas que permita conocer el grado de afectación al sistema para priorizar las fallas atender, si la metodología ágil permite más tiempo para un análisis de este tipo, aumentaría la eficiencia de evaluación de calidad del software.

Respecto al estándar *ISO 9126* contiene un apartado enfocado a la calidad de uso, del cual permite evaluar el sistema en funcionamiento en un campo real, esta posibilidad agregaría un filtro más de calidad para dar seguimiento en la detección de fallas y dar mantenimiento en las funcionalidades del sistema, por lo que se podría diseñar un proceso de calidad de uso y adaptarlo a la metodología *VEyVAA.*

En la planificación general del proyecto de desarrollo del sistema *HES*, están establecidos futuras implementaciones de módulos funcionales para gestionar medidores inteligentes, no solo de energía eléctrica sino de gas y agua, por lo que sería otra oportunidad para aplicar la metodología y procesos de pruebas al sistema, con mejoras en el proceso de evaluación con el apoyo de un equipo de ingenieros especializados en las áreas mencionadas en la metodología.

Para el periodo en que se estima robustecer el sistema tal vez los sistemas informáticos que van a interactuar con él *HES* ya se encuentren disponibles, esto permitiría realizar pruebas más reales en la comunicación y transferencia de datos.

Bibliografia y referencias

Abad, J. H. (2005) Tipos de pruebas de software. Ingeniería de software, Blogspot Google+. Recuperado de: http://ing-sw.blogspot.com/2005/04/tipos-de-pruebas-de-software.html

Beck et al. (2001) Principios de Manifiesto ágil. Recuperado de: http://agilemanifesto.org/iso/es/principles.html

Berger, H., Beynon-Davies, P., y Cleary, P. (2004). The Utility of a Rapid Application Development (RAD) approach for a large complex Information Systems Development. Proceedings of the 13th European Conference on Information Systems, The European IS Profession in the Global Networking Environment, ECIS. Turku, Finland

Boehm B. (1984) Verifying and Validating Software Requirements and Design Specifications, IEEE Software, vol. 1, pp. 75-88.

Boehm B. and R. Turner. (2003). Observations on Balancing Discipline and Agility. Proceedings of the Agile Development Conference (ADC'03). June. Salt Lake City, Utah, USA.

Brunschwiler, C. (2013) Advanced Metering Infrastructure Architecture and Components. Compass Security Network Computing AG. Suiza. Recuperado de: https://blog.compass-security.com/2013/02/advanced-metering-infrastructure-architecture-and-components/

Choudhary G.R., et al. (2018) Empirical analysis of change metrics for software fault prediction. ELSEVIER. Computers and Electrical Engineering, 67. 0045-7906, doi.org/10.1016/j.compeleceng.2018.02.043

CMMI (2002) Capability Maturity Model Integration (CMMI), Version 1.1, CMMI for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1.1), Continuous Representation CMU/SEI-2002-TR-011 ESC-TR-2002-011.

CMMI Institute (2018) CMMI. Pittsburgh Pensilvania, EE.UU. Recuperado de: https://cmmiinstitute.com/cmmi

Corrales J. M. (2012) Controlar el consumo de energía, desde cualquier lugar, con EnviR Recuperado de http://www.efimarket.com/blog/controlar-el-consumo-de-energia-desde-cualquier-lugar-con-envir/

Darwish, S. M. (2016) Software test quality rating: A paradigm shift in swarm computing for software certification. ELSEVIER, Knowledge-Based Systems 110 (2016) 0950-7051. doi.org/10.1016/j.knosys.2016.07.022

Dijkstra, E. (1970) Notes on structures Programming, TH Report 70 –WSK-03, recuperado de: http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF

Eneri (2013) Sistema de infraestructura avanzada de medición. Eneri, The Intelligence of Energy. Recuperado de: http://www.eneri.com.mx/AMI

Enríquez, J. G. Sánchez, J. M., Domínguez F. J., García, J. A., Escalona, M. J. (2018) An approach to characterize and evaluate the quality of Product Lifecycle Management Software Systems. *Computer Standards & Interfaces* (2018), doi: 10.1016/j.csi.2018.05.003

Flores, M., Medina, M. G., Hernández j. j., Sánchez A. (2017) Diseño de una Metodología de Testing en Scrum para un Sistema Integral de tipo Head End enfocado a Smart Cities. Memorias del Congreso Internacional de Investigación Academia Journals Celaya 2017. ISSN 1946-5351 online Vol. 9, No. 6.

Flores, M., Medina, M. G., Hernández j. j., Sánchez A. (2018) Proceso de evaluación de calidad de un Head End System para una red de medidores inteligentes, basado en los requerimientos del cliente y en las características de calidad de la ISO/IEC FDIS 9126. Memorias del Congreso Internacional de investigación Academia Journals Morelia 2018. ISSN 1946-5351 Vol. 10, No. 3.

Galin, D., Software Quality Assurance – From Theory to Implementation, Pearson Education Limited, 2004.

Ho-Won J, Seung-Gweon K, y Chang-Shin C. (2004) Measuring Software Product Quality: A Survey of ISO/IEC 9126, IEEE Software. doi:10.1109/MS.2004.1331309

IEEE Std 610.12 (1990) Standard Glossary of Software Engineering Terminology. IEEE

IEEE (1994). Guide for Software Verification and Validation Plans, in IEEE Std 1059-1993. doi: 10.1109/IEEESTD.1994.121430

IEEE (1998). Standard for Developing Software Life Cycle Processes, in IEEE Std 1074-1997. doi: 10.1109/IEEESTD.1998.88827

IEEE (1998) Standard for Software Verification and Validation, in IEEE Std 1012-1998. doi: 10.1109/IEEESTD.1998.87820.

IEEE (1986) Standard for Software Unit Testing, in ANSI/IEEE Std 1008-1987. doi: 10.1109/IEEESTD.1986.81001.

IEEE (1990) Standard Glossary of Software Engineering Terminology Institute of Electrical and Electronics Engineers, ISBN: 155937067X, 1990.

IEEE (1998) Recommended Practice for Software Requirements Specifications, in IEEE Std 830-1998 doi: 10.1109/IEEESTD.1998.88286.

ISO 8402 (1994) Quality management and quality assurance – Vocabulary. International Standard, Vernier, Geneva Switzerland.

ISO 25000 (2018) ISO/IEC 25010. ISO 25000.com. Calidad del producto software. Recuperado de http://iso25000.com/index.php/normas-iso-25000/iso-25010?limit=3&limitstart=0

ISO/ IEC FDIS 9126-1 (2001). Information technology Software, product quality Part 1: Quality model. International Standard, Vernier, Geneva Switzerland

ISO/IEC TR 9126-2 (2003). Software engineering -Product quality- part2: External metrics. International Organization for Standardization. Vernier, Geneva Switzerland

ISO/IEC TR 9126-3 (2003). Software engineering -Product quality- part3: Internal metrics. De International Organization for Standardization. Vernier, Geneva Switzerland

ISO/IEC TR 9126-4 (2004). Software engineering -Product quality- part4: Quality In Use metrics. De International Organization for Standardization. Vernier, Geneva Switzerland.

ISO/IEC 25010 (2011) Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models. International Organization for Standardization. Vernier, Geneva Switzerland.

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering --Software testing --Part 1: Concepts and definitions. Vernier, Geneva Switzerland.

ISTQB (2016) Standard Glossary of Terms used in Software Testing Version 3.1 All Terms. International Software Testing Qualifications Board. Recuperado de: https://www.istqb.org/downloads/send/20-istqb-glossary/186-glossary-all-terms.html

ISTQB (2016) Pruebas de caja negra. Recuperado de: http://www.pmoinformatica.com/2016/04/pruebas-caja-negra-istqb.html

Khosravi A. et al. (2017) Toward software quality enhancement by Customer Knowledge Management in software companies. ELSEVIER, *Telematics and Informatics 35*. 0736-5853. doi.org/10.1016/j.tele.2017.09.007.

Kit E. (1995) Software Testing In The Real World: Improving The Process, ISBN 0201877562, Addison Wesley.

Lara, W. (2015) Metodología Scrum: Cómo funciona la metodología de trabajo Scrum. Platzi. Recuperado de: https://platzi.com/blog/metodologia-scrum-fases/

Leviton (s.f) Productos y software de recopilación de datos del administrador de energía. recuperado de http://spanish.leviton.com/OA_HTML/SectionDisplay.jsp?section=42096

Lozano, L. (1998). ¿Qué es calidad total?. Revista Medica Herediana, 9(1), 28-34. Recuperado en 20 de enero de 2018, de http://www.scielo.org.pe/scielo.php?script=sci_arttext&pid=S1018-130X1998000100006&lng=es&tlng=es.

Maner, W. (1997) Rapid Application Development. Dalhouse University, Faculty of computer science. Recuperado de: https://web.cs.dal.ca/~jamie/teach/WaltManer/RAD.htm

Martin, J (1991). Rapid Application Development, MacMillan, New York.

McCall, J. A., Richards, P. K., y Waters, G. (1977). Factors in Software Quality: Final Technical Report. Rome Air Development Center, Air Force System Command, Griffith Air Force Base, NY. RADC-TR-77-369.

Mendieta S. (2015). Medidores inteligentes, nueva ruta para la CFE. Recuperado de: http://www.milenio.com/negocios/Medidores-inteligentes-nueva-ruta-CFE_0_629937010.html

Meana S. (2015). La nueva tecnología que te permite ahorrar en el consumo eléctrico. recuperado de http://www.elfinanciero.com.mx/economia/la-nueva-tecnologia-que-te-permitira-ahorrar-en-el-consumo-electrico.html

Minue (2013). Controla tu consumo eléctrico con Efergy Recuperado de https://www.xatakahome.com/iluminacion-y-energia/controla-tu-consumo-electrico-con-efergy

Myers G. (2004) The art of software testing, 2nd edition, ISBN 0-471-46912-2, John Wiley & Sonsm Inc.

NYCE (2016) MoProSoft. CD. México, Mexico Recuperado de: https://www.nyce.org.mx/moprosoft-nyce/

Pande S. et al. (2013) Software Quality Assurance activities of ITER CODACS. ELSEVIER, Fusion Engineering and Design *88*. 0920-3796. doi.org/10.1016/j.fusengdes.2013.02.144.

Pavón, J. (2009) Estructura de las Aplicaciones Orientadas a Objetos El patrón Modelo-Vista-Controlador (MVC). Universidad Complutense Madrid. Dep. Ingeniería del Software e Inteligencia Artificial, Madrid, España.

Pérez, B. (2006) Proceso de testing funcional independiente. PDCIBA Informática, Instituto de computación (InCo) Facultad de ingeniería. Montevideo, Uruguay

Posada M. (2016) Megacable concluye instalación de medidores inteligentes de la CFE. Recuperado de http://www.jornada.unam.mx/ultimas/2016/03/14/megacable-concluye-instalacion-de-medidores-inteligentes-de-cfe-9212.html

Pressman, R. (2002). Ingeniería del software, un enfoque practico, Quinta edición. DF, México: Mc Graw Hill INTERAMERICANA EDITORES.

Pressman, R. (2010). Ingeniería del software, un enfoque practico, séptima edición. DF, México: Mc Graw Hill INTERAMERICANA EDITORES.

Pmoinformatica (2016) Pruebas de software: 10 pasos para elaborar el plan de pruebas. Recuperado de: http://www.pmoinformatica.com/2016/01/elaborar-plan-pruebas-software.html

Ramesh Radhakrishnan, R. R. (2004). IT Infrastructure Architecture Building Blocks. Santa Clara, California: Sun Microsystems, Inc.

Reifer, D. (1985) State of the Art in Software Quality Management, Reifer Consultants.

Reenskaug T. (1979) THING-MODEL-VIEW-EDITOR an Example from a planning system. Universidad de Oslo, Noruega. Recuperado de: http://folk.uio.no/trygver/1979/mvc-1/1979-05-MVC.pdf

Rueda, M. (2017) ¿Qué son las 'Smart cities'? BBVA, España. Recuperado de: https://www.bbva.com/es/las-smart-cities/

Santamaria P. (2012) ODEnergy Home, monitoriza el consumo energético Recuperado de https://www.xatakahome.com/iluminacion-y-energia/odenergy-home-monitoriza-el-consumo-energetico

Schulmeyer, G. G. and McManus, J. I., (1992) eds. Handbook of Software Quality Assurance, 2nd ed, Van Nostrand Rheinhold.

SmartGrid (2010) AMI Network (Moving data elements from the AMI Head-End to Smart Meter & the from Smart Meter to the AMI Head-End). Electric Power Research Institute. Recuperado de: https://www.smartgrid.gov/files/AMI_Network_Moving_Data_Elements_from_AMI_HeadEnd_Smart_Mete.pdf

Sommerville I. (2005) Ingeniería del Software, 7ª Edición, México, Pearson de México-Addison Wesley.

Sommerville I. (2011) Ingeniería del Software, 9ª Edición. México, Pearson de México-Addison Wesley.

Udaondo, M. (1992) Gestión de calidad. Ediciones Díaz de Santos, p. 35, ISBN 9788479780135

Vargas, C. G. y Biagioli G. (2009) Sistema para auditar el cumplimiento de CMMI-SW nivel 2. Facultad de Informática, Universidad Nacional de La Plata, Argentina.

Vásquez, C., y Estrada, E. (2013). Towards the preparation of the Guadalajara's. IEEE Smart Cities, the Smart Cities of the Future Kickoff Event, Guadalajara, Mexico. Recuperado de: https://smartcities.ieee.org/images/files/pdf/whitepapermtx_v8.pdf

Villareal, D. D., Gamboa, S. S., & Gómez, F. L. (2015). Estudio sobre realización y documentación de pruebas software. Recuperado de MASKANA: https://publicaciones.ucuenca.edu.ec/ojs/index.php/maskana/article/view/717/634

Zapata J. (2013). Principios fundamentales del proceso de pruebas recuperado de https://pruebasdelsoftware.wordpress.com/2013/01/07/principios-fundamentales-del-proceso-de-pruebas/

Zapata J. (2013) Niveles de prueba del software. Recuperado de: https://pruebasdelsoftware.wordpress.com/

Zapata J. (2013) Metodología de Pruebas. Recuperado de: https://pruebasdelsoftware.wordpress.com/

Referencias de figuras

Figura 1.1.- Pressman, R. (2010). Ingeniería del software, un enfoque practico. [Figura] séptima edición. DF, México: Mc Graw Hill INTERAMERICANA EDITORES.

Figura 1.2.- Meana S. (2015). La nueva tecnología que te permite ahorrar en el consumo eléctrico. recuperado de http://www.elfinanciero.com.mx/economia/la-nueva-tecnologia-que-te-permitira-ahorrar-en-el-consumo-electrico.html

Figura 1.3.- Leviton (s.f) Productos y software de recopilación de datos del administrador de energía. [Figura] Recuperado de http://spanish.leviton.com/OA_HTML/SectionDisplay.jsp?section=42096

Figura 1.4.- Leviton (s.f) Productos y software de recopilación de datos del administrador de energía. [Figura] Recuperado de http://spanish.leviton.com/OA_HTML/SectionDisplay.jsp?section=42096

Figura 1.5.-Minue (2013). Controla tu consumo eléctrico con Efergy. [Figura] Recuperado de https://www.xatakahome.com/iluminacion-y-energia/controla-tu-consumo-electrico-con-efergy

Figura 1.6.- Minue (2013). Controla tu consumo eléctrico con Efergy. [Figura] Recuperado de https://www.xatakahome.com/iluminacion-y-energia/controla-tu-consumo-electrico-con-efergy

Figura 1.7.-Santamaria P. (2012) ODEnergy Home, monitoriza el consumo energético. [Figura] Recuperado de https://www.xatakahome.com/iluminacion-y-energia/odenergy-home-monitoriza-el-consumo-energetico

Figura 1.8.- Corrales J. M. (2012) Controlar el consumo de energía, desde cualquier lugar con EnviR. [Figura] Recuperado de http://www.efimarket.com/blog/controlar-el-consumo-de-energia-desde-cualquier-lugar-con-envir/

Figura 1.9.- Corrales J. M. (2012) Controlar el consumo de energía, desde cualquier lugar con EnviR. [Figura] Recuperado de http://www.efimarket.com/blog/controlar-el-consumo-de-energia-desde-cualquier-lugar-con-envir/

Figura 1.10.- Eneri (2013) Sistema de infraestructura avanzada de medición. Eneri, The Intelligence of Energy. [Figura] Recuperado de: http://www.eneri.com.mx/AMI

Figura 1.11.- Eneri (2013) Sistema de infraestructura avanzada de medición. Eneri, The Intelligence of Energy. [Figura] Recuperado de: http://www.eneri.com.mx/AMI

Figura 2.1.- Pressman, R. (2010). Ingeniería del software, un enfoque practico. [Figura] séptima edición. DF, México: Mc Graw Hill INTERAMERICANA EDITORES.

Figura 2.3.- Lara, W. (2015) Metodología Scrum: Cómo funciona la metodología de trabajo Scrum. [Figura] Platzi. Recuperado de: https://platzi.com/blog/metodologia-scrum-fases/

Figura 2.7.- Pressman, R. (2010). Ingeniería del software, un enfoque practico. [Figura] séptima edición. DF, México: Mc Graw Hill INTERAMERICANA EDITORES.

Figura 2.10.- Brunschwiler, C. (2013) Advanced Metering Infrastructure Architecture and Components. [Figura] Compass Security Network Computing AG. Suiza. Recuperado de: https://blog.compass-security.com/2013/02/advanced-metering-infrastructure-architecture-and-components/

Figura 2.11.- Flores, M., Medina, M. G., Hernández j. j., Sánchez A. (2017) Diseño de una Metodología de Testing en Scrum para un Sistema Integral de tipo Head End enfocado a Smart Cities. [Figura] Memorias del Congreso Internacional de Investigación Academia Journals Celaya 2017. ISSN 1946-5351 online Vol. 9, No. 6.

Figura 4.1.- Flores, M., Medina, M. G., Hernández j. j., Sánchez A. (2017) Diseño de una Metodología de Testing en Scrum para un Sistema Integral de tipo Head End enfocado a Smart Cities. [Figura] Memorias del Congreso Internacional de Investigación Academia Journals Celaya 2017. ISSN 1946-5351 online Vol. 9, No. 6.

Referencias de tablas

Tabla 2.1.- Pressman, R. (2010). Ingeniería del software, un enfoque practico. [Tabla] séptima edición. DF, México: Mc Graw Hill INTERAMERICANA EDITORES.

Tabla 2.4.- Lara, W. (2015) Metodología Scrum: Cómo funciona la metodología de trabajo Scrum. [Tabla] Platzi. Recuperado de: https://platzi.com/blog/metodologia-scrum-fases/

Tabla 2.8.- Vargas, C. G. y Biagioli G. (2009) Sistema para auditar el cumplimiento de CMMI-SW nivel 2. [Tabla] Facultad de Informática, Universidad Nacional de La Plata, Argentina.

Tabla 2.10.- Pressman, R. (2010). Ingeniería del software, un enfoque practico. [Tabla] séptima edición. DF, México: Mc Graw Hill INTERAMERICANA EDITORES.

Tabla 2.11.- NYCE (2016) MoProSoft. [Tabla] CD. México, Mexico Recuperado de: https://www.nyce.org.mx/moprosoft-nyce/

Tabla 2.12.- NYCE (2016) MoProSoft. [Tabla] CD. México, Mexico Recuperado de: https://www.nyce.org.mx/moprosoft-nyce/

Tabla 2.13.- Pressman, R. (2010). Ingeniería del software, un enfoque practico. [Tabla] séptima edición. DF, México: Mc Graw Hill INTERAMERICANA EDITORES.

Tabla 2.14.- Vargas, C. G. y Biagioli G. (2009) Sistema para auditar el cumplimiento de CMMI-SW nivel 2. [Tabla] Facultad de Informática, Universidad Nacional de La Plata, Argentina.

Tabla 2.15.- ISO/IEC TR 9126-2 (2003). Software engineering -Product quality- part2: External metrics. [Tabla] International Organization for Standardization. Vernier, Geneva Switzerland.

Tabla 2.16.- ISO/IEC 25010 (2011) Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models. [Tabla] International Organization for Standardization. Vernier, Geneva Switzerland.

Tabla 3.1.- ISO/ IEC FDIS 9126-1 (2001). Information technology Software, product quality Part 1: Quality model. [Tabla] International Standard, Vernier, Geneva Switzerland.

Tabla 3.2.- ISO/ IEC FDIS 9126-1 (2001). Information technology Software, product quality Part 1: Quality model. [Tabla] International Standard, Vernier, Geneva Switzerland.

Tabla 3.3.- ISO/ IEC FDIS 9126-1 (2001). Information technology Software, product quality Part 1: Quality model. [Tabla] International Standard, Vernier, Geneva Switzerland.

Tabla 3.4.- ISO/ IEC FDIS 9126-1 (2001). Information technology Software, product quality Part 1: Quality model. [Tabla] International Standard, Vernier, Geneva Switzerland.

Tabla 3.5.- ISO/ IEC FDIS 9126-1 (2001). Information technology Software, product quality Part 1: Quality model. [Tabla] International Standard, Vernier, Geneva Switzerland.

Tabla 3.6.- ISO/ IEC FDIS 9126-1 (2001). Information technology Software, product quality Part 1: Quality model. [Tabla] International Standard, Vernier, Geneva Switzerland.

Glosario de términos

Termino	Descripción
ACS/SQA	Aseguramiento de la calidad del software/ *Software Quality Assurance*.
AES128	*Advanced Encryption Standard (AES)*, es un esquema de cifrado por bloques de 128 bits (existen de más bits) adoptado como un estándar de cifrado
AJAX	*Ajax (Asynchronous JavaScript and XML)* se refiere a un grupo de tecnologías que se utilizan para desarrollar aplicaciones web.
AMI	*Advanced Metering Infrastructure*, Es un sistema de medición remota del consumo de energía eléctrica conformada por una infraestructura de hardware y software integrando 3 subsistemas *(SON, MDMS, HES)*
Base de Datos	Es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso
Bootstrap	Bootstrap es un framework web o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web
Building Blocks	Arquitectura para modularizar un sistema en bloques funcionales e independientes
Confiabilidad	Es cuando el sistema puede mantener un nivel de rendimiento especificado ante alguna falla, y con la capacidad de recuperar los datos perdidos por la falla
CMD	es la consola que interpreta comandos de *DOS* en sistemas operativos de Windows.
Comparative	Herramienta que permite comparar las cadenas de números

CRC	Es la verificación por redundancia cíclica es un código de detección de errores usado frecuentemente en redes digitales y en dispositivos de almacenamiento para detectar cambios accidentales en los datos.
CSS	Son hojas de estilo en cascada es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado
Eclipse neón	es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar aplicaciones basadas en navegadores
Eficiencia	Es cuando el sistema realiza sus funciones de manera rápida sin utilizar excesivamente los recursos de hardware.
EOLO	Es una herramienta que permite generar frame´s de CRC
Estándar de calidad	es un producto de referencia que facilita la tarea de fijar unas condiciones mínimas para que los aspectos y características de un producto, satisfaga de forma eficiente las necesidades de sus usuarios.
Error/fallos/defectos	En el desarrollo de software se considera un error algún código incorrectamente insertada, fallo es lo que se produce cuando el sistema está en operación y deja de funcionar, por último, defecto se llama así a los resultados no deseados que arroja el sistema al realizar una función.
Frame	Paquete de datos que envían los medidores inteligentes del consumo de energía eléctrica
Funcionalidad	Es la correcta forma de operar del sistema y el buen cumplimiento de las necesidades del cliente
Head End System (HES)	sistema integral que gestiona toda la información de forma centralizada, en él se pueden integrar más sistemas para realizar diversas funciones y procesamiento.
Hibernate	*Hibernate* es una herramienta de mapeo *objeto-relacional (ORM)* para la plataforma *Java*
HTML	*HyperText Markup Language* (lenguaje de marcas de hipertexto), es muy utilizado en la elaboración de páginas web.
Interfaz	se utiliza para nombrar a la conexión funcional entre dos sistemas, programas, dispositivos o componentes de cualquier tipo, que proporciona una comunicación de distintos niveles permitiendo el intercambio de información
ISO/IEC 9126	Estándar para evaluar la calidad interna y externa de un software
Java	Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos
JPA	*Java Persistence API*, más conocida por sus siglas *JPA*, es la *API* de persistencia desarrollada para la plataforma *Java EE*
JSON	JSON, acrónimo de *JavaScript Object Notation*, es un formato de texto ligero para el intercambio de datos
Login	ES la interfaz de inicio de sesión con usuario y contraseña
Mantenibilidad	Es cuando el sistema puede ser modificado para corregir agregar o actualizar a otra nueva versión
Maven	Es una herramienta de software para la gestión y construcción de proyectos Java

MDMS	*Meter Data Management System*, se refiere al software que realiza el almacenamiento y la administración de datos a largo plazo para la gran cantidad de datos que entregan los sistemas de medición inteligente.
Metodología	Define y sistematiza el conjunto de técnicas, métodos y procesos que se debe seguir durante algún desarrollo.
Módulo	Componente funcional del software, que puede funcionar sin depender de otros
MVC	Arquitectura que divide en 3 capas al sistema: el modelo (base de datos), la vista (interfaces) y el controlador (lógica de aplicación)
Nivel de pruebas	Fases de pruebas dentro dl desarrollo de software
Oracle database	Es un sistema de gestión de base de datos de tipo objeto-relacional
Pila del producto	Es una lista priorizada sobre las necesidades del cliente o historial de usuario
Portabilidad	El sistema puede ser transferido a otro entorno o ser ejecutado en cualquier plataforma o dispositivo.
Product Backlog	Lista priorizada de las necesidades del cliente
Product Owner	Dueño del producto, se encarga de priorizar las necesidades del cliente en una pila del producto
Retrospectiva sprint	Es un evento de análisis sobre las cosas buenas y malas al desarrollar para implementarlo en el siguiente sprint
Script	es un conjunto de órdenes guardadas en un archivo de texto, que es ejecutado por lotes o línea a línea, en tiempo real por un intérprete.
Scrum	Marco de desarrollo de software ágil.
Scrum Events	Eventos o Actividades que se realizan durante el proceso de desarrollo del software
Scrum Master	Líder del proyecto de Scrum gestiona el desarrollo del software junto con su equipo scrum
Scrum Team	Es el equipo de desarrollo del software
Servicio de Windows	Un servicio de Windows es un programa de ordenador que funciona en segundo plano mediante múltiples hilos que priorizan su ejecución.
Smart cities	Es un esquema basada en tecnologías inteligentes que adoptan las ciudades con grandes índices de habitantes para mejorar la infraestructura que garantice un desarrollo, la calidad de vida de sus ciudadanos y la prestación servicios eficientes.
SON	*System Object Network*, es una red de medidores inteligentes que mide el consumo eléctrico
Spring MVC	*Spring Web MVC* es un sub-proyecto *Spring* que está dirigido a facilitar y optimizar el proceso creación de aplicaciones web utilizando el patrón *MVC* (Modelo-Vista-Controlador), donde el Modelo representa los datos o información que manejará la aplicación web, la Vista son todos los elementos de la UI (Interfaz de Usuario), con ellos el usuario interactúa con la aplicación, ejemplo: botones, campos de texto, etc., finalmente el Controlador será el encargado manipular los datos en base a la interacción del usuario.

Spring Security	*Spring Security* es un *framework* que permitirá gestionar todo lo relativo a la seguridad de nuestra aplicación web, desde el protocolo de seguridad, hasta los roles que necesitan los usuarios para acceder a los diferentes recursos de la aplicación.
Sprint	Es una Iteración o incremento de producto utilizable que se desarrolla en un periodo corto, potencialmente entregable.
Sprint Planning	Planeación del *sprint*
Sprint review	Revisión del sprint para conocer que los módulos se encuentran terminados correctamente
Stakeholder	Son los interesados en el proyecto como el cliente, proveedores y usuarios finales
Técnicas de pruebas	Son los tipos de pruebas que existen para evaluar un software
Test Case (TC)	Casos de prueba, son el conjunto de funcionalidades o características que integran un TP del cual se deben evaluar
Test Procedure (TP)	Procedimiento de prueba, es la función principal que debe realizar un módulo
Tester	Es el responsable de aplicar pruebas
Tester Externo	Es el responsable de validar el sistema terminado mediante un estándar de calidad a favor del cliente
Tester Interno	Es el responsable de Verificar los módulos y trabaja en conjunto con el equipo de scrum
Testing	Es una actividad sobre la aplicación de pruebas
Tomcat	Es un servidor web que soporta la tecnología *J2EE* de *JAVA* y es un contenedor de *servlet y javaserver pages*
Usabilidad	Es cundo el sistema puede ser entendido, aprendido, usado y atractivo para el usuario
Utility	Sistema de servicios públicos (electricidad, gas, agua etc.)
Validación	En la metodología VEyVAA se encarga de evaluar el sistema terminado
VEyVAA	Es una Metodología de aseguramiento de la calidad del software en base a la Verificación y validación ágil con un nuevo enfoque de evaluación.
Verificación	En la metodología VEyVAA se encargará de evaluar los módulos en base a los requerimientos
Web service	Es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones
XML	*Extensible Markup Language*, es un formato universal para datos y documentos estructurados

ANEXO 1

En este anexo 1 se presentan evidencias del trabajo realizado en la empresa *Eos tech*, así como la presentación de documentos oficiales que avalan el proyecto donde se implementó la metodología *VEyVAA* de la presente tesis. Las evidencias son las siguientes:

Carta de liberación de la empresa *Eos Tech* por haber cubierto satisfactoriamente las fases del proyecto. La carta se observa en la figura 1 de este anexo.

EOS TECH

Querétaro, Qro., a 02 de Agosto del 2017

Asunto: Constancia

MTRO. FELIPE PASCUAL ROSARIO AGUIRRE
DIRECTOR DEL INSTITUTO TECNOLÓGICO DE APIZACO
PRESENTE:

AT'N: DR. JOSÉ FEDERICO CASCO VÁZQUEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Por medio del presente le envío un cordial saludo y aprovecho para hacerle constar que el Ing. Manuel Flores Nava, alumno de la Maestría en Sistemas Computacionales con número de control M11370799, de la Institución que usted destacadamente dirige, participó en el sub-proyecto:

"Metodología de aseguramiento de la calidad de un sistema *Head End (HES)* de una red distribuida de medidores inteligentes"

La etapa del proyecto realizada tuvo una duración de 6 meses y se desarrolló en las oficinas centrales de EOS TECH S.A. DE C.V. en la ciudad de Querétaro, Querétaro.

En virtud de que se han cubierto satisfactoriamente la fase del proyecto. Tenemos bien a dar constancia de que dicho trabajo realizado por el Ing. Manuel Flores Nava, cubre y satisface las expectativas planteadas al inicio de la fase de este proyecto.

Agradeciendo sus atenciones a la presente quedo de ustedes.

ATENTAMENTE

Ing. Juan González Salomón
Director General
EOS TECH S.A. DE C.V.

EOS TECH S.A de C. V.
Luis G. Malvaez No. 523 Col. Reforma Agraria 2da Sección.
Querétaro, Qro., México. C.P. 76086
Tel.: (442) 243 1331

www.eostech.mx
"Mejoramos tu mundo con innovación"

Figura 1. Carta de liberación de la empresa Eos Tech

Carta de satisfacción de la empresa *Eos Tech* por cubrir y satisfacer las expectativas planteadas al inicio del proyecto. La carta se observa en la figura 2 de este anexo.

EOS TECH

Querétaro, Qro., a 02 de Agosto del 2017

Asunto: Constancia de satisfacción

MTRO. FELIPE PASCUAL ROSARIO AGUIRRE
DIRECTOR DEL INSTITUTO TECNOLÓGICO DE APIZACO
PRESENTE:

AT'N: DR. JOSÉ FEDERICO CASCO VÁZQUEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Por medio del presente le envío un cordial saludo y aprovecho para hacerle constar que posterior a la recepción del sub-proyecto: **"Metodología de aseguramiento de la calidad de un sistema *Head End (HES)* de una red distribuida de medidores inteligentes"** realizado por el Ing. Manuel Flores Nava con número de control M11370799, y dirigido por la M.C. María Guadalupe Medina Barrera, alumno y profesora respectivos de la Maestría en Sistemas Computacionales, de la Institución que usted destacadamente dirige, **cubre y satisface las expectativas planteadas al inicio de la fase de este proyecto.**

Tenemos bien a dar constancia de que dicho trabajo realizado por el Ing. Manuel Flores Nava. Agradeciendo sus atenciones quedo de ustedes.

ATENTAMENTE

Ing. Juan González Salomón
Director General
EOS TECH S.A. DE C.V.

EOS TECH S.A de C. V.
Luis G. Malvaez No. 523 Col. Reforma Agraria 2da Sección.
Querétaro, Qro., México. C.P. 76086
Tel.: (442) 243 1331

www.eostech.mx
"Mejoramos tu mundo con innovación"

Figura 2. Carta de satisfacción de la empresa Eos Tech

Reconocimiento de la empresa *Eos tech* por la participación en el proyecto *Head End System.* El reconocimiento se observa en la figura 3 de este anexo.

Figura 3. Reconocimiento de la empresa Eos Tech por la participación en el proyecto Head End System

Proyecto *Head End System*

Durante la estancia en la empresa *Eos tech S. A. de C. V.* Querétaro, se realizó una serie de actividades que fueron fundamentales para plantear una propuesta de mejora dentro del área de pruebas de la empresa, así como para cumplir las actividades asignadas de evaluar los módulos desarrollados del *Head End System (HES)* mediante la metodología que se ha propuesto en la presente tesis, del cual se adaptó conforme el reglamento de la empresa.

En la figura 4 se observa una pantalla del sistema de validación del producto de la empresa "*Eos Tech*" del cual se registran los proyectos en construcción que en este caso se muestra al equipo que participo en el desarrollo del *Head End System.*

Figura 4. Equipo de desarrollo de software del Head End System (Eos Tech, 2017)

Para cumplir con los objetivos definidos inicialmente, se tuvieron dos procesos de inducción, uno de manera general con la empresa y otro específicamente en el área de pruebas de validación.

En el primer proceso de inducción general de la empresa se adquirió el conocimiento para manipular los sistemas de calidad, los sistemas de registros de actividades y avances, los sistemas de ingeniería, los sistemas de repositorios, los sistemas de administración de proyectos, esto con el objetivo de registrar de manera correcta cada actividad diaria realizada en la empresa, para poner a disposición toda la información de nuestro trabajo a gerencia, a los jefes de área y a los líderes de proyectos.

Al final de cada inducción se realizaron una serie de evaluaciones de los diferentes cursos proporcionados por los expertos de la empresa con el fin de comprobar el nivel de conocimiento adquirido. Si en cada evaluación los resultados eran positivos la empresa otorgaba un reconocimiento por el buen nivel de conocimientos.

En la figura 5, se muestra un ejemplo de los reconocimientos logrados por el buen desempeño en la capacitación. En las figuras 6 y 6.1 se muestran unas listas de los reconocimientos obtenidos en los cursos de capacitación proporcionados por la empresa *Eos tech*.

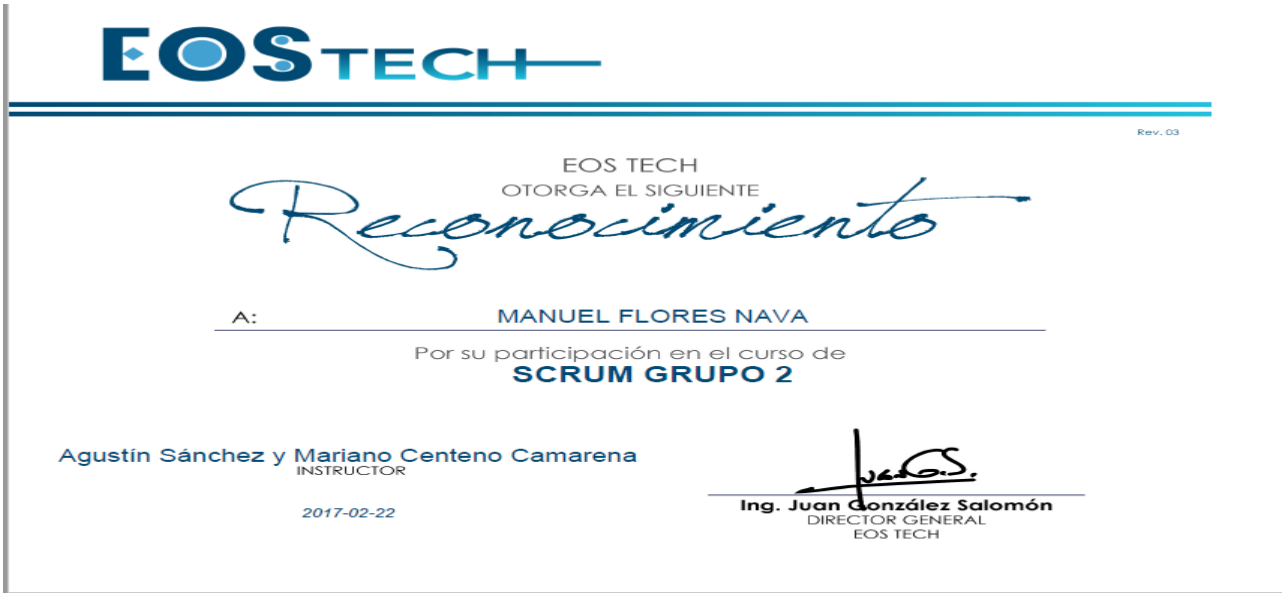
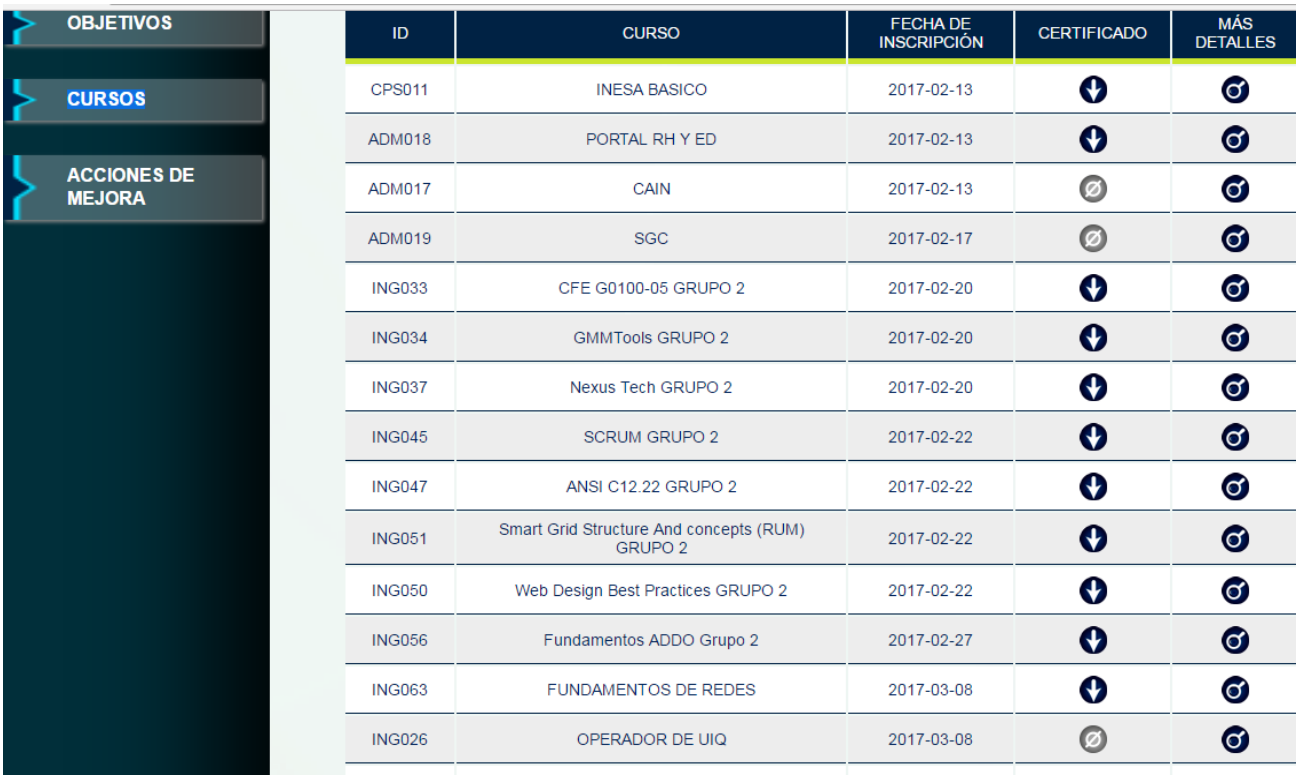


Figura 5. Reconocimiento por el buen desempeño en el curso Scrum (Eos Tech, 2017)



ID	CURSO	FECHA DE INSCRIPCIÓN	CERTIFICADO	MÁS DETALLES
CPS011	INESA BASICO	2017-02-13		
ADM018	PORTAL RH Y ED	2017-02-13		
ADM017	CAIN	2017-02-13		
ADM019	SGC	2017-02-17		
ING033	CFE G0100-05 GRUPO 2	2017-02-20		
ING034	GMMTools GRUPO 2	2017-02-20		
ING037	Nexus Tech GRUPO 2	2017-02-20		
ING045	SCRUM GRUPO 2	2017-02-22		
ING047	ANSI C12.22 GRUPO 2	2017-02-22		
ING051	Smart Grid Structure And concepts (RUM) GRUPO 2	2017-02-22		
ING050	Web Design Best Practices GRUPO 2	2017-02-22		
ING056	Fundamentos ADDO Grupo 2	2017-02-27		
ING063	FUNDAMENTOS DE REDES	2017-03-08		
ING026	OPERADOR DE UIQ	2017-03-08		

Figura 6. lista de cursos con sus respectivos reconocimientos (Eos Tech, 2017)

ING047	ANSI C12.22 GRUPO 2	2017-02-22		
ING051	Smart Grid Structure And concepts (RUM) GRUPO 2	2017-02-22		
ING050	Web Design Best Practices GRUPO 2	2017-02-22		
ING056	Fundamentos ADDO Grupo 2	2017-02-27		
ING063	FUNDAMENTOS DE REDES	2017-03-08		
ING026	OPERADOR DE UIQ	2017-03-08		
SGC025	5 Ss	2017-04-10		
ADM029	ADMINISTRACIÓN DE TIEMPO EN SISTEMA VALIDACIÓN DE PRODUCTO	2017-04-13		
ADM032	Curso de Innovación	2017-05-26		

Figura 6.1 lista de cursos con sus respectivos reconocimientos (Eos Tech, 2017)

Estos reconocimientos son parte de los esfuerzos por lograr adaptarse a los procesos que se realizan en la empresa, estas actividades permitieron analizar y realizar propuestas de mejora en sus procesos y en sus actividades. También fundamentan todas las decisiones tomadas durante el proceso de desarrollo del sistema HES.

A hora bien, en la segunda inducción fue específicamente en el área de pruebas de validación, esta inducción tuvo dos objetivos. el primero de ellos fue conocer los procesos de pruebas que aplica el área, así como su interacción con los sistemas de calidad de la empresa, las normas y reglas a seguir como la documentación y formatos que se utilizan para reportar las pruebas. En la figura 7 se observan las actividades realizadas en la inducción.

Cada actividad realizada en la inducción en el área de pruebas se entregaron reportes al líder del proyecto dando opiniones personales fundamentándolo con estándares investigados.

					Project Owner:	Head End y MDM	
					Project Start Date:	13/mar/2017 (lunes)	
					Display Week:	1	
Tarea	Propietario	Tiempo estimado	Fecha inicio	Fecha fin	Entregable		
Familiarización con conceptos y proceso de pruebas	Mauel Flores Nava	1	13/03/2017	13/03/2017	Reporte de actividad	100%	
Familiarización con documentos	Mauel Flores Nava	1	14/03/2017	14/03/2017	Reporte de actividad	100%	
Iterpretación de requerimientos	Mauel Flores Nava	2	15/03/2017	16/03/2017	Reporte de actividad	100%	
Integración del proceso de pruebas dentro de los sistemas propietarios (INESA / CAIN / SGC)	Mauel Flores Nava	1	17/03/2017	17/03/2017	Reporte de actividad	100%	
Planeación de tren de pruebas	Mauel Flores Nava	3	20/03/2017	22/03/2017	Reporte de actividad	100%	
Proceso de ejecución de tren de pruebas	Mauel Flores Nava	3	23/03/2017	27/03/2017	Reporte de actividad	100%	
Creación de registros de pruebas	Mauel Flores Nava	4	28/03/2017	31/03/2017	Reporte de actividad	100%	

Tabla 7. actividades en área de pruebas de validación (Eos Tech, 2017)

El otro objetivo fue analizar todo el proceso para proponer mejoras, debido a que aparte del proyecto general que es la evaluación del desarrollo del *HES*, también se desarrolló un proyecto de manera individual en la empresa que fue enfocado al aseguramiento de calidad del software.

El producto que se entregó a la empresa, fue un *Handbook* de pruebas estandarizadas, donde se realizan varias propuestas de mejora para los procesos de pruebas de verificación y validación basados en estándares de calidad, gracias a las diversas reuniones que se realizaron con los jefes de áreas interesadas en el proyecto H*andbook*, el trabajo fue estructurado en base a las necesidades del área de validación y del área de ingeniería (específicamente en desarrollo del software).

La propuesta realizada fue bien recibido por gerencia, por el jefe de área de pruebas y por el jefe de ingeniería. En la figura 8 se muestra un formato donde los jefes firmaron por la aceptación del *Handbook.*


LA PRESENTE:

A QUIEN CORRESPONDA

Fecha: 01/08/2017

El que suscribe Ing. Manuel Flores Nava residente de Maestría en la empresa Eos Tech, pone a disposición un handbook para pruebas de software, esto con la finalidad de apoyar al tester como una herramienta que permita ampliar el conocimiento de las técnicas de pruebas, así como estándares útiles para el área de pruebas de validación, siendo así el área propietario de este handbook.

Sin otro en particular, le agradezco la atención prestada a la presente quedando a sus órdenes para cualquier información.

VoBo

Vo.Bo.

Ing. Mariano Centeno Camarena

(Asesor de Ingeniería)

Vo.Bo.

Ing. Gerardo Antonio Pérez Angulo

(Líder de pruebas de validación)

Ing. Manuel Flores Nava (Residente)

Figura 8. Firma de aceptación de handbook por parte de los jefes de áreas de la empresa (Eos Tech, 2017)

BITÁCORA EN LA EMPRESA EOS TECH

La bitácora de las figuras del 9 al 9.8 se visualizan parte de las actividades hechas durante la estancia en *Eos Tech* Querétaro, estas actividades iniciaron el 2/03/2017 después de haber tenido los cursos de inducción y de realizar diversas reuniones para definir y aclarar diferentes aspectos del proyecto, estas actividades fueron registradas oficialmente en los sistemas de proyectos de la empresa. El proyecto es finalizado el 04/08/2017 con la demostración del sistema y presentación de los trabajos realizados durante los 6 meses de estancia, así como la entregad del *Head End System.*

No. Actividad	Responsable	Actividad	Descripción	Fecha Alta	Fecha Inicio	Fecha Final (estimado)	Tiempo Estimado	Tiempo Real	Fecha Final (real)	Estatus Real
-CN001339 -00000005 -AC000001	Manuel Flores	Modelos para desarrollo de software	forma de administrar un proceso de desarrollo de software	27-02-2017	28-02-2017	02-03-2017	20	18	02-03-2017	A tiempo
-CN001339 -00000005 -AC000002	Manuel Flores	Pruebas de caja negra, blanca y gris	descripcion de este prueba	27-02-2017	27-02-2017	27-02-2017	9	0	27-02-2017	A tiempo
-CN001339 -00000005 -AC000003	Manuel Flores	Hibernet	investigaci&#65533;n y an&#65533;lisis de Hiberbet	27-02-2017	02-03-2017	03-03-2017	16	9	03-03-2017	A tiempo
-CN001339 -00000005 -AC000004	Manuel Flores	metodologias de testing	Investigaci&#65533;n y an&#65533;lisis de las metodologias	27-02-2017	06-03-2017	08-03-2017	20	20	08-03-2017	A tiempo
-CN001339 -00000005 -AC000005	Manuel Flores	dise&#65533;o de un Modelo de pruebas	propuesta de modelo de pruebas	27-02-2017	08-03-2017	10-03-2017	25	25	10-03-2017	A tiempo

Figura 9. Bitácora de actividades en Eos tech. (Eos Tech, 2017)

Ú**CN001339-00000005**-AC000006	Manuel Flores	Familiarizaci&#65533;n con conceptos y proceso de pruebas	Conceptos de testing que maneje el &#65533;rea de pruebas de la empresa	13-03-2017	13-03-2017	13-03-2017	9	0	13-03-2017	A tiempo
-CN001339-00000007-AC000007	Manuel Flores	Familiarizaci&#65533;n con conceptos y proceso de pruebas	definiciones que maneje el personal de pruebas	13-03-2017	13-03-2017	13-03-2017	9	9	13-03-2017	A tiempo
-CN001339-00000007-AC000008	Manuel Flores	Familiarizaci&#65533;n con documentos	Formatos, registros, requerimientos y reportes	13-03-2017	14-03-2017	14-03-2017	9	9	14-03-2017	A tiempo
-CN001339-00000007-AC000009	Manuel Flores	Iterpretaci&#65533;n de requerimientos	Como definir los requerimientos	13-03-2017	15-03-2017	16-03-2017	18	18	16-03-2017	A tiempo
-CN001339-00000007-AC000010	Manuel Flores	Integraci&#65533;n del proceso de pruebas dentro de los sistemas propietarios (INESA / CAIN / SGC)	Conocer como van registrando sus pruebas en las plataformas	13-03-2017	17-03-2017	17-03-2017	9	9	17-03-2017	A tiempo

Figura 9.1 Bitácora de actividades en la empresa. (Eos Tech, 2017)

-CN001339 -00000007 -AC000011	Manuel Flores	Planeaci&#65533;n de tren de pruebas	Planificar proceso de pruebas	13-03-2017	20-03-2017	22-03-2017	27	18	22-03-2017	A tiempo
-CN001339 -00000007 -AC000012	Manuel Flores	Proceso de ejecuci&#65533;n de tren de pruebas	Ejecuci&#65533;n de pruebas	13-03-2017	23-03-2017	28-03-2017	27	27	28-03-2017	A tiempo
-CN001339 -00000007 -AC000013	Manuel Flores	Creaci&#65533;n de registros de pruebas	como crear registros de pruebas	13-03-2017	28-03-2017	31-03-2017	36	36	31-03-2017	A tiempo
-CN001339 -00000010 -AC000014	Manuel Flores	pruebas a módulo Encriptacion AES128	Se aplica un tren de pruebas enfocado al modulo	03-04-2017	03-04-2017	04-04-2017	18	18	04-04-2017	A tiempo
-CN001339 -00000010 -AC000015	Manuel Flores	pruebas a modulo CRC	aplicación de un tren de pruebas enfocado al modulo	03-04-2017	05-04-2017	07-04-2017	27	27	07-04-2017	A tiempo
-CN001339 -00000010 -AC000016	Manuel Flores	pruebas a módulo de UserAcces Level	Aplicación de un tren de pruebas enfocado al modulo	03-04-2017	10-04-2017	12-04-2017	27	27	12-04-2017	A tiempo
-CN001339 -00000010 -AC000017	Manuel Flores	pruebas a módulo de Web Services	Aplicación de un tren de pruebas enfocado al modulo	03-04-2017	12-04-2017	18-04-2017	27	27	18-04-2017	A tiempo

Figura 9.2. Bitácora de actividades en la empresa. (Eos Tech, 2017)

Ú **-CN001339** **-00000010** -AC000018	Manuel Flores	Presentación de una propuesta de metodología de pruebas	Exponer una propuesta de una metodología de pruebas a dirección de EOS TECH.	03-04-2017	10-04-2017	18-04-2017	45	5	18-04-2017	A tiempo
-CN001339 **-00000012** -AC000019	Manuel Flores	pruebas a web services	Aplicaci&#65533;n de tren de pruebas segunda iteracion	18-04-2017	18-04-2017	03-05-2017	54	54	03-05-2017	A tiempo
-CN001339 **-00000012** -AC000020	Manuel Flores	Aplicacion de pruebas segunda iteracion user acces level	Aplicacion de tren de pruebas con tomcat, oracle y eclipse	26-04-2017	26-04-2017	28-04-2017	36	36	28-04-2017	A tiempo
-CN001339 **-00000012** -AC000021	Manuel Flores	Propuesta de HandBook	se desarrollara propuesta de manual de procesos de pruebas	04-05-2017	04-05-2017	17-05-2017	90	19	17-05-2017	A tiempo
-CN001339 **-00000012** -AC000022	Manuel Flores	servidor virtual local	Se creara un servidor local de manera virtual para realizar pruebas	04-05-2017	04-05-2017	11-05-2017	72	25	11-05-2017	A tiempo

Figura 9.3. Bitácora de actividades en la empresa. (Eos Tech, 2017)

Ú**CN001339-00000012**-AC000023	Manuel Flores	Presentaci&#65533;n de avances	se presentaran los avances de pruebas y de propuesta para pruebas	04-05-2017	04-05-2017	04-05-2017	5	3	04-05-2017	A tiempo
-CN001339-00000012-AC000024	Manuel Flores	Curso de Tester	Reforzar los conocimientos para la elaboracion de metodologias de pruebas	05-05-2017	05-05-2017	26-05-2017	90	23	26-05-2017	A tiempo
-CN001339-00000012-AC000025	Manuel Flores	configuraci&#65533;n de tomcat con eclipse y pruebas	configuracion de apache tomcat en eclipse neon y pruebas de servicios	12-05-2017	12-05-2017	12-05-2017	9	9	12-05-2017	A tiempo
-CN001339-00000012-AC000026	Manuel Flores	Revisi&#65533;n de E-R y arquitectura del SMCG	An&#65533;lisis de las E-R de la base de datos HES y arquitectura del sistema SMCG	15-05-2017	15-05-2017	15-05-2017	7	7	15-05-2017	A tiempo
-CN001339-00000012-AC000027	Manuel Flores	Pruebas desde apache tomcat	poner a prueba el entorno apache tomcate (para web service y BD)	16-05-2017	16-05-2017	24-05-2017	36	33	24-05-2017	A tiempo
-CN001339-00000012-AC000028	Manuel Flores	an&#65533;lisis y operaci&#65533;n de JIRA y Backlog	Se investiga sobre Backlog y se manipulara el sistema JIRA	17-05-2017	17-05-2017	01-06-2017	100	19	01-06-2017	A tiempo

Figura 9.4. Bitácora de actividades en la empresa. (Eos Tech, 2017)

-CN001339 -00000012 -AC000029	Manuel Flores	propuesta de metodolog&#65533;a de testing	Presentaci&#65533;n de propuesta de la metodolog&#65533;a a gerencia	19-05-2017	19-05-2017	19-05-2017	3	2	19-05-2017	A tiempo
-CN001339 -00000012 -AC000030	Manuel Flores	Desarrollo de la metodologia	Fundamentar la metodolog&#65533;a propuesta	24-05-2017	24-05-2017	01-06-2017	54	14	01-06-2017	A tiempo
-CN001339 -00000013 -AC000031	Manuel Flores	Conteo de inventarios	Conteo de productos en ciertas zonas de la empresa	25-05-2017	25-05-2017	25-05-2017	4	4	25-05-2017	A tiempo
-CN001339 -00000012 -AC000032	Manuel Flores	Documentaci&#65533;n de actividades	Documentaci&#65533;n de actividades de segunda iteracion	26-05-2017	26-05-2017	01-06-2017	36	22	01-06-2017	A tiempo
-CN001339 -00000017 -AC000033	Manuel Flores	Desarrollo de Metodolog&#65533;a de Testing	Desarrollo de Metodolog&#65533;a de Testing	01-06-2017	01-06-2017	30-06-2017	103	34	30-06-2017	A tiempo

Figura 9.5. Bitácora de actividades en la empresa. (Eos Tech, 2017)

-CN001339-00000017-AC000034	Manuel Flores	Desarrollo de Handbook	Desarrollo de Handbook	01-06-2017	01-06-2017	30-06-2017	104	34	30-06-2017	A tiempo
-CN001339-00000017-AC000035	Manuel Flores	Investigaci&#65533;n pruebas unitarias, integracion y de sistema	Conocer los diferentes tipos de pruebas que se pueden aplicar en m&#65533;dulos, integraci&#65533;n de sistema y de sistema compelto	05-06-2017	05-06-2017	12-06-2017	30	18	12-06-2017	A tiempo
-CN001339-00000017-AC000036	Manuel Flores	Investigaci&#65533;n de JUnit	Investigar sobre la herramienta de JUnit y su manipulaci&#65533;n	05-06-2017	05-06-2017	12-06-2017	30	19	12-06-2017	A tiempo
-CN001339-00000017-AC000037	Manuel Flores	Prsentacion de Tesis	exposicion con gerencia y Maestros de ITA	08-06-2017	08-06-2017	08-06-2017	2	2	08-06-2017	A tiempo
-CN001339-00000017-AC000038	Manuel Flores	An&#65533;lisis de estandar de dise&#65533;o java	Revisar el documento de estandar de la empresa sobre dise&#65533;o de programacion en java	13-06-2017	13-06-2017	13-06-2017	9	6	13-06-2017	A tiempo
-CN001339-00000017-AC000039	Manuel Flores	Dise&#65533;o de checklist para an&#65533;lisis est&#65533;tico	en base al est&#65533;ndar de la empresa de dise&#65533;ara un checklist que me permita analisar los c&#65533;digos de m&#65533;dulos desarrollados	13-06-2017	13-06-2017	14-06-2017	18	10	14-06-2017	A tiempo

Figura 9.6. Bitácora de actividades en la empresa. (Eos Tech, 2017)

-CN001339 ؆00000017 -AC000040	Manuel Flores	Investigacion OCL	Todo sobre object Constraint language	15-06-2017	15-06-2017	16-06-2017	15	12	16-06-2017	A tiempo
-CN001339 -00000017 -AC000041	Manuel Flores	Documentacion de pruebas	Se documentan pruebas retomadas del curso de tester	19-06-2017	19-06-2017	20-06-2017	15	14	20-06-2017	A tiempo
-CN001339 -00000017 -AC000042	Manuel Flores	Revisi&#65533;n de checklist de an&#65533;lisis est&#65533;tico para java	se revisa con el l&#65533;der del proyecto el checklist	21-06-2017	21-06-2017	21-06-2017	2	2	21-06-2017	A tiempo
-CN001339 -00000017 -AC000043	Manuel Flores	Configuraci&#65533;n de servidor	Crear entorno para ejecuci&#65533;n de web service	27-06-2017	27-06-2017	29-06-2017	27	13	29-06-2017	A tiempo
-CN001339 -00000017 -AC000044	Manuel Flores	Pruebas Automatizadas	Creaci&#65533;n de programas en java de un cliente para consumir servicio y creador de archivos xml para la web service	28-06-2017	28-06-2017	30-06-2017	27	14	30-06-2017	A tiempo
-CN001339 -00000017 -AC000045	Manuel Flores	Revisi&#65533;n de metodolog&#65533;a SQA	Revisi&#65533;n por parte del Ing. Mariano como experto en los procesos de calidad	30-06-2017	30-06-2017	30-06-2017	2	2	30-06-2017	A tiempo

Figura 9.7. Bitácora de actividades en la empresa. (Eos Tech, 2017)

-CN001339 -00000021 -AC000046	Manuel Flores	Incremento de metodolog&#65533;a SQA	Desarrollo final de la metodolog&#65533;a SQA	03-07-2017	03-07-2017	31-07-2017	90	68	31-07-2017	A tiempo
-CN001339 -00000021 -AC000047	Manuel Flores	HandBook de pruebas	Desarrollo final de HandBook de pruebas	03-07-2017	03-07-2017	31-07-2017	100	88	31-07-2017	A tiempo
-CN001339 -00000021 -AC000048	Manuel Flores	Pruebas automatizadas	Desarrollo de generacion XML autom&#65533;ticas para pruebas	03-07-2017	03-07-2017	21-07-2017	90	19	21-07-2017	A tiempo
-CN001339 -00000021 -AC000049	Manuel Flores	Revisi&#65533;n de handbook	revisi&#65533;n de handbook de pruebas	18-07-2017	18-07-2017	18-07-2017	2	2	18-07-2017	A tiempo
-CN001339 -00000021 -AC000050	Manuel Flores	Configuraci&#65533;n de servidor	preparaci&#65533;n del entorno y configuraci&#65533;n de arranque	21-07-2017	21-07-2017	26-07-2017	20	3	26-07-2017	A tiempo

Figura 9.8. Bitácora de actividades en la empresa. (Eos Tech, 2017)

En las figuras 10 y 10.1 se muestra la planeación del área de proyectos, que llevaba el control del cumplimiento de las actividades a tiempo, en el se puede observar el cumplimiento de evaluación de pruebas a los módulos del Head End System y del desarrollo del *handbook.*

Equipo: Desarrollo de Software

OSTECH

Responsable : Manuel Flores Nava
Fecha de Inicio : 1/1/2017 (domingo)
Semana: 22

Proyecto	Actividad	Desglose	Horas	Comentarios	Entregable	Horas	Comentarios	Entregables	Inicio	Final	% Avance
1		**Segunda iteacion del desarrollo HES**							lun 4/03/17	lun 4/17/17	
1.1	**Aplicación de pruebas a modulos funcionales.**	**Aplicación de pruebas a módulo Encriptacion AES128**	45	Esta actividad contiene más horas pues se inicio desde el 29/03/17	Documentación de resultados de pruebas	27		**Reportes de Resultados**	lun 4/03/17	mié 4/05/17	100%
1..2		Aplicación de pruebas a modulo CRC	45	Inicio el 5/04/17 y su fin fue el 11/04/17		27			jue 4/06/17	lun 4/10/17	100%
1.3		Aplicación de pruebas a módulo de UserAcces Level	9	Los porcentajes se derivan respecto a la semana de iteración		18	Se aplicaron pruebas de Validación de usuarios y conexión cliente servidor		mar 4/11/17	mié 4/12/17	100%
1.4		Aplicación de pruebas a módulo de Web Services	5			18			jue 4/13/17	vie 4/14/17	100%
1.5		Presentación de una propuesta de metodología de pruebas	4		Presentación Power Point	9			lun 4/17/17	lun 4/17/17	100%
1.1	Aplicación de pruebas a modulos funcionales.	Aplicación de pruebas a módulo de UserAcces Level				45	Aplicación de tren de pruebas en Tomcat con Oracle 11G y eclipse neon		mar 4/18/17	vie 4/28/17	100%
1.1		Aplicación de pruebas a módulo de Web Services				54			mar 4/18/17	vie 4/28/17	100%
		Curso de Tester , Creacion de Handbook				36			lun 5/08/17	vie 5/12/17	100%
1.1	**Head End Sistems**	Char Validation Middleware				45			TBD	TBD	100%
1..2		Security Middleware (Code Injection)				45			TBD	TBD	100%
1.3		ORM User Management (Spring Security)				45			TBD	TBD	100%
1.4		On screen Reports				68			TBD	TBD	100%
1.5		UDP Socket				68			TBD	TBD	100%
		SON Device Data Handler							TBD	TBD	100%
		Hibernate ORM Persistence (Operation Data)				72		Reporte de Resultados	TBD	TBD	100%

Figura 10. Planeación de pruebas y Entrega de Handbook. (Eos Tech, 2017)

14	1		Segunda iteacion del desarrollo HES							lun 4/03/17	lun 4/17/17	11
31		Hibernate ORM Persistence (Tasks)	Validation			90				TBD	TBD	100%	#¡VALOR!
32		Hibernate ORM Persistence (Device Data)	Validation			45				TBD	TBD	100%	#¡VALOR!
33		Task Manager Handler	Validation			90				TBD	TBD	100%	#¡VALOR!
34		User Acces Level Management Module	Validation			90				TBD	TBD	100%	#¡VALOR!
35		CIM FTP Web Service	Validation			45				TBD	TBD	1000%	#¡VALOR!
36		GUI (Dashboard)	Validation			297				TBD	TBD	100%	#¡VALOR!
37		Metodología				TBD	Los tiempos sobrantes entre cada prueba de modulos se dedicarán a desarrollar la metodología estandarizada de pruebas de sistemas multiplataforma.	El entregable final será una metodolia estandarizada, que incluye un manual de procesos		lun 5/01/17	vie 8/04/17	100%	70
38		PROPUESTA DE HANDBOOK								lun 5/15/17	vie 8/04/17	100%	60

Figura 10.1. Planeación de pruebas y Entrega de Handbook. (Eos Tech, 2017)

Al termino de las actividades y la entrega del sistema Head End, obtuvimos el visto bueno de gerencia y de los jefes de área, con buenas críticas y con la satisfacción de la empresa, que con mucha emoción se logró con éxito el desarrollo del sistema y la aceptación de la propuesta del *Handbook* de pruebas estandarizadas que involucra parte de la metodología de aseguramiento de calidad.

Una vez cumplido con los objetivos planteados al inicio del proyecto con la empresa se da por terminado la estancia en *Eos tech* Querétaro.

Es importante mencionar que todas las figuras mostradas en este anexo, son pantallas y fotos realizadas a los sistemas de gestión de calidad y a los documentos de la empresa del cual todos los derechos se reservan a *Eos Tech. S.A. de C. V.*

ANEXO 2

En este anexo se presentan los productos logrados en el trabajo de investigación e implementación sobre el proyecto de evaluación de un *Head End System*, esos productos fueron dos artículos presentados en los siguientes congresos internacionales:

- Academia Journal Celaya, Guanajuato, México
- Acdemia Journal Morelia, Michoacan, México

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLOGICO NACIONAL DE MÉXICO

ACADEMIA JOURNALS

CONGRESO INTERNACIONAL DE INVESTIGACIÓN DE ACADEMIA JOURNALS.COM, CELAYA 2017

OTORGAN EL PRESENTE

CERTIFICADO

A

ING. MANUEL FLORES NAVA
M.C. MARÍA GUADALUPE MEDINA BARRERA
M. C. JOSÉ JUAN HERNÁNDEZ MORA
M.S.C. AGUSTÍN SÁNCHEZ ATONAL

POR SU PARTICIPACIÓN CON LA PONENCIA TITULADA

DISEÑO DE UNA METODOLOGÍA DE TESTING EN SCRUM PARA UN SISTEMA INTEGRAL DE TIPO HEAD END ENFOCADO A SMARTCITIES

PUBLICADA EN EL PORTAL DE INTERNET CELAYA.ACADEMIAJOURNALS.COM
VOLUMEN ONLINE CON ISSN 1946-5351 VOL. 9, NO. 6, 2017
E INDIZACIÓN EN FUENTE ACADÉMICA PLUS (EBSCO) Y
LIBRO DIGITAL EBOOK CON ISBN 978-1-939982-32-2, ONLINE.
LA CUAL FUE PRESENTADA EN EL
TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA
LOS DÍAS 8, 9 Y 10 DE NOVIEMBRE DE 2017, CELAYA, GUANAJUATO, MÉXICO.

DR. RAFAEL MORAS
EDITOR, ACADEMIAJOURNALS.COM
PROFESOR DE INGENIERÍA INDUSTRIAL Y ADMINISTRATIVA
ST. MARY´S UNIVERSITY, SAN ANTONIO, TX. EEUU

ING. ALEJANDRO ÁLVAREZ BARCENAS
COORDINADOR GENERAL DEL
CONGRESO INTERNACIONAL DE INVESTIGACIÓN
ACADEMIA JOURNALS, CELAYA 2017

N° 4551

ANAB

Cel1108

Figura 1 Certificado del Congreso Internacional, Academia Journal Celaya

Diseño de una Metodología de *Testing* en *Scrum* para un Sistema Integral de tipo *Head End* enfocado a *Smart cities*

Ing. Manuel Flores Nava[4], M.C. María Guadalupe Medina Barrera[5], M.C. José Juan Hernández Mora[6], M.S.C. Agustín Sánchez Atonal[7]

***Resumen*— El presente artículo describe el diseño de una metodología de *testing* en un desarrollo basado en el framework *Scrum*, este modelo ágil permite modularizar un sistema integral en pequeños componentes de software (*Sprint*). El objetivo es garantizar la calidad de un sistema de tipo *Head End* para *Smart cities*, estructurado bajo la arquitectura Modelo-Vista-Controlador, la arquitectura permite diferenciar la base de datos, de las interfaces y la lógica de aplicación. El *tester* toma ventaja de estos métodos al facilitar la identificación de los módulos a testear, permitiendo la creación de un plan de pruebas por módulo. Para evaluar el sistema, se proponen dos tipos de *tester* (Interno y externo): el interno trabaja del lado del programador, se encarga de verificar cada módulo mediante pruebas objetivas y el externo trabaja del lado del cliente, se encarga de validar a que el sistema completo cumpla las necesidades del cliente mediante pruebas subjetivas.**

Palabras clave—Smartcities, Head End System, Scrum, MVC, Verificación &Validación

Introducción

Las pruebas son de las fases más importantes dentro del desarrollo del software, porque en él se evalúa si el producto desarrollado cumple con la calidad deseada y cumple con las expectativas del cliente, así como también se decide si el producto se encuentra listo para ser entregado al cliente y ponerlo en operación. *Dijkstra* en 1970 afirma que "La prueba de *software* puede ser usada para mostrar la presencia de *bugs,* pero nunca su ausencia". [1]

Algunos modelos de desarrollo ágil de software, le restan importancia a la aplicación de las pruebas debido a los retrasos que estas le pueden ocasionar, pero estas situaciones ponen en riesgo la calidad del software, teniendo como consecuencia un software que no satisfaga las necesidades del cliente.

La aplicación de pruebas no necesariamente debe ser un obstáculo en los tiempos de entrega, ya que se pueden diseñar metodologías que se adapten al proceso de desarrollo ágil de software, con el objetivo de disminuir fallas en el software y aumentar la calidad del producto sin afectar el avance del desarrollo. Para ello es importante que las pruebas sean eficientes, correctas, con un mínimo de tiempo de ejecución y lograr resultados que reflejen el nivel de calidad del sistema para tomar decisiones sobre el producto.

Se toma un caso de prueba de un sistema integral de tipo *Head End*, que ha sido estructurado en modelo-vista-controlador y será desarrollado en el Framework de desarrollo ágil *Scrum*. El producto será implementado en ciudades inteligentes (*Smart cities*), gestionando a los servicios públicos (*Utility*), por lo que es importante evaluar estos tipos de sistemas porque contiene una gran responsabilidad en su operación, la razón es que, en caso de existir alguna falla, puede ocasionar problemas económicos graves a la sociedad y a las empresas que ofrecen los servicios.

¿Qué es Smart cities?

Smart cities es un proyecto para transformar servicios generales en servicios inteligentes, agregando módulos de firmware en las tablas de control de servicios junto con la tecnología inalámbrica, haciendo uso de internet de las cosas (IoT). los gobiernos están decidiendo adoptar la idea de ser una ciudad inteligente para ofrecer mejores servicios, generar economía y mejorar su infraestructura. [2]

---

[4] Ing. Manuel Flores Nava. Egresado del Instituto Tecnológico de Apizaco, estudiante de la Maestría en Sistemas Computacionales del mismo, manuelfloresnava@gmail.com (**autor corresponsal)**

[5] M.C. María Guadalupe Medina Barrera. Catedrática de maestría en Sistemas Computacionales del Instituto Tecnológico de Apizaco, Tlaxcala, México

[6] M.C. José Juan Hernández Mora. Catedrático de maestría en Sistemas Computacionales del Instituto Tecnológico de Apizaco, Tlaxcala, México

[7] M.S.C. Agustín Sánchez Atonal. Egresado de maestría en Sistemas Computacionales del Instituto Tecnológico de Apizaco, Tlaxcala, México

Como parte de la metodología, el *tester* debe realizar un profundo análisis y ser participe con el equipo de desarrollo para conocer los requerimientos del sistema, las decisiones que se han tomado, en donde será implementado, que arquitectura estará basada, que herramientas se usara para el desarrollo y en qué modelo de desarrollo será creado.

Descripción del Método

La metodología que se propone, tiene como objetivo integrarse en el desarrollo *Scrum* y diseñar un plan de pruebas basado en la obtención y evaluación de los módulos funcionales y del sistema completo, así como conseguir un mayor control en sus ejecuciones y resultados. Cabe hacer mención que la metodología implementa dos enfoques para evaluar el producto, uno es la verificación de los módulos y otro la validación del sistema terminado, por lo que podrá funcionar si el modelo de desarrollo permite dividir el sistema en componentes y permita una fase de evaluación final antes de entregar el producto, sino es así, no será posible implementarlo.

La metodología será implementada en un sistema integral de tipo *Head End*, el sistema se presenta como una alternativa de nivel tecnológico en *Smart cities*, para tener la comunicación y el control de los servicios prestados a la ciudadanía, facilitando el trabajo a las compañías responsables a la gestión de *Utility´s* (servicios públicos como el agua, el gas, la electricidad, etc).

¿Qué es un Sistema Integral Head End (HES)?

Un sistema integral gestiona toda la información de forma centralizada, en él se pueden integrar más sistemas para realizar diversas funciones y procesamiento.

El sistema *Head End (HES)* sirve como un puente de comunicación entre la información generada por una red de medidores inteligentes y de un sistema informático que gestiona a la *Utility*, estos sistemas generan las tareas que deben ser ejecutadas por los medidores inteligentes. El *HES* gestiona todo el flujo de datos, para procesar y enviar las tareas correspondientes a la red de medidores inteligentes, la gestión de los datos funciona gracias a un conjunto de operaciones que realiza de manera interna, esas operaciones son claramente identificadas en la figura 1.

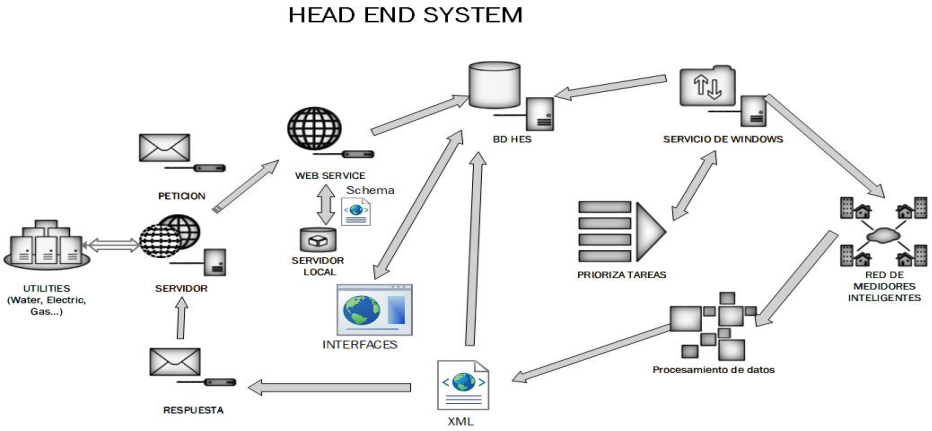


Figura 1. Esquema de funcionamiento de un Sistema integral *Head End*

El funcionamiento del *HES* inicia cuando la *utility* realiza una nueva petición, en ella contiene una o más tareas en un archivo *XML*, esta es recibida por la web service del *HES* que se encarga de realizar la conexión con la *utility*, realiza una comparativa del archivo con su esquema definido en *XML*, para validar que los datos no estén dañados o corrompidos.

Las peticiones validadas son almacenadas en la base de datos del sistema *HES*, para ser consultadas en el momento que se requiera por medio de una interfaz, pero sobre todo para priorizar tareas a ejecutar, ya que existe un módulo de servicio de Windows que monitorea continuamente a la base de datos en caso de existir una tarea, esta toma la tarea con más prioridad y lo envía a la red de medidores para ser ejecutada.

La red de medidores inteligentes genera Paquetes de datos (*Frames*) como respuesta a las peticiones hechas por la *Utility*, la siguiente operación es procesar los datos de respuesta en un archivo *XML* de una manera legible para el usuario, este archivo toma dos rutas una para ser guardado en la base de datos del *HES* y pueda ser

consultado por la interfaz y otro para ser enviado como respuesta a la *utility*. Este funcionamiento es gracias a la arquitectura *MVC*.

El funcionamiento de la arquitectura Modelo-Vista-Controlador (MVC) en el Head End

El sistema *HES* está basado en la arquitectura modelo vista controlador (*MVC*), en la figura 2 se puede observar la arquitectura dividido en tres capas, por lo que el sistema también se puede dividir en tres partes importantes, la primera capa que corresponde el modelo, en el contiene la estructura de datos (base de datos)y la lógica de negocios, la segunda capa que corresponde a las vistas, en el contiene la interfaz, y la tercera capa que corresponde al controlador, en el contiene la lógica de aplicación. El funcionamiento de la arquitectura consiste en una petición del usuario (puede ser inicio de sesión, consulta, etc.), el controlador se encarga de gestionar las peticiones del usuario, realizando un enlace con las capas modelo y vista, ya que no tiene interacción directa con los datos.

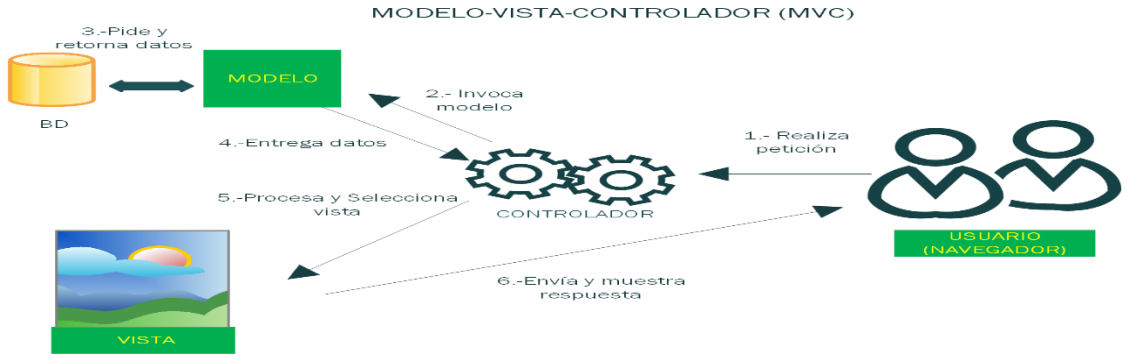


Figura 2 Modelo-Vista-Controlador

El modelo es invocado por el controlador y gestiona los datos realizando diversos mecanismos en su estructura, para que puedan acceder a dichos datos. Una vez ejecutada la acción el modelo entrega los datos de respuesta al controlador y este selecciona una vista para representarlo o mostrarlo desde una interfaz de una manera legible para el usuario.

Tecnologías de desarrollo para el sistema Head End

Las tecnologías de desarrollo que se han propuesto para la realización del sistema, están clasificadas en las tres capas del sistema, cada uno tiene una función en específico que aportan en el desarrollo.

las tecnologías en la capa del modelo son: *Oracle* que se encarga de estructurar los datos, *Maven* aporta la gestión de proyectos java, *JPA* como una Api de persistencia, *Hibernate* es un framework de persistencia de los datos, *spring MVC* proporciona un algoritmo compartido para el procesamiento de solicitudes. Por el lado de la vista las tecnologías son: *HTML 5*, *CSS* y *JSP* brinda la capacidad de construir interfaces web dinámicas, *BOOTSTRAP* es un framework para construir sistemas y sitios web, *JavaScript* y *json* permite la programación orientado a objetos.

En la capa del controlador: se selecciona las tecnologías *Ajax* para la transferencia de datos, *java*, *Oracle Spring mvc* para tener el control de la base de datos y de las interfaces.

Framework Scrum

El desarrollo del sistema integral fue planteado en un framework de desarrollo ágil *Scrum*, se caracteriza por administrar mediante iteraciones, denominadas *sprints*, con una duración de 30 días. [3] En la figura. 3 se muestra el funcionamiento de *Scrum*, este consiste en tener un proyecto que es dividido en un conjunto de iteraciones (*sprint*) que están debidamente clasificadas por orden de prioridad en una lista llamada *Product Backlog*, la lista contiene los requerimientos del cliente, aquí interviene el primer rol llamado *Product Owner* este rol se encarga de proteger los interese del cliente, desarrollando los *Sprints* en la lista.

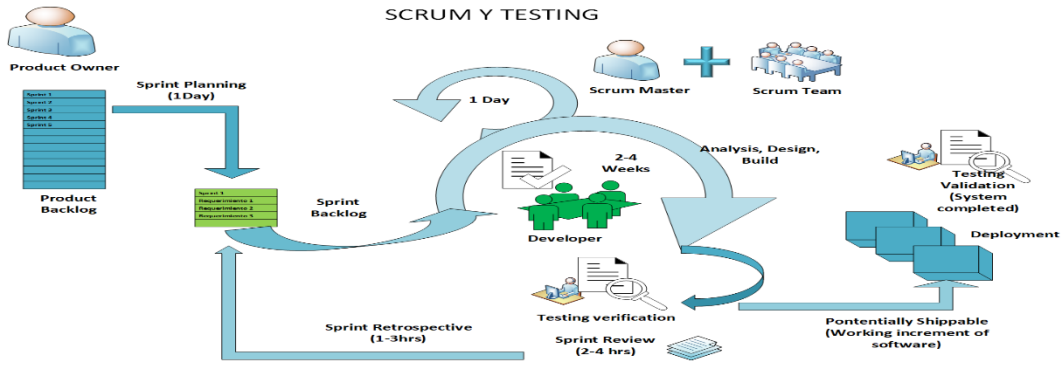


Figura 3. Framework *Scrum*

En la siguiente fase se realiza un *Sprint planning*, en él se planean las actividades que cubren los requerimientos del sprint, las actividades son asignadas al equipo de desarrollo (*Scrum Team*) quienes tienen la responsabilidad de terminarlo en un lapso de tiempo de 2 a 4 semanas. También entra en función otro rol, el *Scrum Master* este rol realiza la función de un líder de proyecto y se encarga de dirigir al equipo de desarrollo, tiene la responsabilidad de solucionar los problemas que pueden obstaculizar el avance del proyecto, mediante reuniones diarias de 15 minutos. Cuando el *sprint* ha sido desarrollado se realiza un *Sprint Review* con el cliente para ser entregado, después se realiza una retrospectiva del *sprint*, se analizan puntos importantes en el proceso de desarrollo para realizar mejoras de trabajo en el siguiente *Sprint*.

Testing de verificación y validación en Scrum

Dentro del esquema de *Scrum* se ha incluido dos fases para evaluar la calidad del software y también para cumplir con las expectativas del cliente, las fases son: la verificación de módulos y la validación del software.

La verificación se refiere al conjunto de tareas que garantizan que el software implementa correctamente una función específica. La validación es un conjunto diferente de tareas que aseguran que el software que se construye sigue los requerimientos del cliente. [4]

las pruebas ágiles son pruebas colaborativas, está involucrado en el proceso a través del diseño, implementación y ejecución del plan de pruebas. Las pruebas ágiles requieren que todo el equipo participe en el proceso de prueba, se requiere mucha comunicación y colaboración. [5]

Para la aplicación de pruebas de verificación y validación al sistema integral *HES* con *Scrum*, se propone dos tipos de *tester*, un interno en el equipo de desarrollo y otro externo en apoyo al cliente como se puede observar en la figura 4, el trabajo de los *tester* es evaluar el proyecto, proporcionar información sobre la calidad del software para que los desarrolladores puedan implementar correcciones y mejoras en las fallas o errores.

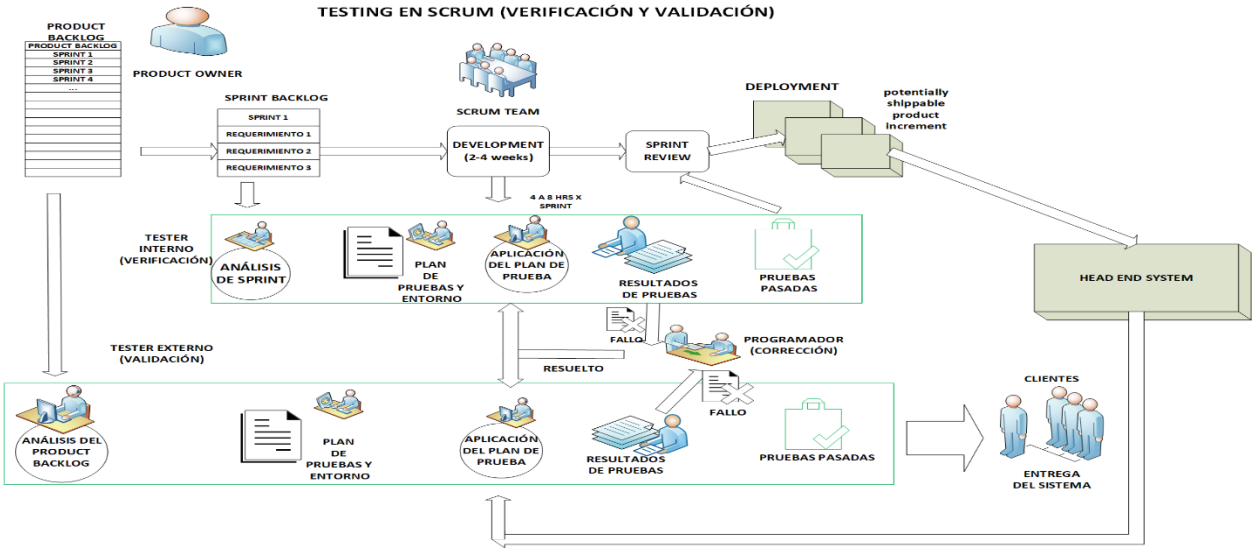


Figura 4. Testing de verificación y validación en Scrum

Tester interno

En la verificación del software, el responsable de ejecutar estas pruebas será un *tester* interno que forma parte del *Scrum Team* y se involucra en sus reuniones diarias para visualizar hacia donde puede dirigir sus pruebas, se encargar de verificar de manera objetiva (su evaluación es que el módulo desarrollado realice su función correctamente). Su intervención es a partir de la selección de un *Sprint* del *product backlog*, para que pueda realizar un análisis en los requerimientos del *sprint*, mientras que el equipo de *Scrum* se encarga de desarrollar dicho *Sprint*, el *tester* interno diseña su plan de pruebas y prepara el entorno donde ejecutará sus técnicas de pruebas, se aprovecha el tiempo para cuando los desarrolladores liberen el sprint, él se encontrará listo para aplicar las pruebas planeadas.

Cuando el plan de pruebas se haya terminado de aplicar, el *tester* obtendrá una serie de resultados, si el resultado es positivo entonces el *sprint* testeado queda liberado para el *sprint review*, pero en caso de que los resultados sean negativos, el *teste*r tiene la obligación de reportar las fallas ocurridas al programador para que pueda ser solucionado lo más pronto posible, todo el proceso que realizará el *tester*, debe ocupar dependiendo de la prueba un tiempo de 4 a 8 horas por *sprint* incluyendo la de regresión.

Tester externo

La segunda perspectiva es un *tester* externo al desarrollo del software, él se enfoca a la validación del software terminado con los *sprint´s* ya integrados, su trabajo es realizar pruebas subjetivas (el *tester* simulara ser un usuario final, pero aplicando también los conocimientos sobre lo que debe ser un software de buena calidad), para ello realizara testeos destructivos con la intención de conocer los alcances y limitantes del software.

El proceso de pruebas del *tester* externo, interviene a partir de la creación del *backlog*, por lo que su plan de pruebas es enfocado en asegurar la implementación de requerimientos en el sistema en terminado y a las características de calidad del software a favor del cliente, la aplicación del plan será después de integrar el sistema y en caso de existir alguna anomalía, se informa al programador para su corrección.

El *tester* externo trabaja junto con el *product owner* que defiende los intereses del cliente por lo que su manera de evaluar es validar que el sistema contiene las siguientes características de calidad según la iso/iec/ieee 29119-1 [6]:

Funcionalidad, el sistema debe tener la capacidad para proporcionar funciones que cumplan con las necesidades bajo condiciones especificadas.

-confiabilidad, el sistema debe tener la capacidad de mantener un nivel de rendimiento cuando se usa bajo condiciones especificadas.

-usabilidad, el sistema debe tener un fácil entendimiento, fácil aprendizaje, fácil uso y atractivo para el usuario y se adapte a las distintas condiciones especificadas.

-eficiencia, el sistema debe tener la capacidad de proporcionar un rendimiento adecuado, en relación con la cantidad de recursos utilizados, bajo condiciones establecidas.

-mantenibilidad, El sistema debe tener la capacidad de ser modificado, estos pueden incluir correcciones, mejoras o adaptación del software a los cambios en el entorno, en los requisitos y especificaciones funcionales.

-portabilidad, El sistema debe tener la capacidad de ser transferido de un entorno a otro.

Planeación

El proceso de pruebas formal, está compuesto al menos por las 5 etapas siguientes: 1. Planeación de pruebas. 2.Diseño de pruebas. 3.implementación de pruebas. 4.Evaluación de criterios de salida. 5.Cierre del proceso. Se definen cuatro niveles de pruebas: pruebas de módulos, pruebas de integración, pruebas del sistema y pruebas de aceptación [7], pero en combinación con *Scrum*, el *sprint review* sustituye a la prueba de aceptación. En cada nivel existen diversas técnicas de pruebas para verificar y validar que los requerimientos solicitados se encuentran dentro del software, es por bien dicho que cada requerimiento del cliente debe contar con al menos una prueba.

Un plan de prueba subraya los tipos de pruebas que se van a realizar y define casos de prueba específicos que se diseñan para garantizar que: se satisfacen todos los requerimientos de funcionamiento, se logran todas las características de comportamiento y de rendimiento, todo el contenido es preciso y se presenta de manera adecuada, y se satisfacen la facilidad de uso. [8]

Mientras que los planes de prueba documentados y los procedimientos caracterizan el éxito de los métodos planificados, los casos de prueba ejecutables definen el éxito de los requisitos y las pruebas en el desarrollo de software ágil. [9] [10]

Resultados

En la tabla 1 Se muestra el sistema modularizado en capas y los sprint que serán probados, también se desglosan los procedimientos de prueba y los casos de prueba según sea los requerimientos, para asignarles algún tipo de prueba. De esta forma se tiene un panorama más accesible, organizada y formal para el *tester*.

CAPAS	SPRINTS	REQUERIMIENTOS	TP (TEST PROCEDURE)	TC (TEST CASE)	TIPOS DE PRUEBAS
Modelo	-Hibernate ORM -Base de datos - HES -XML Persistencia	-Persistencia de datos a una Base de Datos -Tablas -Seguridad -Keys -Persistencia de datos XML	-Mapeo de Objeto-Relacional -Claves -Nivel de acceso -Validación de datos CRUD	-Entidades -Atributos -Asociaciones -Acceso Consulta CRUD	-Análisis Estático -Manual (Consulta) -Caja Negra -Test Unit
Vista	-Mapa de navegación -Interfaces HES	-login -Menú principal -Tablas -formularios -controles -agradable -interactivo -Eficiente -Seguro	-navegación -Login -Seguridad -Validación de datos -Mostrar respuesta de petición -Usabilidad	-Rutas -Utilidad -escalable -Usuario -Password -Caracteres especiales - sesiones -interacción	-Análisis Estático -Prueba de rutas -Caja negra -seguridad -desempeño -contenido -navegación
Controlador	-Web service - Seguridad -validación de datos -Servicio de Windows	-Protocolo de recepción de datos -peticiones -Validación de la petición -Encriptación AES 128	-Web service CRC/checksum -Procesamiento de datos cifrado	-Peticiones -validacion Encriptación Desencriptación	-Desempeño -Caja negra -Caja blanca -Test unit -tipo de datos
General	Sistema HES Terminado	-Funcionalidad -confiabilidad -Usabilidad -Eficiencia -Mantenibilidad -Portabilidad	-Operaciones -Seguridad -Uso - Desempeño -Escalable -Compatible	-Función -Rendimiento -Carga -Acceso -Modificar -ejecución	-Caja Negra -Desempeño -Inyección de código -Nivel de acceso

Tabla 1. Tipos de pruebas a *Sprint´s* de la capa Modelo del sistema *HES*

Formatos

La utilización de formatos para las pruebas se debe realizar en 3 formatos diferentes: 1.- Plan de prueba, el formato describe todos los módulos a probar, los requerimientos, técnicas de prueba, herramientas, criterios de ejecución, el tiempo que tomara realizarlo y responsable. 2.- El Reporte de procedimiento de pruebas, se describen los casos de prueba que contendrá cada procedimiento, el requerimiento que se cubre y criterios de evaluación. 3.-El reporte de resultados contiene las ejecuciones de pruebas que se aplicaron en los módulos con el criterio de evaluación y los resultados de pruebas fallidas o pruebas pasadas.

Cada uno de los documentos tiene un encabezado que muestra información importante como el nombre y rol de los responsables que desarrollaron el módulo y de los que aplicaran las pruebas, el nombre del proyecto, nombre y versión del módulo o del sistema, numero de parte, la fecha de inicio y final de la ejecución y nombre del revisor que avala dicha documentación.

Conclusiones y trabajos a futuro

Gracias a las investigaciones realizadas, se logró diseñar e integrar una metodología de *testing* en el framework Scrum mediante dos tipos de *tester* enfocados a las pruebas de verificación y validación, Permitiendo así el aseguramiento de la calidad del software.

la integración de las pruebas y del *tester* dentro de las actividades de Scrum en el desarrollo del software proporciona beneficios para el proyecto, por ello los *tester* deben ser vistos como aliados para la mejora de los sistemas con la finalidad de enfocar los esfuerzos hacia las vulnerabilidades del sistema y hacia lo que el cliente requiere. Entregar un producto de software de buena calidad que cumpla con los objetivos iniciales, pero sobre todo que logre la confianza y satisfacción del cliente o de los usuarios finales, representa un éxito y buena reputación para cualquier empresa.

Es preciso mencionar que el éxito de la metodología depende de la agilidad de los *tester*, ya que en este tipo de desarrollo se exige una evaluación concreta, rápida y eficiente. Por lo que el *tester* interno sin restar méritos a la habilidad del *tester* externo, debe tener conocimientos en programación por la creación de pruebas automatizadas que se necesiten aplicar en las pruebas unitarias, así como en la integración de los módulos desarrollados, además debe diseñar un plan de pruebas eficiente que no afecte los tiempos de desarrollo, por lo que tiene una gran responsabilidad en sus pruebas.

El *tester* externo al ser subjetivo en su aplicación de pruebas a favor del cliente aporta una visualización diferente al evaluar la calidad del software tocando debilidades no encontradas por el interno. El enfoque de evaluar el sistema completo nos asegura de disminuir las fallas cuando el sistema sea implementado.

Es así que los dos tipos de *tester* tienen tanta importancia dentro de la aplicación de pruebas por las diferentes maneras de evaluar, pero se debe esperar los resultados en la implementación de la metodología, ya que el tiempo que abarque en la ejecución, los resultados de las pruebas y la eficiencia en detectar las fallas, definirá el éxito de la metodología propuesta.

El trabajo a fututo que se realizara próximamente, es implementar y ejecutar la metodología y registrar los resultados positivos o negativos arrojados por las pruebas aplicadas al sistema integral *Head End*,

Referencias

[1] E. W. Dijkstra, “Notes on Structures Programming”,Technical University Eindhoven, The Netherlands, departamento de Matemáticas, Technical Report 70-WSK-03 1970, pp. 15-64 [en línea]. Disponible en: https://www.cs.utexas.edu/users/EWD/ewd02xx/ EWD249.PDF
[2] Vasquez, C., & Estrada, E. (s.f.). Towards the preparation of the Guadalajara’s. Obtenido de IEEE Smart Cities: https://smartcities.ieee.org/images/files/pdf/whitepapermtx_v8.pdf
[3] Canós, J. H., Letelier, P., & Penadés, M. C. (2012). Métodologías Ágiles en el Desarrollo de Software. Obtenido de Universidad Politécnica de Valencia: http://roa.ult.edu.cu/bitstream/123456789/476/1/TodoAgil.pdf
[4] Boehm, B. (1981). Software Engineering Economics,. Redondo Beach, CA: Prentice Hall.
[5] Myers, J. G., Sandler, C., & Badgett, T. (2012). The Art Of Software Testing. Hoboken, New Jersey: WILEY, John wiley & Sons, Inc.
[6] ISO/IEC/IEEE International Standard - Software and systems engineering --Software testing --Part 1:Concepts and definitions," in *ISO/IEC/IEEE 29119-1:2013(E)* , vol., no., pp.1-64, Sept. 1 2013
[7] Villareal, D. D., Gamboa, S. S., & Gómez, F. L. (2015). Estudio sobre realización y documentación de pruebas software. Obtenido de MASKANA: https://publicaciones.ucuenca.edu.ec/ojs/index.php/maskana/article/view/717/634
[8] Pressman, R. (2010). Software Engineering. A Practitioner´s Approach. New York, NY: Mc Graw Hill.
[9] Subhas, C. M., Vinod, K., & Kumar, U. (s.f.). Success Factors of Agile Software Development. Obtenido de Cite Seer X: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.1624&rep=rep1&type=pdf
[10] B. Boehm and R. Turner. 2003. Observations on Balancing Discipline and Agility. Proceedings of the Agile Development Conference(ADC’03). June. Salt Lake City, Utah, USA.

ACADEMIA JOURNALS
OPUS PRO SCIENTIA ET STUDIUM

CEPES Panamericano
CENTRO PANAMERICANO DE ESTUDIOS SUPERIORES

0596

CONGRESO INTERNACIONAL DE INVESTIGACIÓN
DE ACADEMIA JOURNALS MORELIA 2018
ACADEMIAJOURNALS.COM

CERTIFICADO

OTORGADO A

ING. MANUEL FLORES NAVA
M.C. MARÍA GUADALUPE MEDINA BARRERA
M.C. JOSÉ JUAN HERNÁNDEZ MORA
M.S.C. AGUSTÍN SÁNCHEZ ATONAL

POR SU ARTÍCULO INTITULADO

PROCESO DE EVALUACIÓN DE CALIDAD DE UN HEAD END SYSTEM PARA UNA RED DE MEDIDORES INTELIGENTES, BASADO EN LOS REQUERIMIENTOS DEL CLIENTE Y EN LAS CARACTERÍSTICAS DE CALIDAD DE LA ISO/IEC FDIS 9126

EL CUAL FUE PRESENTADO EN EL CONGRESO DESARROLLADO DEL 16 AL 18 DE MAYO DE 2018 EN MORELIA, MICHOACÁN, MÉXICO Y PUBLICADO EN EL PORTAL DE INTERNET *ACADEMIAJOURNALS.COM*, CON ISSN 1946-5351 ONLINE, VOL. 10#3, 2018 Y EN EL LIBRO ELECTRÓNICO ONLINE TITULADO *COMPENDIO DE INVESTIGACIÓN MORELIA 2018* CON ISBN 978-1-939982-36-0

RAFAEL MORAS, PH.D. P.E
EDITOR, ACADEMIA JOURNALS

RECTOR ALDO EMILIO TELLO CARRILLO
CENTRO PANAMERICANO DE ESTUDIOS SUPERIORES

Artículo More136

Instituto de Ciencia, Tecnología e Innovación
Gobierno del Estado de Michoacán

Universidad Nova Spania

Figura 2. Certificado del Congreso Internacional, Academia Journal Morelia

Proceso de evaluación de calidad de un Head End System para una red de medidores inteligentes, basado en los requerimientos del cliente y en las características de calidad de la ISO/IEC FDIS 9126

Ing. Manuel Flores Nava[8], M.C. María Guadalupe Medina Barrera [9],
M.C. José Juan Hernández Mora [10] y M.S.C. Agustín Sánchez Atonal [11]

***Resumen*— El presente artículo expone los resultados que se obtuvieron del proceso para evaluar la calidad de un Head End System para una red de medidores inteligentes, el proceso contiene las siguientes fases: análisis de requerimientos, diseño del plan de pruebas, configuración de herramientas, ejecución y resultados del plan de pruebas, evaluar los criterios de resultados y validación del producto. La evaluación se realizó mediante dos enfoques diferentes, en el primer enfoque se diseñó un plan de pruebas por cada módulo funcional del sistema, para comprobar la implementación de los requerimientos, en la segunda fase se evaluó el sistema terminado basado en las características de calidad interna y externa de la *ISO/IEC 9126 Information technology Software product quality*. El objetivo fue identificar y notificar al programador los errores encontrados para su pronta corrección, las pruebas que se aplicaron en la evaluación fueron de manera estática (*checklist*) y dinámica (*testing*).**

***Palabras clave*— Head End system, calidad, proceso de pruebas, ISO/IEC FDIS 9126, módulos funcionales**

Introducción

Para evaluar la calidad de un software es de vital importancia tener en claro el significado del concepto calidad, existen varias definiciones, por lo que se hace una pequeña recopilación de los que más se adaptan a la propuesta de este artículo, las definiciones son las siguientes:

La Organización Internacional de Normalización (1986) define la calidad como: "la totalidad de las características de un producto o servicio que tienen que ver con su capacidad de satisfacer necesidades específicas o implícitas", la ISO 9000 (2000) lo define como “grado en el que un conjunto de características inherentes cumple con los requisitos”, la IEEE (1998) define la calidad como "el grado en que un sistema, componente o proceso cumple con los requisitos específicos y las necesidades o expectativas del cliente o del usuario".

Ahora bien, sobre la calidad de software Pressman (2010) lo define como: “Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan”. por lo que el proceso que se diseñó para aplicar las pruebas y evaluar la calidad de un sistema se enfoca en lograr un producto que cumpla con las necesidades y la satisfacción del cliente.

En la evaluación por módulo, los requerimientos impondrán los criterios de evaluación, con el objetivo de comprobar que han sido implementados en el sistema, como lo mencionan los autores (león, N. E., Pinto, N., & Gómez, L. C., 2011) “la calidad es un tema subjetivo dependiente del nivel de satisfacción del usuario sienta frente al producto utilizado”.

Para conocer la calidad de un software, se deben realizar un proceso de pruebas que nos permita evaluar el correcto funcionamiento y eficiencia del software. Las actividades de un proceso de pruebas según International Software Testing Qualifications Board deben cumplir con lo siguiente: planificar y controlar, seleccionar las condiciones de la prueba, diseñar y ejecutar casos de prueba, comprobar los resultados, evaluar los criterios de resultados, elaborar informes del proceso y la aplicación objeto de las pruebas, incluyendo bitácoras de experiencia. (ISTQB, 2013).

No siempre se logra detectar los errores que contiene un software durante el proceso de desarrollo debido a que

[8] Ing. Manuel Flores Nava. Egresado del Instituto Tecnológico de Apizaco, estudiante de la Maestría en Sistemas Computacionales del mismo, manuelfloresnava@gmail.com (**autor corresponsal**)
[9] M.C. María Guadalupe Medina Barrera. Catedrática de maestría en Sistemas Computacionales del Instituto Tecnológico de Apizaco, Tlaxcala, México
[10] M.C. José Juan Hernández Mora. Catedrático de maestría en Sistemas Computacionales del Instituto Tecnológico de Apizaco, Tlaxcala, México
[11] M.S.C. Agustín Sánchez Atonal. Egresado de maestría en Sistemas Computacionales del Instituto Tecnológico de Apizaco, Tlaxcala, México

algunos errores aparecen en funcionamiento con el usurario, Dijkstra en 1970 afirma que “La prueba de software puede ser usada para mostrar la presencia de *bugs*, pero nunca su ausencia”. por lo que es importante emplear un plan de pruebas eficiente para prevenir todos los errores posibles.

El caso de prueba que se presenta en este artículo es sobre un Head End system (HES), funciona como un puente de comunicación entre la información generada por una red de medidores inteligentes y gestión de un sistema informático de servicios públicos (utility), estos sistemas generan las tareas que deben ser ejecutadas por los medidores inteligentes. El HES gestiona todo el flujo de datos, para procesar y enviar las tareas correspondientes a la red de medidores inteligentes, la gestión de los datos funciona gracias a un conjunto de operaciones que realiza de manera interna. (Flores, M., Medina, M. G., Hernández, J. J. & Sánchez, A., 2017).

Proceso para la aplicación de pruebas

El proceso para realizar la evaluación del sistema se basa en los requerimientos del cliente, así como los requisitos técnicos que debe cumplir cada módulo funcional del sistema, los responsables en evaluar el sistema son dos tipos de tester, uno es el interno que forma parte del equipo de desarrollo, que evalúa cada módulo funcional basado en requerimientos del cliente y el externo que evalúa el sistema terminado basado en la ISO/IEC FDIS 9126-2001 para calidad interna y externa del sistema terminado.

Antes de comenzar el proceso de aplicación de pruebas, los programadores deben realizar una solicitud de pruebas al equipo de testers para evaluar un módulo funcional o en otro caso el sistema completo, los tester deben analizar que los documentos y el código fuente han sido proporcionados de manera correcta por los programadores, en caso contrario deber ser rechazado dicha solicitud.

En la figura 1 se puede observar un diagrama de flujo que describe las fases importantes del proceso de aplicación de pruebas. Para llevar a cabo la evaluación de un módulo del sistema, antes debe existir lo siguiente:

- Análisis de requerimientos: Cuando ya se tiene la documentación completa del módulo o del sistema a evaluar, inmediatamente el equipo de testers realiza un análisis a los requerimientos técnicos del desarrollador, para estructurarlo con los requerimientos del cliente como un objetivo que debe cumplir el software a evaluar.
- Diseño del plan de pruebas: se diseñan los criterios de evaluación, las métricas, así como las técnicas de pruebas que comprobarán los criterios basados en el cumplimiento de los requerimientos y del buen funcionamiento del módulo.
- Configuración de herramientas: se seleccionan un conjunto de herramientas que permitan ejecutar cada una de las pruebas, también se debe configurar y preparar el entorno para ejecutar las herramientas.
- Ejecución y resultados del plan de pruebas: Se realiza la ejecución del plan de pruebas que se diseñó de manera especial para el módulo o para el sistema completo a evaluar.
- Evaluar los criterios de resultados: se realiza una comparación de los resultados con los criterios de evaluación (requerimientos) para la toma de decisiones, si los resultados no coinciden con los criterios, entonces las pruebas son identificadas como fallidas y se toma la decisión de notificar los fallos al programador para su pronta corrección. Cuando los errores han sido corregidos, se vuelve aplicar el plan de pruebas diseñadas con anterioridad, estas pruebas aplicadas son nombradas como pruebas de regresión.
- Validación del producto: se obtienen los resultados de las pruebas, en caso de que las pruebas han sido positivas entonces el producto se libera y se cierra el proceso para recibir una nueva solicitud de pruebas de un nuevo módulo funcional, pero en caso de haber terminado con las iteraciones y módulos funcionales, se realiza una carta de liberación para la producción del Software (o distribución al cliente).

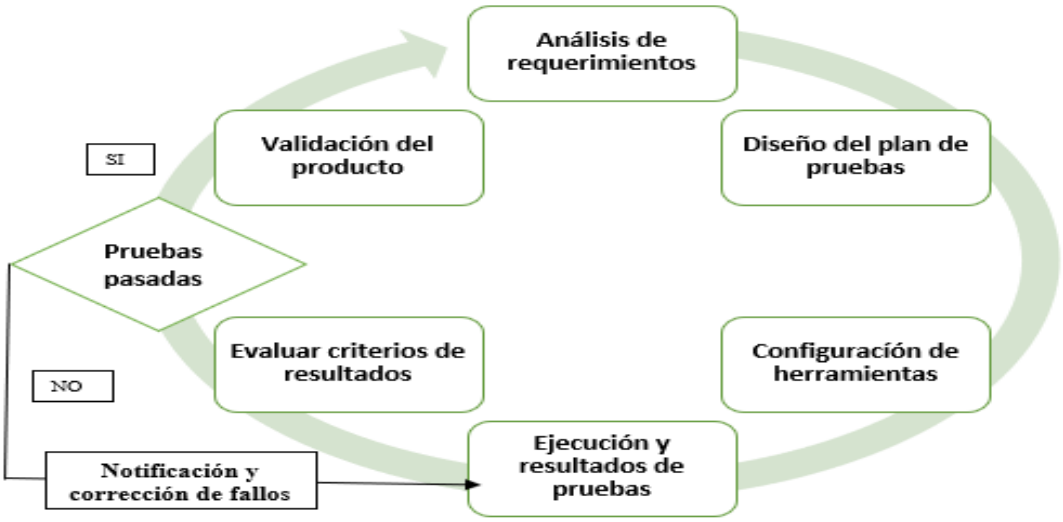


Figura 1.- Ciclo del proceso de diseño y aplicación de pruebas

Pruebas en base a las características de calidad interna y externa de la ISO/IEC FDIS 9126

En el libro de ingeniería de software novena edición Sommerville (2011) menciona que "La calidad subjetiva de un sistema de software se basa principalmente en sus características no funcionales", por lo que se ha implementado dentro del proceso de evaluación de un sistema terminado las características de la calidad interna y externa ISO/IEC FDIS 9126.

El estándar ISO/IEC FDIS 9126, contiene cuatro documentos que describen las características de calidad en el software: Parte 1. Ingeniería de software-Calidad del producto-Modelo de calidad (ISO/IEC 9126-1,2001), Parte 2. Ingeniería de software-Calidad del producto-Métricas externas (ISO/IEC 9126-2,2002), Parte 3. Ingeniería de software-Calidad del producto-Métricas internas (ISO/IEC 9126-3,2002), Parte 4. Ingeniería de software-Calidad del producto-Métricas de calidad en el uso (ISO/IEC 9126-1,2002).

Para la evaluación del sistema terminado se basó en la calidad interna y externa del estandar, que identifica la calidad de un producto de software por medio de seis características que se pueden observar en la figura 2, de las cuales cada uno de ellas se subdivide en subcaracteristicas que se relacionan entre sí. "Cada subcaracterística se describe con más detalle mediante atributos de calidad externos e internos apropiados que se pueden medir mediante métricas específicas" (Al-Kilida, H., Cox, K & Kitchenham, B, 2005).

Dentro del proceso de aplicación de pruebas que se propuso para evaluar un sistema terminado, se seleccionó las características y sub características de calidad interna y externa que contiene el estándar ISO/IEC FDIS 9126, para obtener el grado que calidad que contiene en las características no funcionales y funcionales del sistema.

El encargado de evaluar el sistema terminado es el tester externo al desarrollo del software, por lo que la aplicación de las pruebas fue de manera subjetiva a favor del usuario final, las pruebas se realizaron de manera estáticas, con el apoyo de checklist diseñadas para identificar las características del estándar mencionado con anterioridad y con pruebas dinámicas con técnicas de caja negra.

Figura 2.- Diagrama de la estructura de calidad interna y externa de ISO/IEC FDIS 9126

Caso de prueba: evaluación del *Head End system*

El *Head End System* (HES) será implementado para gestionar una red de medidores inteligentes y por sistemas *utility*, por lo que realizará una serie de tareas e instrucciones para el control de los datos de consumo de energía, es así que se evaluaron cada una de las funciones que debe realizar el sistema.

En la tabla 1 refleja los módulos que componen cada iteración de desarrollo, siendo una solución que cubre los requerimientos del cliente, el tester organiza de esta manera para verificar que los requerimientos realmente se han implementado en el módulo a testear.

Iteraciones	Módulos	Requerimientos a cubrir
1ra iteración	-FTP Web Service -AES 128 Encryptation Module -User Acces Level Management - (CRC Calculator)	*Cifrado de datos para una comunicación segura. * integridad de datos en las comunicaciones con los dispositivos remotos. *Configuración de roles y contraseñas. *Seguridad
2da iteración	-XML Persistence -GUI (Dashboard) -File Creator -CIM FTP Web Service -User Acces Level Management Spring Security	*Garantizar el almacenamiento de la información en la base de datos. *Contemplar mecanismos de redundancia y recuperación de desastres. * almacenamiento de datos sin procesar de los dispositivos automático- *Interfaz de importación de la información de los dispositivos. *parámetros de las tareas automáticas del dispositivo, -Las lecturas de registros. -Demanda. -Las alarmas. - eventos de medidores. -El perfil de carga. -Las tarifas horarias.
3ra iteración 21/06/2017 – 30/07/2017	-Validation Middleware - Spring Security -On screen Reports -UDP Socket -Task Manager -User Acces Level -Web Service -GUI (Dashboard)	*Comunicación bidireccional con dispositivos remotos. *Ejecución automática de tareas (jobs) periódicas *Ejecución automática de programas de tareas (schedules). *Adquisición de datos o envío de instrucciones a-petición del operador. *Consulta de datos a-petición del operador *Recibir y registrar de manera automática las alarmas de un dispositivo *Generación de informes de manera automática o a-petición. *Visualización de parámetros, atributos, estadísticas, y estado.

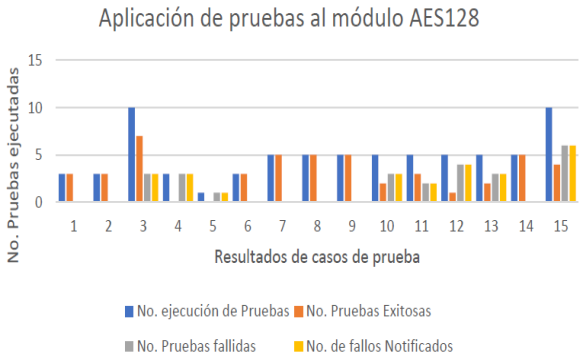
Tabla 1.- módulos desarrollados que satisfacen los requerimientos del cliente

Resultados de módulos del sistema

En el módulo AES 128 se aplicaron pruebas de caja negra, pruebas de seguridad, pruebas de carga y de rendimiento, para obtener resultados que se observan en la figura 3, de la eficiente encriptación y desencriptación a pesar de la cantidad de la información obtenida por los medidores.

En la figura 4 se observa los resultados del módulo CRC, el módulo permite validar la estructura de un frame enviado por los medidores inteligentes, con la intención de asegurar que la información no ha sido corrompida.

En él se aplicaron pruebas caja negra, pruebas de rendimiento y carga, en el manejo de excepciones se utilizó las pruebas de caja blanca.



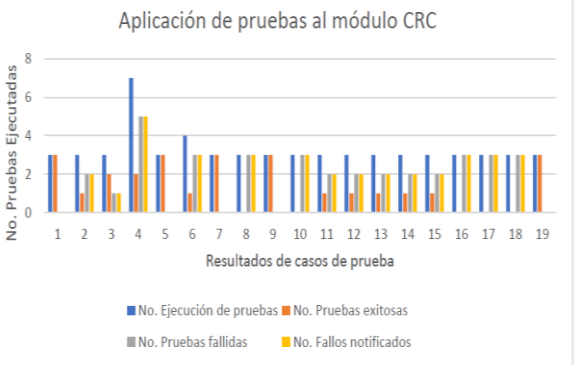
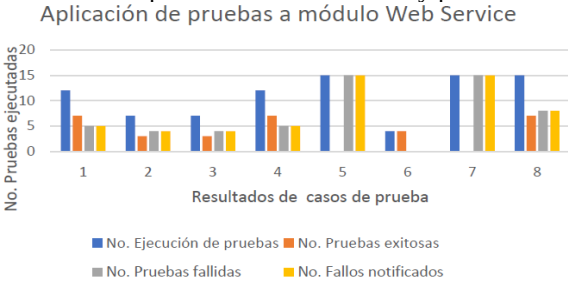


Figura 3.- Resultado de pruebas a módulo aes128 Figura 4.- Resultado de pruebas a módulo CRC

Los resultados de la web service s observa en la figura 5, el módulo es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones, el módulo será montado en un servidor para recibir múltiples peticiones de diferentes sistemas por lo que se tuvieron que generar scripts que simulen las peticiones y scripts que responden a las peticiones (pruebas automáticas), debido a que la web service contiene un esquema XML que las peticiones deben respetar, se tuvieron que aplicar pruebas de caja negra y caja blanca.

Los resultados del módulo de servicio de Windows se observan en la figura 6, en él se realizaron pruebas automatizados y pruebas de caja blanca, para conocer el comportamiento de múltiples hilos (threadpool), para la gestión de tareas priorizadas en el sistema y para controlar las excepciones generadas.



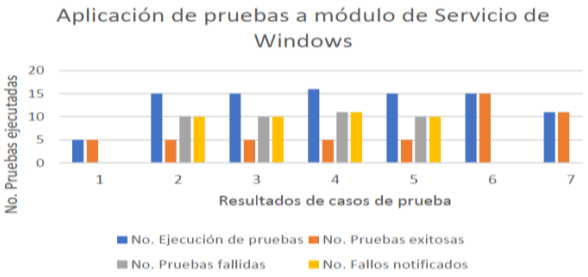
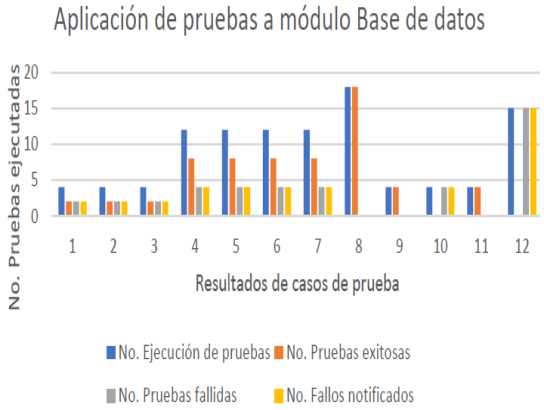


Figura 5.- resultado de pruebas a web service Figura 6.- Resultado de pruebas a servicio Windows

En la figura 7, se observan los resultados de la base de datos, en este módulo se guardarán los datos generados por los medidores inteligentes, por lo que debe contener una estructura de tablas con sus respectivos campos que debe coincidir con la estructura que contiene los medidores, por lo que se aplicaron pruebas de caja negra en base a consultas de tablas y campos, además de pruebas de seguridad e inyección de código.

En el módulo de interfaces se aplicaron pruebas de caja negra para conocer el funcionamiento de las interfaces y la validación de los datos al ingresar en los campos, también se aplicaron pruebas de seguridad como permisos de acceso a ciertas interfaces según sea el usuario también se realizaron la inyección de código en los campos de cada formulario, los resultados se observan en la figura 8.



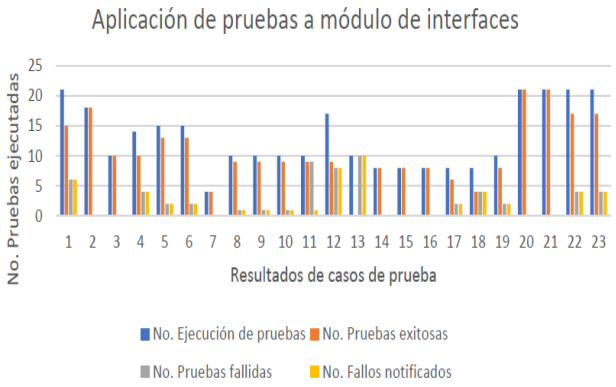


Figura 7.- resultado de pruebas a Base de datos Figura 8.- resultado de pruebas a interfaces

Cada uno de los fallos detectados en cada uno de los módulos, fueron notificados al desarrollador para su pronta solución, en la mayoría de los casos se lograron resolver excepto en la implementación de spring security en la base de datos que contenía 15 pruebas fallidas, por lo que afecto en la integración de los módulos, por ende, los desarrolladores tuvieron que recurrir a soluciones prontas debido al corto tiempo de entrega. Los resultados de cada módulo se ven reflejados en la figura 9.

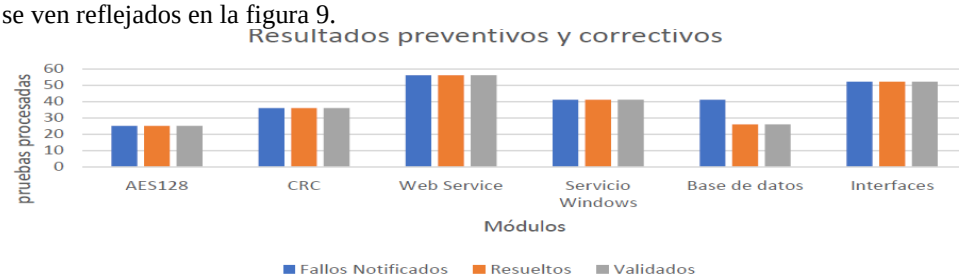


Figura 9.- Resultado de total de casos de fallos notificados, resueltos y validados

Resultados de la evaluación del sistema terminado

El Head End System se ha desarrollado cumpliendo con los requerimientos iniciales, la evaluación del sistema es enfocado a comprobar la implementación de funciones deseadas y la interacción entre módulos. La evaluación también califica las características no funcionales del sistema (colores, utilidad, seguridad, etc.) estas características fueron comprobadas por medio de checklist diseñados para identificar atributos de calidad en el sistema.

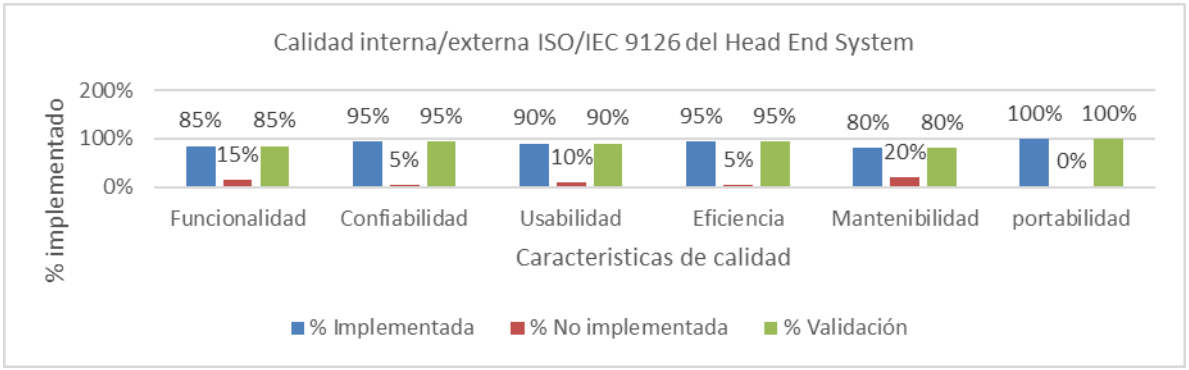


Figura 10.- resultados de evaluación de calidad ISO/IEC FDIS 9126

Las características y sub-características del estándar ISO/IEC FDIS 9126, facilitaron la evaluación del sistema identificando los atributos necesarios, enfocando los esfuerzos en la comprobación del grado de existencia de cada característica de calidad en el sistema. En la figura 10 se muestran los porcentajes generales de cada característica de calidad, fundamentado en los resultados de los checklist aplicados durante la manipulación del sistema.

Conclusiones

La aplicación del proceso de pruebas ha permitido prevenir e identificar los fallos de los módulos funcionales en fases tempranas para su pronta corrección, esta acción aumenta la calidad de los módulos, teniendo así menos fallos en el sistema. La validación de cada módulo ha sido gracias al trabajo en conjunto del tester y programador.

La selección de las técnicas de pruebas fue fundamental para la identificación de los fallos, la más compleja de las pruebas realizadas fueron las de comunicación, rendimiento y carga, debido a la limitación de hardware para ejecutar miles de peticiones y de las múltiples funcionalidades.

El Head End System aún se encuentra limitado para comprobar funcionalidades de comunicación con otros sistemas, esto es debido a que los sistemas externos aún se encuentran en desarrollo, por lo que se tuvieron que realizar simulaciones por medio de un entorno cliente servidor, del cual facilitó la manipulación del sistema y de la ejecución de las pruebas para comprobar y evaluar las funcionalidades del sistema.

El proceso de aplicación de pruebas es parte de una metodología de aseguramiento de calidad que se adapta al desarrollo del sistema, por lo que la calidad es vigilada desde que se construye y se termina el sistema, interactuando constantemente con los desarrolladores y cliente, esto con la intención de facilitar y fortalecer la evaluación del sistema, con criterios avalados por los stakeholders (las personas interesadas en el sistema). Es importante mencionar, que las acciones de corrección de errores son ajenas a la metodología, esa responsabilidad recae sobre el programador, la metodología se limita en prevenir, evaluar, detectar y notificar errores, fallas y defectos.

El sistema fue presentado ante el cliente con una demostración de funcionamiento del sistema, para que exprese su punto de vista, como resultado de ello en esta primera entrega del sistema, ha cumplido con los requerimientos que fueron planteados al equipo encargado para el desarrollo del sistema, logrando así la satisfacción del cliente.

Trabajos futuros

En la planificación general del proyecto de desarrollo del sistema HES, están establecidos futuras implementaciones de módulos funcionales para gestionar medidores inteligentes, no solo de energía eléctrica sino de gas y agua, por lo que sería otra oportunidad para aplicar la metodología y procesos de pruebas al sistema,

Para el periodo en que se estima robustecer el sistema, talvez los sistemas informáticos que van a interactuar con él *HES* ya se encuentren disponibles, esto permitiría realizar pruebas más reales en la comunicación y transferencia de datos.

Referencias

Dijkstra, E. W. (1970). Notes on Structured Programming. Technical University Eindhoven, The Netherlands, department of Mathematics, Technical Report, pp. 15-64. Recuperado de https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF

Flores, M., Medina M. G., Hernández J. J. y Sánchez A. (2017). Diseño de una Metodología de Testing en Scrum para un Sistema Integral de tipo Head End enfocado a Smart cities. Academia journals Celaya 2017, ISSN 1946-5351 online Vol. 9, No. 6,

H. Al-Kilidar, K. Cox y B. Kitchenham (2005). The use and usefulness of the ISO/IEC 9126 quality standard. International Symposium on Empirical Software Engineering, 2005. pp. 7 doi: 10.1109/ISESE.2005.1541821

IEEE std 830-1998 (1998) Recommended Practice for Software Requirements Specifications," in IEEE, doi: 10.1109/IEEESTD.1998.88286

IEEE Std 1074 -1997 (1998) " Standard for Software Life Cycle Processes,". doi:10.1109/IEEESTD.1998.88827

ISTQB (2013), “Certified Tester Foundation Level Syllabus. Released version 2013”, International Software Testing Qualifications Board. Recuperado de http://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html

ISO (1986), "ISO 8402 Quality Vocabulary," de International Organisation for Standardization.

ISO/IEC, "ISO/IEC 9126-1(2001). Software engineering- Product quality- Part 1: Quality model. De International Organization for Standardization

ISO/IEC, "ISO/IEC 9126-2(2002). Software engineering -Product quality- part2: External metrics. De International Organization for Standardization

ISO/IEC, "ISO/IEC 9126-3 (2002). Software engineering -Product quality- part3: Internal metrics. De International Organization for Standardization

ISO/IEC, "ISO/IEC 9126-4(2002). Software engineering -Product quality- part4: Quality In Use metrics. De International Organization for Standardization

Hiyam Al-Kilidar, Cox, K.y Kitchenham, B. (2005). The Use and Usefulness of the ISO/IEC 9126. Quality Standard 0-7803-9508-5/05

López A. M., Cabrera, C. y Valencia, L. E. (2008). Introducción a la calidad de software. Scientia et Technica, Universidad Tecnológica de Pereira. ISSN 0122-1701

León, N. E. Pinto, N. y Gómez, L. C. (2011). Herramienta computacional para la evaluación de calidad de productos software enmarcados en actividades de investigación. Universidad Tecnológica de Pereira. ISSN 0122-1701

Norma Internacional ISO 9000, (2000). Sistemas de gestión de la calidad-Conceptos y vocabulario. De International Organization for Standardization.

Pressman, R. S. (2010). Ingeniería del software, un enfoque práctico (7a. ed.). D.F., México. McGraw Hill latinoamericana editores. ISBN: 978-607-15-0314-5

Sommerville I. (2011). Ingeniería del software (9a. ed.). Edo. México, México. Pearson Education. ISBN: 978-607-32-0603-7.