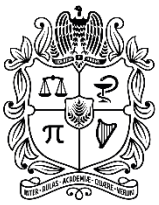


UNIVERSIDAD
NACIONAL
DE COLOMBIA

Representación de las mejores prácticas de verificación de sistemas de software en el núcleo de la Esencia de *Semat*

A *Semat*-Essence-kernel-based representation of software system verification best practices

Carlos Puerto García

Universidad Nacional de Colombia

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Medellín, Colombia

2019

Representación de las mejores prácticas de verificación de sistemas de software en el núcleo de la Esencia de *Semat*

A *Semat*-Essence-kernel-based representation of software system verification best practices

Carlos Puerto García

Tesis de investigación presentada como requisito parcial para optar al título de:

Magíster en Ingeniería–Ingeniería de Sistemas

Director:

Ph.D. Carlos Mario Zapata Jaramillo

Codirector:

Ph.D. Albeiro Espinosa Bedoya

Línea de Investigación:

Ingeniería de Sistemas

Grupo de Investigación:

Lenguajes computacionales

Universidad Nacional de Colombia

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Medellín, Colombia

2019

Dedicatoria

A mis padres Rita Leonor García y Carlos Julio Puerto Flórez, por darme la vida, la libertad de tomar decisiones y ser responsable de mis actos, por darme confianza en mí mismo para salir adelante con los proyectos que me propongo realizar, por ser quienes son y por apoyarme en todo momento.

A mi abuelita Concha por su amor.

A mi hermano gemelo Camilo Puerto García, compañero del alma, de viaje, de aventuras, hasta siempre, donde quiera que estés, te dedico este trabajo que es la vida misma.

A mis hermanos Harold, Mónica y Andrés, quienes me han acompañado por el camino de la vida.

A Lina Marcela Rodas Fernández, quien me acompañó durante el empeño, esfuerzos, perseverancia invertida en el proceso de realización de la Maestría.

Agradecimientos

A Dios por darme la fortaleza y mostrarme el camino, a la Universidad Nacional de Colombia, por acogerme y brindarme los medios de conocimiento e investigación necesarios para la producción de esta Tesis de Maestría, al Doctor Carlos Mario Zapata Jaramillo por introducirme en el mundo de *Semat* y por enseñarme a escribir, al Doctor Albeiro Espinosa Bedoya por introducirme en el mundo de la verificación y la validación y por ser la persona que es, a los dos gracias por su acompañamiento y paciencia en el proceso de enseñanza-aprendizaje en la investigación, para lograr que esta Tesis de Maestría sea una realidad y adquiera dimensión y el nivel adecuados; a mis compañeros de posgrado que me animaron a salir avante con la Tesis, a Paola Andrea Noreña, Maria Clara Gómez Álvarez, Juan Ricardo Cogollo, Claudia Durango, Grissa Maturana, Antony Henao. A todos ellos les auguro éxitos y el debido reconocimiento institucional. A Wasif Afzal, Lisa Crispin y Janet Gregory por animarme y orientarme en el comienzo de esta investigación.

Resumen

La verificación de sistemas de software (VSS) es un proceso fundamental que permite evaluar si los requisitos que se establecen son correctos, construyendo así software con calidad. Existen varias técnicas de VSS donde se utilizan prácticas en común, las cuales se pueden llamar mejores prácticas de VSS; unas se representan gráficamente y otras se expresan en lenguaje natural. En ambas propuestas se evidencian diferencias en sus elementos principales, en las técnicas que se representan gráficamente los símbolos que se utilizan en el modelo de representación son distintos y en las técnicas que se expresan en lenguaje natural no existe representación gráfica. Aunque en los dos casos algunos elementos son similares, no se establecen elementos universales y en ninguna de estas propuestas hay un marco de trabajo estándar donde se puedan capturar sus elementos comunes. La Esencia de *Semat* (Teoría y Método de la Ingeniería de Software, por sus siglas en inglés) es un estándar donde se define un lenguaje universal con un conjunto de elementos ampliamente aceptados en la ingeniería de software (IS). En el núcleo de *Semat,* los elementos comunes de las prácticas de IS se pueden capturar, adaptar y representar. Por ello, en esta Tesis de Maestría se propone la representación de las mejores prácticas de VSS en el núcleo de la Esencia de *Semat*. De esta forma, se pueden representar los elementos de las mejores prácticas de VSS, para implementarlas cuando el contexto del proyecto de software lo requiera. Esto puede apoyar el proceso de VSS para que se desarrolle un producto de acuerdo con los requisitos establecidos.

Palabras clave: verificación, sistemas, software, técnicas, prácticas, requisitos, calidad, *Semat*, representación.

Abstract

Software System verification (SSV) is a crucial process allowing for assessing whether the established requirements are correct or not. SSV is intended to build software with quality. Common practices are used in some SSV techniques. Some of the so-called SSV best practices are graphically represented and other ones are expressed in natural language. Such representations have their main elements: Graphical representations always have different symbols in the representation model and natural-language-based representations have no graphic representation. Even though in both cases some elements are similar, universal elements are still missed. Also, all proposals lack a standard framework with common elements. Semat (Software Engineering Method and Theory) Essence is a standard with a universal language defined and a set of elements widely accepted in Software Engineering (SE). In the Semat kernel, SE practice common elements can be captured, adapted, and represented. Consequently, in this M.Sc. Thesis we propose the identification and representation of system and software verification best practices in the Semat Essence kernel. Hence, SSV best practice processes can be represented in order to be implemented when the context of the software project requires them. Such practices are intended to support the SSV process for developing a product according to the established requirements.

Keywords: verification, systems, software, techniques, practices, requirements, quality, Semat, representation.

Contenido

Lista de Figuras

Lista de tablas

Lista de Símbolos y abreviaturas

Abreviaturas

Abreviatura	Término
IS	Ingenería de Software
VSS	Verificación de Sistemas de Software
OMG	Object Management Group
IEEE	The Institute of Electrical and Electronics Engineers
ISO	International Organization for Standarization
IEC	International Electrotechnical Commission
Std	Standard
TDD	Test Driven Development
A-TDD	Acceptance Test Driven Development
BDD	Behauvior Driven Development
Alpha	Abstract-Level Progress Health Attribute
Semat	Software Engineering Method and Theory
SWEBOK	Software Engineering Body of Knowledge
V&V	Verificación y Validación
NASA	National Aeronautics and Space Administration
MIL	Model In Loop
SIL	Software In Loop
PIL	Processor In Loop
HIL	Hardware In Loop
TLD	Test Last Development
SoC	System on Chip
RTL	Register Transfer Level
HW	Hardware
SW	Software
A&T	Analysis and Test patterns
MBAT	Model-Based Analysis and Testing
ISTQB	International Software Testing Qualification Board

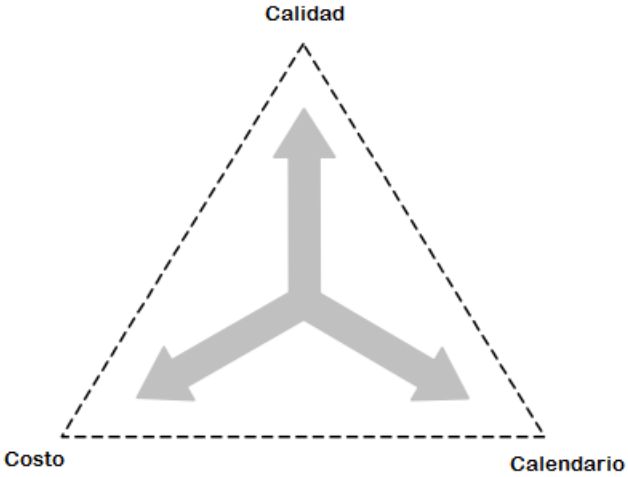
1. Introducción

Boehm (1984) define informalmente la verificación de software mediante la pregunta ¿se está construyendo el producto de software correctamente? En varias normas se define la verificación de sistemas de software (VSS) como “la confirmación, mediante la aportación de evidencia objetiva, de que se cumplen los requisitos especificados” (IEEE Std12207, 2008; IEEE Std15288, 2008; ISO/IEC 25000, 2014; ISO 9000, 2015). Además, en este proceso se debe tratar de encontrar defectos lo más tempranamente posible para que el costo de errores de requisitos no sea mayor en etapas posteriores. La VSS se realiza en paralelo con todas las etapas del ciclo de vida del software, tal como se establece en la norma para la verificación de software (IEEE Std1012, 2016).

Las mejores prácticas de VSS se realizan en las fases del ciclo de vida durante el desarrollo del producto, para evaluar si se construye un producto que cumpla con los requisitos establecidos y lograr así el desarrollo de productos de software con calidad. La calidad se define como el “grado en el que un conjunto de características inherentes de un objeto cumple con los requisitos” (ISO 9000, 2015).

El objetivo de la VSS es aumentar la calidad del software, buscar defectos y verificar que el software que se desarrolla cumple con los requisitos esperados (Merayo y Salaün, 2017). De acuerdo con el cuerpo de conocimiento de la ingeniería del software (SWEBOK por sus siglas en inglés), en una cultura saludable de ingeniería de software se debe entender el balance entre costo, calendario y calidad (Bourque *et al.,* 2014). Así, los intereses de las actividades de verificación se ligan con las actividades de calidad y éstas se combinan con los intereses de los costos y el calendario, formando un triángulo (véase la Figura 1). Por lo anterior, las mejores prácticas de VSS se comprometen con estos tres objetivos de la ingeniería del software, principalmente con la calidad.

Figura 1-1: Los tres objetivos de la ingeniería del software
(Adaptado de Andersson, 2003)



Las propuestas de mejores prácticas de verificación de sistemas de software se pueden encontrar en diferentes técnicas de verificación de sistemas de software que se representan gráficamente y otras se expresan en lenguaje natural. Algunos ejemplos de técnicas de verificación que se representan gráficamente son; verificación formal de sistemas en *chip* (Bhunia *et al*., 2019), verificación de software basado en la arquitectura (Schmidt, 2013), verificación basada en análisis y pruebas en sistemas críticos de seguridad (Herzner *et al*., 2014) y verificación de modelos (Murphy *et al*., 2008), entre otros; otras técnicas de verificación de software que se expresan en lenguaje natural se utilizan en la industria y en otras resultan casos de estudios realizados en la industria y en la academia. Algunas propuestas se encuentran en verificación formal automática basada en análisis estático en Facebook (Calcagno *et al*., 2015), verificación de requisitos de software embebido para naves espaciales en la Nasa (Mirantes *et al*., 2014), revisiones y auditorías técnicas (Holdaway, 2009; Schmidt, 2013; Antinyan *et al*., 2017), inspecciones de documentación técnica (Komssi *et al*., 2010; Dautovic *et al*., 2011), verificación de aplicaciones móviles (Sahinoglu *et al*., 2014; Santos *et al*., 2015), verificación de software embebido (Espinosa-Bedoya *et al*., 2016; Garousi *et al*., 2018) y técnicas de pruebas de software (Garousi *et al*., 2013; Bjarnason *et al*., 2013; Kassab *et al*., 2017), entre otros.

En las representaciones gráficas se evidencian diferencias entre sus elementos principales y en el modelo de representación; por ejemplo los símbolos que se utilizan en verificación formal de sistemas en *chip* son símbolos de diagramas de flujo para representar gráficamente el proceso de verificación (Bhunia *et al*., 2019); en verificación de software basado en la arquitectura se utilizan elipses entrelazadas y rectángulos unidos con líneas (Schmidt, 2013), en verificación de software embebido se utiliza el diseño basado en modelos, donde se usan modelos específicos de pruebas para representar procesos sistémicos de simulación con vectores y figuras como cuadros y rectángulos (Murphy *et al.,* 2008); en sistemas ciberfísicos críticos de seguridad también se trabaja con software embebido y también se propone en sus representaciones cuadros y rectángulos (Herzner *et al.,* 2014).

En la revisión de estas técnicas de verificación de sistemas de software se encuentran buenas prácticas en común que se utilizan en estas técnicas de verificación, como por ejemplo, verificación continua de sistemas de software por un equipo independiente, planificación continua del proceso de verificación, detección temprana de errores y defectos, retroalimentación continua entre equipo de verificación y demás miembros del equipo, utilización continua de herramientas automáticas, realización continua de pruebas de componentes, realización continua de pruebas de integración y revisión en pares de código y requisitos.

Aunque en las propuestas de representaciones de técnicas de VSS anteriormente mencionadas donde se encuentran mejores prácticas de VSS, algunos elementos son similares, no se establecen elementos universales y, además, en ninguna de estas propuestas hay un marco de trabajo estándar donde se pueda capturar sus elementos comunes. En las propuestas de mejores prácticas de VSS que se expresan en lenguaje natural, los elementos principales de las prácticas de verificación son diferentes y no se representan gráficamente. Por ejemplo en verificación formal automática basada en análisis estático en Facebook, el desarrollo de software es permanente, de modo que los cambios en el código son continuos, iterativos e incrementales; en la técnica de verificación se analiza estáticamente el código mediante herramientas automáticas de forma continua y asíncrona. Los resultados que arrojan las herramientas se revisan en pares y se realiza retroalimentación con los desarrolladores de software (Calcagno *et al*., 2015). En verificación de requisitos de software embebido durante el desarrollo de software científico para naves espaciales, en los resultados de sus análisis y pruebas se realizan listados

de lecciones aprendidas y de recomendaciones para futuras misiones espaciales (Mirantes *et al*., 2014). En las propuestas de las mejores prácticas para revisiones de documentación técnica se utilizan marcos para estructurar experiencias y hacer uso de la palabra en comunicación escrita y oral, lo cual puede desmotivar a los miembros del equipo de trabajo (Holdaway, 2009). En inspecciones de documentos de software donde las revisiones en pares se reconocen como una de las mejores prácticas en ingeniería de requisitos para el control de la calidad, se propone realizar inspecciones con diferentes actividades (Komssi *et al*., 2010) y también se presenta un enfoque que se basa en herramientas que facilitan el proceso de inspección (Dautovic *et al*., 2011).

Si las propuestas de mejores prácticas de VSS tuvieran un marco común donde se pudieran representar e identificar para que se apliquen durante el proceso de construcción de un producto de software, se podría contribuir a que los productos se entreguen con calidad, satisfaciendo los requisitos que se establecen. Para superar los desafíos de verificación, se requiere una estrategia de verificación para obtener una alta calidad (Ghosh *et al*., 2015).

Para poder capturar los elementos comunes de las mejores prácticas de VSS se plantea el uso del núcleo y el lenguaje de la Esencia de *Semat* (*Software Engineering Method And Theory*), estándar que se plantea en el OMG (*Object Management Group*; OMG, 2018). *Semat* es una comunidad internacional que promueve el proceso para crear un estándar universal, que se basa en una teoría sólida, principios probados y mejores prácticas (OMG, 2018). La comunidad *Semat* se centra en dos objetivos principales: (i) encontrar un núcleo de elementos ampliamente acordados; (ii) definir una base teórica sólida de la ingeniería de software (OMG, 2018). Este estándar contiene elementos ampliamente aceptados en la ingeniería de software, incluyendo un núcleo y un lenguaje. Con la Esencia de la ingeniería de software se introduce el núcleo y el lenguaje, mostrando cómo aplicarlos en el desarrollo software, donde el equipo de trabajo evalúa y escoge las mejores prácticas que pueden ser útiles para un proyecto de software determinado. Juntos, el núcleo y el lenguaje se conocen como la Esencia y conforman un marco común. El núcleo de la Esencia proporciona el terreno común para ayudar a los profesionales a comparar métodos y tomar mejores decisiones sobre sus prácticas (OMG, 2018). El núcleo de *Semat* no

compite con los métodos de desarrollo de sistemas de software existentes. No es un nuevo método para el desarrollo de sistemas de software (Simonette *et al*., 2014), si no una respuesta a la necesidad de redefinir la ingeniería de software (Jacobson *et al*., 2013).

Por esta razón, en esta Tesis de Maestría se propone la identificación y representación de mejores prácticas de VSS en un marco común, para poder aplicarlas cuando sea necesario, dependiendo del contexto determinado de un proyecto de software. De esta forma se puede contribuir en el proceso de verificación de un sistema de software que se desarrolla, para que se cumpla con los requisitos que se especifican. Con las representaciones de las mejores prácticas de VSS en el núcleo de la Esencia de *Semat*, los elementos principales de estas mejores prácticas se pueden extraer, combinar o adaptar cuando el contexto del proyecto de software lo amerite y a medida que el proceso de VSS avance. Esto puede contribuir al desarrollo de productos con calidad. En esta propuesta se abarcan algunas de las vistas típicas que posibilitan la separación de intereses con el núcleo de la Esencia de *Semat,* al ligar el símbolo de las prácticas con los alfas, los espacios de actividad, los productos de trabajo, las actividades, las fases, los roles y las competencias necesarias en esta labor.

Esta Tesis se organiza de la siguiente manera: En el Capítulo 2 se presenta el marco teórico relacionado con la verificación de sistemas de software y la sintaxis gráfica de los elementos del núcleo de *Semat*; en el Capítulo 3 se describen antecedentes de representaciones gráficas sobre mejores prácticas de verificación de sistemas de software, propuestas de mejores prácticas de VSS expresadas en lenguaje natural, se realiza una discusión, se planea el problema de investigación, la hipótesis, el objetivo general, los objetivos específicos, se identifican las técnicas de VSS revisadas y las mejores prácticas de VSS con sus elementos principales; en el Capítulo 4 se proponen las representaciones de las mejores prácticas de verificación de sistemas de software en el núcleo de la esencia de *Semat*, en el Capítulo 5 se presenta la validación de las representaciones de las mejores prácticas de VSS propuestas; en el Capítulo 6 se discuten las conclusiones, y trabajo futuro que se puede derivar de esta Tesis y, finalmente, en el Capítulo 7 se presentan las referencias.

2. Marco teórico

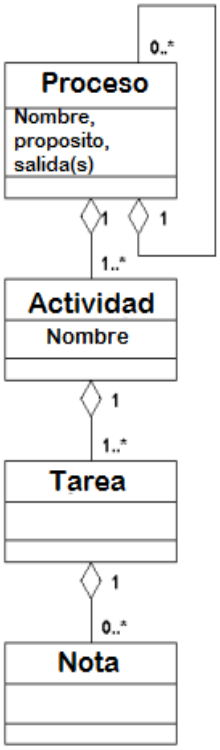
2.1 Verificación de Sistemas y Software

La verificación se define formalmente y específicamente en la norma para la verificación de sistemas y software (IEEE Std1012, 2016), como “el proceso para proporcionar evidencia objetiva de que el sistema, el software y sus productos asociados cumplen con los requisitos de todas las actividades del ciclo de vida durante cada proceso del ciclo de vida (adquisición, suministro, desarrollo, operación y mantenimiento)”. En la norma para los procesos de aseguramiento de la calidad (IEEE 730, 2014) se explica que la verificación busca la consistencia, la completitud, la exactitud del software y su documentación de respaldo mientras se desarrolla y proporciona soporte para una conclusión posterior para que el software se valide. La validación se define como “el proceso para proporcionar evidencia de que el sistema, el software o el hardware y sus productos asociados satisfacen los requisitos asignados al final de cada actividad del ciclo de vida, resuelven el problema correcto y satisfacen el uso previsto y las necesidades del usuario” (IEEE Std1012, 2016). Para los propósitos de verificación y validación (V&V), no se concluye ningún proceso de ciclo de vida hasta que sus productos de desarrollo se verifican y validan de acuerdo con las tareas definidas en el plan de V&V; el esfuerzo de V&V es el trabajo asociado para realizar los procesos, las actividades y tareas de V&V (IEEE Std1012, 2016).

La VSS dentro del ciclo de vida es un proceso con un conjunto de actividades que compara un producto del ciclo de vida con las características requeridas para ese producto. Esto puede incluir requisitos específicos, descripción del diseño y el propio sistema, entre otros (IEEE Std12207, 2008).

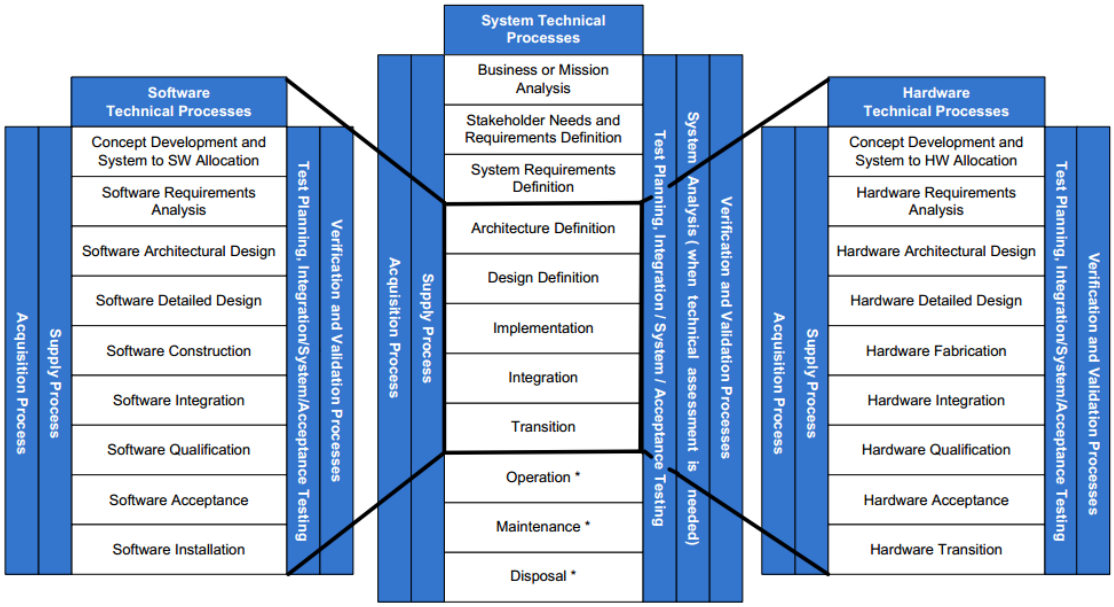
Los procesos del ciclo de vida de software contienen actividades, que a su vez tienen tareas (véase la Figura 2-1). Los procesos requieren un propósito y unas salidas y contienen por lo menos una actividad. Las actividades se construyen con un grupo de tareas y las notas se usan cuando se necesita explicación o una mejor descripción de un proceso o actividad (IEEE Std12207, 2008).

Figura 2-1: Construcción de los procesos
(Adaptado de IEEE Std12207, 2008)



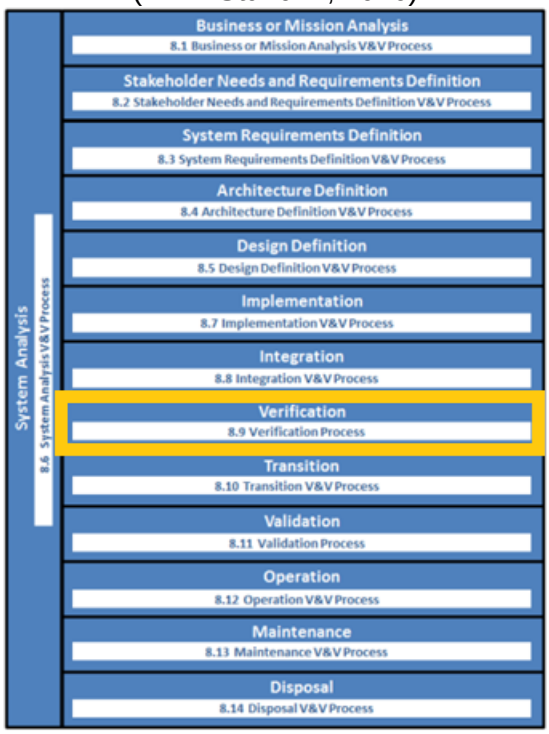
Los procesos de verificación se encuentran en los procesos técnicos, los cuales se clasifican en procesos técnicos de software, procesos técnicos de sistema y procesos técnicos de hardware (IEEE Std1012, 2016; Véase la Figura 2-2). Las actividades de VSS se pueden aplicar a cualquier proceso de ciclo de vida (IEEE Std1012, 2016).

Figura 2-2: Relación de los procesos técnicos de software, procesos técnicos de sistema y procesos técnicos de hardware
(IEEE Std1012, 2016)



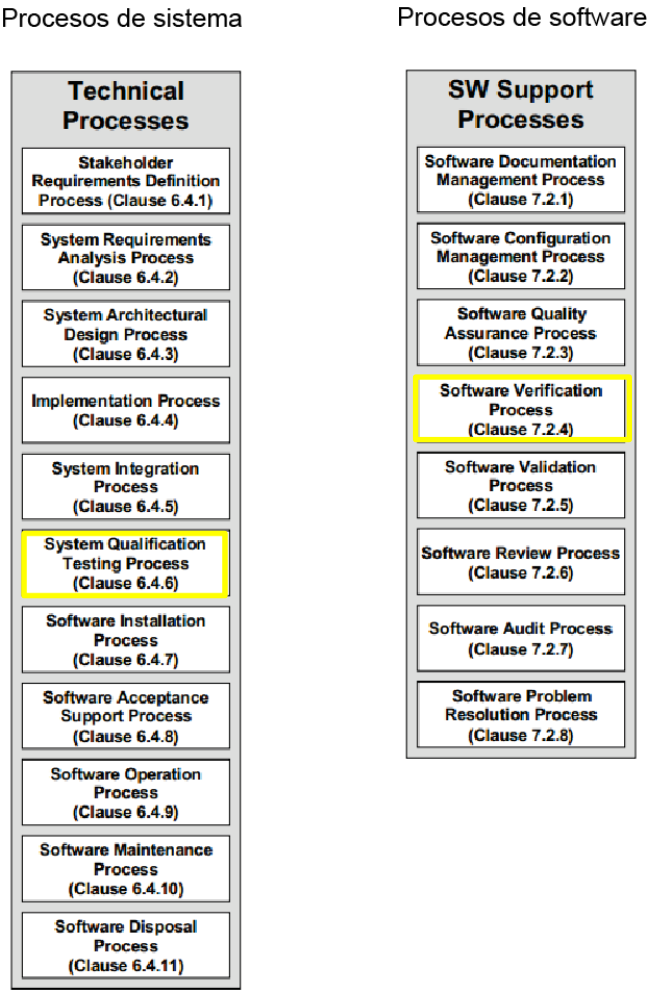
El propósito del proceso de verificación es confirmar que cada producto o servicio de software de un proceso o proyecto refleje adecuadamente los requisitos que se especifican (IEEE Std12207, 2008). Una propiedad esencial de todos los requisitos del software es que sean verificables (Bourque *et al*., 2014). Los procesos de verificación proporcionan una evaluación objetiva de productos y procesos a lo largo del ciclo de vida. Esta evaluación demuestra si los requisitos son correctos, completos, precisos, consistentes y verificables. Los procesos de verificación permiten determinar si el producto satisface el uso previsto y las necesidades del usuario, incluyendo análisis, evaluación, revisión, inspección y prueba de productos y procesos (IEEE Std1012, 2016). En los procesos técnicos del ciclo de vida del sistema se puede apreciar el proceso de verificación (Véase la Figura 2-3).

Figura 2-3: Procesos de verificación alineados con otros procesos técnicos del ciclo de vida (IEEE Std1012, 2016)



Los grupos de procesos del ciclo de vida del sistema y del software también se pueden apreciar en la norma IEEE 12207; La verificación se encuentra en los procesos técnicos del sistema y en los procesos de soporte del software como se muestra en la Figura 2-4.

Figura 2-4: Grupos de procesos del ciclo de vida de sistema y de software (IEEE 12207, 2008)



2.1.1 Niveles de integridad

El esfuerzo para el proceso de verificación contiene unas actividades y tareas de acuerdo con el nivel de integridad del sistema o software. Los niveles de integridad se utilizan para determinar las actividades, las tareas, el rigor y el nivel de intensidad de la verificación que se realizará, establecen la importancia del sistema basado en la complejidad, criticidad, riesgo, nivel de seguridad y otras características (IEEE Std1012, 2016). Los niveles de integridad son cuatro:

1) Insignificante
2) Marginal
3) Crítico
4) Catastrófico

Los cuatro niveles de integridad establecidos en la norma para la documentación de pruebas de software y sistemas (IEEE Std.829, 2008) se describen en Tabla 2-1.

Tabla 2-1: Niveles de integridad
(Adaptado de IEEE Std.829, 2008)

Descripción	Nivel
El software se debe ejecutar correctamente o se producirán graves consecuencias (pérdida de vida, pérdida del sistema, daños ambientales, pérdida económica o social). Ninguna mitigación es posible.	4
El software se debe ejecutar correctamente o el uso previsto (misión) del sistema/software no se realizará, lo que causará graves consecuencias (lesiones permanentes, mayor degradación del sistema, daños ambientales, impacto económico o social). La mitigación parcial o completa es posible.	3
El software se debe ejecutar correctamente; de lo contrario, no se podrá realizar una función deseada con consecuencias menores. Mitigación completa posible.	2
El software se debe ejecutar correctamente o la función deseada no se realizará causando consecuencias insignificantes. No se requiere mitigación.	1

2.1.2 Verificación de sistemas

El propósito del proceso de verificación de sistemas es proporcionar evidencia objetiva cuando los resultados alcancen lo siguiente (IEEE Std1012, 2016):

a) Cumplir con los requisitos (por ejemplo, para la corrección, integridad, consistencia y precisión) para todas las actividades durante cada proceso del ciclo de vida.
b) Satisfacer los estándares, prácticas y convenciones durante los procesos del ciclo de vida.

c) Completar con éxito cada actividad de ciclo de vida y satisfacer todos los criterios para iniciar actividades de ciclo de vida sucesivas (es decir, el producto se construye correctamente).

Resultados:

Como resultado de la implementación exitosa del proceso de verificación de sistemas:

a) Se desarrolla e implementa un plan de verificación.
b) El sistema de interés y todos los componentes del sistema de interés tienen asignados niveles de integridad que se reevalúan a lo largo del ciclo de vida del sistema.
c) El sistema y cada uno de sus componentes se evalúan para satisfacer los requisitos en función de los niveles de integridad asignados.
d) La evidencia objetiva se desarrolla para determinar si el sistema y cada uno de sus componentes cumplen con los requisitos y satisfacen todos los criterios para cada actividad sucesiva del ciclo de vida.

El esfuerzo de verificación de sistemas se ajusta a las descripciones de tareas, entradas y salidas para los niveles de integridad.

En la tabla 2-2, 2-3 y 2-4 se pueden observar las actividades y tareas de entrada para la verificación de sistemas.

Tabla 2-2: Actividades y tareas de entrada para la verificación de sistemas (Adaptado de ISO/IEC 15288, 2014)

	Proceso: Análisis del negocio o misión	**Proceso:** Necesidades del interesado y definición de requisitos	**Proceso:** Definición de requisitos del sistema	**Proceso:** Definición de arquitectura	**Proceso:** Definición del diseño	**Proceso:** Análisis del sistema	**Proceso:** Implementación	**Proceso:** Integración	**Proceso:** Transición	**Proceso:** Operaciones	**Proceso:** Mantenimiento	**Proceso:** Disposición
Entradas	1) Resultados de analisis del negocio o misión	1) Documentacion conceptual	1) Documentacion conceptual	1) Vistas y modelos de arquitectura	1) Vistas y modelos de arquitectura	1) Estrategia de analisis del sistema	1) Documentacion conceptual	1) Documentacion conceptual	1) Plan de pruebas de aceptación	1) Documentacion conceptual	1) Estrategia de mantenimiento del sistema	1) Leyes aplicables
	2) Informe del análisis de peligros	2) Informe del análisis de riesgos	2) Informe de tareas críticas	2) Análisis de compensación de arquitectura	2) Análisis de compensación de arquitectura	2) Resultados del análisis del sistema	2) Informe de tareas críticas	2) Elementos de hardware	2) Acuerdo de adquisición	2) Restricciones	2) Datos de rendimiento del sistema	2) Documentación conceptual
	3) Soluciones candidatas identificadas	3) Procedimientos y procesos organizacionales	3) Informe del analisis de peligros	3) Necesidades del negocio	3) Necesidades del negocio		3) Documentos de diseño	3) Secuencia y estrategia de	3) Documentacion conceptual	3) Cambios ambientales	3) Requerimientos del sistema	3) Repositorio de configuración
	4) problema identificado o espacio de oportunidad	4) Soluciones candidatas preferidas	4) Procedimientos y procesos organizacionales	4) Informe de tareas críticas	4) Informe de tareas críticas		4) Procesos de fabricación, herramientas y equipos.	4) Otros elementos externos del sistema	4) Paquete de instalación	4) Reporte de analisis de peligros		4) plan de disposición
	5) Espacio de solución identificado	5) Evaluación preliminar de amenazas y riesgos.	5) Evaluación preliminar de amenazas y riesgos.	5) Informe del analisis de riesgos	5) Documentos de diseño		5) Dibujos de hardware	5) Elementos del software	5) Requisitos de los interesados	5) Paquete de instalación		5) Registros de inventario
	6) Nuevos elementos de mercado o misión.	6) Reporte de analisis de seguridad	6) Informe de análisis de riesgo	6) Procedimientos y procesos organizacionales	6) Informe del analisis de riesgos		6) Especificaciones de requerimientos de hardware	6) Requisitos de los interesados	6) Resultados de pruebas de aceptación del sistema	6) Nuevas restricciones		6) Lista de restricciones
	7) concepto organizativo de operaciones	7) Requisitos del interesado	7) Reporte de analisis de seguridad	7) Evaluación preliminar de amenazas y riesgos.	7) Procedimientos y procesos organizacionales		7) Analisis de trazabilidad de hardware	7) Arquitectura del sistema	7) Requisitos del sistema	7) Procedimientos de operación		7) Decisiones de gestión y técnicas

Tabla 2-3: (Continuación) Actividades y tareas de entrada para la verificación de sistemas (Adaptado de ISO/IEC 15288, 2014)

	Proceso: Análisis del negocio o misión	**Proceso:** Necesidades del interesado y definición de requisitos	**Proceso:** Definición de requisitos del sistema	**Proceso:** Definición de arquitectura	**Proceso:** Definición del diseño	**Proceso:** Análisis del sistema	**Proceso:** Implementación	**Proceso:** Integración	**Proceso:** Transición	**Proceso:** Operaciones	**Proceso:** Mantenimiento	**Proceso:** Disposición
Entradas	8) Objetivos y planes estratégicos organizacionales	8) Planes y horarios de desarrollo de proveedores.	8) Requisitos del interesado	8) Informe de análisis de riesgo	8) Evaluación preliminar de amenazas y		8) Informe del analisis de peligros	8) Estrategia de integración	8) Estrategia de transición	8) Reportes de problemas de operación		8) Procedimientos y pólizas organizacional
	9) Estrategia organizacional	9) Resultados de tareas de verificación	9) Planes y horarios de desarrollo de proveedores.	9) Reporte de analisis de seguridad	9) Informe de análisis de riesgo		9) Procedimientos de implementación	9) Resultados de pruebas de calidad del sistema	9) Sistema de transición	9) Cambios propuestos		
	10) Procesos y procedimientos organizacionales		10) Plan de prueba de aceptación del sistema	10) Planes y horarios de desarrollo de proveedores	10) Reporte de analisis de seguridad		10) Estrategias de implementación	10) Requisitos del sistema	10) Documentación del usuario	10) Reporte de análisis de riesgos		
	11) Soluciones candidatas preferidas		11) Plan de prueba de integración del sistema	11) Plan de prueba de aceptación del sistema	11) Analisis de seguridad de subsistemas		11) Reporte de analisis de riesgos			11) Reporte de análisis de seguridad		
	12) Evaluación preliminar de amenazas y riesgos.		12) Plan de prueba de calidad del sistema	12) diseño de prueba de integración del sistema	12) Planes y horarios de desarrollo de proveedores.		12) Reporte de analisis de seguridad			12) Planes y horarios de desarrollo de proveedores.		
	13) reporte de analisis de seguridad		13) Requisitos del sistema	13) Diseño de prueba de calidad del sistema	13) casos de prueba de aceptación del sistema		13) Documentos de diseño de software			13) Plan de verificación del sistema		
	14) Resultados de las tareas de verificación		14) Resultados de las tareas de verificación	14) Requisitos del sistema	14) diseño de prueba de integración del sistema		14) Especificación de requerimientos de interfaz de software			14) Documentación del usuario		
				15) Resultados de las tareas de verificación	15) Diseño de prueba de calidad del sistema		15) Especificación de requerimientos de software			15) Resultados de las tareas de verificación		

Tabla 2-4: (Continuación) Actividades y tareas de entrada para la verificación de sistemas Adaptado de ISO/IEC 15288, 2014)

Entradas	**Proceso:** Análisis del negocio o misión	**Proceso:** Necesidades del interesado y definición de requisitos	**Proceso:** Definición de requisitos del sistema	**Proceso:** Definición de arquitectura	**Proceso:** Definición del diseño	**Proceso:** Análisis del sistema	**Proceso:** Implementación	**Proceso:** Integración	**Proceso:** Transición	**Proceso:** Operaciones	**Proceso:** Mantenimiento	**Proceso:** Disposición
					16) Requisitos del sistema		16) Analisis de tasabilidad de software					
					17) Resultados de las tareas de verificación		17) Subsistemas					
							18) Casos de prueba de aceptación del sistema					
							19) Arquitectura del sistema					
							20) Diseño del sistema					
							21) Descripción del elemento del sistema desde el diseño del sistema.					
							22) Resultados del análisis de la implementación del elemento del sistema					
							23) Requisitos del elemento del sistema, diseños e implementación.					
							24) Elementos del sistema					
							25) Casos de prueba de integración del sistema					
							26) Diseño de prueba de calidad del sistema					
							27) Requisitos del sistema					
							28) Resultados de las tareas de verificación					

En la tabla 2-5 se especifican las actividades y tareas para el proceso de verificación de sistemas en el nivel de integridad 4.

Tabla 2-5: Actividades y tareas mínimas requeridas en el nivel de integridad 4 para la verificación de sistemas

(Adaptado de IEEE Std.1012, 2016)

	Proceso: Análisis del negocio o misión	**Proceso:** Necesidades del interesado y definición de requisitos	**Proceso:** Definición de requisitos del sistema	**Proceso:** Definición de arquitectura	**Proceso:** Definición del diseño	**Proceso:** Análisis del sistema	**Proceso:** Implementación	**Proceso:** Integración	**Proceso:** Transición	**Proceso:** Operaciones	**Proceso:** Mantenimiento	**Proceso:** Disposición
Tareas	1) Evaluación de los resultados del analisis del negocio o misión	1) Necesidades del interesado y evaluación de requisitos	1) Evaluación de requisitos	1) Evaluación de la arquitectura	1) Evaluación del diseño	1) Evaluación de la estrategia de análisis del sistema	1) Evaluación de la estrategia de implementación	1) Evaluación de la estrategia de integración	1) Evaluación de la estrategia de transición	1) Evaluación de los procesos operacionales	1) Evaluación de la estrategia del mantenimiento	1) Evaluación del plan de disposición
	2) Análisis de trazabilidad	2) Análisis de trazabilidad	2) Análisis de la interfaz	2) Análisis de la interfaz	2) Análisis de la interfaz	2) Resultados de evaluación del análisis del sistema	2) Análisis de la implementación de los elementos del sistema	2) Ejecución de las pruebas de ingegración del sistema	2) Evaluación de la demostración de la transición	2) Análisis de peligro	2) Evaluación de la ejecución del mantenimiento del sistema	
	3) Análisis de criticidad	3) Análisis de criticidad	3) Análisis de trazabilidad	3) Análisis de asignación de requisitos	3) Análisis de trazabilidad		3) Análisis de la interacción de los elementos del sistema	3) Análisis de la interacción de los elementos del sistema	3) Ejecución de las pruebas de aceptación del sistema	3) Análisis de seguridad		
	4) Análisis de peligro	4) Análisis de peligro	4) Análisis de criticidad	4) Análisis de trazabilidad	4) Análisis de criticidad		4) Análisis de criticidad	4) Ejecución de las pruebas de cualificación del sistema		4) Análisis de riesgos		
	5) Análisis de seguridad	5) Análisis de seguridad	5) Planeación de pruebas integración del sistema	5) Análisis de criticidad	5) Casos de prueba de integración del sistema		5) Procedimiento de las pruebas de integración del sistema					
	6) Análisis de riesgos	6) Análisis de riesgos	6) Planeación de pruebas de cualificación del sistema	6) Planeación de pruebas de integración del sistema	6) Casos de pruebas de la cualificación del sistema		6) Procedimiento de las pruebas de cualificación del sistema					
			7) Planeación de pruebas aceptación del sistema	7) Planeación de pruebas de cualificación del sistema	7) Casos de pruebas de la aceptación del sistema		7) Procedimiento de las pruebas de aceptación del sistema					
			8) Análisis de peligro	8) Planeación de pruebas de aceptación del sistema	8) Análisis de peligro		8) Análisis de peligro					
			9) Análisis de seguridad	9) Análisis de peligro	9) Análisis de seguridad		9) Análisis de seguridad					
			10) Análisis de riesgos	10) Análisis de seguridad	10) Análisis de riesgos		10) Análisis de riesgos					
				11) Análisis de riesgos								

En la tabla 2-6 se especifican las actividades y tareas de salida para el proceso de verificación de sistemas en el nivel de integridad 4.

Tabla 2-6: Actividades y tareas de salida en el nivel de integridad 4 para la verificación de sistemas

(Adaptado de IEEE Std.1012, 2016)

Salidas	**Proceso:** Análisis del negocio o misión	**Proceso:** Necesidades del interesado y definición de requisitos	**Proceso:** Definición de requisitos del sistema	**Proceso:** Definición de arquitectura	**Proceso:** Definición del diseño	**Proceso:** Análisis del sistema	**Proceso:** Implementación	**Proceso:** Integración	**Proceso:** Transición	**Proceso:** Operaciones	**Proceso:** Mantenimiento	**Proceso:** Disposición
	1) Reporte de anomalías	1) Reporte de anomalías	1) Reporte de anomalías	1) Reporte de anomalías	1) Reporte de anomalías	1) Reporte de tareas	1) Reporte de anomalías	1) Reporte de anomalías	1) Reporte de anomalías	1) Reporte de anomalías	1) Reporte de anomalías	1) Reporte de anomalías
	2) Reporte de tareas	2) Reporte de tareas	2) Reporte de tareas	2) Reporte de tareas	2) Reporte de tareas		2) Reporte de tareas	2) Reporte de tareas	2) Reporte de tareas	2) Reporte de tareas	2) Reporte de tareas	2) Reporte de tareas
			3) Plan de pruebas del sisema -Aceptación -Integración -Calidad	3) Diseño de pruebas del sistema -Aceptación -Integración -Calidad	3) Casos de prueba del sistema -Aceptación -Integración -Calidad		3) Procedimientos de pruebas del sistema -Aceptación -Integración -Calidad	3) Resultados de la ejecución de las pruebas del sistema -Integración -Calidad	3) Resultados de la ejecución de las pruebas de aceptación del sistema	3) Análisis de seguridad actualizado	3) Evaluaciones de acciones correctivas	

2.1.3 Verificación de software

El propósito del proceso de verificación de software es proporcionar evidencia objetiva si los resultados alcanzan lo siguiente (IEEE Std1012, 2016):

a) Cumplir con los requisitos (por ejemplo, para la corrección, integridad, consistencia y precisión) de todas las actividades durante cada proceso del ciclo de vida.

b) Satisfacer los estándares, prácticas y convenciones durante los procesos del ciclo de vida.

c) Completar con éxito cada actividad de ciclo de vida y satisfacer todos los criterios para iniciar actividades de ciclo de vida sucesivas (es decir, el producto está construido correctamente).

Resultados:

Como resultado de la implementación exitosa del proceso de verificación de Software:

a) Se desarrolla e implementa un plan de verificación.

b) El sistema de interés (software) y todos los componentes del sistema de interés tienen asignados niveles de integridad que se reevalúan a lo largo del ciclo de vida del sistema.

c) El software y cada uno de sus componentes se evalúan para satisfacer los requisitos en función de los niveles de integridad asignados.

d) Se desarrollan pruebas objetivas para determinar si el software y cada uno de sus componentes cumplen con los requisitos y satisfacen todos los criterios para cada actividad sucesiva del ciclo de vida.

El esfuerzo de verificación de software se ajusta a las descripciones de tareas, entradas y salidas.

En la tabla 2-7 se observan las actividades y tareas de entrada para el proceso verificación de software.

Tabla 2-7: Actividades y tareas de entrada para la verificación de software (Adaptado de IEEE Std.1012, 2016)

	Proceso: Análisis de los requisitos del *software*	**Proceso:** Análisis de los requisitos del *software*	**Proceso:** Diseño de la arquitectura del *software*	**Proceso:** Construcción del *software*	**Proceso:** Integración del *software*	**Proceso:** Pruebas de calificación del *software*	**Proceso:** Soporte de aceptación del *software*	**Proceso:** Instalación del *software*	**Proceso:** Operación	**Proceso:** Mantenimiento	**Proceso:** Disposición
Entradas	1) Necesidades de adquisición	1) Documentación conceptual	1) Documentación conceptual	1) Estándar de codificación.	1) Informe de análisis de peligros	1) Informe de análisis de peligros	1) Informe de análisis de peligros	1) Informe de análisis de peligros	1) Documentación conceptual	1)Cambios aprobados	1) Estrategia de disposición del *software*
	2) Documentación conceptual	2) Informe de criticidad	2) Informe de criticidad	2) Documentación conceptual	2) Informe de análisis de riesgos.	2) Informe de análisis de riesgos.	2) Informe de análisis de riesgos.	2) Paquete de instalación	2) Cambios de entorno	2) Reportes de anomalías	
	3) Asignación del nivel de integridad al desarrollador	3) Informe de análisis de peligros	3) Normas de diseño	3) Informe de criticidad	3) Informe de análisis de seguridad	3) Informe de análisis de seguridad	3) Descripción del diseño del software, documento del diseño de la interfaz	3) Informe de análisis de riesgos.	3) Informe de análisis de seguridad	3) Paquete de instalación	
	4) Informe de análisis de peligros	4) Evaluación preliminar de amenazas y riesgos	4) Informe de análisis de peligros	4) Informe del análisis de peligros	4) Fuente y codigo ejecutable	4) Plan de pruebas de calidad del *software*, diseño, casos, procedimientos y resultados de ejecución de pruebas	4) Informe de análisis de seguridad	4) Informe de análisis de seguridad	4) Paquete de instalación	4) Reporte de análisis de seguridad	
	5) Evaluación preliminar de amenazas y riesgos	5) Informe de análisis de riesgos.	5) Informe de análisis de riesgos.	5) Informe de análisis de riesgos.	5) Plan de pruebas de *software*, diseño de pruebas, casos de prueba, procedimiento de pruebas: - Integración - Calidad sw - Aceptación	5) Fuente y codigo ejecutable	5) Plan de pruebas de aceptación del *software*, diseño, casos, procedimientos y resultados de ejecución de pruebas	5) Planes y horarios de desarrollo de proveedores	5) Nuevas restricciones	5) Niveles de integridad de mantenimiento	
	6) Informe de análisis de seguridad	6) Informe de análisis de seguridad	6) Informe de análisis de seguridad	6) Informe de análisis de seguridad	6) Resultados de ejecución de las pruebas de integración del *software*	6) Planes y horarios de desarrollo de proveedores.	6) Fuente y codigo ejecutable	6) Documentación del usuario	6) Procedimientos operacionales	6) Reportes de problemas operacionales	

Tabla 2-8: (Continuación) Actividades y tareas de entrada para la verificación de software (Adaptado de IEEE Std.1012, 2016)

	Proceso: Análisis de los requisitos del software	**Proceso:** Análisis de los requisitos del software	**Proceso:** Diseño de la arquitectura del software	**Proceso:** Construcción del software	**Proceso:** Integración del software	**Proceso:** Pruebas de calificación del software	**Proceso:** Soporte de aceptación del software	**Proceso:** Instalación del software	**Proceso:** Operación	**Proceso:** Mantenimiento	**Proceso:** Disposición
Entradas	7) Arquitectura del sistema	7) Especificación de los requisitos del software.	7) Especificación de los requisitos del software, especificación de los requisitos de la interfaz, descripción del diseño del software, documento del diseño	7) Descripción del diseño del software, documento del diseño de la interfaz, fuente y código ejecutable.	7) Planes y horarios de desarrollo de proveedores	7) Resultados de las tareas de verificación	7) Planes y horarios de desarrollo de proveedores.	7) Resultados de las tareas de verificación	7) Reportes de problemas operacionales	7) Cambios propuestos	
	8) Horarios y planes y de desarrollo de proveedores	8) Requisitos del sistema	8) Planes y horarios de desarrollo de proveedores.	8) Requisitos del sistema	8) Resultados de las tareas de verificación		8) Documentación del usuario		8) Cambios propuestos	8) Reporte de analisis de riesgos	
	9) Documentos de requerimientos del sistema	9) Planeación de las pruebas de software. - Cualificación de los desarrolladores - Aceptación del	9) Requisitos del sistema	9) Resultados de la ejecución de las pruebas de componentes			9) Resultados de las tareas de verificación		9) Reporte de analisis de riesgos	9) Reporte de análisis de seguridad	
	10) Necesidades de los usuarios	10) Planes y horarios de desarrollo de proveedores.	10) Planeación de las pruebas de software. - Componente - Integración - Calidad de sw - Aceptación	10) Diseño de pruebas de software y casos de prueba - Componente - Integración - Calidad de sw - Aceptación					10) Reporte de análisis de seguridad	10) Planes y horarios de desarrollo de proveedores.	
	11) Resultados de las tareas de V&V	11) Documentación del usuario	11) Diseño de las pruebas de software. - Componente - Integración - Calidad sw - Aceptación	11) V&V de procedimientos de pruebas de software - Componente - Integración - Calidad sw - Aceptación					11) Planes y horarios de desarrollo de proveedores	11) Plan de verificación	
		12) Resultados de las tareas de verificación	12) Documentación del usuario	12) Planes y horarios de desarrollo de proveedores.					12) Documentación del usuario	12) Resultados de las tareas de verificación	
			13) Resultados de las tareas de verificación	13) Documentación del usuario					13) Plan de verificación		
				14) Resultados de las tareas de verificación					14) Resultados de las tareas de verificación		

En la tabla 2-9 se aprecian las actividades y tareas para el proceso de verificación de software.

Tabla 2-9: Actividades y tareas para la verificación de software (Adaptado de IEEE Std.1012, 2016).

	Actividad: Concepto de software	**Actividad:** Análisis de requerisitos de software.	**Actividad:** Diseño de software.	**Actividad:** Construcción de software.	**Actividad:** Integración de software.	**Actividad:** Pruebas de calidad del software	**Actividad:** Pruebas de aceptación de software.	**Actividad:** Instalación de software y revisión	**Actividad:** Operación	**Actividad:** Mantenimiento	**Actividad:** Disposición
Tareas	1) Evaluación de la documentación conceptual	1) Evaluación de requisitos	1) Evaluación del diseño	1) Evaluación de la documentación y código fuente	1) Verificación de la ejecución de las pruebas de integración de software	1) Verificaciónde la ejecución de las pruebas de calificación del software	1) Verificación del procedmiento de pruebas de aceptación de software.	1) Auditoria de la configuración de la instalación	1)Evaluación de nuevas restricciones	1)Revisión del plan de verificación	1)Evaluación de la disposición del software
	2) Análisis de criticidad	2) Análisis de trazabilidad	2) Análisis de trazabilidad	2) Análisis de trazabilidad	2) Análisis de trazabilidad	2) Análisis de trazabilidad	2) Verificación de la ejecución de las pruebas de aceptación de software.	2) Revisión de la instalación	2)Evaluación de procedimientos operacionales	2) Evaluación de anomalías	
	3) Análisis de asignación de requisitos	3) Análisis de interfaz	3) Análisis de interfaz	3) Análisis de interfaz	3) Análisis de peligros	3) Análisis de peligros	3) Análisis de trazabilidad	3) Análisis de peligros	3) Análisis de peligros	3)Análisis de criticidad	
	4) Análisis de trazabilidad	4) Análisis de criticidad	4) Análisis de criticidad	4) Análisis de criticidad	4) Análisis de seguridad	4) Análisis de seguridad	4) Análisis de peligros	4) Análisis de seguridad	4) Análisis de seguridad	4)Evaluación de migración	
	5) Análisis de peligros	5) Verificación del plan de purebas de calidad de software	5) Verificación del plan de pruebas de componentes de software	5) Verificación de casos de pruebas de componentes de software	5) Análisis de riesgos	5) Análisis de riesgos	5) Análisis de seguridad	5) Análisis de riesgos	5) Análisis de riesgos	5)Evaluación de retiro	
	6) Análisis de seguridad	6) Verificación del plan de purebas de aceptación de software	6) Verificación del plan de purebas de integración de software	6) Verificación de casos de pruebas de integración de software			6) Análisis de riesgos			6) Análisis de peligros	
	7) Análisis de riesgos	7) Análisis de peligros	7) Verificación del plan de diseño de pruebas de componentes de software	7) Verificación de casos de pruebas de calidad de software						7) Análisis de seguridad	
		8) Análisis de seguridad	8) Verificación del plan de diseño de pruebas de integración de software	8) Verificación de casos de pruebas de aceptación de software						8) Análisis de riesgos	
		9) Análisis de riesgos	9) Verificación del diseño de pruebas de calidad de software	9) Verificación de procedmimientos de pruebas de componentes de software						9)Iteración de tareas	
			10) Verificación del diseño de pruebas de aceptación de software	10) Verificación de procedmimientos de pruebas de integración de software							
			11) Análisis de peligros	11) Verificación de procedmimientos de pruebas de caliad de software							
			12) Análisis de seguridad	12) Verificación de la ejecución de componentes del software							
			13) Análisis de riesgos	13) Análisis de peligros							
				14) Análisis de seguridad							
				15) Análisis de riesgos							

En tabla 2-10 se observan las actividades y salidas para el proceso de verificación de software.

Tabla 2-10: Salidas para la verificación de software
(Adaptado de IEEE Std.1012, 2016)

	Actividad: Concepto de software	**Actividad:** Análisis de requerisitos de software.	**Actividad:** Diseño de software.	**Actividad:** Construcción de software.	**Actividad:** Integración de software.	**Actividad:** Pruebas de calidad del software	**Actividad:** Pruebas de aceptación de software.	**Actividad:** Instalación de software y revisión.	**Actividad:** Operación.	**Actividad:** Mantenimiento.	**Actividad:** Disposición.
Salidas	1) Informe(s) de anomalías	1) Informe(s) de anomalías	1) Informe(s) de anomalías	1) Informe(s) de anomalías	1) Informe(s) de anomalías	1) Informe(s) de anomalías	1) Informe(s) de anomalías	1) Informe(s) de anomalías	1) Informe(s) de anomalías	1) Informe(s) de anomalías	1) Informe(s) de anomalías
	2) Informe(s) de tareas	2) Informe(s) de tareas	2) Informe(s) de tareas	2) Informe(s) de tareas	2) Informe(s) de tareas	2) Informe(s) de tareas	2) Informe(s) de tareas	2) Informe(s) de tareas	2)Reporte de tareas	2) Informe(s) de tareas	2) Informe(s) de tareas
		3) Verificación del plan de pruebas de software	3) Verificación del plan de pruebas de software - Componente - Integración	3) Casos de pruebas del software - Componente - Integración - Calidad sw - Aceptación	3) Informe de ejecución de verificación de las pruebas de software - Integración	3) Resultados de ejecución de verificación de las pruebas de calificación del software	3) Verificación del procedimiento de prueba de aceptación del software.			3)Plan de verificación	
			4) Verificación del diseño de pruebas de software - Componente - Integración - Calificación - Aceptación	4) Procedimiento de pruebas del software - Componente - Integración - Calidad sw			4) Resultados de ejecución de verificación de las pruebas de aceptación del software				
				5) Informe de ejecución de verificación de las pruebas de componentes de software							

2.1.4 Verificación formal

La verificación formal hace parte de los métodos formales, los cuales son métodos de ingeniería de software que se utilizan para especificar, desarrollar y verificar el software a través de la aplicación de una notación matemática y lenguajes rigurosos, se puede verificar la calidad del modelo de software, la integridad y la corrección de manera sistemática, automatizada o semiautomática (Bourque *et al.*, 2014).

2.1.5 Pruebas de verificación

Las pruebas de verificación son actividades que se deben realizar con un nivel mínimo, dependiendo del nivel de integridad (IEEE Std.1012, 2016).

Es importante aclarar que las pruebas son una de las muchas actividades de verificación destinadas a confirmar que la salida de desarrollo de software cumple con sus requisitos de entrada, son actividades importantes de verificación (ISO/IEC 29119-1, 2013).

En la Tabla 2-8 se puede observar el nivel mínimo para las pruebas de verificación de sistemas y software de acuerdo al nivel de integridad.

Tabla 2-11: Nivel mínimo para las pruebas de verificación de sistemas y software en cada nivel de integridad
(Adaptado de IEEE Std.1012, 2016).

Software	**Pruebas para el nivel de integridad**			
	4	**3**	**2**	**1**
Pruebas de componentes	Realizar	Realizar	Revisar	No hay acción
Pruebas de integración	Realizar	Realizar	Revisar	No hay acción
Pruebas de sistema	Realizar	Realizar	Revisar	No hay acción
Pruebas de aceptación	Realizar	Realizar	Revisar	No hay acción

Sistema	**Pruebas para el nivel de integridad**			
	4	**3**	**2**	**1**
Pruebas de integración	Realizar	Revisar	Revisar	No hay acción
Pruebas de sistema	Realizar	Revisar	Revisar	No hay acción
Pruebas de aceptación	Realizar	Revisar	Revisar	No hay acción

El requisito “Realizar” la prueba, significa que en el esfuerzo de verificación se crean criterios de prueba, se confirma que la prueba se lleva a cabo adecuadamente, aumenta la detección de errores del sistema/software y se revisan los resultados (IEEE Std1012, 2016).

Las pruebas de componentes, de integración, de sistema y de aceptación se catalogan como los niveles de las pruebas según la *ISTQB* (International Software Testing Qualification Board, por sus siglas en inglés), mientras que en el *SWEBOK* (Software Engineering Body Of Knowledge, por sus siglas en inglés) se catalogan los niveles de las pruebas solamente a las pruebas de componentes, integración y de sistema, no entran las pruebas de aceptación.

Para las siguientes definiciones se toma como referencia la ISTQB.

Pruebas de componentes

También conocidas como pruebas de unidad o unitarias o de módulo, se enfocan en los componentes que se pueden probar por separado del resto del sistema; generalmente las realiza el desarrollador que escribió el código, se requiere acceso al código que se está probando.

Pruebas de Integración

Se centran en las interacciones entre componentes o sistemas.

a) Las pruebas de integración de componentes se centran en las interacciones e interfaces entre los componentes integrados, se realizan después de las pruebas de componentes, y generalmente son automatizadas, hacen parte del proceso de integración continua.
b) Las pruebas de integración de sistemas se centran en las interacciones e interfaces entre sistemas, paquetes y microservicios, se pueden realizar después de las pruebas del sistema o en paralelo con las actividades de prueba del sistema en curso.

Pruebas de sistema

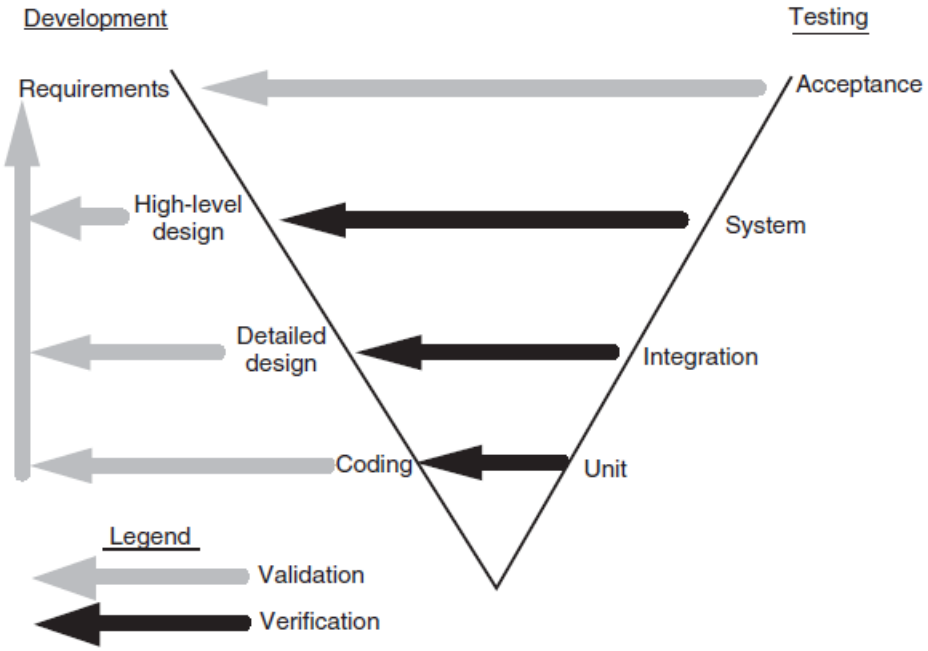
Se centra en el comportamiento y las capacidades de todo un sistema o producto, consideran las tareas de extremo a extremo que se pueden realizar en el sistema y los comportamientos no funcionales que se muestran al realizar esas tareas.

Pruebas de aceptación

Las pruebas de aceptación se enfocan en el comportamiento y las capacidades de todo un sistema o producto, ayudan a verificar los comportamientos funcionales y no funcionales del sistema, y se realizan con el usuario final donde se utilizan técnicas como alfa y beta las cuales son técnicas donde se realizan pruebas con grupos de usuarios pequeños y más grandes respectivamente.

En la Figura 2-5 se encuentra el modelo clásico en V, en el cual se pueden apreciar los niveles de las pruebas anteriormente descritos, en el lado izquierdo y en derecho las fases del ciclo de vida del software, donde se relaciona la verificación.

Figura 2-5: Fases de desarrollo y niveles de pruebas en el modelo en V
(Naik *et al*., 2008)



Aunque los procesos ágiles e iterativos son cada vez más utilizados, el modelo V sigue siendo el principal caballo de batalla y el principal modelo de referencia en la verificación de sistemas y software (Nielsen, 2014).

Sommerville (2011) considera que la verificación comprueba si el producto cumple con los requisitos funcionales y no funcionales especificados.

De acuerdo a la *ISTQB*, las pruebas funcionales y no funcionales se definen de la siguiente forma:

Pruebas funcionales

Las pruebas funcionales especifican lo que el sistema debería hacer.

Se deberían ejecutar en todos los niveles de las pruebas ya que consideran el comportamiento del software, por lo que se pueden usar técnicas de caja negra para derivar condiciones de prueba y casos de prueba para la funcionalidad del componente o sistema.

Pruebas no funcionales

Especifican que tan bien se comporta un sistema de software.

Evalúan las características y subcaracterísticas de calidad de los sistemas y el software, como la usabilidad, la eficiencia de rendimiento o la seguridad. Estas características y subcaracterísticas se plasman en la norma ISO/IEC 25010 (ISO/IEC, 2014).

Pruebas de caja Negra

Se basan únicamente en el comportamiento de las entradas y salidas del software.

Pruebas de caja blanca

Se basan en la estructura interna o la implementación del sistema. La estructura interna puede incluir código, arquitectura, flujos de trabajo y/o flujos de datos dentro del sistema

Otras actividades de verificación incluyen varios análisis estáticos y dinámicos, inspecciones de código y de documentos, tutoriales y otras técnicas. En el *SWEBOK* se indica que para determinar si un sistema de software satisface los requisitos establecidos, se pueden usar varias técnicas para controlar la calidad de software, las cuales pueden clasificarse de muchas maneras, pero un enfoque sencillo utiliza sólo dos categorías, las técnicas dinámicas y técnicas estáticas (Bourque *et al*., 2014).

La verificación abarca las técnicas dinámicas y las técnicas estáticas, a continuación, se definen resumidamente de la siguiente forma:

Pruebas dinámicas

Son las pruebas de software que requieren la ejecución del elemento de prueba.

Las pruebas dinámicas consisten en más que "solo" ejecutar elementos de prueba ejecutables, también incluyen actividades de preparación y actividades de seguimiento (ISO/IEC 29119-1, 2013).

En la revisión sistemática de literatura se pueden encontrar más clases de pruebas interesantes que se utilizan hoy en día como las pruebas TDD (Test Driven Development, por sus siglas en inglés), A-TDD (Acceptance Test Driven Development, por sus siglas en inglés), BDD (Behauvior Driven Development, por sus siglas en inglés) y muchas más, estas clases de pruebas se mencionan de nuevo en esta Tesis de maestría en una de las mejores prácticas llamada detección temprana de errores y defectos, estas pruebas se pueden revisar más a fondo como una técnica de verificación por lo cual se consideran como trabajo futuro sobre pruebas de software al igual que el proceso de validación de sistemas y software.

Pruebas estáticas

Son las pruebas de software en que un elemento de prueba se examina con un conjunto de criterios de calidad sin que el código se ejecute.

En las pruebas estáticas también se pueden incluir el uso de herramientas de análisis estático con las cuales se detectan defectos en el código o en los documentos sin que se ejecute el código (por ejemplo, un compilador o un analizador de seguridad para el código).

Las pruebas dinámicas y estáticas se complementan al encontrar diferentes tipos de defectos (ISTQB, 2018). Las pruebas dinámicas son necesarias, pero no suficientes para proporcionar una seguridad razonable de que el software funcionará según lo previsto. Las actividades de pruebas estáticas adicionales, como las revisiones por pares y el análisis estático, deben realizarse en combinación con actividades de pruebas dinámicas eficaces (ISO/IEC 29119-1, 2013).

Las técnicas estáticas se clasifican en cinco tipos de acuerdo al estándar para las revisiones y auditorias de software ISO/IEC 1028 (2008):

a) Revisiones de gestión
b) Revisiones técnicas
c) Inspecciones
d) Walkthroughs
e) Auditorias

Revisiones de gestión

Una evaluación sistemática de un producto o proceso de software realizado por o en nombre de la administración que monitorea el progreso, determina el estado de los planes y cronogramas, confirma los requisitos y la asignación de su sistema, o evalúa la efectividad de los enfoques de administración utilizados para lograr la adecuación para el propósito.

Revisiones técnicas

Una evaluación sistemática de un producto de software realizado por un equipo calificado que examina la idoneidad del producto de software para su uso previsto e identifica las discrepancias de las especificaciones y estándares.

Inspecciones

Un examen visual de un producto de software para detectar e identificar anomalías de software, incluidos errores y desviaciones de las normas y especificaciones.

Inspecciones de software

Técnicas destinadas a verificar de forma sistemática los artefactos de software no ejecutables con la intención de encontrar tantos defectos como sea posible, tan pronto como sea posible.

Auditorias

Un examen independiente de un producto de software, proceso de software o conjunto de procesos de software realizado por un tercero para evaluar el cumplimiento de las especificaciones, estándares, acuerdos contractuales u otros criterios.

Walkthroughs

Una técnica de análisis estático en la que un diseñador o programador dirige a los miembros del equipo de desarrollo y otras partes interesadas a través de un producto de software, y los participantes hacen preguntas y comentarios sobre posibles anomalías, violaciones de los estándares de desarrollo y otros problemas.

2.2 Semat (Software engineering method and theory)

Semat es una iniciativa que fundan en septiembre de 2009 Ivar Jacobson, Bertrand Meyer y Richard Soley, cuando perciben la necesidad de cambiar la forma en que la gente trabaja con métodos de desarrollo de software (Jacobson *et al*., 2013).

El dominio de la Esencia son los métodos de la ingeniería del software. Utiliza una arquitectura simple en capas que se muestra en la Figura 2-6, donde un método es una composición simple de prácticas que se describen utilizando el núcleo de la Esencia y el lenguaje de la Esencia. El núcleo y el lenguaje permiten que una práctica se fusione de una manera segura con otras prácticas relevantes para formar un método de “alto-nivel”.

El enfoque del núcleo se basa en establecer una base común para la definición de prácticas de desarrollo de software que permita definir y aplicar las prácticas independientemente. Las prácticas se pueden combinar para crear métodos de ingeniería de software específicos adaptados a las necesidades específicas de una comunidad, proyecto, equipo u organización de ingeniería de software específico (OMG, 2018).

El marco común lo componen los métodos, las prácticas, el núcleo y el lenguaje los cuales conforman la Esencia. Véase la Figura 2-6.

Figura 2-6: Arquitectura del método.
Adaptado de (OMG, 2018).

Los conceptos claves para entender la arquitectura del método son los siguientes (OMG, 2018):

- **Método:** Es una composición de prácticas. Los métodos no son sólo descripciones que leen los desarrolladores, los métodos son dinámicos e incluyen las actividades diarias.
- **Práctica:** Es un enfoque repetible para hacer algo con un objetivo específico en mente. Una práctica proporciona una manera sistemática y verificable de abordar un aspecto particular del trabajo en cuestión. Una práctica puede ser parte de muchos métodos.

En la Esencia de la ingeniería del software se menciona que una práctica se usa para describir cómo manejar un aspecto específico de un esfuerzo de ingeniería de software, incluyendo las descripciones de todos los elementos relevantes necesarios para expresar la orientación de trabajo deseada que se requiere para lograr el propósito de la práctica. Una práctica también se puede definir como una composición de otras prácticas (OMG, 2018).

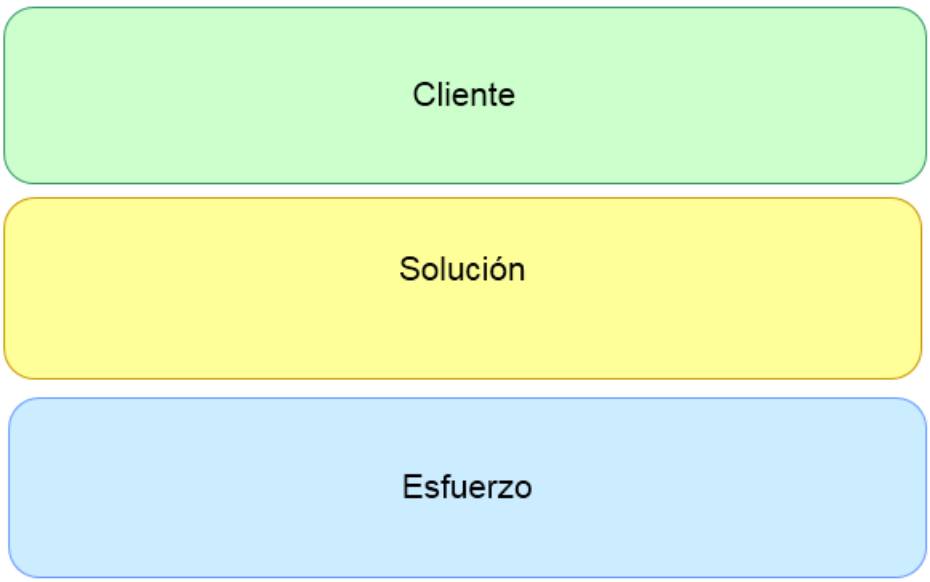
Esta Tesis de Maestría se enfoca en las prácticas de desarrollo, específicamente en las prácticas de evaluación de productos de sistemas y software.

2.2.1 Áreas de interés

El núcleo de *Semat* se organiza en tres áreas de interés, que se enfocan en aspectos específicos de la IS y se diferencia cada área con un color:

- **Cliente:** Contiene todo lo concerniente con el uso y explotación del sistema de software que se produce y se distingue con el color verde.
- **Solución:** Contiene todo lo concerniente a la especificación y el desarrollo del sistema de software y se distingue con el color amarillo.
- **Esfuerzo:** Contiene todo lo concerniente al equipo, y a la forma en que se realiza el trabajo y se distingue con el color azul. (OMG, 2018).

Figura 2-7: Áreas de interés
Adaptado de (OMG, 2018).

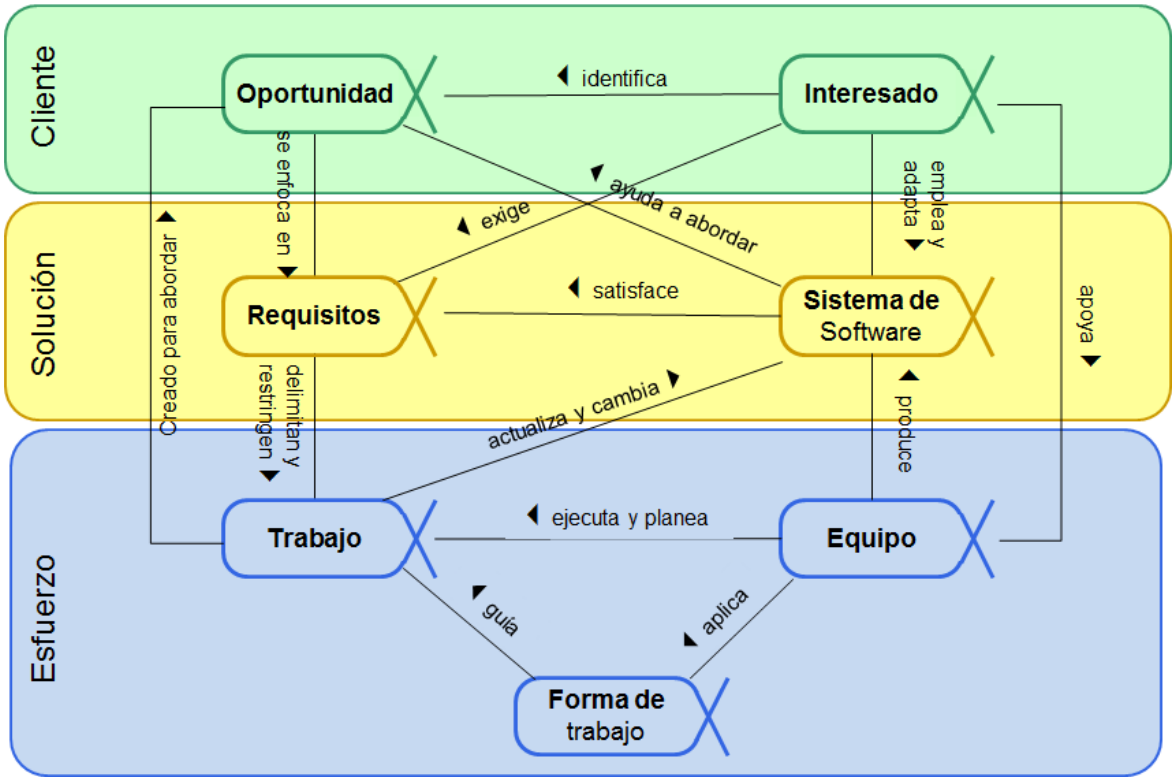


Los elementos que definen el núcleo de *Semat* son los alfas, los espacios de actividad y las competencias.

2.2.2 Alfas

Los alfas (por sus siglas en inglés *Alpha, Abstract -Level Progress Health Attribute*). Mediante los alfas se pueden capturar los conceptos clave involucrados en la ingeniería de software, permiten el seguimiento y la evaluación del progreso y la salud de cualquier esfuerzo de ingeniería de software y proporcionan el terreno común para la definición de métodos y prácticas de ingeniería de software (OMG, 2018). También se pueden definir como las “cosas con las que se trabaja” (Jacobson *et al*., 2013). (véase la Figura 2-8).

Figura 2-8: Alfas del núcleo
(Jacobson et al., 2013)



En el área de interés cliente, el equipo necesita comprender a los interesados e identificar las oportunidades, por lo que contiene los alfas interesados y oportunidad.

Alfa interesado: "Las personas, grupos, u organizaciones que afectan o se afectan por un sistema de software". Los interesados proveen la oportunidad y son la fuente de los requisitos, el equipo de trabajo también hace parte de los interesados.

Alfa oportunidad: "Conjunto de circunstancias que hace apropiado el desarrollo o el cambio de un sistema de software". Con el alfa oportunidad se articula la razón para construir un sistema de software. Representa la comprensión del equipo de las necesidades de las partes interesadas y ayuda a configurar los requisitos para el nuevo sistema de software al proporcionar una justificación para su desarrollo.

En el área de interés solución, el equipo necesita establecer un entendimiento compartido de los requisitos y construir, probar, implementar y dar soporte a un sistema de software que cumpla con los requisitos establecidos, por lo que contiene los alfas requisitos y sistema de software.

Alfa requisitos: "Lo que el sistema de software debe hacer para abordar la oportunidad y satisfacer a los interesados." Se debe entender lo que se necesita del sistema de software, comunicarse con los interesados y los miembros del equipo, e impulsar el desarrollo y las pruebas del nuevo sistema.

Alfa sistema de software: "Un sistema que se compone por software, hardware y datos que proporcionan su valor principal mediante la ejecución del software." El sistema de software brinda solución a un problema determinado y satisface los requisitos que exige el interesado.

En el área de interés esfuerzo se forma el equipo y la forma como se trabaja, y se realiza el trabajo para llegar al objetivo planeado, por lo que esta área contiene los alfas equipo, trabajo y forma de trabajo.

Alfa equipo: “Un grupo de personas que participan activamente en el desarrollo, mantenimiento, entrega o soporte de un sistema de software específico.”

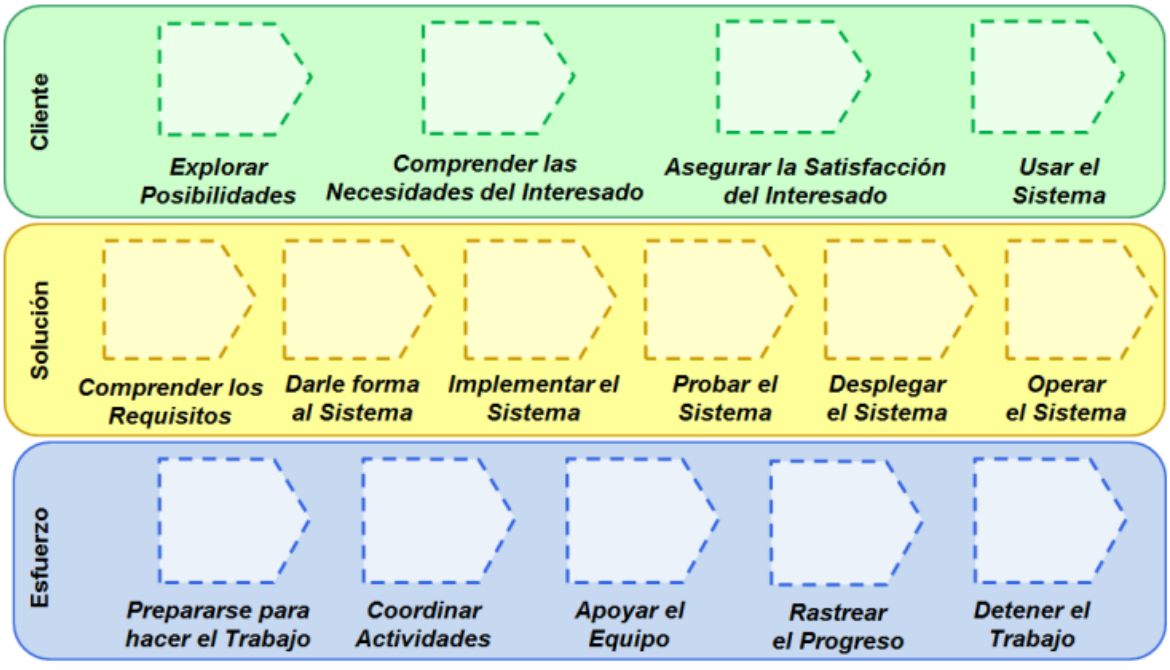
Alfa trabajo: “Actividad que involucra esfuerzo mental o físico para lograr un resultado.”

Alfa forma de trabajo: “El conjunto personalizado de prácticas y herramientas que utiliza un equipo para guiar y apoyar su trabajo.”

2.2.3 Espacios de actividad

El núcleo de *Semat* también proporciona un conjunto de espacios de actividad que complementan a los alfas para proporcionar una vista de las actividades esenciales en los esfuerzos de la ingeniería de software (OMG, 2018), estos espacios de actividad también se llaman “las cosas que se hacen” (Jacobson *et al*., 2013). En la Figura 2-9 se presentan los espacios de actividad.

Figura 2-9: Espacios de Actividad
(Jacobson et al., 2013)



El área de interés cliente contiene cuatro espacios de actividad los cuales involucran la oportunidad y los interesados.

Explorar las posibilidades

Explorar las posibilidades que se presentan por la creación de un sistema de software nuevo o que se mejora. Esto incluye el análisis de la oportunidad que se aborda y la identificación de las partes interesadas.

Comprender las necesidades del interesado

Involucrarse con las partes interesadas para comprender sus necesidades y asegurar que se produzcan los resultados correctos. Esto incluye identificar y trabajar con los representantes de las partes interesadas para avanzar en la oportunidad.

Asegurar la satisfacción del interesado

Compartir los resultados del trabajo de desarrollo con las partes interesadas para obtener la aceptación del sistema que se produce y verificar que la oportunidad se aborde con éxito.

Usar el sistema

Observar el uso del sistema en un entorno operativo y cómo beneficia a los interesados.

El área de interés solución contiene seis espacios de actividad los cuales involucran los alfas requisitos y sistema de software.

Comprender los requisitos

Establecer una comprensión compartida de lo que debe hacer el sistema a que se crea.

Darle forma al sistema

Darle forma al sistema para que sea fácil de desarrollar, cambiar y mantener, y pueda hacer frente a las demandas actuales y futuras esperadas. Esto incluye el diseño general y la arquitectura del sistema que se producirá.

Implementar el sistema

Construir un sistema implementando, probando e integrando uno o más elementos del sistema. Esto incluye la corrección de errores y las pruebas unitarias.

Probar el sistema

Verificar que el sistema que se produce cumpla con los requisitos de las partes interesadas.

Desplegar el sistema

Tomar el sistema probado y ponerlo a disposición para su uso fuera del equipo de desarrollo.

Operar el sistema

Apoyar el uso del sistema de software en producción.

El área de interés esfuerzo contiene cinco espacios de actividad los cuales involucran los alfas equipo, forma de trabajo y trabajo.

Prepararse para hacer el trabajo

Establecer el equipo y su entorno de trabajo. Comprender y comprometerse a completar el trabajo.

Coordinar actividades

Coordinar y dirigir el trabajo del equipo. Esto incluye toda la planificación continua y la planificación del trabajo, y la adición de los recursos adicionales necesarios para completar la formación del equipo.

Apoyar el equipo

Ayudar a los miembros del equipo para que se ayuden entre ellos mismos, colaborar y mejorar su forma de trabajar.

Rastrear el progreso

Medir y evaluar el progreso realizado por el equipo.

Detener el trabajo

Detener el esfuerzo de ingeniería de software y entregar las responsabilidades del equipo.

2.2.4 Competencias

El núcleo de *Semat* también proporciona un conjunto de competencias que complementan los alfas y los espacios de actividad para proporcionar una visión de las capacidades clave requeridas y el conocimiento para llevar a cabo el trabajo de IS. En la Figura 2-11 se presentan las competencias.

Figura 2-10: Competencias.
Adaptado de (OMG, 2018)

El área de interés esfuerzo contiene dos competencias.

Liderazgo

Esta competencia permite a una persona inspirar y motivar a un grupo de personas para lograr una conclusión exitosa de su trabajo y cumplir sus objetivos.

Gestión

Esta competencia encapsula la capacidad de coordinar, planificar y rastrear el trabajo realizado por un equipo.

La competencia de gestión es la capacidad administrativa y organizativa que permite hacer lo correcto en el momento adecuado para maximizar las posibilidades de éxito de un equipo.

El área de interés solución contiene tres competencias

Análisis

Esta competencia encapsula la capacidad de comprender las oportunidades y sus necesidades relacionadas con las partes interesadas, y transformarlas en un conjunto acordado y consistente de requisitos.

La competencia de análisis es la capacidad deductiva de comprender la situación, contexto, conceptos y problemas, identificar soluciones apropiadas de alto nivel, y evaluar y sacar conclusiones aplicando el pensamiento lógico.

Desarrollo

Esta competencia encapsula la capacidad de diseñar y programar sistemas de software efectivos siguiendo los estándares y normas acordados por el equipo.

La competencia de desarrollo es la capacidad mental de concebir y producir un sistema de software, o uno de sus elementos, para una función o fin específico. Permite a un equipo producir sistemas de software que cumplan los requisitos.

Pruebas

Esta competencia encapsula la capacidad de probar un sistema, verificando que sea utilizable y que cumpla con los requisitos.

La competencia de prueba tiene habilidades de observación, comparación, investigación y destrucción que permite probar el sistema.

El área de interés cliente contiene una competencia.

Representación del interesado

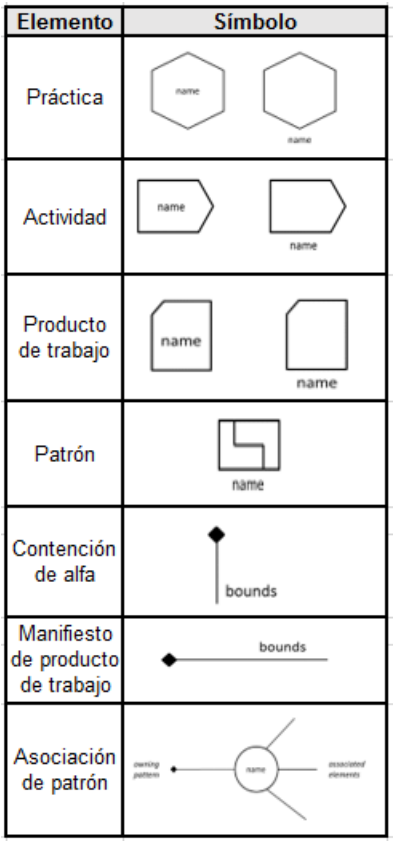
Esta competencia encapsula la capacidad de reunir, comunicar y equilibrar las necesidades de otras partes interesadas, y representar con precisión sus puntos de vista. La competencia de representación de las partes interesadas es la capacidad empática de representar y reflejar con precisión las opiniones, derechos y obligaciones de otras partes interesadas.

2.2.5 Otros elementos del núcleo de *Semat*

La sintaxis gráfica del lenguaje de *Semat* incluye otros elementos como métodos, prácticas, actividades, patrones, productos de trabajo y asociaciones los cuales ayudan a representar gráficamente las prácticas y describir o comparar los métodos.

- **Práctica:** Es un enfoque sistemático para realizar actividades. Ayuda a describir cómo deben realizarse las actividades de ingeniería de software.
- **Actividad:** Define uno o más tipos de elementos de trabajo y brinda orientación sobre cómo realizarlos.
- **Producto de trabajo:** Es un resultado de actividad. Además, ayuda a proporcionar evidencia para los estados alfas, por ejemplo, documentos o una pieza de software.
- **Patrón:** Es una descripción de una estructura en una práctica, por ejemplo, roles, fases e hitos.
- **Contención de alfa:** Una asociación de un alfa se visualiza como una línea continua para conectar dos alfas. La línea puede tener uno o más segmentos.
- **Manifesto producto de trabajo:** Conectar un alfa con un producto de trabajo. Esta conexión se representa con una línea horizontal con un diamante relleno en la punta.
- **Asociación de patrón:** Conecta el patrón con sus elementos asociados. Se representa con una línea sólida y un diamante en la punta en la cual se conecta el patrón. La línea tiene origen en un círculo en el que se pone el nombre de la asociación y del que se conectan los elementos asociados al patrón con una línea sólida por cada elemento.

Figura 2-11: Sintaxis gráfica de elementos del núcleo de la Esencia de Semat. Adaptado de (OMG, 2018)



3. Antecedentes

En este Capítulo se presentan varias técnicas de verificación de sistemas de software que se expresan en lenguaje natural y que se representan gráficamente. Estas técnicas de verificación son relevantes encontradas en la revisión sistemática de literatura de diferentes técnicas de VSS de acuerdo con su aplicabilidad e importancia. También se realiza una discusión sobre las técnicas de VSS revisadas, se plantea el problema de investigación sobre las mejores prácticas de VSS encontradas entre las técnicas revisadas en esta Tesis de maestría, se plantea una hipótesis, los objetivos de la Tesis, alcances del trabajo, se identifican las mejores prácticas de verificación de sistemas de software encontradas entre las técnicas de VSS revisadas en esta Tesis y se identifican los elementos principales de estas técnicas susceptibles a representar con el lenguaje de la Esencia de *Semat*.

3.1 Verificación de sistemas de software expresadas en lenguaje natural

En esta Sección se presentan hallazgos asociados a propuestas de verificación de sistemas de software que se expresan en lenguaje natural.

3.1.1 Verificación formal automática basada en análisis estático

Moving Fast with Software Verification (Calcagno *et al*., 2015)

Con esta técnica de VSS se verifica el comportamiento correcto de una aplicación de software mediante un análisis para detectar errores de forma estática, antes de que el código se envíe a producción; esta técnica de verificación de software la utilizan en Facebook, donde el software

de alta calidad es importante; se utilizan herramientas automáticas que permiten analizar de forma estática el código, se implementan y ejecutan de manera continua para verificar las propiedades seleccionadas de cada modificación de código en las aplicaciones móviles de Facebook; éstas incluyen las principales aplicaciones de Facebook para Android e iOS, Facebook *messenger*, Instagram y otras aplicaciones.

La forma de desarrollo de software en Facebook es continua y perpetua, lo cual quiere decir que nunca termina, es rápida, iterativa e incremental. En Facebook se introducen nuevos cambios en el código dos veces al día.

El proceso de desarrollo de Facebook es el siguiente:

- El programador desarrolla una nueva característica o hace algún cambio en el código base.
- Mediante el sistema de control de código, el cambio va un proceso de revisiones por pares que realizan otros ingenieros. El autor del cambio recibe sugerencias sobre mejoras o solicitudes de cambios adicionales de los revisores.
- Por lo tanto, el autor y los revisores inician un ciclo de iteraciones destinadas a hacer que el código cambie de manera robusta y eficiente, además de ser comprensibles, legibles y mantenibles por otros.
- Cuando los revisores están satisfechos, "aceptan" el cambio de código y el cambio se puede enviar mediante del sistema de control de fuente a la base de código principal Facebook.
- Cada dos semanas se congela una versión del código base en la versión candidata. El lanzamiento del código candidato entra en período de pruebas, al ponerlo a disposición de los empleados de Facebook para uso interno. Durante este período, los empleados realizan retroalimentaciones, las cuales ayudan a corregir errores que se manifiestan en el tiempo de ejecución.
- Después de dos semanas de uso interno, la versión candidata se implementa para los usuarios, primero con una pequeña fracción de usuarios y, si no genera ninguna alerta, finalmente se implementa para todos los usuarios realizando pruebas de aceptación.

Las pruebas de regresión se ejecutan automáticamente y, antes de aceptar cualquier cambio de código, un revisor requiere que todas las pruebas pasen. Las pruebas se ejecutan de forma asíncrona y los resultados están disponibles automáticamente en la herramienta de colaboración *phabricator* utilizada en la revisión por pares.

La herramienta de automatización de análisis estático llamada INFER se ejecuta y el proceso es completamente automático. Una vez que el código se envía para revisión por pares, un análisis se ejecuta de forma asincrónica en uno de los centros de datos de Facebook y los resultados se informan en *phabricator* en forma de comentarios. Con INFER se insertan comentarios en las líneas de código donde detecta un posible error. Además, en Facebook se ayudan con otras herramientas para navegar por el rastreo de errores y facilitar que el desarrollador inspeccione el informe de errores. Para proporcionar comentarios útiles sobre errores, se desarrolló un sistema de detección de errores en los diferentes cambios.

3.1.2 Verificación de requisitos de software embebido para naves espaciales

Confidence in Spacecraft Software: Continuous Process Improvement in Requirements Verification (Mirantes *et al*., 2014)

En verificación de requisitos de software embebido durante el desarrollo de software científico para naves espaciales, en los resultados de sus análisis y pruebas se realizan listados de lecciones aprendidas y de recomendaciones para futuras misiones espaciales. Se realiza la verificación de requisitos independientes para lograr altos niveles de confianza en la calidad del software de naves espaciales y se proporciona mejora continua del proceso de verificación de requisitos de software independiente para apoyar futuras misiones de naves espaciales.

Los autores resumen las lecciones aprendidas, la eficiencia de costos y la efectividad de la verificación independiente de los requisitos de software para naves espaciales en cuatro misiones de naves espaciales de la NASA. El resultado del estudio proporciona recomendaciones para mejorar la forma en que se llevan a cabo la verificación de requisitos en la construcción de software para naves espaciales.

La verificación de los requisitos funcionales la debe realizar un equipo de ingenieros de prueba que sean independientes del desarrollo del código. Un equipo independiente formado por varios ingenieros para pruebas desarrolla los diseños de prueba de verificación de requisitos de naves espaciales. Los ingenieros para pruebas realizan las ejecuciones de pruebas utilizando un entorno de simulación de prueba con emuladores de instrumentos científicos y utilizando una variedad de herramientas de prueba de software. En la siguiente lista se presentan las lecciones aprendidas en el estudio de verificación de requisitos de software de naves espaciales (Mirantes *et al*., 2014).

Lecciones aprendidas:

- Invertir en testing: formación, herramientas, comunicaciones.
- Aumentar el enfoque en la rentabilidad
- Utilizar métricas para la mejora de procesos.
- Administrar recursos
- Alcance del esfuerzo
- Involucrar al equipo de desarrollo.
- Aprovechar todos los esfuerzos de pruebas
- Rastrear cambios para probar documentaciones y herramientas
- Plan de prueba
- Plan tiempo de “espera”

En la siguiente lista se presentan recomendaciones del estudio de verificación de requisitos del software de naves espaciales para futuras misiones espaciales (Mirantes *et al*., 2014).

Recomendaciones para misiones futuras:

- Involucrar a probadores experimentados en gran medida en la revisión de requisitos, y hacer su membresía en el panel de revisión de requisitos obligatorio.
- Revisar cuidadosamente los requisitos para garantizar la capacidad de prueba y para filtrar la información de diseño extraña.

- Realizar un análisis de riesgos para determinar si las áreas funcionales se probarán mediante pruebas de escenario o pruebas tradicionales basadas en requisitos.
- Construir pruebas basadas en escenarios de forma iterativa e incremental.
- Mantener todos los planes de prueba en la herramienta de seguimiento de requisitos, vincularlos a los requisitos y capturar los métodos de prueba.
- Utilizar los escenarios como base de la prueba de regresión: usar la automatización para los casos que se incluirán en la regresión; y hacer la prueba restante manualmente.
- Construir y revisar casos de prueba y documentación de forma incremental. Rastrear los cambios usando un sistema de control de versiones.
- Reunir y usar métricas a lo largo de los ciclos de prueba para permitir ajustes dinámicos del proceso cuando sea necesario.
- Planificar la reutilización cuando se desarrolle casos de prueba y se busque formas de simplificarlos.
- Utilizar una matriz de verificación y métodos de prueba para identificar dónde se pueden aprovechar otros esfuerzos de prueba para reducir la duplicación en todas las configuraciones de software.

Una valiosa lección aprendida de las misiones fue invertir más en la capacitación de un grupo central de ingenieros de prueba e involucrarlos en las fases tempranas del proceso de desarrollo (por ejemplo, requisitos).
El equipo central del proyecto para la verificación de requisitos de naves espaciales consistió en cuatro ingenieros de prueba. Además del equipo central, había un líder técnico y dos ingenieros de prueba a tiempo parcial.

El líder de la prueba técnica para el programa de verificación de revisión del software espacial Van Allen Probes estuvo muy involucrado en la fase de revisión de requisitos y sirvió como miembro de la junta de revisión por pares para la revisión de requisitos y la revisión crítica de diseño.

El software para naves espaciales se considera un software crítico y, por lo tanto, requiere que todos los requisitos sean verificados por un equipo de verificación independiente, se utiliza un equipo de verificación interno para verificar los requisitos de software para naves espaciales para cada versión incremental antes de su entrega para su uso durante las pruebas de integración.

Las especificaciones de los casos de prueba se realizaron por revisión por pares para su aprobación y cada caso de prueba se asignó al menos a un requisito del software de naves espaciales. Adoptar esta estrategia de prueba demostró ser eficaz para encontrar problemas y cumplir con el presupuesto planificado y los hitos del cronograma.

El proceso de desarrollo del software de naves espaciales requiere que la documentación de la verificación de requisitos (plan de prueba y especificaciones de prueba) sea revisada por pares.

Un objetivo del programa de verificación de requisitos de software de naves espaciales es agregar valor al programa general de verificación y validación de software de naves espaciales.

Un aspecto importante del proceso fue aprovechar la verificación realizada para reducir la duplicación de esfuerzos. Muchos *scripts* fueron desarrollados y utilizados para pruebas de componentes y de sistema. La reutilización de estos scripts existentes, al extender los *scripts*, fue esencial para una verificación de revisión de software de naves espaciales rentable y eficiente. Otra ventaja del conjunto de pruebas de regresión fue la automatización.

3.1.3 Revisiones y auditorías técnicas

Technical Reviews: Framing the Best of the Best Practices (Holdaway, 2009)

Las revisiones son técnicas estáticas de VSS, los autores comparten mejores prácticas para las revisiones de documentación técnica. Estas prácticas se pueden usar para cambiar las actitudes acerca de las revisiones técnicas para beneficiar a los revisores, a los desarrolladores de información y, en última instancia, a los clientes que dependen de la documentación técnica para aprender sobre tecnologías y completar tareas con productos de software.

La documentación del producto debe ser precisa y completa. Los desarrolladores de información, necesitan el aseguramiento de la calidad que proviene de las revisiones técnicas de expertos

comprometidos en la materia capacitados para proporcionar revisiones técnicas. Obtener las revisiones de calidad (o cualquier revisión) requiere crear marcos que fomenten actitudes positivas y productivas sobre las revisiones técnicas, evocar palabras positivas utilizando los marcos de los desarrolladores de información y los revisores, y mantener esos marcos para reforzar las mejores prácticas para las revisiones técnicas. Este marco posiciona las revisiones técnicas como colaboraciones en lugar de un conjunto de prescripciones correctivas para el contenido incorrecto.

En el proceso de revisión técnica, para los desarrolladores de información, la revisión técnica es de importancia crítica para determinar si el contenido es preciso y completo.

La base de todas las mejores prácticas para las revisiones técnicas exitosas es establecer la revisión técnica no solo como una actividad central de desarrollo de productos, sino como una actividad satisfactoria y de colaboración. Para asegurarse de que el marco tenga una mayor resonancia entre los revisores, debe desarrollarse mucho antes del inicio de la revisión como parte de la planificación del proyecto. Se formula un acuerdo para evitar problemas de comunicación con los ingenieros. Se tiene en cuenta el tono de colaboración en lugar de suponer que no se proporcionará cooperación.

Desarrolladores de información:

- Asistir a reuniones de planificación de sprint de ingeniería, reuniones de entrega de sprint y retrospectivas.
- Colaborar en el texto de la interfaz de usuario con los ingenieros antes de agregar el texto a la interfaz de usuario.
- Fortalecer las relaciones laborales con expertos en la materia realizando reuniones.
- Documentar las características específicas del sprint al final de un sprint y enviarlas a revisar al comienzo del próximo sprint.

Ingenieria:

- Revisar las características específicas del sprint tan pronto como se reciben del desarrollador de información
- Revisar documentos completos durante el tercer al último sprint.

- Identificar el impacto potencial de la documentación de las nuevas funciones y proporciona los nombres de los temas de los expertos en la materia que participan
- Presentar nueva información a los desarrolladores de información tan pronto como se planifica, en lugar de esperar hasta que se implemente
- Permitir el acceso a la interfaz de usuario.

Tanto los desarrolladores de información como los ingenieros:

- Comunicar lo que han hecho hoy, lo que harán mañana, lo que se avecina y cualquier problema de bloqueo.
- Capturar todas las tareas en *ScrumWorks* (una herramienta compartida de ingreso y seguimiento de tareas en línea).
- Sincronizar las fechas de entrega de manera que si las fechas de entrega del código de ingeniería se resuelven, la documentación se deslice un número igual de días.

Reforzar el marco a lo largo del proceso de revisión y más allá requiere un uso cuidadoso de las palabras en la comunicación escrita y hablada para mantener los marcos fuertes y positivos.

Proactive Reviews of Textual Requirements (Antinyan et al., 2017)

En este artículo los autores analizan que en productos de desarrollo de software grandes, la cantidad de requisitos textuales puede llegar a ser muy grande también. Cuando se entrega una cantidad tan grande de requisitos a los desarrolladores de software, existe el riesgo de que los requisitos vagos o complejos permanezcan sin que se detecten hasta el final del proceso. Para detectar dichos requisitos, las empresas realizan revisiones manuales de los requisitos. Sin embargo, las revisiones manuales requieren mucho esfuerzo y la eficiencia es baja. La revisión manual de los requisitos es una técnica que los ingenieros de software utilizan hoy en día para mitigar los riesgos.

Al identificar los desafíos asociados con una gran cantidad de requisitos, los ingenieros de la industria encuentran que el soporte de herramientas tiene un papel crucial para un proceso de revisión eficiente. La necesidad de soporte de herramientas y automatización es algo natural a medida que crece la cantidad de requisitos. Podría haber tres beneficios principales al usar revisiones automáticas:

- Reducción sustancial del esfuerzo dedicado a revisar los requisitos
- Mitigar los riesgos técnicos de modificaciones tardías en el diseño.
- Obtener retroalimentación "justo a tiempo" de los requisitos de escritura, para facilitar el proceso de mejora

La mayoría de los miembros del equipo estaban contentos con la solución diseñada y consideraban que la herramienta era útil en su trabajo diario. La herramienta proporcionó retroalimentación instantánea sobre los requisitos de calidad interna.

La automatización de las revisiones, ahorró varios cientos de horas de tiempo de trabajo en cada lanzamiento de producto (6 meses). La adopción de la herramienta Rendex en las empresas colaboradoras dio un paso notablemente nuevo hacia revisiones proactivas de los requisitos. Además, como notaron los ingenieros de software, recibir retroalimentación instantánea sobre los requisitos recién escritos ayudó a los diseñadores de requisitos a mejorar los requisitos de manera muy eficiente, porque la retroalimentación de los requisitos escritos se recibió a tiempo.

Las actividades clave para lograr esto son involucrar a varios ingenieros en las discusiones y considerar sus comentarios. Siempre es una buena práctica sugerir el uso del método a un equipo más pequeño de la organización, que está más abierto a usar nuevos métodos y puede brindar comentarios valiosos sobre lo que se puede mejorar.
Los autores desarrollaron una herramienta (RQA) para el análisis automatizado, que se introdujo en varias organizaciones. El método ayudó a las organizaciones colaboradoras a ser proactivas en el proceso de revisión.

Software Engineering - Architecture-Driven Software Development **(Schmidt, 2013)**

El autor menciona varias clases de revisiones dentro de las cuales la mayoría de ellas se realizan en pares. Algunas de estas revisiones se realizan a pruebas de componentes y de integración, cada componente debe someterse a una revisión de preparación de integración, se asegura que cada unidad o componente de implementación que participe en la integración haya pasado satisfactoriamente la revisión de calidad, se evalúa el enfoque de integración y se evalúan los procedimientos de prueba de integración de componentes. La revisión debe ser realizada por un miembro senior del equipo de implementación de software con la asistencia de un representante del equipo de Ingeniería de Software.

3.1.4 Inspecciones de documentación técnica

Persuading Software Development Teams to Document Inspections: Success Factors and Challenges in Practice **(Komssi *et al*., 2010).**

Las inspecciones son técnicas estáticas de VSS, la técnica de revisión en pares más formal, la inspección, es efectiva para el descubrimiento de defectos en los documentos. Las revisiones en pares se reconocen como una de las mejores prácticas en ingeniería de requisitos. Este artículo describe los factores de éxito y los desafíos involucrados en persuadir a los equipos de revisión para que documenten las inspecciones.

El modelo de inspección tradicional comienza con las etapas de planificación y lanzamiento. Mientras que la etapa de planificación se centra en cuidar los detalles prácticos y preparar la estrategia de inspección, la etapa de lanzamiento se concentra en compartir información y aclarar las tareas de los revisores. Los revisores realizan una verificación individual. Su objetivo es encontrar la cantidad máxima de defectos en el documento al verificarlo contra las reglas de inspección y la lista de verificación inicialmente proporcionada en la reunión inicial.

Se proponen reuniones para generar una lluvia de ideas sobre las posibles causas de los defectos más dañinos y encontrar posibles mejoras. Se llegó a la conclusión que la planificación de la inspección y el análisis causal eran más creativos y agradables como reuniones cara a cara que como registro de defectos.

Además, la participación de ingenieros en el trabajo de adaptación fue una forma útil de obtener ideas para mejoras. En consecuencia, por ejemplo, se puso a prueba el momento oportuno de las inspecciones y la creación de reglas de inspección específicas del contexto. El resultado del uso de inspecciones personalizadas se identificó como uno de los factores clave en el éxito de los proyectos de software.

Este articulo recomienda involucrar a los equipos de desarrollo de software en el trabajo de adaptación para obtener ideas de mejora para el proceso y las técnicas de inspección.

Automatic Checking of Quality Best Practices in Software Development Documents **(Dautovic *et al*., 2011)**

Este artículo presenta un enfoque basado en herramientas que facilita el proceso de inspección del software para determinar los defectos de las mejores prácticas de documentación en los documentos de desarrollo de software. Mediante estudio empírico realizado y se muestra cómo este enfoque basado en herramientas ayuda a facilitar las tareas de inspección y a apoyar la recopilación de información sobre la calidad de los documentos que se inspeccionan.

Se presenta un enfoque basado en herramientas que trata de identificar posibles defectos de calidad del documento. Este análisis basado en herramientas se basa en las mejores prácticas para la documentación del software. Además, para proporcionar soporte para la metodología en el nivel lingüístico de las especificaciones de requisitos, presentan herramientas que analizan la calidad de especificaciones de requisitos, se describen otras herramientas que también admiten el análisis automático de documentos de requisitos de lenguaje natural.

La herramienta analiza las propiedades del documento, verifica automáticamente las referencias cruzadas dentro del documento bajo inspección, así como del documento bajo inspección a otros documentos relacionados. Los estudios empíricos muestran que el soporte de herramientas puede aumentar significativamente el rendimiento del proceso general de inspección de software, se pueden utilizar herramientas de inspección de software para facilitar las tareas de detección de defectos, La herramienta de detección automática de defectos es capaz de encontrar defectos de documentación significativos descubiertos adicionales que los inspectores humanos habían pasado por alto.

3.1.5 Verificación de software embebido

A Review on Verification and Validation for Embedded Software **(Espinosa-Bedoya *et al*., 2016)**

Los autores de este artículo proporcionan una revisión sobre la verificación y validación de software embebido. Un software embebido contiene un microprocesador y un software para realizar determinadas funciones.

La detección temprana de defectos implica la utilización de metodologías y herramientas apropiadas, la racionalización y eficacia de la automatización de pruebas y depuración con el fin de evitar la pérdida de calidad, costo y tiempo.

Se referencia la norma ISO/IEC/IEEE 1012 la cual establece actividades para llevar a cabo el proceso de verificación. Se clasifican en actividades comunes (Gestión de la verificación, soporte de adquisiciones, planeación de suministros, planeación del proyecto), actividades de sistema, actividades de software y actividades de hardware.

La verificación se realiza en paralelo con todas las etapas del ciclo de la vida, no sólo al Final. Los autores nombran técnicas estáticas y dinámicas de verificación donde se encuentran pruebas de componentes y pruebas de Integración.

Para realizar las pruebas de componentes se ejecuta un módulo específico y se verifican los atributos funcionales del módulo, una vez que los módulos se prueban individualmente, se integran para ver cómo funcionan y si se comunican bien con los otros módulos, realizando las pruebas de integración. La automatización de las pruebas de regresión se considera para que sean efectivas.

La etapa de requisitos es crucial en la reducción temprana de defectos en los sistemas embebidos, la especificación debe ser inspeccionada en una etapa temprana del ciclo de vida de desarrollo del producto, antes de usarla para el diseño de software. Las técnicas estáticas de verificación de software han probado ser efectivas en la identificación prematura de defectos.

Se analizan retos en el equipo de verificación y validación con relación a sistemas embebidos críticos en seguridad. Algunos de estos retos son: conocimiento inadecuado de los procesos, carencia de datos en campos electromagnéticos, comportamiento humano en interfaz hombre-máquina, entre otros.

Los estudios muestran que el enfoque principal se centra en la fase de requisitos en busca de una detección temprana de defectos. La generación de modelos formales, verificación automática, chequeo de modelos y pruebas de seguridad son aspectos abiertos a investigación.

Testing embedded software: A survey of the literature **(Garousi *et al*., 2018).**

En este artículo se revisan una gran cantidad de técnicas de prueba, enfoques, herramientas y *frameworks* que proponen en la industria y en la academia para probar software embebido de manera efectiva y eficiente. Se escogen las técnicas correctas de pruebas para software embebido.

Se analiza que el software embebido generalmente se desarrolla en paralelo con el hardware, puede haber solo unas pocas muestras del hardware recientemente desarrollado, lo que afecta el alcance de los esfuerzos de los equipos de prueba.

El equipo de verificación utiliza varios enfoques de prueba basados en simulación en la industria del software embebido, por ejemplo, pruebas de Modelo en bucle (MIL), Software en bucle (SIL), Procesador en bucle (PIL) y pruebas de hardware en el bucle (HIL).

Los niveles de prueba que se utilizaron en este estudio fueron, pruebas unitarias, de integración y de sistema. Al centrarse en las pruebas unitarias, se aplicó el desarrollo basado en pruebas (TDD) al software embebido.

Las actividades de pruebas que se utilizaron fueron planeación y gestión de las pruebas, diseño de casos de pruebas, automatización de pruebas, ejecución de pruebas, evaluación de pruebas, codificación de pruebas, reporte de resultados de pruebas, entre otras.

Los artefactos generados por las técnicas propuestas son requisitos de casos de prueba, entradas de casos de prueba (valores), salidas esperadas, código de prueba, entre otros.

Se nombran las pruebas automatizadas para sistemas automotrices embebidos a partir de modelos de especificación de requisitos.

3.1.6 Verificación de aplicaciones móviles

Mobile Application Verification: A systematic Mapping Study **(Sahinoglu *et al*., 2014)**

En este artículo se presenta un mapeo sistemático de la verificación de software en el campo de las aplicaciones móviles. Las pruebas de software son una técnica de verificación que se requiere

para lograr un sistema más confiable y de calidad, es la técnica de verificación de software más utilizada. Se resumen estudios existentes en la verificación de aplicaciones móviles.

Los autores realizan un análisis de los desafíos en esta área. indican la necesidad de investigación en la automatización de pruebas de aplicaciones móviles. Los niveles de prueba más estudiados son las pruebas de componentes, de integración, sistema y aceptación.

Mobile Testing in Software Industry using Agile: Challenges and Opportunities **(Santos *et al*., 2015)**

En este artículo, se comparten y discuten los mayores desafíos más comunes que enfrentan el desarrollo y las pruebas de aplicaciones móviles en un entorno ágil, desde el punto de vista de la calidad del producto con la colaboración de todo el equipo. Para este propósito, se utilizó un proyecto real, donde a través de informes de la experiencia, se proporciona la oportunidad de conocer algunas prácticas utilizadas en el desarrollo y las pruebas de aplicaciones móviles.

Las demandas específicas y las características de los dispositivos deben considerarse en el desarrollo de la aplicación, el producto debe tener una alta calidad desde el principio, teniendo en cuenta las variaciones existentes de los dispositivos. Se proponen muchas técnicas, métodos y herramientas para realizar las pruebas en software para dispositivos móviles, en estos casos es importante tener en cuenta las características de ambos: dispositivos y software para obtener un producto de calidad.

Se debe seleccionar correctamente los métodos y técnicas de prueba para garantizar que se alcance el nivel de calidad que los usuarios esperan. La mayoría de los equipos deben crear o adaptar su propia estrategia en función de su situación única. Si el equipo ágil utiliza TDD (desarrollo basado en pruebas), por ejemplo, y automatiza gran parte de las pruebas de unidad y regresión, el equipo no puede llevar a cabo integraciones completas del sistema, rendimiento a gran escala y pruebas de seguridad sin ayuda. Para este proyecto, se combinaron pruebas funcionales y unitarias.

Las pruebas durante todo el proceso en lugar de al final del proyecto pueden brindar una serie de beneficios para los desarrolladores de aplicaciones especialmente en el descubrimiento de fallas de codificación u otros errores. Cuando las pruebas se realizan de forma regular durante el proceso de desarrollo, los desarrolladores de aplicaciones y los ingenieros de pruebas detectan fácilmente los problemas en el camino y los corrigen a medida que avanzan. Esto asegura que el proyecto continuará sin problemas y que cualquier problema se corregirá más fácilmente.

Además, al introducir la automatización de pruebas temprano y conectar las pruebas funcionales, de carga y rendimiento a la integración continua como parte de las pruebas de regresión, el equipo de desarrollo se responsabiliza de los errores en el código.

3.1.7 Pruebas de software

***Challenges and Practices in Aligning Requirements with Verification and Validation: A Case Study of Six Companies* (Bjarnason *et al*., 2013)**

Los autores realizan un caso de estudio realizado en seis empresas de software, donde se identifican los métodos y prácticas utilizadas. Se hace énfasis en la importancia de una coordinación efectiva entre la ingeniería de Requisitos con la verificación de sistemas y software lo cual cubre la coordinación entre diferentes roles y las actividades de ingeniería de requisitos, no sólo la organización funcional a través de la alineación horizontal (diseñadores, desarrolladores, testers y documentación), sino también la alineación vertical de la responsabilidad. Esto conlleva a efectos positivos en la verificación de sistemas y software, como el mejoramiento de la cobertura de las pruebas, la gestión de riesgos, dando lugar al aumento de la productividad y la calidad del producto.

La débil alineación de la ingeniería de requisitos con la verificación puede generar problemas para entregar los productos necesarios a tiempo con la calidad adecuada. Por ejemplo, una comunicación débil de los cambios de requisitos a los probadores puede resultar en la falta de verificación de los nuevos requisitos y la verificación incorrecta de los requisitos inválidos antiguos, lo que lleva a problemas de calidad del software, esfuerzo perdido, retrasos, problemas con el tiempo de entrega, y como consecuencia la insatisfacción de las necesidades de los clientes. Cuando la ingeniería de requisitos y la VSS están alineados, pueden apoyar eficazmente las actividades de desarrollo entre la definición inicial de requisitos y las pruebas de aceptación del producto final.

El estudio identificó que los aspectos humanos son centrales, es decir, la cooperación y la comunicación. Además, se identificaron otros tres problemas relacionados con la alineación que afectan a equipos de prueba independientes, a saber, cobertura de prueba incierta, no saber si se pretende cambiar el comportamiento del software y la falta de canales de comunicación establecidos para tratar problemas y preguntas.

La alineación de Ingeniería de requisitos con verificación también cubre la coordinación entre roles y actividades de ingeniería de requisitos y verificación. La colaboración exitosa entre los requisitos y pruebas se puede lograr asignando y conectando roles tanto de los requisitos como de pruebas haciéndolos responsables para garantizar que se realice correctamente el proceso de verificación.

Los resultados de los estudios, incluyen una serie de prácticas que aumentan la comunicación e interacción entre los requisitos y los roles de prueba, como lo son, la participación temprana del probador, las políticas de trazabilidad, considerar las solicitudes de características de los probadores y vincular las personas de prueba y requisitos.

Los profesionales de las seis compañías en el estudio encontraron que la alineación de la ingeniería de requisitos con la verificación es un factor importante, pero desafiante, en el desarrollo de productos. Se consideró que la alineación de ingeniería y verificación de requisitos

afectaba todo el ciclo de vida del proyecto, desde el contacto con el cliente hasta el desarrollo del software.

Dentro de las prácticas realizadas en las empresas de software involucradas en este caso de estudio están la verificación temprana y pruebas independientes. Tener una cobertura de prueba completa con pruebas unitarias brinda una mejor seguridad.

Se mencionó que tener un proceso de verificación está directamente relacionado con una buena alineación entre los requisitos y la prueba, involucrar a los desarrolladores y probadores en detallar los requisitos es otra práctica, especialmente mencionada por las empresas.

Los probadores pueden participar para garantizar la capacidad de prueba de los requisitos, o incluso especificar los requisitos en forma de casos de prueba.

Revisiones de requisitos entre los ingenieros de requisitos y probadores es otra práctica aplicada para garantizar que los requisitos se entiendan y se puedan probar.
La mayoría de los entrevistados mencionan las revisiones de requisitos por los probadores como una buena práctica que mejora tanto la comunicación como la calidad de los requisitos.

Las prácticas para verificar que las propiedades del sistema estén alineadas con los requisitos del sistema incluyen comenzar la verificación tempranamente para permitir tiempo para retroalimentación y cambios, usar equipos de prueba independientes, reutilizar retroalimentación de los clientes obtenidos de proyectos anteriores y capacitar a los probadores.

La verificación se mejora mediante actividades de verificación temprana y pruebas independientes. Los asuntos organizacionales y humanos están relacionados con varios de los desafíos identificados. La cooperación y colaboración exitosa es un asunto humano.

A survey of software testing practices in Canada (Garousi *et al*., 2013)

Los autores realizan un estudio que se centra en métodos y prácticas de pruebas de software, realizan encuestas sobre prácticas de prueba de software entre practicantes en Canadá. Los resultados de la encuesta revelan hallazgos importantes e interesantes sobre las prácticas y métodos de pruebas de software. También se analizan resultados de otras encuestas realizadas por otros autores sobre prácticas y métodos de pruebas de software.

Algunos de los hallazgos son los siguientes: las pruebas funcionales y unitarias son dos tipos de pruebas comunes que reciben la mayor atención y los esfuerzos dedicados a ellas, el estilo tradicional de Desarrollo de Prueba al final (TLD) sigue dominando y algunas compañías están intentando los nuevos enfoques de desarrollo como el Desarrollo Dirigido por Prueba (TDD) y el Desarrollo Conducido por el Comportamiento (BDD), en términos de las herramientas de prueba más populares, las herramientas de prueba de aplicaciones web y *NUnit* superaron a las herramientas *JUnit* e IBM *Rational*, la mayoría de las empresas canadienses dedicaron menos del 40% de sus esfuerzos (presupuesto y tiempo) a las pruebas durante el desarrollo, se analizaron pruebas de integración, herramientas de automatización, pruebas de regresión las cuales son las que más se automatizan, pruebas de componentes, los autores identificaron dos tipos de automatización de pruebas: automatización de datos grabados y automatización de *scripts*, en otras encuestas también se encontraron que los asuntos más investigados fueron sobre pruebas automatizadas y herramientas para realizarlas.

Los autores concluyen que las pruebas unitarias y funcionales del sistema son los dos tipos de pruebas más comunes, los enfoques de prueba de caja negra son más populares que las pruebas de caja blanca.

***Software Testing Practices in Industry: The State of the Practice* (Kassab *et al*., 2016).**

En este artículo los autores realizaron una encuesta a más de 167 profesionales de software para determinar el uso de diversas prácticas y métodos de pruebas de software. Los datos incluyen características de proyectos, prácticas, empresas y profesionales relacionados con las pruebas de software. A los encuestados, que informaron sobre actividades de pruebas de software, también se les hizo una serie de preguntas sobre técnicas y herramientas en uso dentro de sus proyectos.

Los resultados de las encuestas demostraron que las pruebas de software más utilizadas fueron pruebas de componentes, pruebas de integración, pruebas de sistema, pruebas de aceptación y pruebas de regresión, los autores también encontraron que las pruebas automatizadas se incrementaron desde el 2004. Otra área de investigación de pruebas es cómo construir un equipo de pruebas efectivo.

3.2 Verificación de sistemas de software representadas gráficamente

Existen diferentes representaciones gráficas de verificación de sistemas de software, en esta Tesis de maestría se abarcan algunas de las que se consideran más relevantes:

3.2.1 Verificación formal de sistemas en *chips*

***System On Chip (SoC) Design and Test* (Bhunia *et al*., 2019)**

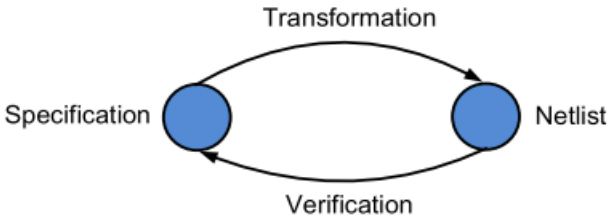
SoC (*System on Chip*) es un sistema completo embebido en un solo *chip* (IEEE Std.1500, 2005). Los SoC se usan ampliamente en varias aplicaciones debido a su alta densidad funcional y bajo consumo de energía (Bhunia *et al*., 2019), como por ejemplo los procesadores de computadores.

SoC es un circuito integrado con todos los componentes necesarios para un sistema. Por lo general, SoC incluye señales analógicas, digitales y mixtas de núcleos de propiedad intelectual (IP Core) (Bhunia *et al*., 2019). Un núcleo es un bloque de circuito prediseñado que se puede probar como una unidad individual (IEEE Std.1500, 2005). IP (Diseño electrónico de propiedad intelectual) es un término utilizado en la comunidad de diseño electrónico para referirse a una colección reutilizable de especificaciones de diseño que representan el comportamiento, las propiedades y/o la representación del diseño en diversos medios (IEEE Std.1734, 2011).

Flujo de verificación de SoC

La verificación de SoC, se refiere principalmente al proceso para garantizar la corrección funcional y la transformación correcta de las especificaciones de diseño a la conectividad de la lista de componentes (Netlist) de un circuito electrónico, antes de salir a producción, este proceso se muestra en la Figura 3-1.

Figura 3-1: Proceso de verificación de Sistemas en chips (Bhunia et al., 2019).



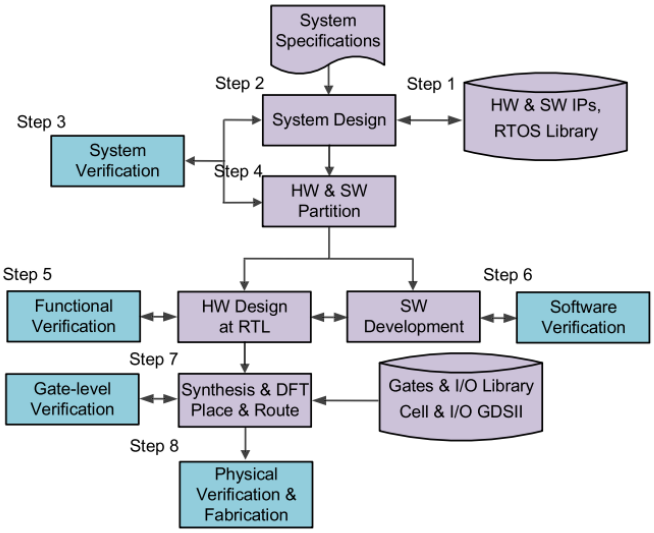
La Figura 3-1 describe el flujo de verificación de SoC utilizado en la industria. Este flujo comienza con la creación de la especificación del sistema, que, en cierta medida, determina e impulsa la estrategia de verificación.

La planificación de la verificación del *chip* se considera simultáneamente con la creación de la especificación de diseño.

En el Paso 1, todos los semiconductores IP (*Intelectual Property*) en el sistema se deben verificar antes de la integración en el sistema. El IP se verifica antes de enviarla a un integrador de SoC o a un usuario de IP. Dado que las IP se pueden entregar en diferentes formatos de varios

proveedores de IP, se requiere que el integrador vuelva a verificar la IP a través de la traducción de los archivos de diseño y bancos de pruebas a su propio entorno de aplicación.

Figura 3-2: Flujo de verificación de Sistemas en *chips* (Bhunia et al., 2019).



Después de la verificación individual de la IP, la IP adquirida debe ser envuelta con lógica adicional para comunicarse con las IP existentes. Entonces, está listo para integrarse en el SoC.

En el Paso 2, Se basa en los protocolos de interfaz, la verificación de la interfaz entre bloques en un *chip* se realiza para reducir los esfuerzos de integración final y permitir la detección temprana de errores en el sistema.

En el paso 3, se lleva a cabo la verificación del nivel de SoC. Para ser exactos, el comportamiento de un SoC se modela según su especificación y se verifica a través del banco de pruebas de simulación de comportamiento. El banco de pruebas podría describirse utilizando varios idiomas, por ejemplo, Verilog / VHDL, C / C ++. Además, se requiere que el banco de pruebas se convierta al formato que se especifica, lo cual es adecuado para la verificación de hardware y software en los siguientes pasos.

En el Paso 4, una vez que se completa la verificación a nivel del sistema, el diseño de SoC se divide en partes de software y hardware basadas en la biblioteca de IP de software y hardware. Luego, la arquitectura, incluidas las partes de software y hardware, se verifica utilizando el banco de pruebas creado en el último paso.

En el Paso 5, la verificación funcional se realiza en el diseño del hardware en el Registro del nivel de Transferencia (RTL Register Transfer Level) obtenido del último paso. La verificación de hardware utiliza el banco de pruebas creado durante el proceso de verificación del comportamiento del sistema. La verificación en RTL involucra principalmente la verificación de líneas, simulación lógica, verificación formal (es decir, verificación de equivalencia y verificación de modelos), verificación basada en transacciones y análisis de cobertura de códigos.

En el Paso 6, la verificación del software se realiza según las especificaciones del sistema. La verificación de software y la integración de hardware / software (HW/SW se pueden ejecutar con diferentes métodos, que incluyen prototipo blando, prototipo rápido, emulación y co-verificación de HW/SW. La co-verificación de HW/SW se requiere para SoC con núcleos de tipo de procesador. Durante este proceso, la integración y verificación de HW/SW se produce simultáneamente. La co-simulación se realiza para acoplar los simuladores de hardware actuales con emuladores/depuradores de software, lo que permite que el software se ejecute en el diseño de hardware. A su vez, el diseño del hardware se proporciona y se estimula con el estímulo real. Por lo tanto, esta co-verificación reduce los esfuerzos para crear el banco de pruebas de hardware y permite una integración más temprana de hardware y además, proporciona una mejora significativa del rendimiento para la verificación del sistema.

En el Paso 7, el diseño del registro del nivel de transferencia RTL se sintetiza con la biblioteca de tecnología para generar el nivel de puerta para la conectividad de la lista de componentes del circuito. La conectividad de la lista de componentes del circuito se verifica mediante la herramienta de verificación de equivalencia formal para garantizar que el diseño del registro del nivel de transferencia RTL sea lógicamente equivalente con la conectividad de la lista de componentes del circuito.

En el Paso 8, la verificación se lleva a cabo en el diseño físico del circuito integrado para garantizar que el diseño cumpla con ciertos criterios. Esto implica la verificación de reglas de diseño. Cualquier violación debe ser resuelta antes de que el chip sea fabricado. Una vez que se completa la verificación de la disposición física, el diseño está listo para el cierre.

3.2.2 Verificación de software basado en la arquitectura

Software Engineering - Architecture-Driven Software Development
(Schmidt, 2013)

El material presentado en este libro proporciona un conjunto de prácticas de ingeniería de software que se basan en la ingeniería de sistemas. Este material propone "mejores prácticas" populares que se fomentan debido a la falta de una forma perfecta de diseñar software.

Al adaptar las prácticas de ingeniería de sistemas, este libro presenta un enfoque integral para diseñar un producto de software mediante el establecimiento de principios y prácticas rigurosas de ingeniería de software Estas prácticas se aplican durante el proceso de desarrollo de software para controlar, revisar y administrar la arquitectura de software en un contexto típico de proyecto de desarrollo de software. Los contenidos de este libro están alineados con el Cuerpo de Conocimientos de Ingeniería de Software (SWEBOK).

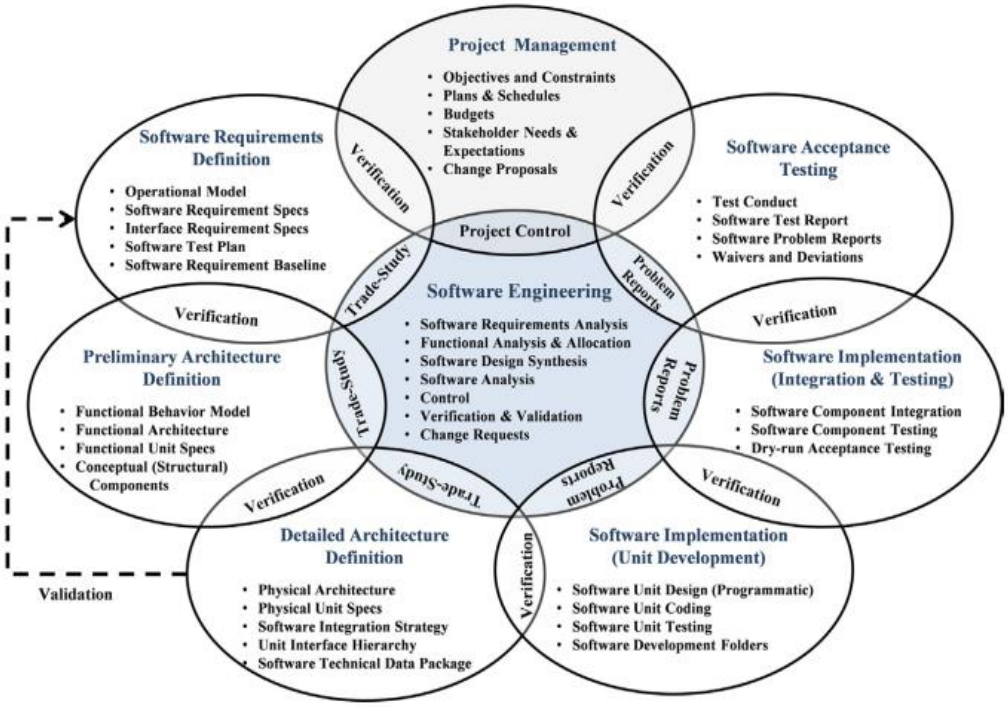
El capítulo 15 de este libro trata explícitamente sobre las prácticas de verificación de software, identifica las tareas específicas que se deben realizar para garantizar que las tareas de verificación deben realizarse para garantizar que la implementación del software y los esfuerzos de prueba y evaluación estén sincronizados con las especificaciones de la arquitectura del software y la documentación de diseño.

El autor propone que el enfoque general para ejecutar el desarrollo integrado de productos y procesos es formar equipos multidisciplinarios para todos los productos y procesos posteriores al desarrollo para abordar problemas técnicos, equilibrar los requisitos y ayudar a integrar los diversos equipos. Se propone equipos integrados donde involucran el equipo de verificación y el

equipo de desarrollo. Durante la planificación de la prueba, se correlacionan los casos y escenarios de prueba con los requisitos.

Como se puede apreciar en la Figura 3-3 la verificación se aplica en todo el ciclo de vida del modelo de desarrollo orientado en la arquitectura, donde se pueden relacionar prácticas de verificación como: planificación del proceso de verificación, realización continua de pruebas de componentes, realización continua de pruebas de integración.

Figura 3-3: Modelo de desarrollo orientado en la arquitectura de software
(Schmidt, 2013)



La verificación del software debe realizarse periódicamente para garantizar que la arquitectura de software en evolución sea coherente.

Se estudia la verificación de comportamiento funcional, el análisis de requisitos y especificación, las pruebas de software generalmente se estiman solo durante las fases de prueba. Sin embargo, la planificación de la prueba y el establecimiento de casos y procedimientos de prueba ocurren durante las primeras fases del desarrollo del software. Las pruebas de software deben reconocerse como un esfuerzo significativo, y su planificación debe comenzar lo antes posible.

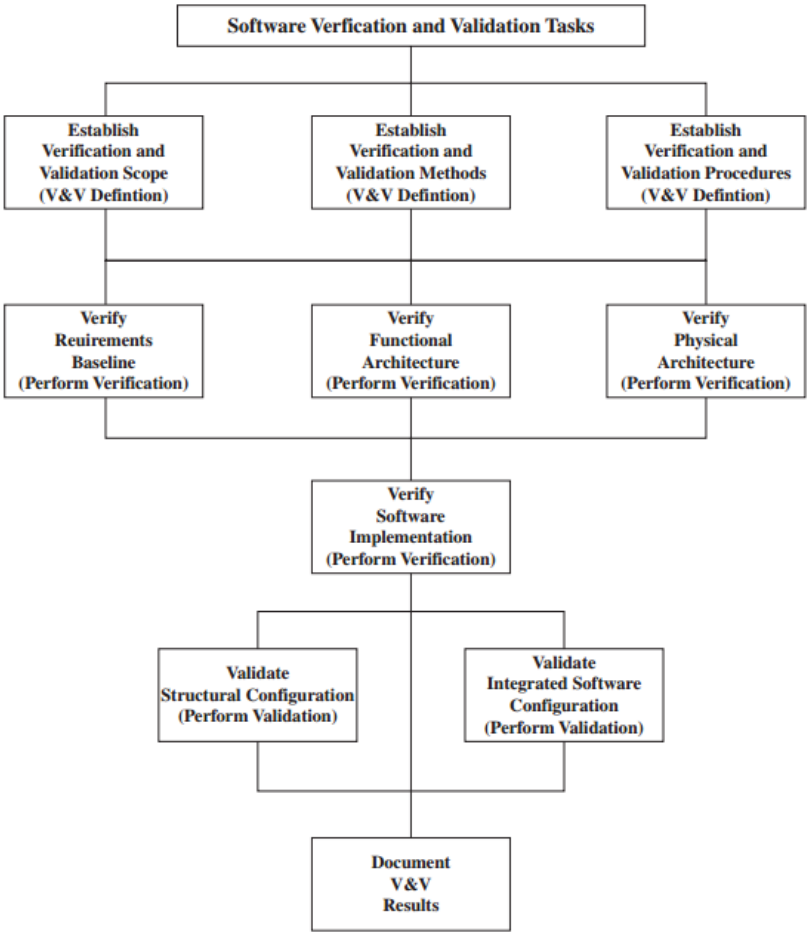
Mientras se analizan, evalúan y formalizan los requisitos de software, se deben identificar los desafíos para probar y calificar el producto de software. La sección de calificación del producto de las especificaciones del software identifica los métodos de análisis, inspección, demostración y prueba que se utilizarán para confirmar la satisfacción de los requisitos en función de los resultados de la prueba.

Se propone la verificación como una práctica, la práctica de verificación de software confirma la consistencia entre los elementos de la arquitectura del software. Esto implica la determinación de que el diseño del software, tal como lo expresa la arquitectura, se formula y se configura adecuadamente para satisfacer las necesidades de las partes interesadas.

La arquitectura del software debe verificarse continuamente para que sea internamente consistente y validada según las necesidades de los interesados. la verificación debe corroborar que el diseño del software cumplirá su propósito previsto y, en consecuencia, justifica una mayor inversión en el proyecto de desarrollo.

Se proponen unas tareas de verificación de la arquitectura véase Figura 3-4.

Figura 3-4: Tareas de verificación de la arquitectura del software
(Schmidt, 2013)



Las siguientes son tareas de verificación donde se encuentran las siguientes prácticas:

Tareas de verificación de línea de base de requisitos

Verificar las especificaciones de los requisitos de software

- Confirmar que todos los requisitos de software se pueden rastrear hasta las necesidades o expectativas legítimas de las partes interesadas.
- Confirmar que cada requisito es único, cuantificable, y comprobable.
- Confirmar que el conjunto de requisitos son colectivos consistente y no repetitivo.
- Confirmar que los requisitos de software se establecen de manera consistente con los requisitos de los registros de estudios comerciales.

Tareas de verificación de arquitectura funcional

Verificar especificaciones funcionales

- Confirmar que la unidad funcional y las especificaciones de los componentes se asimilan adecuadamente para satisfacer las especificaciones de nivel superior
- Confirmar que la unidad funcional y las especificaciones de los componentes reflejen adecuadamente las características de rendimiento derivadas de los modelos de comportamiento.

Tareas de verificación de arquitectura física

Verificar especificaciones unidad estructural

- Confirmar que las especificaciones de la unidad estructural asimilan adecuadamente las características de especificación de la unidad funcional.

Verificar la estrategia de integración de software

- Confirmar que la estrategia de integración de software cierra ingeniosamente el abismo de diseño de software.
- Confirmar que las pruebas de integración de software ejercitan adecuadamente los componentes estructurales integrados.

Tareas de verificación de implementación de software

Verificar planes de implementación de software - implementación de unidades estructurales

- Confirmar que los planes de implementación de software representan adecuadamente el esfuerzo necesario para diseñar, codificar y probar cada unidad estructural.

Verificar planes de implementación de software - cumplimiento de la estrategia de integración

- Confirmar que los planes de implementación de software representan adecuadamente el esfuerzo necesario para ensamblar, integrar y probar cada componente de la estructura.

Verificar la conformidad de la unidad de software

- Confirmar que la implementación de cada unidad de software se ajusta a su especificación estructural.

Verificar la conformidad del componente de software

- Confirmar que la implementación de cada componente de software se ajusta a su especificación estructural.

3.2.3 Análisis y pruebas en sistemas críticos de seguridad

Expressing Best Practices in (Risk) Analysis and Testing of Safety-Critical Systems Using Patterns **(Herzner *et al*., 2014).**

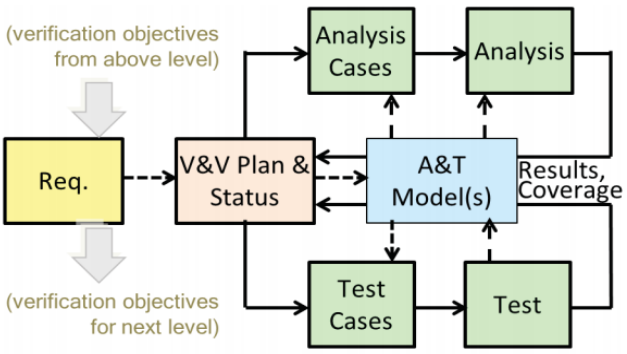
Herzner *et al*. (2014), presentan un enfoque para alcanzar mejores prácticas de VSS; utilizando un modelo avanzado que se basa en análisis y pruebas de software, denominado A & T patterns (Analysis and Test patterns, por sus siglas en inglés); y mediante el uso complementario y la combinación optimizada de técnicas y herramientas de verificación. Esta técnica de verificación se utilizó en un contexto industrial para desarrollar un proyecto llamado Artemis MBAT (*Combined Model-based Analysis and Testing of Embedded Systems*, por sus siglas en inglés); se utilizó esta técnica combinada basada en análisis (riesgo/seguridad) y pruebas de software de sistemas embebidos, se aplicaron técnicas de análisis estático y pruebas dinámicas para la verificación de sistemas físicos cibernéticos críticos para la seguridad, los cuales se aplican como por ejemplo, en el transporte automotriz ferroviario y aeroespacial. Recopilando, describiendo, e interrelacionando los patrones de análisis y pruebas hay un gran potencial para capturar las mejores prácticas y experiencias valiosas para soluciones avanzadas de VSS.

Un patrón es una solución para cierto problema en un contexto dado. En el proyecto ARTEMIS MBAT, se desarrolló una metodología para combinar el análisis y la prueba (basados en modelos). Una parte esencial de esta metodología fue la descripción detallada de soluciones genéricas para problemas y problemas específicos, para lo cual se desarrolló el concepto de "Patrones de flujo de trabajo de análisis y prueba".

Un patrón de análisis y pruebas es un flujo de trabajo utilizado para combinar análisis y pasos de prueba para satisfacer requisitos complejos, mejorar la eficiencia, ahorrar esfuerzo, etc. Además del diagrama de flujo de trabajo, la descripción de un patrón de análisis y prueba contiene información adicional que no sólo ayuda a aplicar el patrón, sino que también sirve para evaluar rápidamente si un determinado patrón puede ayudar a resolver o mitigar un cierto problema, describiendo un flujo de trabajo eficiente, integrando análisis de riesgo y seguridad, utilizando técnicas adecuadas de análisis, pruebas y verificación de riesgo, también se proporciona una forma de relacionar diferentes patrones, haciéndolos más fáciles de identificar y usar, un buen patrón define una mejor práctica que puede ser re-usada en varias situaciones (Herzner *et al*., 2014).

En la Figura 3-5 se describe esta idea general de integración de análisis integrados de riesgos, pruebas y análisis.

Figura 3-5: Método de combinación de análisis de riesgos y pruebas de software (Herzner et al. 2014).



Los ingenieros desarrollan un plan inicial de verificación que define cómo y con qué técnicas se aborda cada requisito para el componente considerado. Esto incluye una evaluación inicial de riesgos (probabilidad de defectos), que guiará el esfuerzo requerido y la elección de las técnicas de verificación.

Con base en este plan inicial, los ingenieros definen y ejecutan una serie de análisis y casos de prueba que resultan en un conjunto de veredictos (aprobado, reprobado, no terminado), medidas de cobertura, confianza, complejidad, etc., y posibles indicios de comportamiento sospechoso que requieren una mayor investigación.

El flujo es el siguiente:

- Los requisitos formales se crean sobre la base de los requisitos del lenguaje natural.
- El modelo de implementación del sistema se crea en base a los requisitos formales. Incluso si el flujo de trabajo pudiera comenzar alternativamente con los requisitos del lenguaje natural, se hace hincapié en los requisitos formalizados para evitar la ambigüedad semántica del lenguaje natural.
- Extender el modelo de sistema nominal con comportamiento de mal funcionamiento utilizando técnicas de inyección de fallas, lo que resulta en un modelo de sistema extendido.
- Se genera un observador de fallas de los requisitos de seguridad analizados. Este autómata de observación se utilizará más adelante en el nivel del análisis para controlar si el sistema ha alcanzado un estado de falla.
- Se crea una tarea de análisis de seguridad y se vincula a modelo extendido y el observador de fallas.
- Se elige el tipo de tarea de análisis de seguridad.
- Se ejecuta el análisis de seguridad basado en el modelo.
- Finalmente, un árbol de fallas es generado dependiendo del tipo de análisis elegido.
- Se evalúa los resultados y los resultados de la violación de los requisitos de seguridad (causada por los modos de falla inyectados).
- El ingeniero de sistemas modificará el diseño de acuerdo con los resultados del análisis mediante la introducción de mecanismos de seguridad adicionales.

3.2.4 Verificación de modelos

***Best Practices for Verification, Validation, and Test in Model-Based Design* (Murphy et al., 2008).**

Los autores proponen un diseño basado en modelos los cuales se utilizan para verificar, validar y probar un diseño de manera temprana y continua durante todo el proceso de diseño. Esto mejora la capacidad de un equipo para implementar un sistema embebido de alta calidad a tiempo en comparación con los métodos tradicionales, que se basan en la verificación, validación y pruebas al final del proceso. Se presentan un conjunto de mejores prácticas: desarrollar pruebas de modelo con el diseño, probar exhaustivamente en la simulación, y usar todas las técnicas disponibles. La aplicación de estas mejores prácticas ayuda a garantizar un proceso de desarrollo que aproveche al máximo el diseño basado en modelos para proporcionar una verificación, validación y prueba más rigurosas.

Se mencionan desafíos en el desarrollo de sistemas integrados críticos para la seguridad, muchos de los cuales se encuentran en aplicaciones automotrices y aeroespaciales. El diseño basado en modelos ayuda a abordar los desafíos del desarrollo de sistemas embebidos.

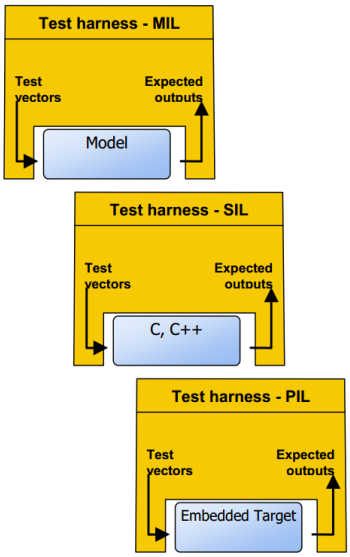
Uno de los aspectos más valiosos del diseño basado en modelos es la disponibilidad de modelos ejecutables para realizar verificación, validación y pruebas durante todo el proceso de desarrollo, especialmente en sus primeras etapas.

Los autores mencionan que en empresas que intentan garantizar un proceso de desarrollo riguroso utilizando el diseño basado en modelos necesitan definir y establecer actividades de verificación y pruebas efectivas, especialmente al principio del proceso de desarrollo.
Este documento presenta un conjunto de mejores prácticas que se encuentran en varias empresas con procesos rigurosos.

Los grupos de desarrollo que utilizan técnicas de simulación desarrollan y aplican conjuntos de pruebas a los modelos y luego reutilizan estos conjuntos para probar la implementación del

software. El conjunto completo puede ejecutarse repetidamente en simulación. Este proceso se denomina prueba de modelo en bucle (MIL) Models in the loop.

Figura 3-6: Desarrollo pruebas para ejecutar contra el modelo
(Murphy et al., 2008).



Como se muestra en la Figura 3-6, la mejor práctica es reutilizar las pruebas desarrolladas para las pruebas MIL contra la implementación de software del modelo en el entorno de desarrollo.

Esta práctica, conocida como cosimulación del código o software-in-the-loop (SIL), proporciona la confianza de que el código coincide con el comportamiento deseado desarrollado y especificado en el modelo. Debido a que las pruebas son las mismas en las pruebas MIL y SIL, es más fácil comparar el resultado de ambas pruebas y replicar cualquier error que se identifica.

Por lo general, los ingenieros elaboran el conjunto de pruebas utilizadas en SIL, agregando pruebas que no se pueden realizar en las pruebas MIL. Además de garantizar que no hay cambios en el comportamiento de una etapa a otra, las pruebas SIL se utilizan para garantizar que el paso de desarrollo no introduce errores. Este paso de cosimulación se denomina prueba de procesador en bucle (PIL).

Una piedra angular del diseño basado en modelos es hacer que las pruebas continuas formen parte del proceso de diseño. En este proceso, los componentes y subsistemas se construyen, prueban, integran con sistemas más grandes y se prueban nuevamente. Este enfoque expone los defectos en la interfaz entre componentes o subsistemas lo antes posible. Al integrar varios subsistemas, es importante tener un conocimiento profundo de los requisitos de cada sistema y componente.

3.3 Discusión

Las anteriores técnicas de verificación de sistemas de software que se expresan en lenguaje natural y que se representan gráficamente contienen prácticas en común que se utilizan en la industria y en la academia, estas prácticas en común entre las técnicas de VSS revisadas en esta Tesis de Maestría son las que se denominan mejores prácticas de VSS, las cuales buscan alcanzar sistemas de software de alta calidad. Cada mejor práctica de VSS tiene elementos principales susceptibles a representar con los elementos básicos de la Esencia de *Semat*.

Estas mejores prácticas de VSS se realizan en diferentes contextos donde se desarrollan diferentes sistemas de software con diferentes niveles de integridad, participan varios roles los cuales realizan diferentes actividades y tareas en diferentes fases en el ciclo de vida de sistemas y software, se utilizan tecnologías y herramientas de acuerdo al contexto de cada técnica de VSS, se aplican competencias no solo técnicas, sino que también blandas en el proceso de verificación.

Cada equipo de trabajo debe identificar cuál práctica y cuales actividades se acoplan mejor en su contexto, estas mejores prácticas de VSS contienen elementos comunes susceptibles a representar con el lenguaje de la Esencia de *Semat*.

3.4 Problema de investigación

Las mejores prácticas de verificación de sistemas de software no tienen un marco de trabajo estándar donde se agrupen los elementos comunes de estas mejores prácticas, no se incluyen elementos universales, y no existen representaciones gráficas en *Semat* de mejores prácticas de VSS, por ende, no se pueden combinar ni adaptar al contexto de los diversos proyectos de software que se estén desarrollando, debido a la ausencia de este marco de trabajo estándar, se dificulta identificar y utilizar buenas prácticas de verificación de sistemas y software para mejorar los procesos de verificación y desarrollo, y poder adaptar estas prácticas a la forma de trabajo de cada equipo y así ayudar a construir sistemas de software con calidad.

3.5 Hipótesis

Con las representaciones de las mejores prácticas de VSS en el núcleo de la Esencia de Semat; se pueden extraer sus elementos principales y éstas buenas prácticas de VSS se pueden combinar o adaptar cuando el contexto del proyecto de software lo amerite y a medida que los procesos evolucionen, lo que puede conllevar a un mejoramiento constante en la forma de trabajar en el proceso de verificación de sistemas de software.

En esta propuesta se abarca algunas de las vistas típicas que posibilita la separación de intereses con el núcleo de la Esencia de *Semat,* al ligar el símbolo de las prácticas con los alfas, los espacios de actividad, los productos de trabajo, las actividades, los patrones y las competencias necesarias en esta labor.

3.6 Objetivo General

Representar las mejores prácticas de VSS; empleando para ello el núcleo de la Esencia de *Semat*, de tal forma que se puedan capturar, integrar, adaptar y mejorar según el contexto del proyecto de software.

3.7 Objetivos Específicos

- Identificar las mejores prácticas de verificación de sistemas de software y caracterizarlas en términos de sus elementos principales.

- Correlacionar los elementos identificados en el objetivo anterior con los elementos básicos del núcleo de la Esencia de *Semat.*

- Crear los diagramas que permitan representar las mejores prácticas de verificación de sistemas de software, mediante un marco común donde se agrupen dichas prácticas con base en la sintaxis del núcleo de la Esencia de *Semat*.

- Validar la representación obtenida por medio de un caso de laboratorio.

3.8 Identificación de las mejores prácticas de verificación de sistemas de software y sus elementos principales

En esta sección se presentan las técnicas de verificación de sistemas de software revisadas con sus respectivas referencias, se identifican las mejores prácticas de VSS y también se identifican los elementos principales de las mejores prácticas de VSS.

3.8.1 Técnicas de verificación de sistemas de software revisadas.

En la Tabla 3-1 se presentan las técnicas de verificación de sistemas de software revisadas con sus respectivas referencias y nombradas de acuerdo con su definición.

Tabla 3-1: Técnicas de VSS revisadas y sus referencias
Elaboración propia.

Referencias	Técnica de verificación
(Calcagno *et al*., 2015)	Verificación formal automática basada en análisis estático en Facebook
(Mirantes *et al*., 2014)	Verificación de requisitos de software embebido para naves espaciales en la Nasa
(Holdaway, 2009; Schmidt, 2013; Antinyan *et al*., 2017)	Revisiones y auditorías técnicas
(Komssi *et al*., 2010; Dautovic *et al*., 2011)	Inspecciones de documentación técnica
(Espinosa-Bedoya *et al*., 2016; Garousi *et al*., 2018)	Verificación de software embebido
(Sahinoglu *et al*., 2014; Santos *et al*., 2015)	Verificación de aplicaciones móviles
(Garousi et al., 2013; Bjarnason *et al*., 2013; Kassab *et al*., 2016;)	Pruebas de software
(Bhunia *et al*., 2019)	Verificación formal de sistemas en *chips*
(Herzner *et al*., 2014)	Análisis y pruebas en sistemas críticos de seguridad
(Murphy *et al*., 2008)	Verificación de modelos
(Schmidt, 2013)	Verificación de software basado en la arquitectura

3.8.2 Mejores prácticas de verificación de sistemas de software

Las mejores prácticas de VSS que se proponen en esta Tesis de maestría son prácticas en común entre las técnicas de VSS revisadas en la literatura.

En la Tabla 3-2 se proponen mejores prácticas de VSS que son más comunes encontradas entre las técnicas de VSS revisadas con sus respectivas referencias.

Tabla 3-2: Mejores prácticas de VSS entre las técnicas revisadas y sus referencias. Elaboración propia.

Prácticas / Referencias	Verificación continua de sistemas de software por un equipo independiente	Planificación continua del proceso de verificación	Detección temprana de errores y defectos	Retroalimentación continua entre equipo de verificación y demás miembros del equipo	Utilización continua de herramientas automáticas	Realización continua de pruebas de componentes	Realización continua de pruebas de integración	Revisión en pares de código y requisitos
(Calcagno *et al*., 2015)	X		X	X	X	X	X	X
(Mirantes *et al*., 2014)	X	X	X	X	X	X	X	X
(Holdaway, 2009; Schmidt, 2013; Antinyan *et al*., 2017)	X	X	X	X	X	X	X	X
(Komssi et al., 2010; Dautovic *et al*., 2011)	X	X	X	X	X			X
(Espinosa-Bedoya *et al*., 2016; Garousi *et al*., 2018)	X	X	X		X	X	X	
(Sahinoglu *et al*., 2014; Santos *et al*., 2015)	X	X	X	X	X	X	X	
(Garousi *et al*., 2013; Bjarnason *et al*., 2013; Kassab *et al*., 2016)	X		X	X	X	X	X	
(Bhunia *et al*., 2019)	X		X			X	X	
(Herzner *et al*., 2014)	X	X	X					
(Murphy *et al*., 2008)	X	X	X		X	X	X	
(Schmidt, 2013)	X	X	X	X		X	X	X

Al identificar las mejores prácticas de verificación de sistemas de software se cumple con el primer objetivo específico que se plantea en esta Tesis de maestría.

3.8.3 Elementos principales de las mejores prácticas de verificación de sistemas de software

A continuación, se caracterizan los elementos principales de las mejores prácticas de VSS susceptibles a representar con el lenguaje de la Esencia de *Semat.*

Los elementos principales identificados en la práctica verificación continua de sistemas de software por un equipo independiente se presentan en la Tabla 3-3:

Tabla 3-3: Elementos principales identificados en la práctica verificación continua de sistemas de software por un equipo independiente. Parte 1/2
Elaboración propia.

Actividades	Productos de trabajo	Roles
Definir el proceso de verificación	Definición del proceso de verificación	Equipo de verificación Revisores
Verificar el soporte de adquisición	Verificación del soporte de adquisición	
Verificar la planificación del suministro	Verificación de la planificación del suministro	
Verificar la planificación del proyecto	Verificación de la planificación del proyecto	
Verificar la gestión de la configuración	Verificación de la gestión de la configuración	
Revisar documentación	Revisión de la documentación	
Encontrar defectos en la documentación	Corrección de errores en la documentación	
Documentar la inspección	Documentación de la inspección	
Verificar requisitos funcionales	Calidad en los requisitos funcionales	
Diseñar las pruebas de requisitos funcionales	Diseño de pruebas de requisitos funcionales	
Ejecutar las pruebas	Satisfacción de los requisitos que se establecen	
Construir pruebas basadas en escenarios de forma iterativa e incremental	Especificación continua de criterios claros	
Analizar de forma continua y asíncrona el código	Cumplimiento continuo de los requisitos	
Sugerir mejoras en el código	Corrección en el código	
Requerir que todas las pruebas pasen	Calidad en el código	
Realizar pruebas durante todo el proceso en lugar de al final del proyecto	Aseguramiento de la calidad del producto	
Especificar los requisitos en forma de casos de prueba	Casos de prueba coherentes para verificar los requisitos	

Tabla 3-4: Elementos principales identificados en la práctica verificación continua de sistemas de software por un equipo independiente. Parte 2/2
Elaboración propia.

Actividades	Productos de trabajo	Roles
Utilizar técnica combinada de análisis y pruebas de software	Sistemas de software con calidad	Equipo de verificación Revisores
Crear casos de análisis	Casos de análisis	
Analizar los riesgos de seguridad	Análisis de riesgos de seguridad	
Crear casos de prueba	Cumplimiento de los requisitos	
Probar el sistema	Calidad del sistema demostrable	
Verificar continuamente la arquitectura del software	Satisfacción de los criterios de la arquitectura de software	
Desarrollar pruebas de modelo con el diseño	Pruebas de modelo con el diseño	
Probar la simulación	Calidad de la simulación	
Probar el modelo en Loop (MIL)	Calidad del modelo en bucle (MIL)	
Probar el Software en Loop (SIL)	Calidad del software en el bucle (SIL)	
Probar procesador en Loop (PIL)	Calidad de los procesadores en el bucle (PIL)	
Verificar los Intellectual Property	Satisfacción de los requisitos de los Intellectual Property	
Verificar la interfaz	Calidad de la Interfaz	
Verificar el Sistema en *Chip*	Sistema en Chip operable y con calidad	
Verificar el software	Software operable y con calidad	
Verificar la conectividad de la lista de componentes del circuito	Conectividad de la lista de componentes del circuito con calidad	

Los elementos principales identificados en la práctica planificación continua del proceso de verificación se presentan en la Tabla 3-5:

Tabla 3-5: Elementos principales identificados en la práctica planificación continua del proceso de verificación
Elaboración propia.

Actividades	Productos de trabajo	Roles
Desarrrollar un plan de verificación	Identificación de requisitos y estrategias	Equipo de verificación
Definir técnicas para abordar los requisitos del componente	Integración de técnicas para cumplir con los requisitos	
Evaluar los riesgos	Identificación y mitigación de riesgos	
Planificar la reutilización de casos de prueba	Reutilización de casos de prueba	
Planificar la inspección	Estrategia de inspección	
Reunirse cara a cara	Comunicación asertiva	
Seleccionar métodos y técnicas de prueba	Preparación del trabajo	
Revisar los requisitos	Requisitos coherentes y con calidad	
Planear los suministros	Planificación de los suministros	
Planear el proyecto	Planificación del proyecto	
Correlacionar los casos y escenarios de prueba con los requisitos	Mejoramiento de calidad de sistemas de software	

Los elementos principales identificados en la práctica detección temprana de errores y defectos se presentan en la Tabla 3-6.

Tabla 3-6: Elementos principales identificados en la práctica detección temprana de errores y defectos
Elaboración propia.

Actividades	Productos de trabajo	Roles
Verificar los requisitos	Requisitos coherentes	Equipo de verificación Equipo de desarrollo de software
Analizar los riesgos de seguridad	Identificación de riesgos de seguridad	
Crear casos de análisis	Casos de análisis	
Mitigar los riesgos	Control de riesgos	
Crear casos de prueba	Casos de prueba	
Probar el software continuamente	Aseguramiento de la calidad del software	
Involucrar a probadores en la revisión de requisitos	Mejoramiento en la revisión de requisitos	
Analizar de forma continua y asíncrona los cambios de código con herramienta automática	Código con calidad	
Revisar las características específicas del sprint tan pronto como se reciben	Identificación de cambios y estimación esfuerzos	
Verificar la interfaz del Sistema en *chip*	Verificación del Desempeño de Interfaz de SoC	
Desarrollar orientandose en pruebas	Desarrollo de software ágil	
Construir escenarios de prueba para cada uno de los requisitos en el diseño de arquitectura	Arquitectura con calidad	

Los elementos principales identificados en la práctica retroalimentación continua entre equipo de verificación y demás miembros del equipo se presentan en la Tabla 3-7.

Tabla 3-7: Elementos principales identificados en la práctica retroalimentación continua entre equipo de verificación y demás miembros del equipo
Elaboración propia.

Actividades	Productos de trabajo	Roles
Informar resultados del análisis estático en forma de comentarios	Código eficiente, comprensible, legible y mantenible	Equipo de verificación Revisores Desarrolladores de software Desarrolladores de información Ingenieros de requisitos
Analizar riesgos de seguridad	Mitigación de riesgos de seguridad	
Sugerir mejoras en el código	Corrección del código	
Inspeccionar informe de errores	Detección de errores	
Requerir que todas las pruebas pasen	Aceptación de cambios en el código	
Interactuar entre revisores y desarrolladores de software	Código robusto y de calidad	
Corregir errores en el código	Código correcto	
Aceptar correcciones en el código	Software funcional	
Retroalimentar en tiempo de ejecución	Corrección de errores en tiempo de ejecución	
Coordinar y colaborar con diferentes roles atraves de la alineación horizontal y vertical	Mejoramiento de la cobertura de pruebas	
Involucrar a los ingenieros de requisitos	Colaboración entre miembros del equipo	
Involucrar a los ingenieros de desarrollo	Verificación de requisitos colaborativa	
Involucrar a los revisores por parte de los desarrolladores de información	Revisiones técnicas precisas y completas	
Establecer las revisiones técnicas como colaboraciones en lugar de correcciones	Revisiones técnicas colaborativas	
Asistir a *sprints* de planificación, *daily* y retrospectivas.	Participación de los revisores y desarrolladores de información en los sprints	

Los elementos principales identificados en la práctica utilización continua de herramientas automáticas se presentan en la Tabla 3-8.

Tabla 3-8: Elementos principales identificados en la práctica utilización continua de herramientas automáticas.
Elaboración propia.

Actividades	Productos de trabajo	Roles
Introducir la automatización de pruebas temprano	Aseguramiento de la calidad	Ingeniero de automatización
Generar pruebas	Control de la calidad del software	
Implementar y ejecutar de manera continua herramienta automática	Disminución de esfuerzos y costos	
Revisar requisitos textuales con herramienta automática	Mejoramiento en la revisión de requisitos textuales	
Analizar de forma estática el código con herramienta automática	Corrección temprana de errores en el código	
Automatizar pruebas de regresión	Pruebas de regresión efectivas y reducción de esfuerzos	

Los elementos principales identificados en la práctica realización continua de pruebas de componentes se presentan en la Tabla 3-9.

Tabla 3-9: Elementos principales identificados en la práctica realización continua de pruebas de componentes.
Elaboración propia.

Actividades	Productos de trabajo	Roles
Diseñar casos de prueba	Estrategia para realizar las pruebas de componentes	Desarrollador de software
Ejecutar casos de prueba	Resultados de la ejecución de los casos de prueba	
Verificar correctitud funcional de los componentes de software	Funcionalidad de los componentes con calidad	
Requerir que todas las pruebas pasen	Aseguramiento de la calidad de los componentes	
Reportar resultados de pruebas de componentes	Documentación de pruebas	

Los elementos principales identificados en la práctica realización continua de pruebas de integración se presentan en la Tabla 3-10.

Tabla 3-10: Elementos principales identificados en la práctica realización continua de pruebas de integración.
Elaboración propia.

Actividades	Productos de trabajo	Roles
Revisar estrategia de integración de componentes	Enfoque y proceso de integración acordes	Revisores Equipo de verificación
Integrar componentes	Sistema de software funcional	
Verificar comunicación entre módulos	Comunicación entre módulos correcta	
Verificar la conectividad de la lista de componentes del circuito	Conectividad de la lista de componentes del circuito correcta	
Reportar los resultados de pruebas de integración	Documentación de pruebas de integración	

Los elementos principales identificados en la práctica revisión en pares de código y requisitos se presentan en la Tabla 3-11.

Tabla 3-11: Elementos principales identificados en la práctica revisión en pares de código y requisitos.
Elaboración propia.

Actividades	Productos de trabajo	Roles
Revisar el plan de prueba	Plan de prueba correcto	Revisores en pares
Revisar los requisitos	Requisitos correctos	
Revisar el diseño	Diseño correcto	
Especificar los casos de prueba	Asignación de requisitos de software	
Revisar la calidad del componente	Componente con calidad	
Revisar el código	Calidad en el código	
Analizar de forma continua y asíncrona el código	Inspección continua de la calidad del código	
Revisar preparación de integración de componentes	Integración de componentes efectiva	
Encontrar defectos	Corrección de defectos	
Requerir que todas las pruebas pasen	Código correcto	
Sugerir mejoras en el código	Mejoramiento continuo en el código	
Inspeccionar documentación	Documentación correcta	

Al caracterizar los elementos principales de las mejores prácticas de verificación de sistemas de software los cuales son susceptibles a representar con el lenguaje de la Esencia de Semat se cumple con el primer objetivo específico propuesto en esta Tesis de maestría.

4. Representación de las mejores prácticas de verificación de sistemas y software en el núcleo de la Esencia de *Semat*

En este Capítulo se presenta la correlación de los elementos principales que se identificaron en las mejores prácticas de VSS con los elementos básicos del núcleo de la Esencia de *Semat* y se propone la representación en el núcleo de la Esencia de *Semat* de las mejores prácticas de VSS que fueron más comunes entre las diferentes técnicas de VSS revisadas.

En la Figura 4-1 se representan en el núcleo de la Esencia de *Semat* las mejores prácticas de VSS que fueron más comunes entre las técnicas de verificación revisadas.

Figura 4-1: Representación de las mejores prácticas de verificación de sistemas y software en el núcleo de la Esencia de Semat.
Elaboración propia.

Para realizar la representación gráfica con los elementos de la Esencia del núcleo de *Semat* de las mejores prácticas de VSS más comunes entre las técnicas de verificación que se revisaron, se propone primero correlacionar los principales elementos de las prácticas de VSS con los elementos básicos del lenguaje de *Semat*, después definir las prácticas mediante los alfas, las actividades, productos de trabajo, roles y las asociaciones que se comprometen, y por ultimo representar las prácticas mediante espacios de actividad, actividades, competencias y sus asociaciones.

Al correlacionar los elementos principales de las mejores prácticas de VSS con los elementos básicos del lenguaje de *Semat* y al representar las mejores prácticas de VSS en el núcleo de la Esencia de *Semat*, se cumple con el segundo y tercer objetivo específico propuestos en esta Tesis de maestría.

4.1 Representación gráfica en el núcleo de la Esencia de *Semat* de la práctica verificación continua de sistemas de software por un equipo independiente

Esta práctica se realiza en la mayoría de técnicas de verificación revisadas en la literatura. Es importante que el proceso de verificación se realice por un equipo independiente a los demás miembros del equipo (Desarrolladores de software, los encargados de levantar requisitos, desarrolladores de información que generan la documentación, etc.).

Para la representación en *Semat* de la práctica verificación continua de sistemas de software por un equipo independiente se correlacionaron los elementos principales de esta práctica encontrados en las diferentes técnicas de VSS susceptibles a representar con los elementos de *Semat.* Dentro de las actividades de las diferentes técnicas de VSS revisadas se encuentran las actividades comunes de del proceso de VSS que se proponen en la norma IEEE 1012.

A continuación, se correlacionan los elementos principales de las mejores prácticas de VSS entre las técnicas de VSS revisadas con los elementos básicos del núcleo de la Esencia de *Semat*.

En la Tabla 4-1 y 4-2 se correlacionan los elementos principales identificados en la práctica verificación continua por un equipo independiente con los elementos básicos del núcleo de la Esencia de *Semat*.

Tabla 4-1: Correlación de los elementos principales identificados en la práctica verificación continua por un equipo independiente con los elementos básicos del núcleo de la Esencia de *Sema*t. Parte 1/2
Elaboración propia.

Práctica	Actividad	Producto de trabajo	Alfas		Espacios de actividad		Roles	
Verificación continua de sistemas de software por un equipo independiente	1) Definir el proceso de verificación	A) Definición del proceso de verificación	Interesado	B, C	Probar el sistema	2, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 17	Equipo de verificación	A,B,C,D,E, K,L,M,N,Ñ, O,P
	2) Verificar el soporte de adquisición	B) Verificación del soporte de adquisición						
	3) Verificar la planificación del suministro	C) Verificación de la planificación del suministro	Trabajo	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O, P	Comprender las necesidades del interesado	1, 2, 3		
	4) Verificar la planificación del proyecto	D) Verificación de la planificación del proyecto						
	5) Verificar la gestión de la configuración	E) Verificación de la gestión de la configuración			Explorar las posibilidades	1, 2, 3		
	6) Revisar documentación	F) Revisión de la documentación						
	7) Encontrar defectos en la documentación	G) Defectos en la documentación	Forma de trabajo	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O, P	Comprender los requisitos	1, 2, 3, 4, 6, 7, 9, 10, 12, 13, 14, 15, 16, 17	Revisores	F, G, H, I, J
	8) Documentar la inspección	H) Documentación de la inspección						
	9) Verificar requisitos funcionales	I) Verificación de los requisitos funcionales						
	10) Diseñar las pruebas de requisitos funcionales	J) Diseño de pruebas de requisitos funcionales			Apoyar el equipo	1, 2, 3, 4, 5, 8, 14,		
	11) Ejecutar las pruebas	K) Ejecución de las pruebas	Equipo	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O, P				
	12) Construir pruebas basadas en escenarios de forma iterativa e incremental	L) Construcción de pruebas						
	13) Analizar de forma continua y asíncrona el código	M) Análisis continuo y asíncrono del código			Coordinar actividades	1, 2, 3, 4, 5		
	14) Sugerir mejoras en el código	N) Corrección en el código	Requisitos	A, B, C, F, G, I, J, K, L, M, O, P	Prepararse para hacer el trabajo	1, 2, 3, 4		
	15) Requerir que todas las pruebas pasen	Ñ) Calidad en el código						
	16) Realizar pruebas durante todo el proceso en lugar de al final del proyecto	O) Realización de pruebas durante todo el proceso del proyecto	Sistema de software	K, M, N, Ñ, O	Rastrear el progreso	1, 2, 3, 4, 5		
	17) Especificar los requisitos en forma de casos de prueba	P) Requisitos en forma de casos de prueba						

Tabla 4-2: Correlación de los elementos principales identificados en la práctica verificación continua por un equipo independiente con los elementos básicos del núcleo de la Esencia de Semat. Parte 2/2
Elaboración propia.

Práctica	Actividad	Producto de trabajo	Alfas		Espacios de actividad		Roles	
Verificación continua de sistemas de software por un equipo independiente	1) Utilizar técnica combinada de análisis y pruebas de software	A) Sistemas de software con calidad	Trabajo	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O	Probar el sistema	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16	Equipo de verificación	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O
	2) Crear casos de análisis	B) Casos de análisis						
	3) Analizar los riesgos de seguridad	C) Análisis de riesgos de seguridad	Forma de trabajo	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O	Implementar el sistema	1,2 4, 7, 9, 10, 11, 12, 14, 15, 16		
	4) Crear casos de prueba	D) Cumplimiento de los requisitos						
	5) Probar el sistema	E) Calidad del sistema demostrable						
	6) Verificar continuamente la arquitectura del software	F) Satisfacción de los criterios de la arquitectura de software						
	7) Desarrollar pruebas de modelo con el diseño	G) Pruebas de modelo con el diseño	Equipo	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O	Darle forma al sistema	6, 7		
	8) Probar la simulación	H) Calidad de la simulación						
	9) Probar el modelo en Loop (MIL)	I) Calidad del modelo en bucle (MIL)						
	10) Probar el Software en Loop (SIL)	J) Calidad del software en el bucle (SIL)						
	11) Probar procesador en Loop (PIL)	K) Calidad de los procesadores en el bucle (PIL)	Requisitos	A, B, D, G, H, I, J, M, N				
	12) Verificar los Intellectual Property	L) Satisfacción de los requisitos de los Intellectual Property						
	13) Verificar la interfaz	M) Calidad de la Interfaz			Comprender los requisitos	1, 2, 4, 5, 7, 8, 9, 10, 11, 13, 14, 15, 16		
	14) Verificar el Sistema en *Chip*	N) Sistema en Chip operable y con calidad						
	15) Verificar el software	Ñ) Software operable y con calidad						
	16) Verificar la conectividad de la lista de componentes del circuito	O) Conectividad de la lista de componentes del circuito con calidad	Sistema de software	A, E, F, I, J, K, L, M, N, Ñ, O				

En las Figuras 4-2 y 4-3 se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica verificación continua de sistemas de software por un equipo independiente, mediante alfas, roles, actividades y productos de trabajo

Figura 4-2: Representación gráfica en el núcleo de la Esencia de Semat, de la práctica verificación continua de sistemas de software por un equipo independiente, mediante alfas, roles, actividades y productos de trabajo. Parte 1/2.
Elaboración propia.

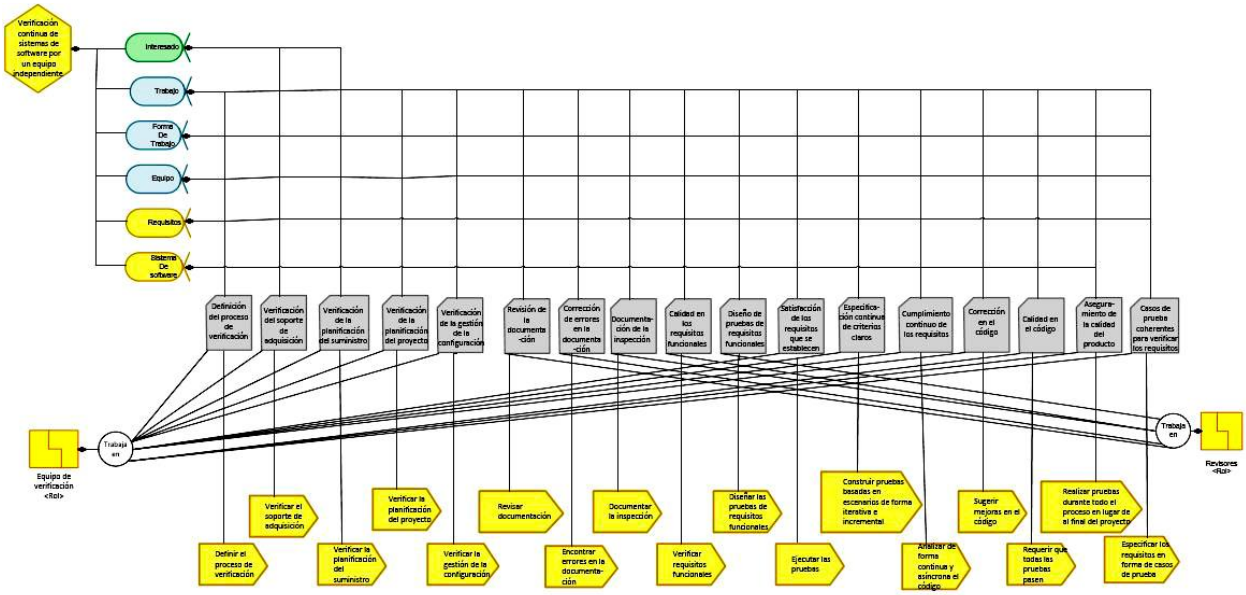
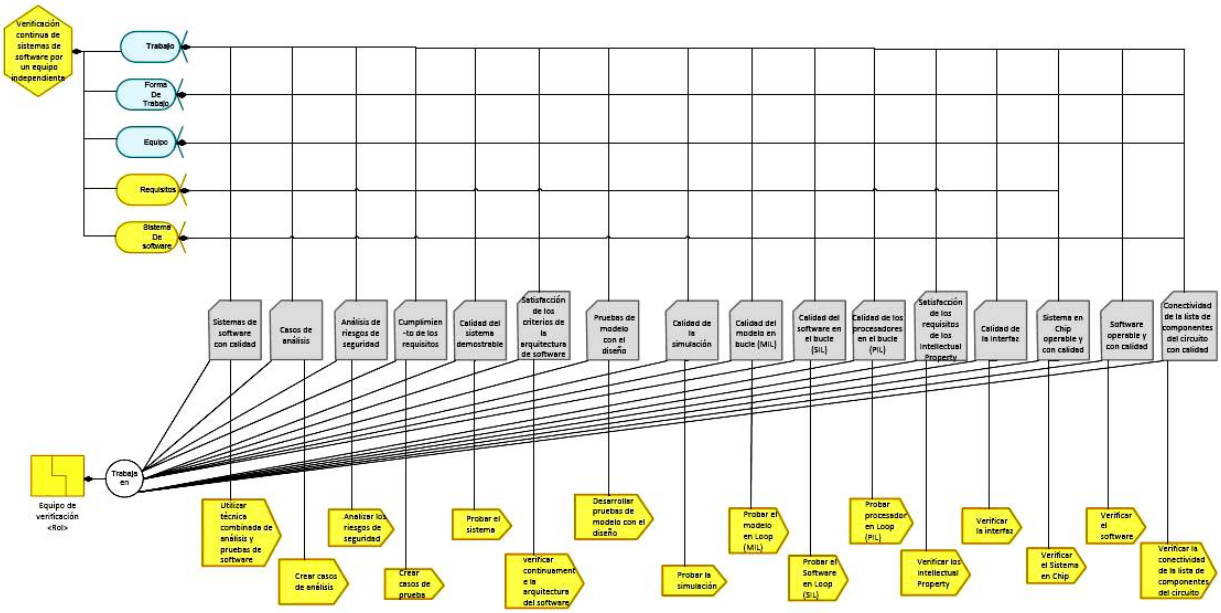


Figura 4-3: Representación gráfica en el núcleo de la Esencia de Semat, de la práctica verificación continua de sistemas de software por un equipo independiente, mediante alfas, roles, actividades y productos de trabajo. Parte 2/2.

Elaboración propia.



En las figuras 4-4 y 4-5, se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica verificación continua de sistemas de software por un equipo independiente, mediante espacios de actividad, actividades, competencias y sus asociaciones.

Figura 4-4: Representación gráfica en el núcleo de la Esencia de Semat, de la práctica verificación continua de sistemas de software por un equipo independiente, mediante espacios de actividad, actividades, competencias y sus asociaciones. Parte 1/2.
Elaboración propia.

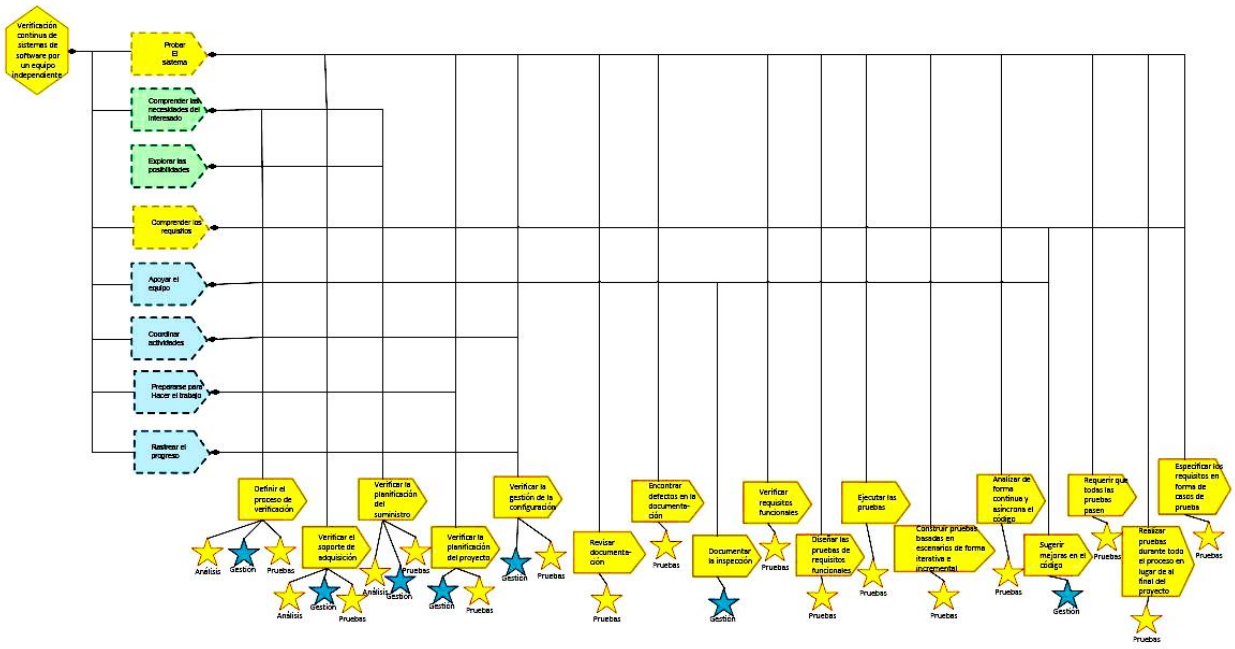
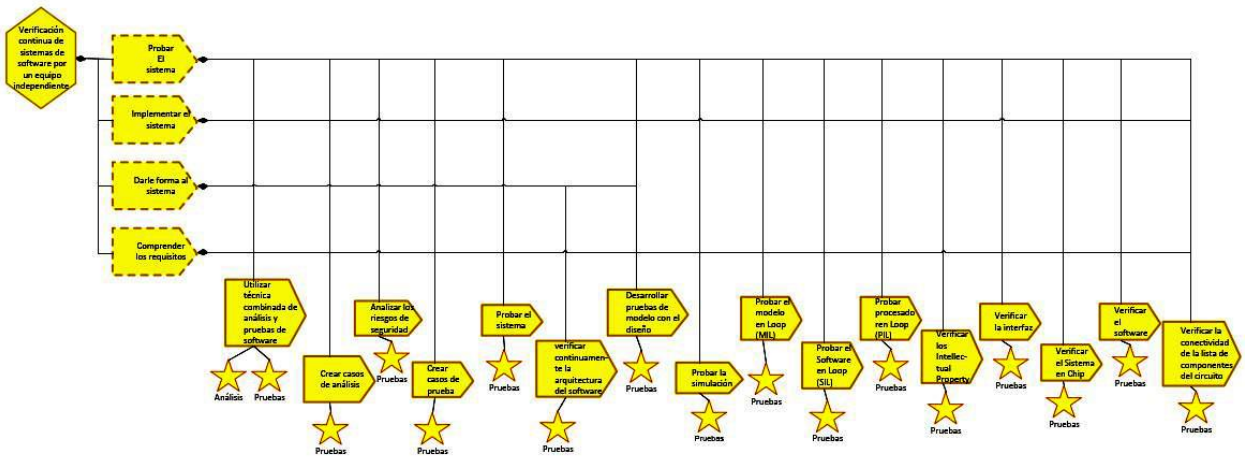


Figura 4-5: Representación gráfica en el núcleo de la Esencia de Semat, de la práctica verificación continua de sistemas de software por un equipo independiente, mediante espacios de actividad, actividades, competencias y sus asociaciones. Parte 2/2.
Elaboración propia.



4.2 Representación gráfica en el núcleo de la Esencia de *Semat* de la práctica planificación continua del proceso de verificación.

La planificación es fundamental para llevar a cabo un buen proceso de verificación, en la revisión de literatura, la mayoría de técnicas de VSS revisadas en esta Tesis realizan planificación para lograr el objetivo de construir sistemas y software con calidad. Sin embargo, cada técnica de verificación y en general cada proyecto puede tener un contexto diferente, por lo que en cada técnica se puede planificar el proceso de VSS de una forma diferente, así que las actividades propuestas se pueden utilizar dependiendo del contexto, En Anexo de esta Tesis se propone además la representación gráfica de la práctica planificación del proceso de verificación con las actividades principales para la planificación del proceso de verificación propuestas por la norma IEEE 829.

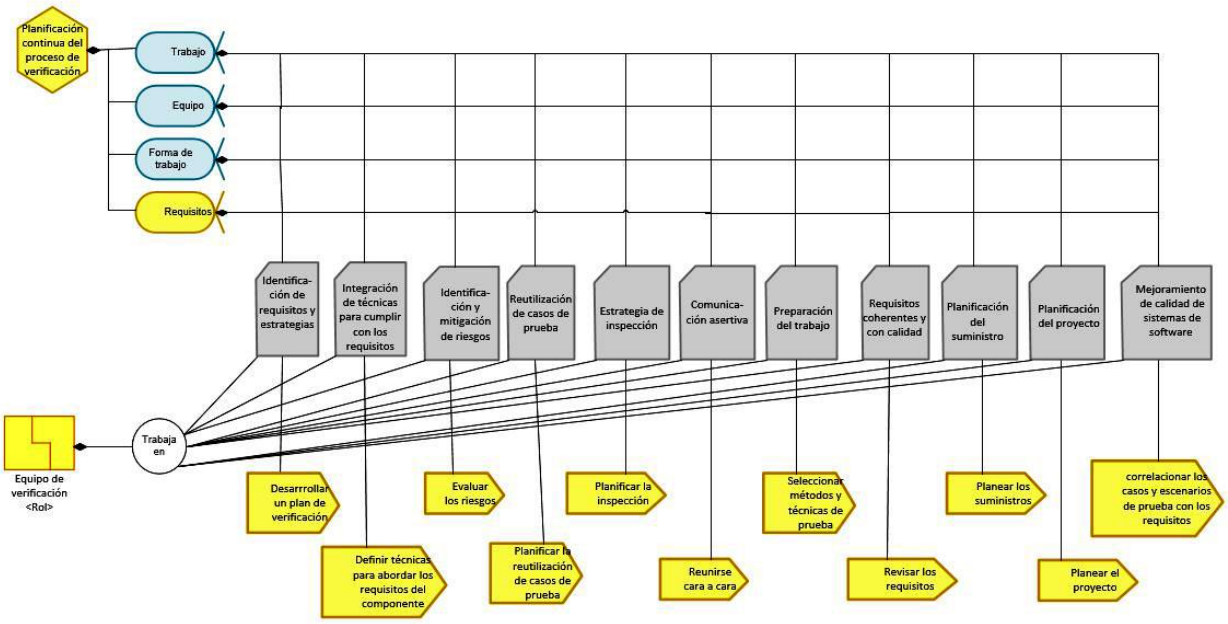
En la Tabla 4-3 se correlacionan los elementos principales identificados en la práctica planificación del proceso de verificación con los elementos básicos del núcleo de la Esencia de *Semat*.

Tabla 4-3: Correlación de los elementos principales identificados en la práctica planificación continua del proceso de verificación con los elementos básicos del núcleo de la Esencia de Semat.
Elaboración propia.

Práctica	Actividad	Producto de trabajo	Alfas		Espacios de actividad		Roles	
Planificación continua del proceso de verificación	1) Desarrrollar un plan de verificación	A) Identificación de requisitos y estrategias	Trabajo	A, B, C, D, E, F, G, H, I, J, K	Apoyar el equipo	1, 2, 3, 4, 5, 6, 7	Equipo de verificación	A, B, C, D, E, F, G, H, I, J, K
	2) Definir técnicas para abordar los requisitos del componente	B) Integración de técnicas para cumplir con los requisitos						
	3)Evaluar los riesgos	C) Identificación y mitigación de riesgos	Forma de trabajo		Prepararse para hacer el trabajo	1, 2, 3, 5, 6, 7, 9, 10		
	4) Planificar la reutilización de casos de prueba	D) Reutilización de casos de prueba						
	5) Planificar la inspección	E) Estrategia de inspección						
	6) Reunirse cara a cara	F) Comunicación asertiva						
	7) Seleccionar métodos y técnicas de prueba	G) Preparación del trabajo	Equipo		Coordinar actividades	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11		
	8) Revisar los requisitos	H) Requisitos coherentes y con calidad						
	9) Planear los suministros	I) Planificación del suministro						
	10) Planear el proyecto	J) Planificación del proyecto	Requisitos	A, B, C, F, G, H, I, J, K	Comprender los requisitos	1, 2, 3, 4, 5, 7, 8, 9, 10, 11		
	11) correlacionar los casos y escenarios de prueba con los requisitos	K) Mejoramiento de calidad de sistemas de software						

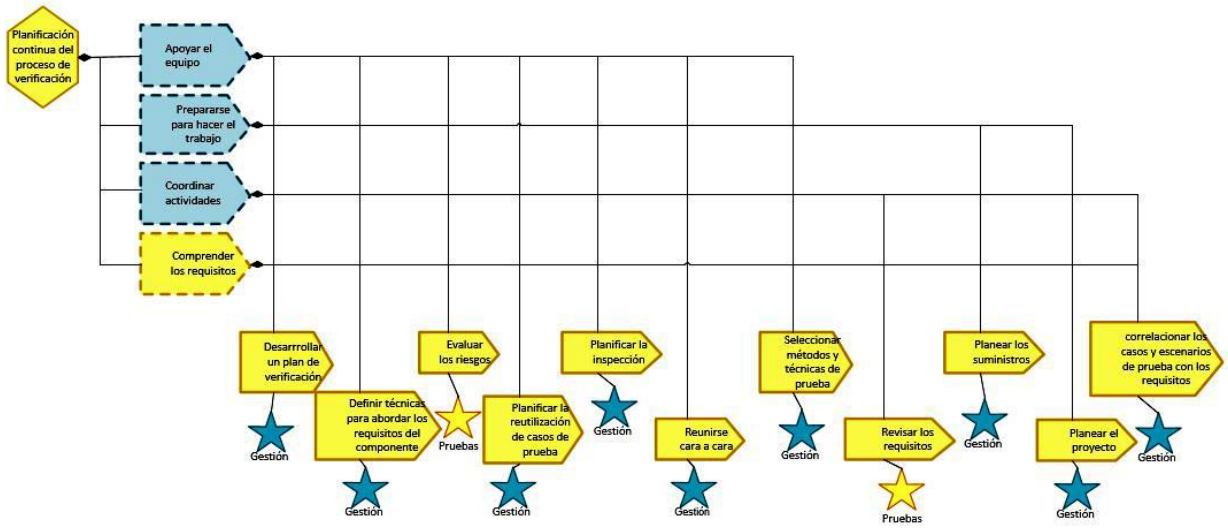
A continuación, se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica planificación continua del proceso de verificación, mediante alfas, productos de trabajo, actividades y roles asociados a las técnicas de VSS revisadas. Véase Figura 4-6.

Figura 4-6: Representación gráfica en el núcleo de la Esencia de Semat, de la práctica planificación continua del proceso de verificación, mediante alfas, actividades, roles y productos de trabajo asociados a las técnicas de VSS revisadas.
Elaboración propia.



En la Figura 4-7 se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica planificación continua del proceso de verificación, mediante espacios de actividad, actividades, competencias y sus asociaciones.

Figura 4-7: Representación gráfica en el núcleo de la Esencia de Semat, de la práctica planificación continua del proceso de verificación, mediante espacios de actividad, actividades, competencias y sus asociaciones.
Elaboración propia.



4.3 Representación gráfica en el núcleo de la Esencia de *Semat* de la práctica detección temprana de errores y defectos.

Esta práctica es fundamental para alcanzar productos de calidad sin malgastar esfuerzos en el proceso de construcción de sistemas y software, en el proceso de verificación si se empieza a verificar desde los requisitos se tendrá un mejor resultado con la satisfacción de las necesidades del cliente. La práctica de verificación detección temprana de errores y defectos también se aplica en todo el ciclo de vida de los sistemas y software en proyectos ágiles.

Los proyectos ágiles tienen iteraciones cortas que permiten al equipo del proyecto recibir comentarios tempranos y continuos sobre la calidad del producto durante todo el ciclo de vida del desarrollo (ISTQB, 2014).

Los probadores ágiles son proactivos. No se sientan a esperar a que les llegue el trabajo.

El equipo de verificación debe trabajar con los programadores para que produzcan algún fragmento de código comprobable desde el principio (Crispin *et al*., 2009).

Hay ciertas prácticas de prueba que se pueden seguir en cada proyecto de desarrollo (ágil o no) para producir productos de calidad. Estos incluyen escribir pruebas por adelantado para expresar el comportamiento adecuado, enfocándose en la prevención, detección y eliminación temprana de defectos, y asegurando que los tipos de prueba correctos se ejecuten en el momento correcto y como parte del nivel de prueba correcto. Los profesionales ágiles pretenden introducir estas prácticas temprano.

Los probadores en proyectos ágiles juegan un papel clave en la guía del uso de estas prácticas de prueba a lo largo del ciclo de vida (ISTQB, 2014). Algunas técnicas utilizadas son TDD, ATDD, BDD.

El concepto de la pirámide de prueba se basa en el principio de prueba de control de calidad y pruebas tempranas (es decir, eliminar defectos lo antes posible en el ciclo de vida). Cuantos más errores pueda solucionar de inmediato, menos deuda técnica generará su aplicación y menos inventario de "defectos" tendrá. Los defectos también son más baratos de solucionar cuanto antes se descubran. Es más barato corregir errores que se encuentran durante el desarrollo que después (ISTQB, 2014).

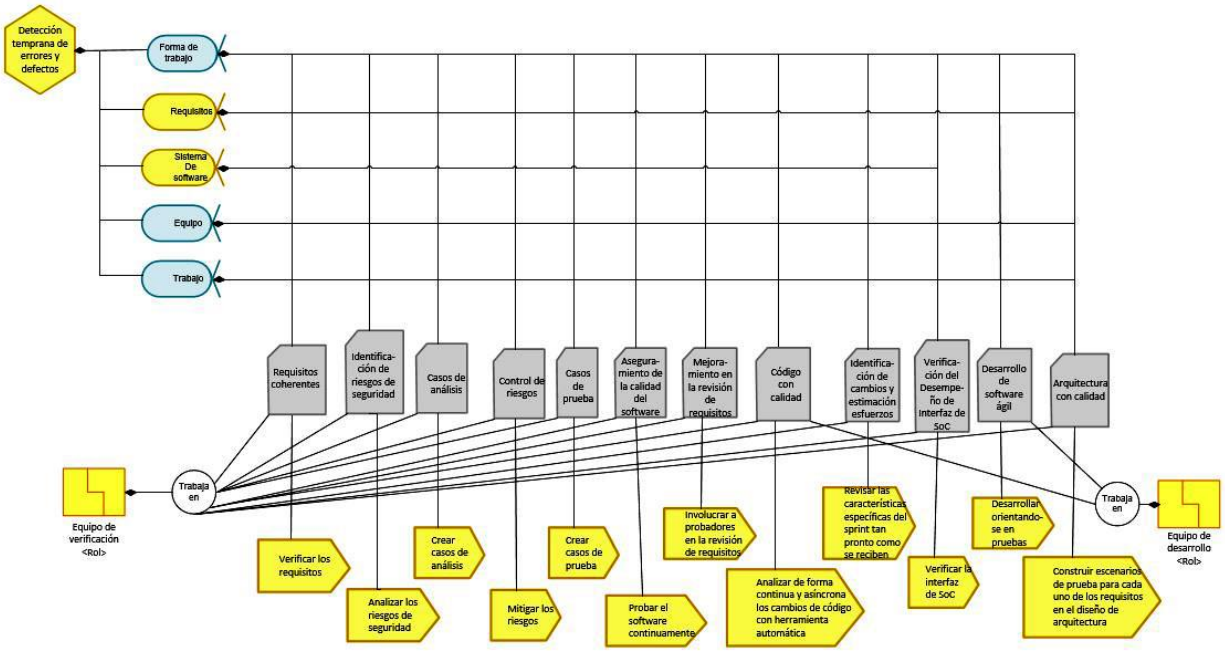
En la Tabla 4-4 se correlacionan los elementos principales identificados en la práctica detección temprana de errores y defectos con los elementos básicos del núcleo de la Esencia de *Semat*.

Tabla 4-4: Correlación de los elementos principales identificados en la práctica detección temprana de errores y defectos con los elementos básicos del núcleo de la Esencia de Semat. Elaboración propia.

Práctica	Actividad	Producto de trabajo	Alfas		Espacios de actividad		Roles	
Detección temprana de errores y defectos	1) Verificar los requisitos	A) Requisitos coherentes	Forma de trabajo	D, E, F, G, H, I, J, K, L	Apoyar el equipo	7, 9	Equipo de verificación	A, B, C, D, E, F, G, H, I, J, L
	2) Analizar los riesgos de seguridad	B) Identificación de riesgos de seguridad						
	3) Crear casos de análisis	C) Casos de análisis	Requisitos	A, B, C, D, E, K, L				
	4) Mitigar los riesgos	D) Control de riesgos						
	5) Crear casos de prueba	E) Casos de prueba						
	6) Probar el software continuamente	F) Aseguramiento de la calidad del software						
	7) Involucrar a probadores en la revisión de requisitos	G) Mejoramiento en la revisión de requisitos	Sistema de software	F, H, J	Probar el sistema	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12		
	8) Analizar de forma continua y asíncrona los cambios de código con herramienta automática	H) Código con calidad						
	9) Revisar las características específicas del sprint tan pronto como se reciben	I) Identificación de cambios y estimación esfuerzos	Equipo	A, B, C, D, E, F, G, H, I, J, L			Equipo de desarrollo	H, K
	10) Verificar la interfaz del Sistema en chip	J) Verificación del Desempeño de Interfaz de SoC			Comprender los requisitos	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12		
	11) Desarrollar orientandose en pruebas	K) Desarrollo de software ágil	Trabajo	A, B, C, D, E, F, G, H, I, J, K, L				
	12) Construir escenarios de prueba para cada uno de los requisitos en el diseño de arquitectura	L) Arquitectura con calidad						

A continuación, se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica detección temprana de errores y defectos, mediante alfas, productos de trabajo, actividades y roles relacionados a las técnicas de VSS revisadas en la literatura, véase Figura 4-8.

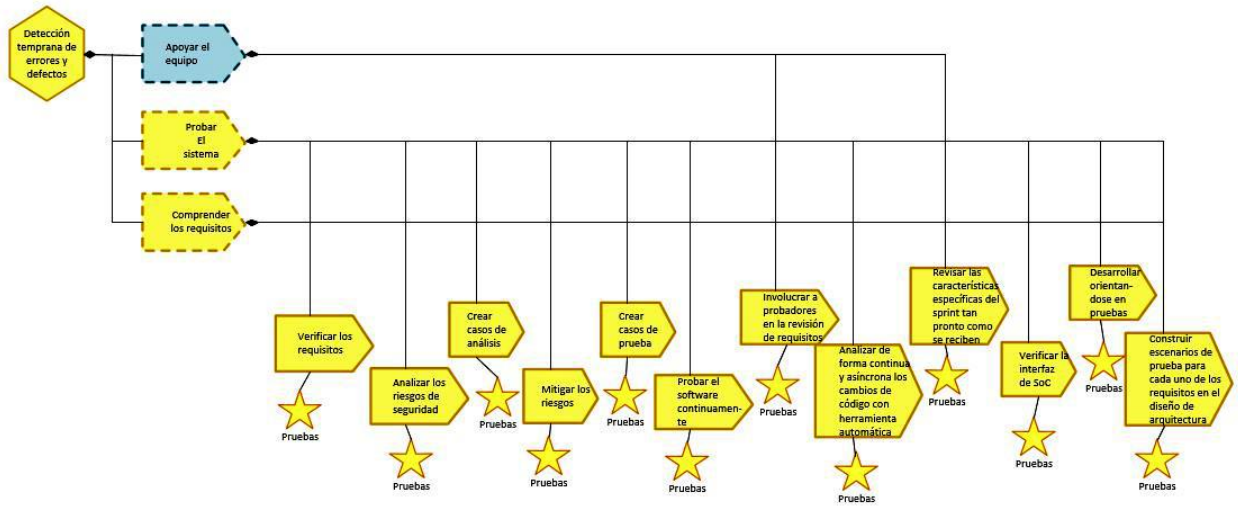
Figura 4-8: Representación gráfica en el núcleo de la Esencia de Semat, de la práctica detección temprana de errores y defectos, mediante alfas, actividades, roles y productos de trabajo.
Elaboración propia.



En la Figura 4-9 se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica detección temprana de errores y defectos, mediante espacios de actividad, actividades, competencias y sus asociaciones relacionados con las técnicas de VSS revisadas en la literatura.

Figura 4-9: Representación gráfica en el núcleo de la Esencia de Semat de la práctica detección temprana de errores y defectos, mediante espacios de actividad, actividades, competencias y sus asociaciones.

Elaboración propia.



4.4 Representación gráfica en el núcleo de la Esencia de *Semat* de la práctica retroalimentación continua entre equipo de verificación y demás miembros del equipo.

Esta práctica es una de las más comunes entre las técnicas de verificación revisadas, la retroalimentación continua tiene que ver mucho con la comunicación, la interacción entre miembros del equipo de trabajo, es muy importante en toda clase de equipos de trabajo. El equipo de verificación debe tener una retroalimentación continua con el equipo de desarrollo y demás miembros del equipo. Las actividades propuestas para esta práctica involucran actividades en común entre las técnicas de verificación revisadas.

De acuerdo con Crispin y Gregory (2009), la retroalimentación temprana y frecuente ayuda al equipo a enfocarse en las características con el mayor valor comercial o riesgo asociado, y estas se entregan primero al cliente. También ayuda a administrar mejor al equipo, ya que la capacidad del equipo es transparente para todos.

Los beneficios de la retroalimentación temprana y frecuente incluyen:

- Evitar malentendidos de requisitos, que pueden no haberse detectado hasta más adelante en el ciclo de desarrollo, cuando son más caros de solucionar.
- Aclarar las solicitudes de funciones del cliente, haciéndolas disponibles para el uso temprano del cliente. De esta manera, el producto refleja mejor lo que el cliente quiere.
- Descubrir (a través de la integración continua), aislar y resolver problemas de calidad de manera temprana.
- Proporcionar información al equipo ágil sobre su productividad y capacidad de entrega.
- Promover el impulso constante del proyecto.

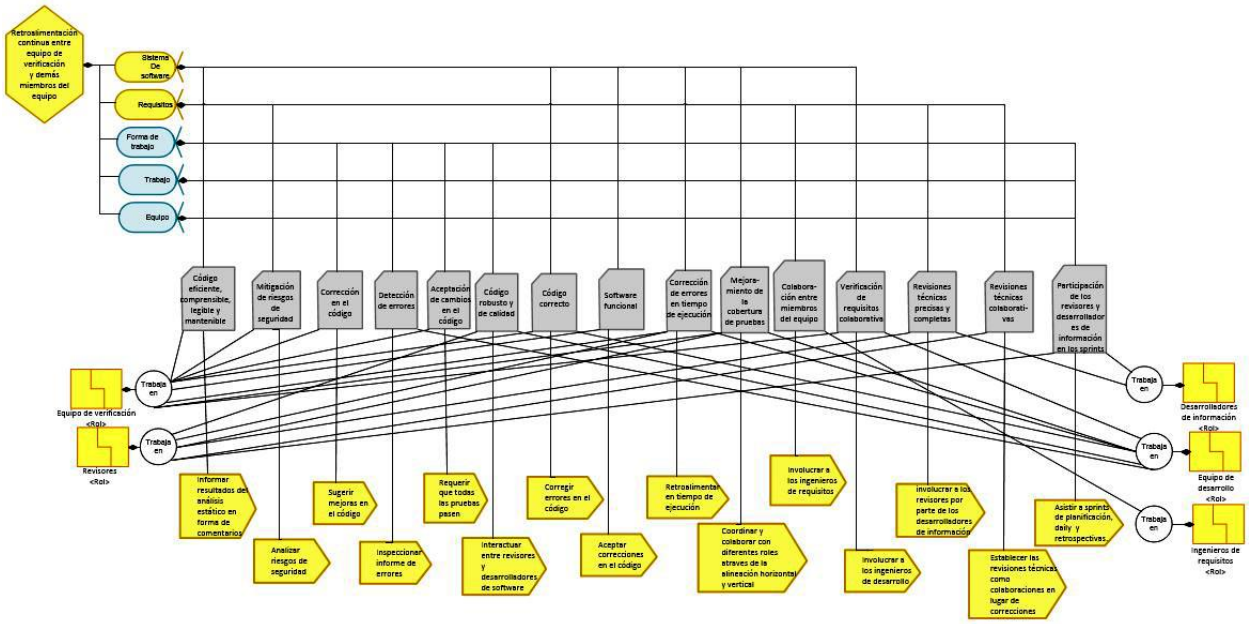
En la Tabla 4-5 se correlacionan los elementos principales identificados en la práctica retroalimentación continua entre equipo de verificación y demás miembros del equipo con los elementos básicos del núcleo de la Esencia de *Semat*.

Tabla 4-5: Correlación de los elementos principales identificados en la práctica retroalimentación continua entre equipo de verificación y demás miembros del equipo con los elementos básicos del núcleo de la Esencia de Semat.
Elaboración propia.

Práctica	Actividad	Producto de trabajo	Alfas		Espacios de actividad		Roles	
Retroalimentación continua entre equipo de verificación y demás miembros del equipo	1) Informar resultados del análisis estático en forma de comentarios	A) Código eficiente, comprensible, legible y mantenible	Sistema de software	A, G, H, I, J, L	Rastrear el progreso	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	Revisores	F, I, J, M, N, Ñ
	2) Analizar riesgos de seguridad	B) Mitigación de riesgos de seguridad						
	3) Sugerir mejoras en el código	C) Corrección en el código	Requisitos	B, H, J, K, L, M, N	Apoyar el equipo	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15		
	4) Inspeccionar informe de errores	D) Detección de errores					Equipo de desarrollo	D, F, G, I, J, L
	5) Requerir que todas las pruebas pasen	E) Aceptación de cambios en el código						
	6) Interactuar entre revisores y desarrolladores	F) Código robusto y de calidad						
	7) Corregir errores en el código	G) Código correcto	Forma de trabajo	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ	Coordinar actividades	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	Desarrolladores de información	M, Ñ
	8) Aceptar correcciones en el código	H) Software funcional						
	9) Retroalimentar en tiempo de ejecución	I) Corrección de errores en tiempo de ejecución						
	10) Coordinar y colaborar con diferentes roles atraves de la alineación horizontal y vertical	J) Mejoramiento de la cobertura de pruebas					Ingenieros de requisitos	K
	11) Involucrar a los ingenieros de requisitos	K) Colaboración entre miembros del equipo	Trabajo	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ	Probar el sistema	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12		
	12) Involucrar a los ingenieros de desarrollo	L) Verificación de requisitos colaborativa						
	13) Involucrar a los revisores por parte de los desarrolladores de información	M) Revisiones técnicas precisas y completas					Equipo de verificación	A, B, C, E, G, H, I, J, M
	14) Establecer las revisiones técnicas como colaboraciones en lugar de correcciones	N) Revisiones técnicas colaborativas	Equipo	A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ	Comprender los requisitos	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15		
	15) Asistir a sprints de planificación, daily y retrospectivas.	Ñ) Participación de los revisores y desarrolladores de información en los sprints						

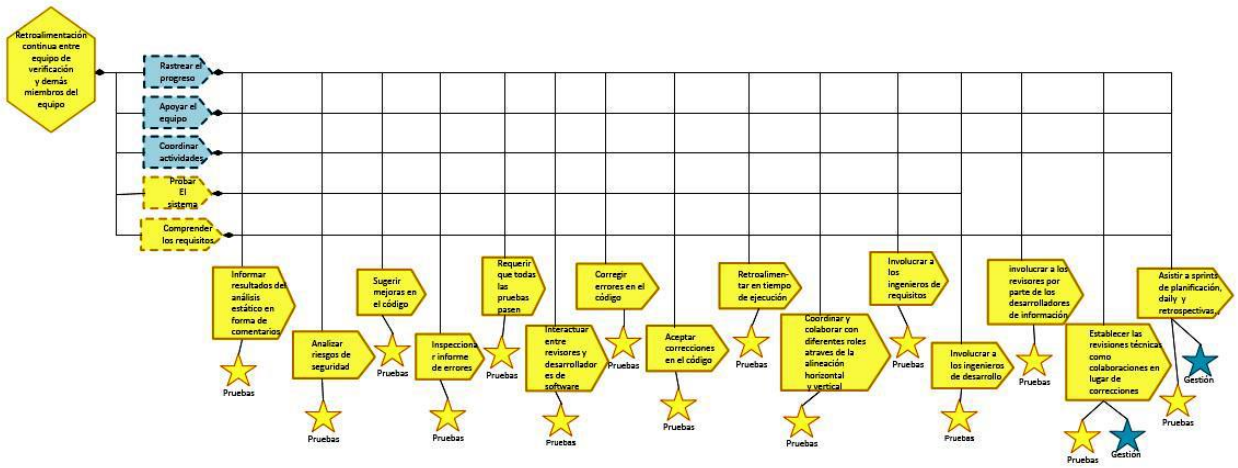
A continuación, se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica retroalimentación continua entre equipo de verificación y demás miembros del equipo, mediante alfas, productos de trabajo, actividades y roles, véase Figura 4-10.

Figura 4-10: Representación gráfica en el núcleo de la Esencia de Semat de la práctica retroalimentación continua entre equipo de verificación y demás miembros del equipo, mediante alfas, actividades, roles y productos de trabajo.
Elaboración propia.



En la Figura 4-11 se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica retroalimentación continua entre equipo de verificación y equipo de desarrollo, mediante espacios de actividad, actividades, competencias y sus asociaciones.

Figura 4-11: Representación gráfica en el núcleo de la Esencia de Semat de la práctica retroalimentación continua entre equipo de verificación y demás miembros del equipo, mediante espacios de actividad, actividades, competencias y sus asociaciones
Elaboración propia.



4.5 Representación gráfica en el núcleo de la Esencia de *Semat* de la práctica utilización continua de herramientas automáticas.

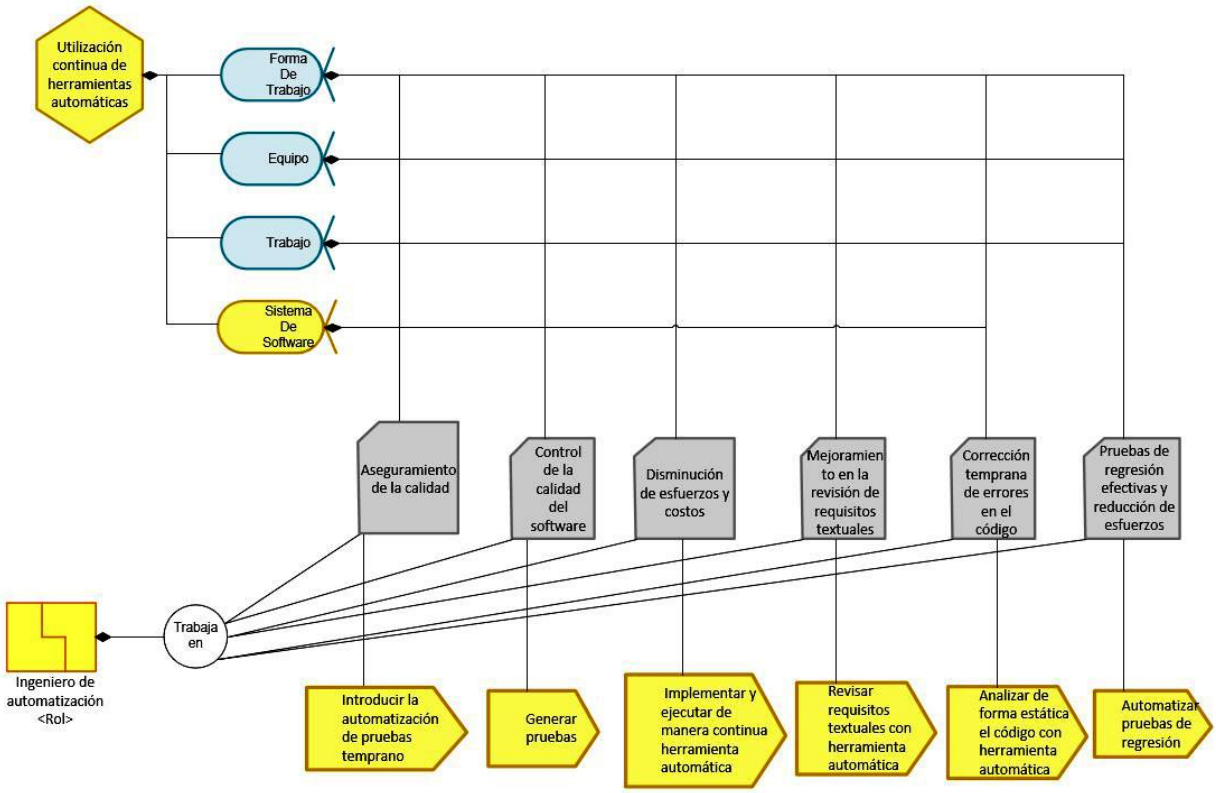
En la Tabla 4-6 se correlacionan los elementos principales identificados en la práctica utilización continua de herramientas automáticas con los elementos básicos del núcleo de la Esencia de *Semat*.

Tabla 4-6: Correlación de los elementos principales identificados en la práctica utilización continua de herramientas automáticas con los elementos básicos del núcleo de la Esencia de Semat.
Elaboración propia.

Práctica	Actividad	Producto de trabajo	Alfas		Espacios de actividad		Roles	
Utilización continua de herramientas automáticas	1) Introducir la automatización de pruebas temprano	A) Aseguramiento de la calidad	Forma de trabajo	A, B, C, D, E, F	Rastrear el progreso	3, 4, 5, 6	Ingeniero de automatización	A, B, C, D, E, F
	2) Generar pruebas	B) Control de la calidad del software						
	3) Implementar y ejecutar de manera continua herramienta automática	C) Disminución de esfuerzos y costos	Equipo	A, B, C, D, E, F	Apoyar el equipo	1, 2, 3, 4, 5, 6		
	4) Revisar requisitos textuales con herramienta automática	D) Mejoramiento en la revisión de requisitos textuales	Trabajo	A, B, C, D, E, F	Coordinar actividades	1, 2, 3, 4, 5, 6		
	5) Analizar de forma estática el código con herramienta automática	E) Corrección temprana de errores en el código			Probar el sistema	1, 2, 3, 5, 6		
	6) Automatizar pruebas de regresión	F) Pruebas de regresión efectivas y reducción de esfuerzos	Sistema de software	A, B, E	Implementar el sistema	1, 2, 3, 5, 6		
					Comprender los requisitos	1, 2, 3, 4, 5, 6		

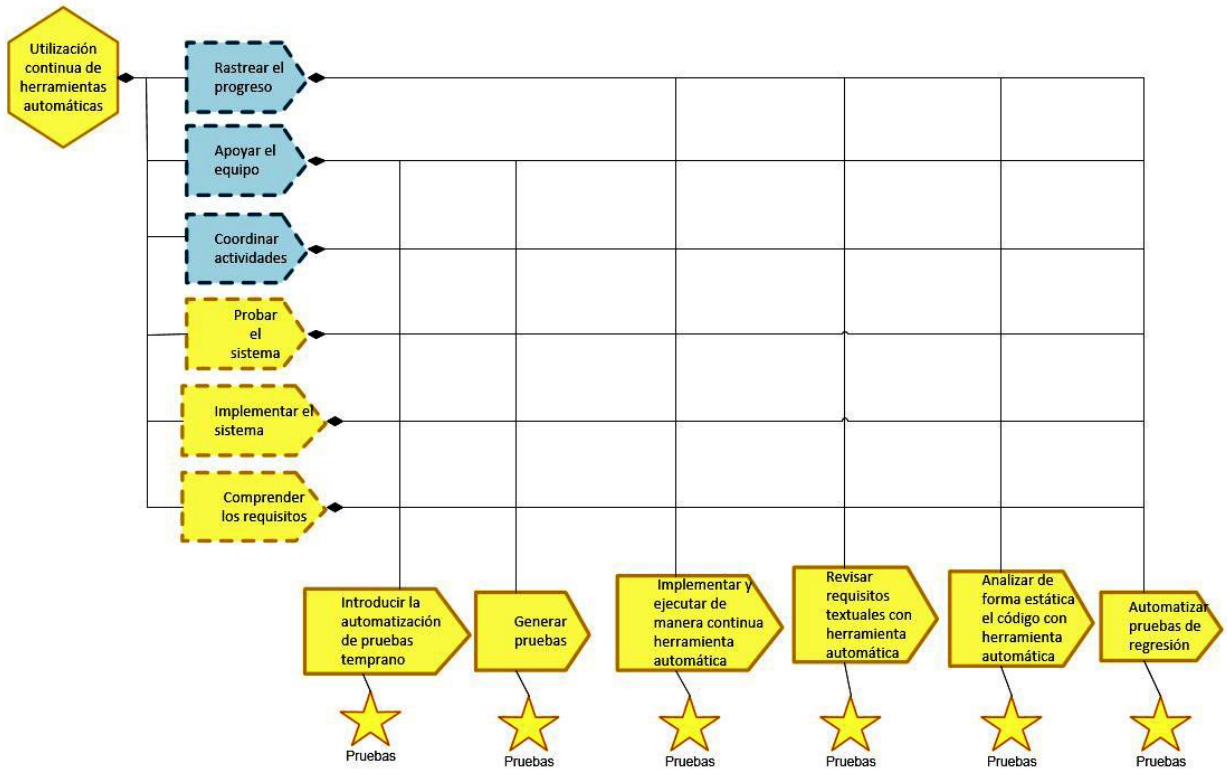
A continuación, se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica utilización continua de herramientas automáticas, mediante alfas, productos de trabajo, actividades y roles, véase Figura 4-12.

Figura 4-12: Representación gráfica en el núcleo de la Esencia de Semat de la práctica utilización continua de herramientas automáticas, mediante alfas, actividades, roles y productos de trabajo.
Elaboración propia.



En la Figura 4-13 se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica utilización continua de herramientas automáticas, mediante espacios de actividad, actividades, competencias y sus asociaciones.

Figura 4-13: Representación gráfica en el núcleo de la Esencia de Semat de la práctica utilización continua de herramientas automáticas, mediante espacios de actividad, actividades, competencias y sus asociaciones.
Elaboración propia.



4.6 Representación gráfica en el núcleo de la Esencia de *Semat* de la práctica realización continua de pruebas de componentes.

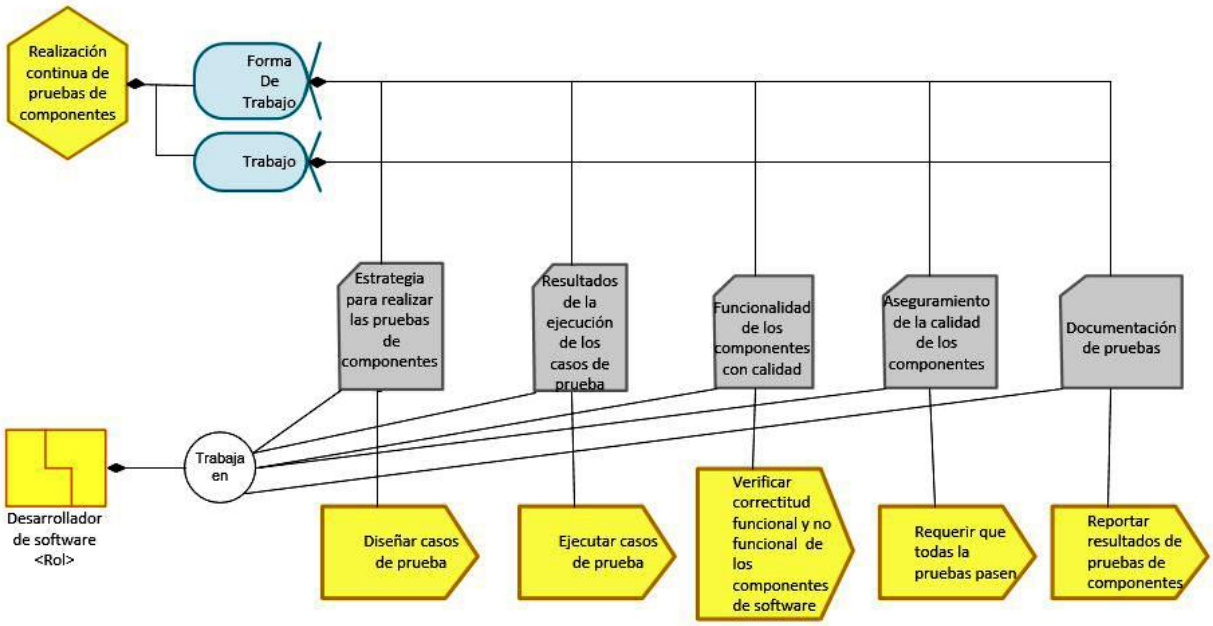
En la Tabla 4-7 se correlacionan los elementos principales identificados en la práctica realización continua de pruebas de componentes con los elementos básicos del núcleo de la Esencia de *Semat*.

Tabla 4-7: Correlación de los elementos principales identificados en la práctica realización continua de pruebas de componentes con los elementos básicos del núcleo de la Esencia de Semat.
Elaboración propia.

Práctica	Actividad	Producto de trabajo	Alfas		Espacios de actividad		Roles	
Realización continua de pruebas de componentes	1) Diseñar casos de prueba	A) Estrategia para realizar las pruebas de componentes	Forma de trabajo	A, B, C, D, E	Rastrear el progreso	3, 4, 5	Desarrollador de software	A, B, C, D, E, F
	2) Ejecutar casos de prueba	B) Resultados de la ejecución de los casos de prueba			Coordinar actividades	3, 4, 5		
	3) Verificar correctitud funcional de los componentes de software	C) Funcionalidad de los componentes con calidad			Prepararse para hacer el trabajo	1		
	4) Requerir que todas las pruebas pasen	D) Aseguramiento de la calidad de los componentes	Trabajo	A, B, C, D, E	Probar el sistema	1, 2, 3, 4, 5		
	5) Reportar resultados de pruebas de componentes	E) Documentación de pruebas			Implementar el sistema	1, 2, 3, 4, 5		
					Comprender los requisitos	1, 3, 4		

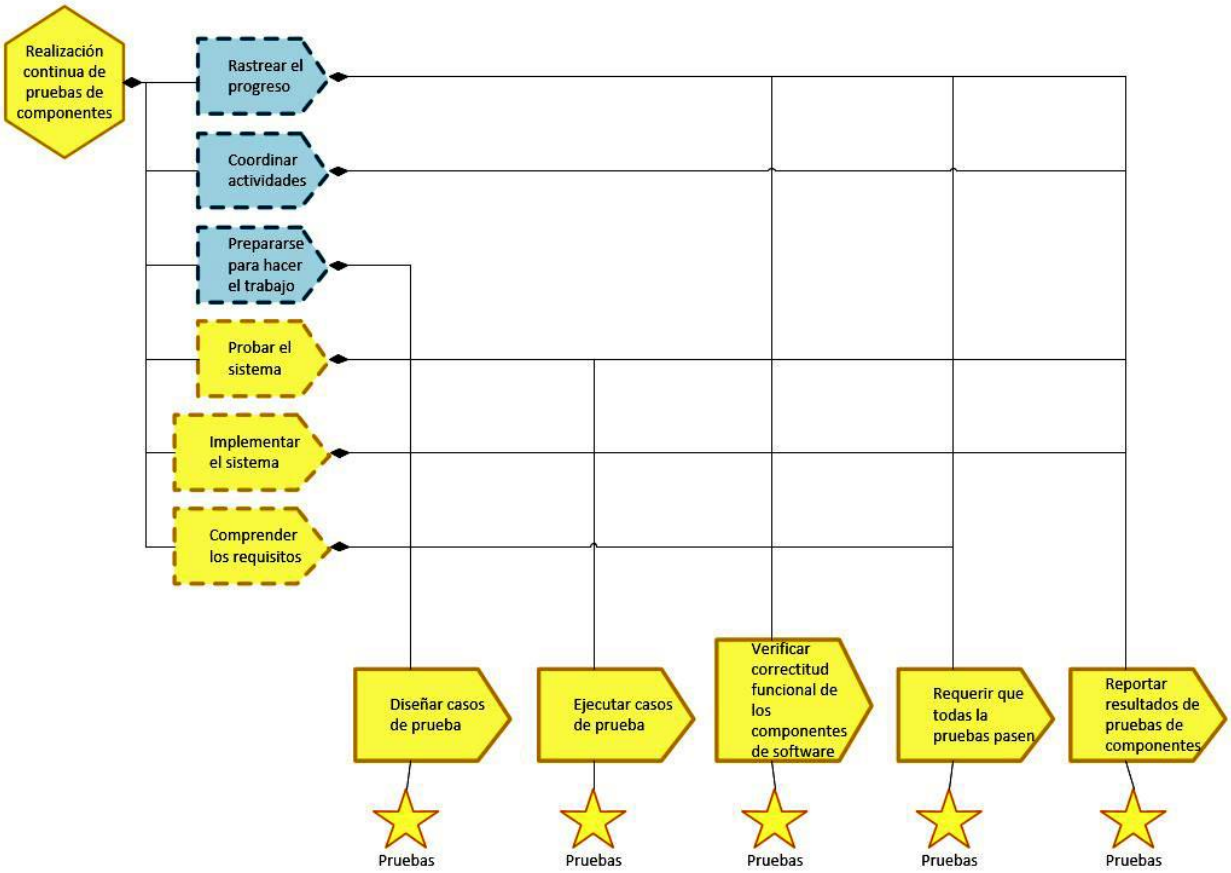
A continuación, se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica realización continua de pruebas de componentes, mediante alfas, productos de trabajo, actividades y roles, véase Figura 4-14.

Figura 4-14: Representación gráfica en el núcleo de la Esencia de Semat de la práctica realización continua de pruebas de componentes, mediante alfas, actividades, roles y productos de trabajo.
Elaboración propia.



A continuación, se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica realización continua de pruebas de componentes, mediante espacios de actividad, actividades, competencias y sus asociaciones, véase la Figura 4-15.

Figura 4-15: Representación gráfica en el núcleo de la Esencia de Semat de la práctica realización continua de pruebas de componentes, mediante espacios de actividad, actividades, competencias y sus asociaciones.
Elaboración propia.



4.7 Representación gráfica de la práctica realización continua de pruebas de integración en el núcleo de la Esencia de *Semat*.

Las pruebas de integración de componentes y las pruebas de integración del sistema deben concentrarse en la integración misma. Por ejemplo, si integra el módulo A con el módulo B, las

pruebas deben centrarse en la comunicación entre los módulos, no en la funcionalidad de los módulos individuales, ya que eso se realiza durante las pruebas de componentes. Si se integra el sistema X con el sistema Y, las pruebas deben centrarse en la comunicación entre los sistemas, no en la funcionalidad de los sistemas individuales, ya que eso se realiza durante las pruebas del sistema. Se aplican los tipos de pruebas funcionales, no funcionales y estructurales (ISTQB, 2018).

Las pruebas de integración de componentes generalmente la realizan los desarrolladores. Las pruebas de integración del sistema son generalmente responsabilidad de los probadores. Idealmente, los probadores que realizan pruebas de integración de sistemas deben comprender la arquitectura del sistema y realizar la planificación de la integración (ISTQB, 2018).

Para simplificar el aislamiento de defectos y detectar defectos temprano, la integración normalmente debería ser incremental (es decir, un pequeño número de componentes o sistemas adicionales a la vez) en lugar de "big bang" (es decir, integrar todos los componentes o sistemas en un solo paso) (ISTQB, 2018).

Cuanto mayor sea el alcance de la integración, más difícil será aislar los defectos de un componente o sistema específico, lo que puede generar un mayor riesgo y tiempo adicional para la resolución de problemas. Esta es una razón por la que la integración continua, donde el software se integra componente por componente (es decir, integración funcional), es una práctica común. Dicha integración continua a menudo incluye pruebas de regresión automatizadas, idealmente en múltiples niveles de prueba (ISTQB, 2018).

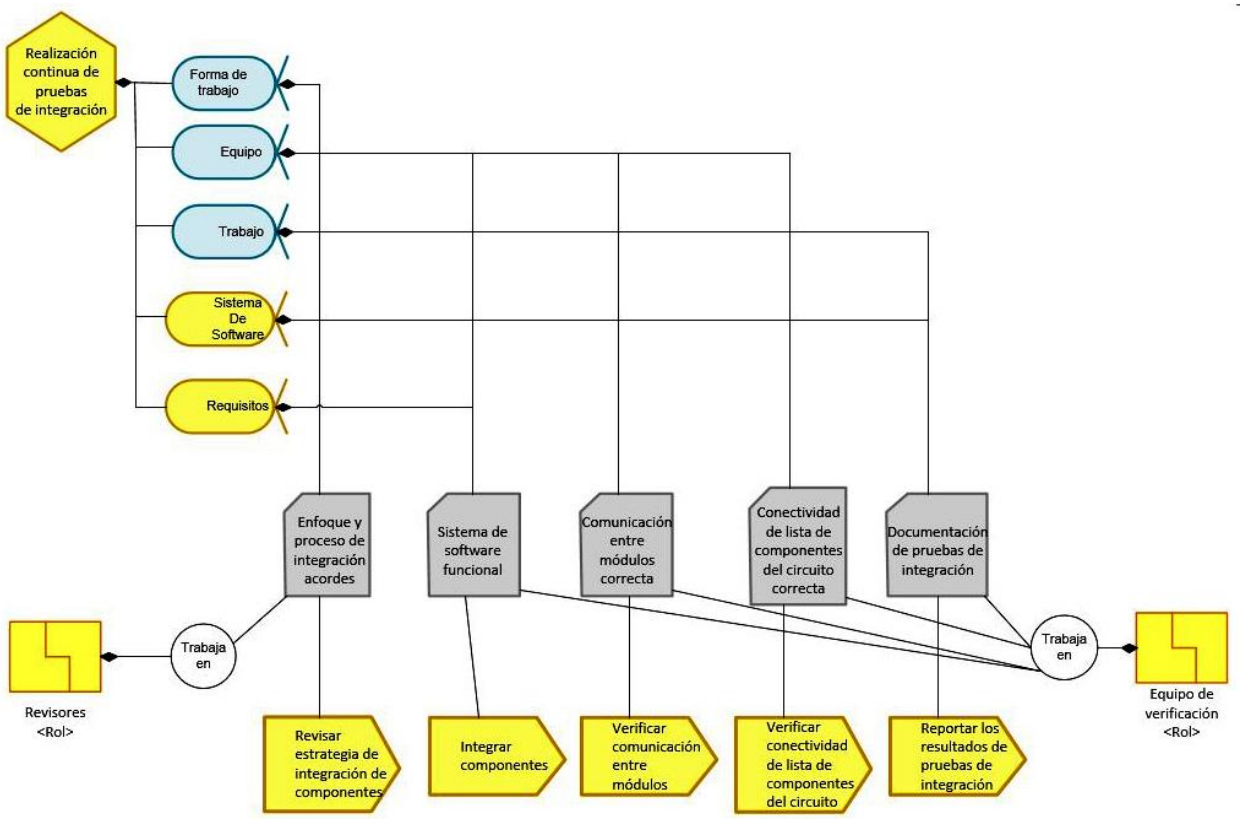
En la Tabla 4-8 se correlacionan los elementos principales identificados en la práctica realización continua de pruebas de integración con los elementos básicos del núcleo de la Esencia de *Semat*.

Tabla 4-8: Correlación de los elementos principales identificados en la práctica realización continua de pruebas de integración con los elementos básicos del núcleo de la Esencia de Semat.
Elaboración propia.

Práctica	Actividad	Producto de trabajo	Alfas		Espacios de actividad		Roles	
Realización continua de pruebas de integración	1) Revisar estrategia de integración de componentes	A) Enfoque y proceso de integración acordes	Forma de trabajo	A	Rastrear el progreso	5	Revisores	A
	2) Integrar componentes	B) Sistema de software funcional	Equipo	A, B, C, D	Probar el sistema	1, 2, 3, 4		
	3) Verificar comunicación entre módulos	C) Comunicación entre módulos correcta	Trabajo	A, B, C, D, E	Comprender los requisitos	1, 2, 3, 4, 5		
	4) Verificar la conectividad de la lista de componentes del circuito	D) Conectividad de lista de componentes del circuito correcta	Sistema de software	A, B, C, D, E	Apoyar el equipo	1, 2, 3, 5	Equipo de verificación	B, C, D, E
	5) Reportar los resultados de pruebas de integración	E) Documentación de pruebas de integración			Coordinar actividades	1, 2, 3, 4, 5		
			Requisitos	B	Prepararse para hacer el trabajo	1		

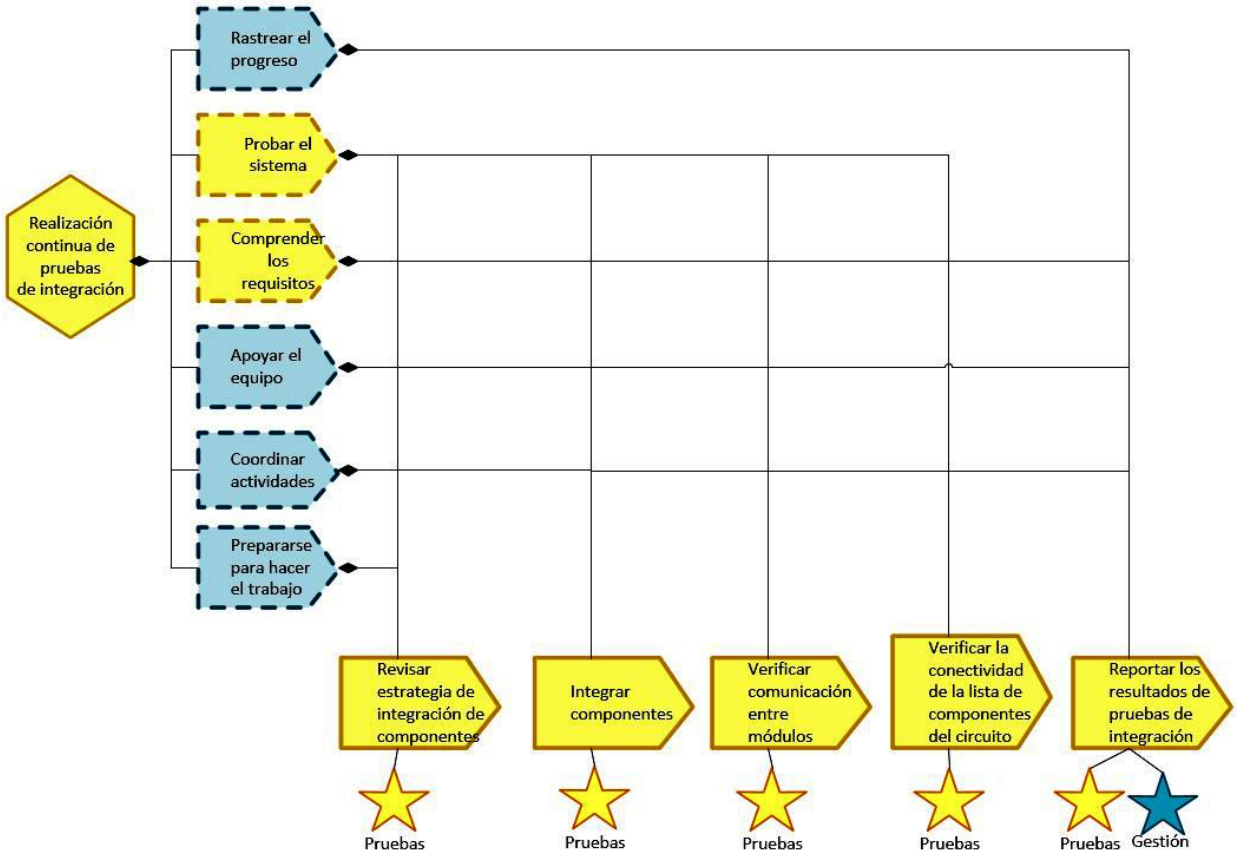
A continuación, se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica realización continua de pruebas de integración, mediante alfas, productos de trabajo, actividades y roles, véase la Figura 4-16.

Figura 4-16: Representación gráfica en el núcleo de la Esencia de Semat de la práctica realización continua de pruebas de integración, mediante alfas, actividades, roles y productos de trabajo.
Elaboración propia.



En la Figura 4-17 se representa gráficamente en el núcleo de la Esencia de *Semat*, la práctica realización continua de pruebas de integración, mediante espacios de actividad, actividades, competencias y sus asociaciones.

Figura 4-17: Representación gráfica en el núcleo de la Esencia de Semat de la práctica realización continua de pruebas de integración, mediante espacios de actividad, actividades, competencias y sus asociaciones.
Elaboración propia.



4.8 Representación gráfica de la práctica revisión en pares de código y requisitos en el núcleo de la Esencia de *Semat*.

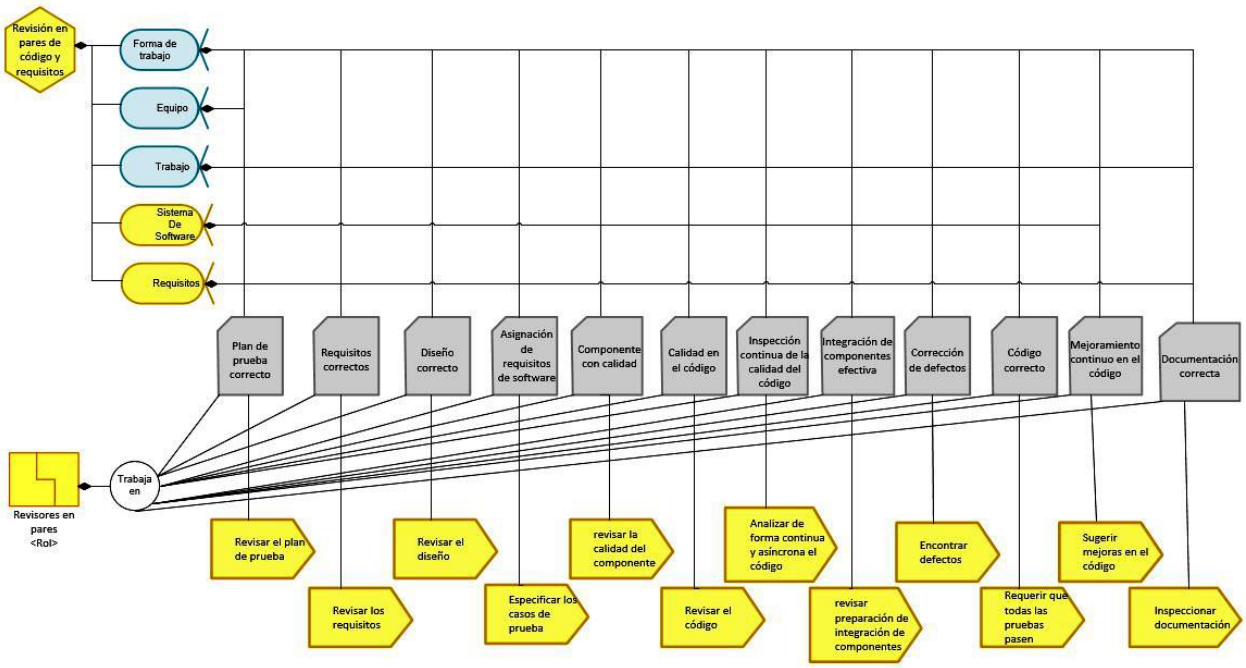
En la Tabla 4-9 se correlacionan los elementos principales identificados en la práctica revisión en pares de código y requisitos con los elementos básicos del núcleo de la Esencia de *Semat*.

Tabla 4-9: Correlación de los elementos principales identificados en la práctica revisión en pares de código y requisitos con los elementos básicos del núcleo de la Esencia de Semat. Elaboración propia.

Práctica	Actividad	Producto de trabajo	Alfas		Espacios de actividad		Roles	
Revisión en pares de código y requisitos	1) Revisar el plan de prueba	A) Plan de prueba correcto	Forma de trabajo	A, B, C, D, E, F, G, H, I, J, K, L	Rastrear el progreso	9, 10, 11	Revisores en pares	A, B, C, D, E, F, G, H, I, J, K, L
	2) Revisar los requisitos	B) Requisitos correctos			Apoyar el equipo	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12		
	3) Revisar el diseño	C) Diseño correcto	Equipo	A	Coordinar actividades	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12		
	4) Especificar los casos de prueba	D) Asignación de requisitos de software						
	5) Revisar la calidad del componente	E) Componente con calidad			Prepararse para hacer el trabajo	1		
	6) Revisar el código	F) Calidad en el código			Probar el sistema	5, 6, 7, 8, 9, 10, 11		
	7) Analizar de forma continua y asíncrona el código	G) Inspección continua de la calidad del código	Trabajo	A, B, C, D, E, F, G, H, I, J, K, L				
	8) Revisar preparación de integración de componentes	H) Integración de componentes efectiva			Implementar el sistema	4, 5, 6, 7		
	9) Encontrar defectos	I) Corrección de defectos	Sistema de software	E, F, G, H, I, J, K				
	10) Requerir que todas las pruebas pasen	J) Código correcto			Darle forma al sistema	3		
	11) Sugerir mejoras en el código	K) Mejoramiento continuo en el código	Requisitos	A, B, D, E, L	Comprender los requisitos	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12		
	12) Inspeccionar documentación	L) Documentación correcta						

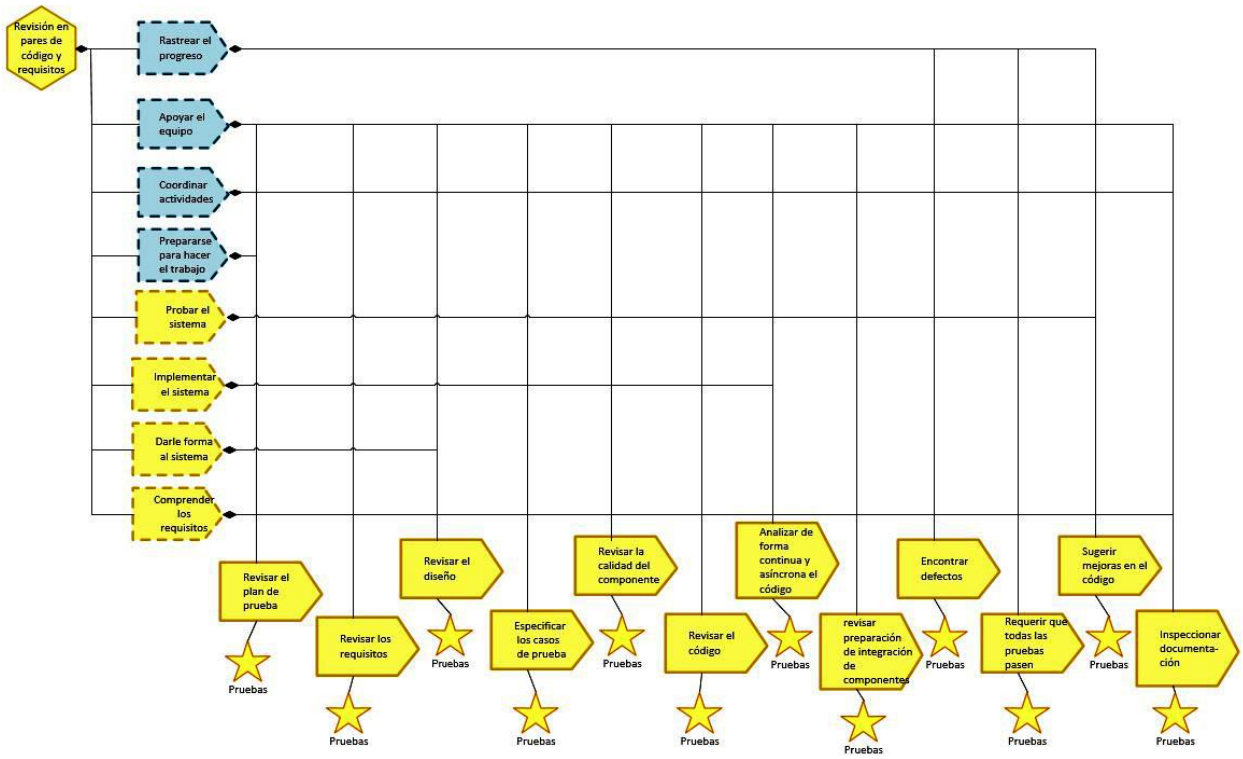
La representación gráfica en el núcleo de la Esencia de *Semat* de la práctica revisión en pares de código y requisitos, mediante alfas, productos de trabajo, actividades y roles, se presenta en la Figura 4-18.

Figura 4-18: Representación gráfica en el núcleo de la Esencia de Semat de la práctica revisión en pares de código y requisitos, mediante alfas, actividades, roles y productos de trabajo.
Elaboración propia.



La representación gráfica en el núcleo de la Esencia de *Semat* de la práctica revisión en pares de código y requisitos, mediante espacios de actividad, actividades, competencias y sus asociaciones, se presenta en la Figura 5-19.

Figura 4-19: Representación gráfica en el núcleo de la Esencia de Semat de la práctica revisión en pares de código y requisitos, mediante espacios de actividad, actividades, competencias y sus asociaciones.
Elaboración propia.



5. Validación

El objetivo de la validación de esta Tesis de Maestría es evaluar las representaciones en *Semat* de las prácticas de verificación de sistemas y software, validar la facilidad de entendimiento de las representaciones en el núcleo de la Esencia de *Semat* de las prácticas de VSS y la coherencia de las representaciones en *Sema*t frente a las prácticas de VSS.

Se realiza un caso de laboratorio, en el cual se realizaron actividades y encuestas a estudiantes de posgrado del curso de verificación y validación de la Universidad Nacional de Colombia sede Medellín y se invitaron estudiantes de doctorado de Ingeniería de sistemas de la Universidad Nacional de Colombia, los cuales tienen experiencia con el lenguaje de *Semat* en estos dos grupos los participantes tienen experiencia laboral en la industria y en la academia, igualmente se realizó la actividad y la encuesta a estudiantes de pregrado de octavo semestre de ingeniería de sistemas en el curso de pruebas de software de la Corporación Universitaria Americana, donde algunos tienen experiencia laboral en la industria.

Para realizar la validación, inicialmente, se realizó una introducción al lenguaje de *Semat*, se presentaron algunas representaciones de mejores prácticas de verificación de sistemas y software en el núcleo de la Esencia de *Semat* entre las técnicas de VSS revisadas, se pidió que explicaran alguna representación en *Semat* de alguna práctica de verificación, para validar su entendimiento y coherencia, realizando una retroalimentación y por último y se realizó la siguiente encuesta:

El objetivo de esta encuesta es validar representaciones en *Semat* de prácticas de verificación de sistemas y software revisadas en la tesis de maestría llamada representación de las mejores prácticas de verificación de sistemas de software en el núcleo de la Esencia de *Semat* elaborada por el estudiante Carlos Puerto García teniendo como director al Dr. Carlos Mario Zapata Jaramillo y como codirector al Dr. Albeiro Espinosa Bedoya.

De antemano muchas gracias por su valiosa participación.

IDENTIFICACIÓN

Sexo	Rango de Edad	Nivel Finalizado	Área
Masculino__ Femenino__	18-25 años __ 26-35 años __ 36-45 años___ 46-60 años___	Bachillerato___ Pregrado___ Especialista__ Maestría ___ Doctorado__	Requisitos__ Desarrollo__ Pruebas__ Gestión__ Docencia__ Otra__

1. Labora en:
   Industria__ Academia __ Estudiante __ del programa de:

2. Años de experiencia laboral

3. Conocimiento en *Semat*

   Si__ No__

4. Asigne una calificación a la facilidad de entendimiento de las representaciones de prácticas de verificación de sistemas y software en *Semat*.

   5. Muy alta

4. Alta
3. Media
2. Baja
1. Muy baja

5. ¿Cómo considera la coherencia de las representaciones en *Semat* frente a las prácticas de verificación de sistemas y software?

   5. Muy alta
   4. Alta
   3. Media
   2. Baja
   1. Muy baja

6. Evalúe las prácticas de verificación de sistemas y software analizadas
   5. Las mejores
   4. Muy buenas
   3. Buenas
   2. Regulares
   1. Malas

7. Si eligió regulares o malas en la anterior pregunta, exprese por qué.

La actividad se realizó inicialmente con un grupo de 7 estudiantes de posgrado de la Universidad Nacional de Colombia sede medellín en el curso de verificación y validación, tres estudiantes de especialización en ingeniería de software y un estudiante de maestría en ingeniería de sistemas y se invitaron a tres estudiantes de doctorado en ingeniería de sistemas, obteniendo los siguientes resultados:

Figura 5-1: Resultados de la identificación encuesta 1, sexo.
Elaboración propia del autor.

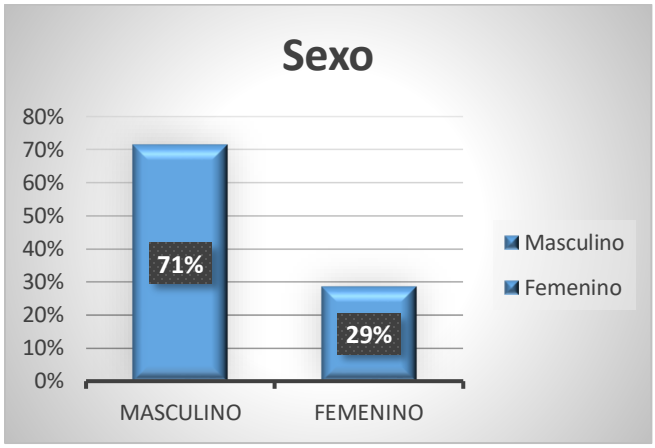


Figura 5-2: Resultados de la identificación encuesta 1, rango de edad.
Elaboración propia del autor.

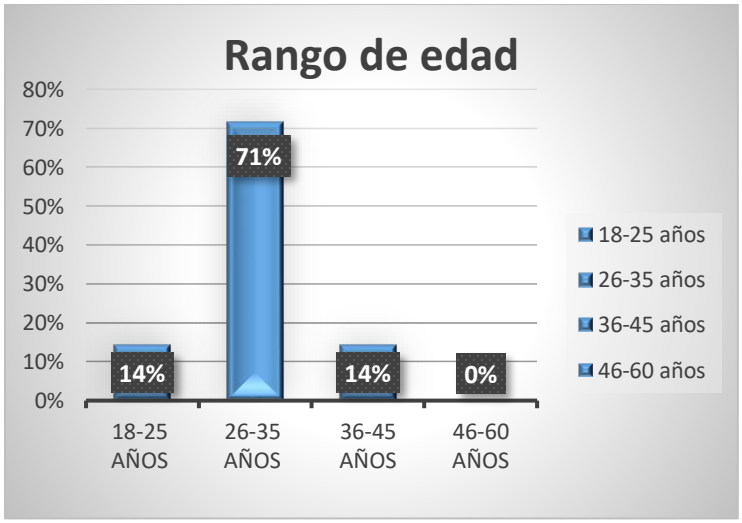


Figura 5-3: Resultados de la identificación encuesta 1, nivel escolaridad finalizado.

Elaboración propia del autor.

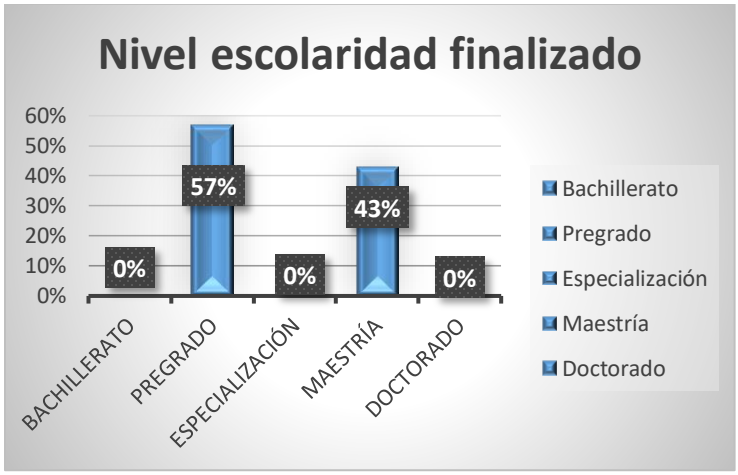


Figura 5-4: Resultados de la pregunta 1 de la encuesta 1. Labora/estudia
Elaboración propia del autor.

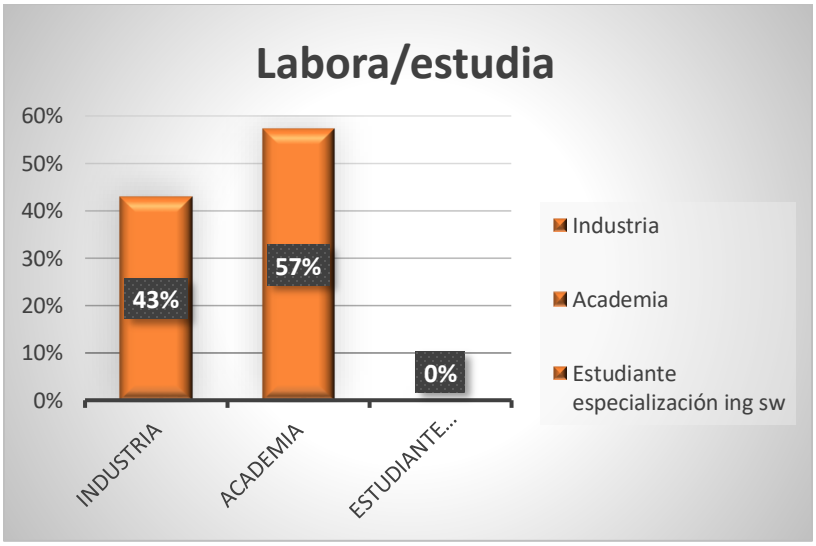


Figura 5-5: Resultados de la pregunta 1 de la encuesta 1. Área laboral
Elaboración propia del autor.

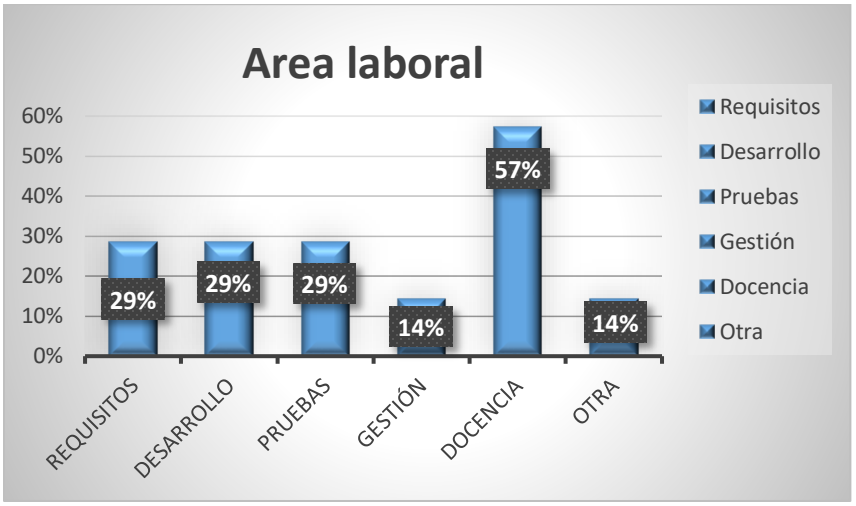


Figura 5-6: Resultados de la pregunta 2 de la encuesta 1. Años de experiencia laboral
Elaboración propia del autor.

Figura 5-7: Resultados de la pregunta 3 de la encuesta 1. Conocimiento en Semat
Elaboración propia del autor.

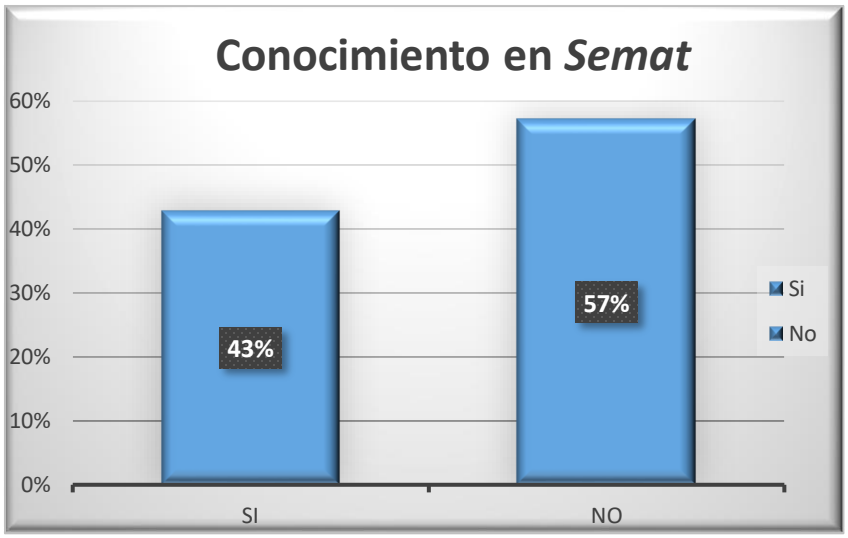


Figura 5-8: Resultados de la pregunta 4 de la encuesta 1. Entendimiento representaciones de las prácticas de VSS
Elaboración propia del autor.

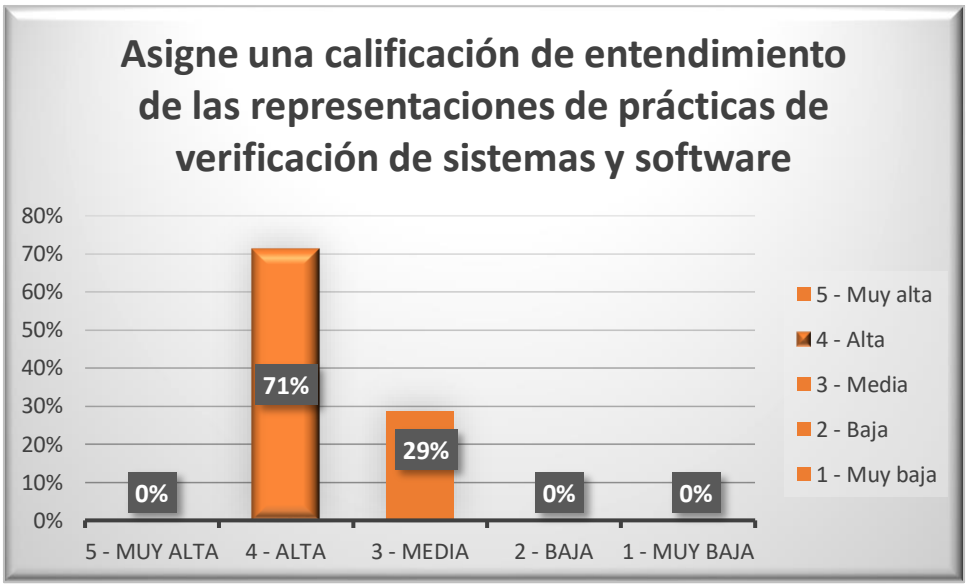


Figura 5-9: Resultados de la pregunta 5 de la encuesta 1. Coherencia de las representaciones en Semat de las prácticas de VSS
Elaboración propia del autor.

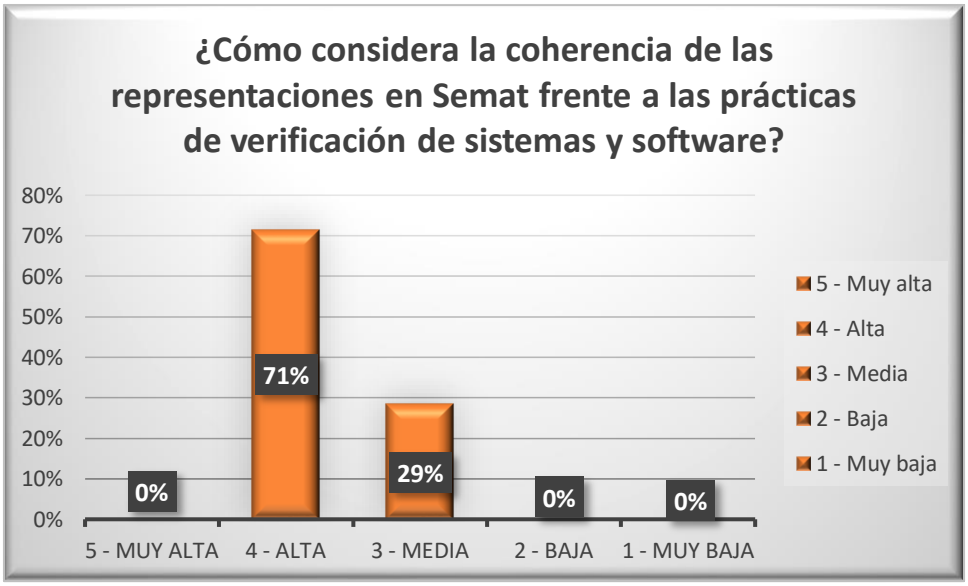
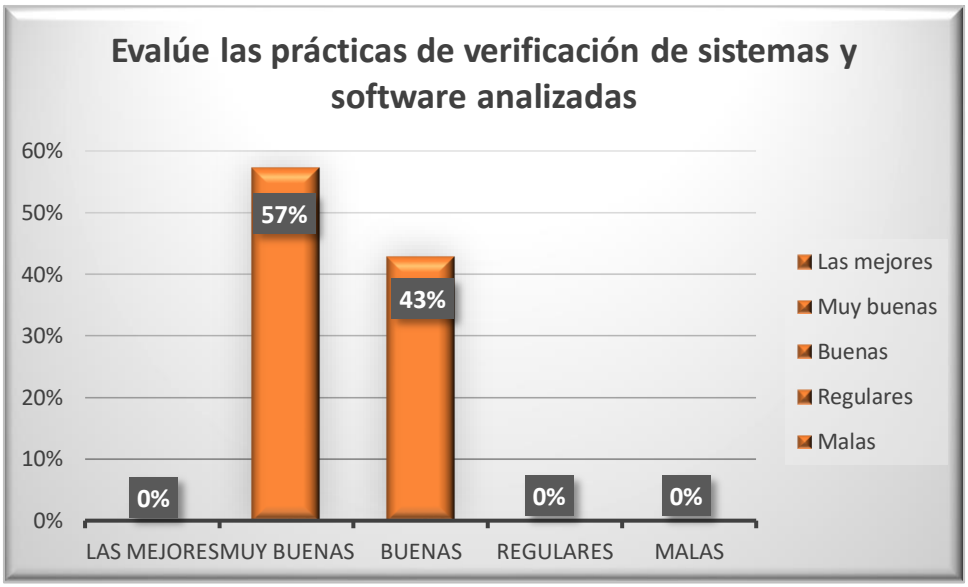


Figura 5-10: Resultados de la pregunta 6 de la encuesta 1. Evaluación de las prácticas de VSS Elaboración propia del autor.



Resultados de la pregunta 7 de la encuesta 1:

Si eligió regulares o malas en la anterior pregunta, exprese por qué.
R/ Nadie eligió regulares o malas

La actividad se realizó por segunda vez con un grupo de 14 estudiantes de pregrado de ingeniería de sistemas de la Corporación Universitaria Americana, obteniendo los siguientes resultados:

Identificación:

Figura 5-11: Resultados de la identificación, encuesta 2. Sexo.
Elaboración propia del autor.

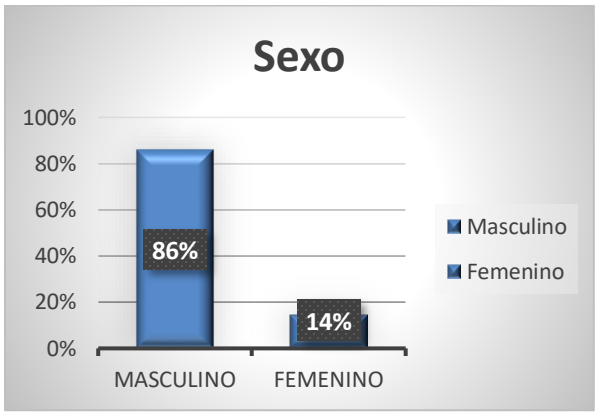


Figura 5-12: Resultados de la identificación, encuesta 2. Rango de edad.
Elaboración propia del autor.

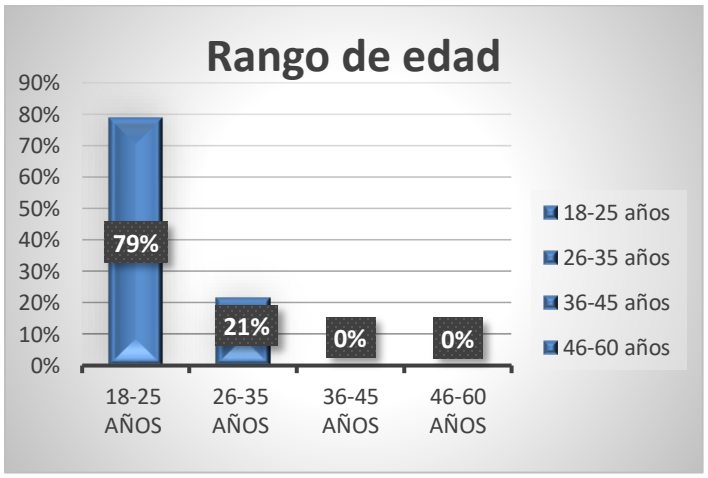


Figura 5-13: Resultados de la identificación, encuesta 2. Nivel de escolaridad finalizado. Elaboración propia del autor.

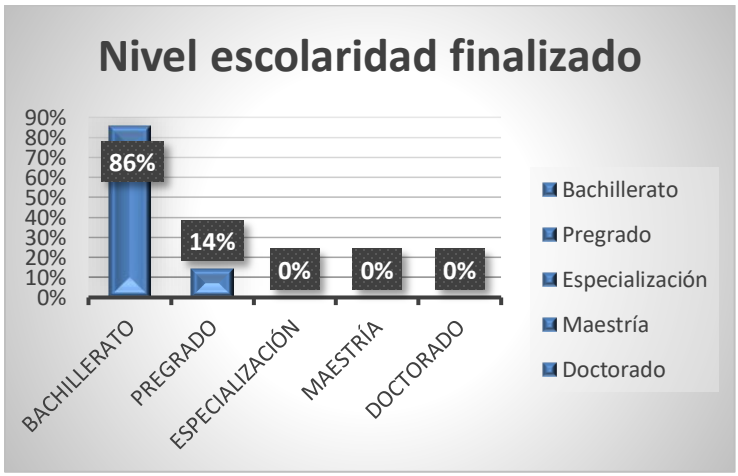


Figura 5-14: Resultados de la pregunta 1 de la encuesta 2. Labora/estudia Elaboración propia del autor.

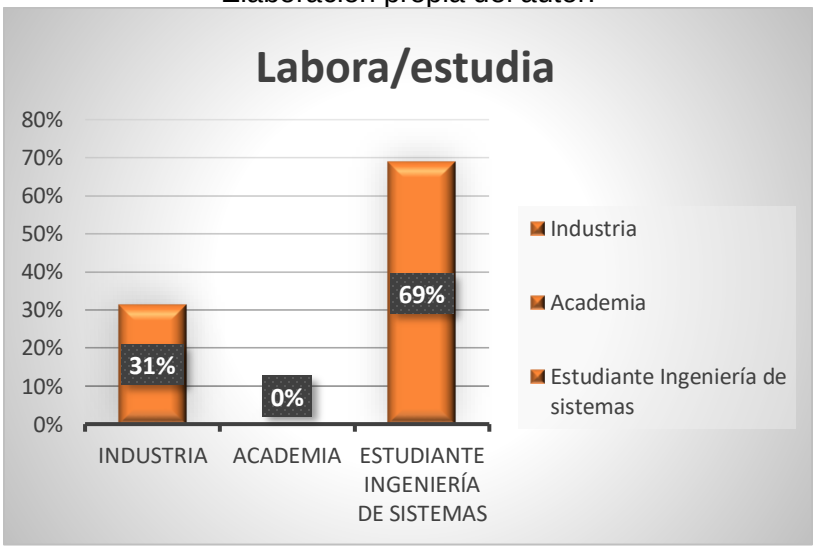


Figura 5-15: Resultados de la pregunta 1 de la encuesta 2. Área laboral.
Elaboración propia del autor.

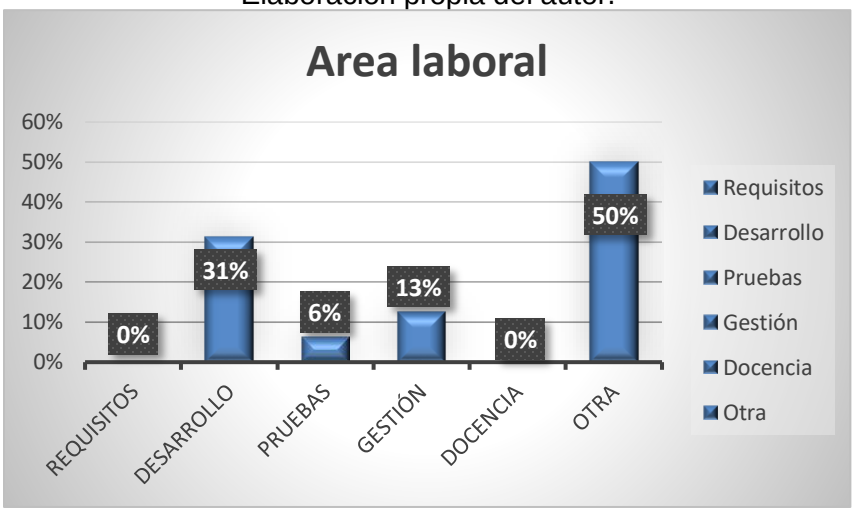


Figura 5-16: Resultados de la pregunta 2 de la encuesta 2. Años de experiencia laboral.
Elaboración propia del autor.

Figura 5-17: Resultados de la pregunta 3 de la encuesta 2. Conocimiento en Semat.
Elaboración propia del autor.

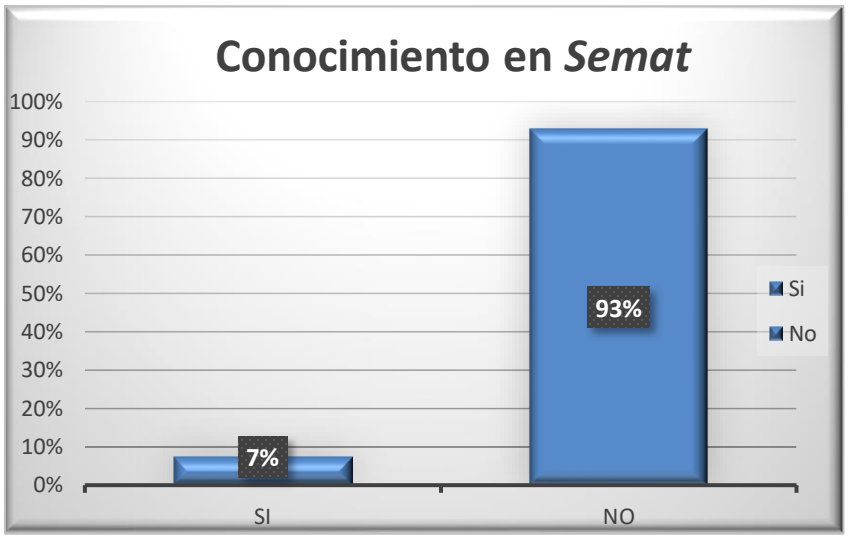


Figura 5-18: Resultados de la pregunta 4 de la encuesta 2. Entendimiento de las representaciones de las prácticas de VSS.
Elaboración propia del autor.

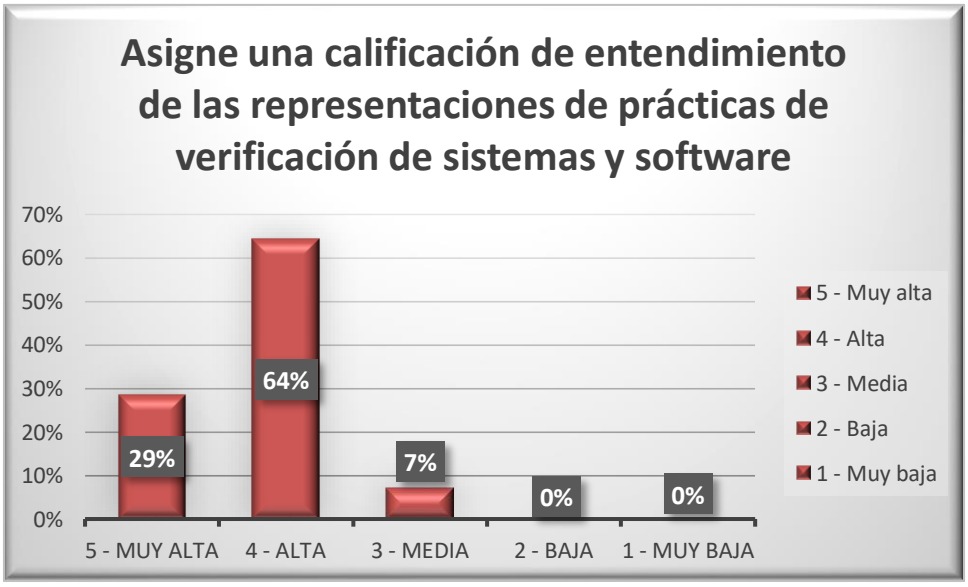


Figura 5-19: Resultados de la pregunta 5 de la encuesta 2. Coherencia de las representaciones en Semat frente a las prácticas de VSS.
Elaboración propia del autor.

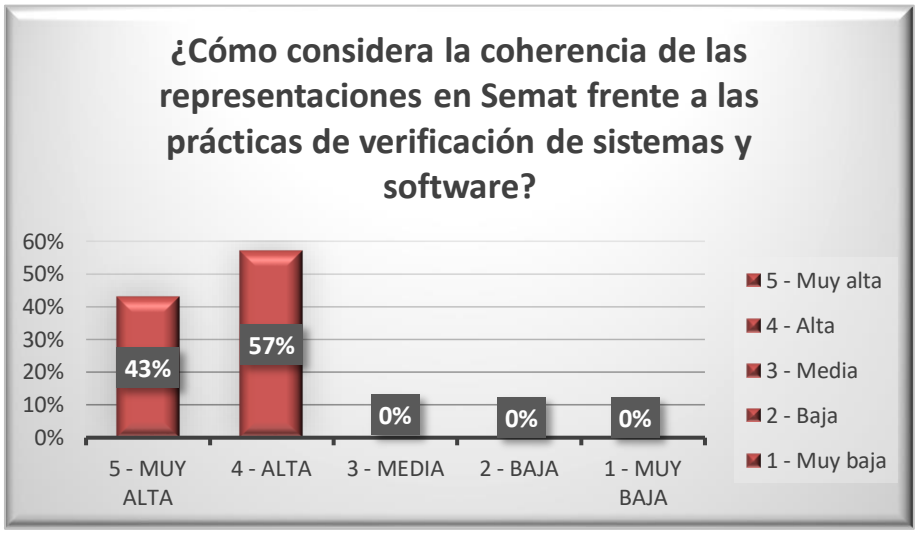
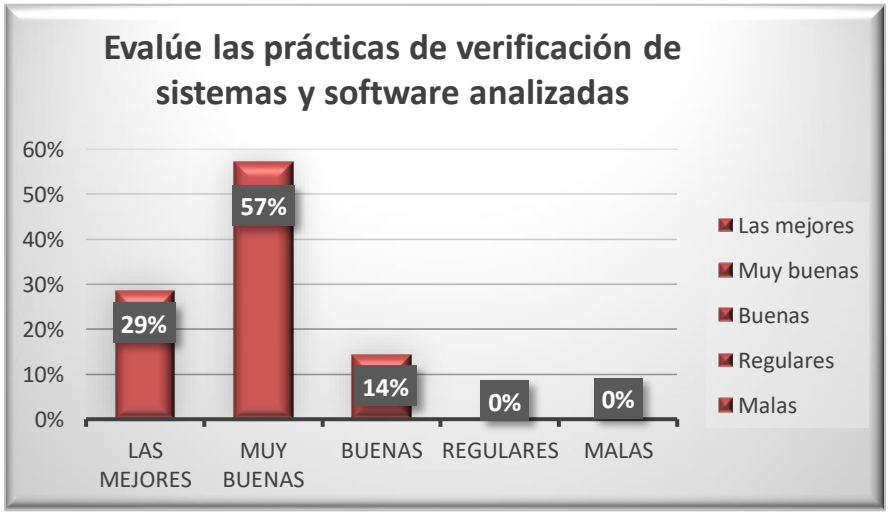


Figura 5-20: Resultados de la pregunta 6 de la encuesta 2. Evaluación de las prácticas de VSS.
Elaboración propia del autor.



Resultados de la pregunta 7 de la encuesta 2:

Si eligió regulares o malas en la anterior pregunta, exprese por qué.
R/ Nadie eligió regulares o malas

Con base en los resultados obtenidos al realizar las dos encuestas llevadas a cabo en las dos actividades, primero con estudiantes de posgrado y luego con estudiantes de pregrado, donde en los dos grupos hay ingenieros que tienen experiencia laboral en la industria y en la academia y conocen los procesos de sistemas y software, se puede concluir que las representaciones de prácticas de verificación de sistemas y software en el núcleo de la Esencia de *Semat* son claras y coherentes, los participantes lograron identificar y explicar los elementos que define *Semat* y relacionarlos con las prácticas de verificación de sistemas y software, evaluaron en un nivel alto el entendimiento y coherencia de las representaciones de prácticas de verificación de sistemas y software en el núcleo de la Esencia de *Semat* y estuvieron de acuerdo en que las prácticas de VSS propuestas son buenas prácticas de verificación de sistemas y software.

6. Conclusiones y trabajo futuro

En esta Tesis de Maestría se realizó una revisión sistemática de literatura de diferentes técnicas de verificación de sistemas de software, entre las cuales se encontraron prácticas en común que se utilizan entre estas técnicas, considerando estas prácticas como mejores o buenas prácticas de VSS.

La VSS se aplica en todo el ciclo de vida de los sistemas de software, es un proceso fundamental para alcanzar productos con calidad, sin embargo, la VSS no tiene un marco común, un lenguaje universal donde se puedan capturar y combinar las mejores prácticas de VSS cuando el contexto de un proyecto determinado lo amerite.

El núcleo de la Esencia de *Semat* permite representar gráficamente prácticas de verificación de sistemas de software que se identificaron en el contexto de diferentes técnicas de VSS.

Las representaciones en *Semat* de las mejores prácticas de verificación de sistemas y software tienen como objetivo apoyar el proceso de verificación de sistemas y software, al poder utilizarlas cuando el contexto del proyecto de software lo requiera.

Cada equipo de trabajo debe identificar cuál práctica y cuales actividades se acoplan mejor en su contexto, el dominio del negocio, el nivel de riesgo, el tipo de producto, quiénes son los usuarios, qué herramientas están usando, cuáles son los costos, etc. Lo que funciona "mejor" para un equipo puede no ser apropiado para lo que use otro equipo.

En las actividades realizadas con estudiantes de pregrado y posgrado donde algunas personas tienen experiencia laboral en la industria y en la academia, se observó que la mayoría de participantes no tenían conocimientos en *Semat*, sin embargo, comprendieron las representaciones de las prácticas de VSS en el lenguaje de *Semat*, logrando explicar las prácticas de VSS, estuvieron de acuerdo que las prácticas propuestas las consideran buenas prácticas y se hizo una retroalimentación en las actividades analizando las representaciones de las prácticas de VSS y sus actividades con los elementos de *Semat*.

Se llega a la conclusión que las representaciones de las prácticas de VSS son claras, comprensibles y coherentes de acuerdo al proceso de desarrollo y verificación de sistemas y software.

Como trabajo futuro se propone revisar buenas prácticas para una determinada técnica de verificación y representarlas en *Semat* (pruebas de software, verificación formal, revisiones, etc.), igualmente para técnicas de validación de sistemas de software.

7. Referencias

Andersson, C. (2003). Exploring the Software Verification and Validation Process with focus on Efficient Fault Detection (Tesis de Maestría). Universidad de Lund, Lund, Suecia.

Antinyan, V., Staron, M. (2017). Proactive reviews of textual requirements. *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Klagenfurt, Austria.

Bjarnason, E., Runeson, P., Borg, M., Unterkalmsteiner, M., Engström, E., Regnell, B., Feldt, R. (2013). Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering*, 19(6), 1809–1855.

Boehm, B. (1984). Verifying and Validating Software Requirements and Design Specifications. *IEEE Transactions on software engineering,* (1), 75-88.

Bourque, P. & Fairley, R. E. (eds.) (2014). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Los Alamitos, CA: IEEE Computer Society.

Bhunia, S., & Tehranipoor, M. (2019). System on Chip (SoC) Design and Test. Hardware Security, 47–79.

Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P., Papakonstantinou, I., Purbrick, J., y Rodriguez, D. (2015). Moving Fast with Software Verification. *NASA Formal Methods.* 7° simposio internacional. Pasadena, CA, USA.

Crispin, L., y Gregory, J. (2009). *Agile Testing A practical guide for testers and agile teams*. New jersey: Addison-Wesley.

Dautovic, A., Plösch, R., y Saft, M. (2011). Automatic Checking of Quality Best Practices in Software Development Documents. *11th International Conference On Quality Software*, Madrid, España.

Espinosa-Bedoya, A., Perez, Y., y Marin, H. (2016). A Review on Verification and Validation for Embedded Software. *IEEE Latin america transactions, 14*, 2339 – 2347.

Garousi, V., y Zhi, J. (2013). A survey of software testing practices in Canada. *The Journal of Systems and Software, 86,* 1354-1376.

Garousi, V., Felderer, M., Karapıçak, Ç., y Yılmaz, U. (2018). Testing embedded software: A survey of the literature. *Information and Software Technology, 104*, 14-45.

Herzner, W., Sieverding, S., Kacimi, O., Böde, E., Bauer, T., y Nielsen, B. (2014). Expressing Best Practices in (Risk) Analysis and Testing of Safety-Critical Systems Using Patterns. *25th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 299-304). IEEE. Nápoles, Italia.

Holdaway, J. (2009), Technical Reviews: Framing the Best of the Best Practices. 2009 *IEEE International Professional Communication Conference*. Waikiki, HI, USA

IEEE (2008). IEEE 12207-2008. *Standard for Systems and software engineering – Software life cycle processes IEEE Computer Society*, IEEE Standards.

IEEE (2008). IEEE 15288-2008. *Standard for Systems and software engineering – System life cycle processes IEEE Computer Society*, IEEE Standards.

IEEE (2008). IEEE 829-2008. *Standard for Software and System Test Documentation – IEEE Computer Society*, IEEE Standards.

IEEE (2008). *IEEE 1028-2008. Standard for Software Reviews and Audits IEEE Computer Society*, IEEE Standards.

IEEE (2014). IEEE 730-2014. *Standard for Software Quality Assurance Processes IEEE Computer Society,* IEEE Standards.

IEEE (2016). *IEEE 1012-2016. Standard for System and Software Verification and Validation IEEE Computer Society*, IEEE Standards.

ISO (2015). Norma internacional ISO 9000 - *Sistemas de gestión de la calidad – Fundamentos y vocabulario*, ISO 9000, 2015.

ISO/IEC (2013). ISO/IEC/IEEE 29119-1:2013 *Software and systems engineering — Software testing* — Part 1: Concepts and definitions. Ginebra, Suiza: International Organization for Standarization.

ISO/IEC (2014). ISO/IEC 25000-2014. *Standard for Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE*, ISO/IEC 25000, 2014.

ISTQB (2014). Foundation Level Extension Syllabus Agile Tester, International Software Testing Qualification Board, 2014.

ISTQB (2018). Foundation Level Syllabus Version 2018, International Software Testing Qualification Board, 2018.

Jacobson, I., Ng, P., McMahon, E., Spence, I., Lidman, S y Zapata, C. (Traductor). (2013). La Esencia de la Ingeniería de Software: El Núcleo de Semat. *Revista Latinoamericana de Ingeniería de Software*, 1(3), 71-78.

Kassab, M., DeFranco, J., y Laplante, P. (2017). Software Testing Practices in Industry: The State of the Practice. *IEEE Software, 34(5)*, 46–52.

Komssi, M., Kauppinen, M., Pyhäjärvi, M., Talvio, J., y Männistö, T. (2010). Persuading Software Development Teams to Document Inspections: Success Factors and Challenges in Practice, *18th IEEE International Requirements Engineering Conference*, Sydney, NSW, Australia.

Merayo, M., y Salaün, G. (2017). Preface: Special issue on software verification and testing. *Journal of Systems and Software,* (132), 317-318.

Mirantes, M. A., Spezio, M., y Wortman, K. A. (2014). Confidence in Spacecraft Software: Continuous Process improvement in requirements verification. *IEEE Aerospace Conference*. (pp. 1-11). Big Sky, MT, USA.

Murphy, B., Wakefield, A., y Friedman, J. (2008). Best Practices for Verification, Validation, and Test in Model- Based Design. *SAE World Congress & Exhibition*, Detroit, USA.

Naik, K., Priyadarshi, T. (2008). *Software testing and quality assurance*. New Jersey: Wiley.

Nielsen, B. (2014). Towards a Method for Combined Model-based Testing and Analysis. *2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD).* 609 – 618.

OMG (2018). “Essence–Kernel and Language for Software Engineering Methods”. V 1.2, Object Management Group.

Probert, R., Chen, Y., Ghazizadeh, B., Sims, P., y Cappa, M. (2003). Formal verification and validation for e-commerce: theory and best practices. *Information and Software Technology*, (45), 763–777.

Sahinoglu, M., Koray, I., y Aktas, M. (2014). Mobile Application Verification: A Systematic Mapping Study. *Lecture Notes in Computer Science Springer*. 9159, 147-163.

Santos, A., Correia, I. (2015). Mobile Testing in Software Industry Using Agile: Challenges and Opportunities**.** *IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, Graz, Austria.

Schmidt, R. (2013). Software Engineering: Architecture-driven Software Development. MA, USA: Morgan Kaufmann.

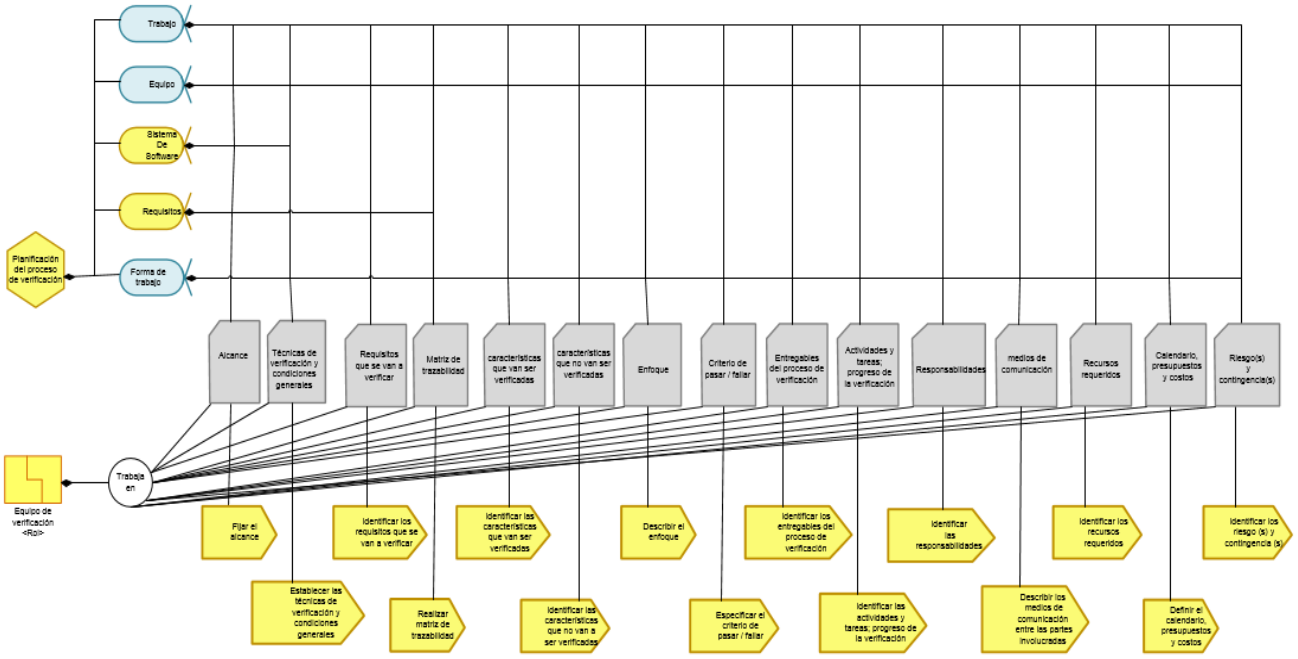
Simonette, M., Lago, L. y Spina, E. (2014). Extending SEMAT kernel to deal with developer error, *INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING, 8,* 253-258.

Somerville, I. (2011). *Software Engineering*. Boston: Addison-Wesley.

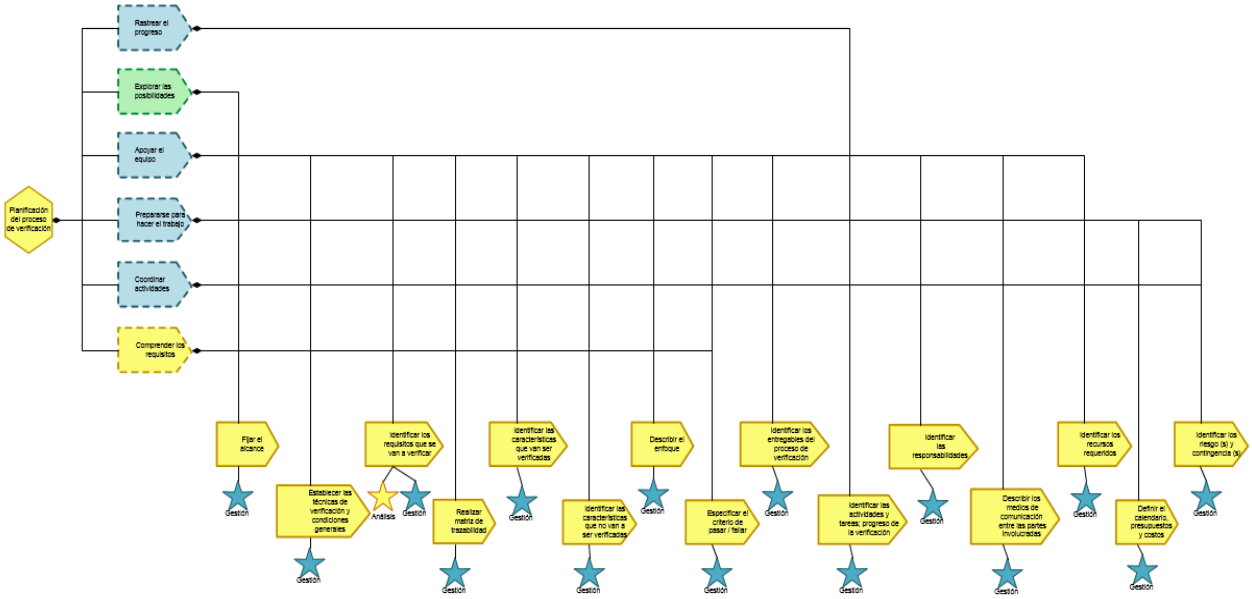
Zapata, C. y Jacobson, I. (2014). A first course in software engineering method and theory, *Dyna, 81*(183), 231–241.

Anexos

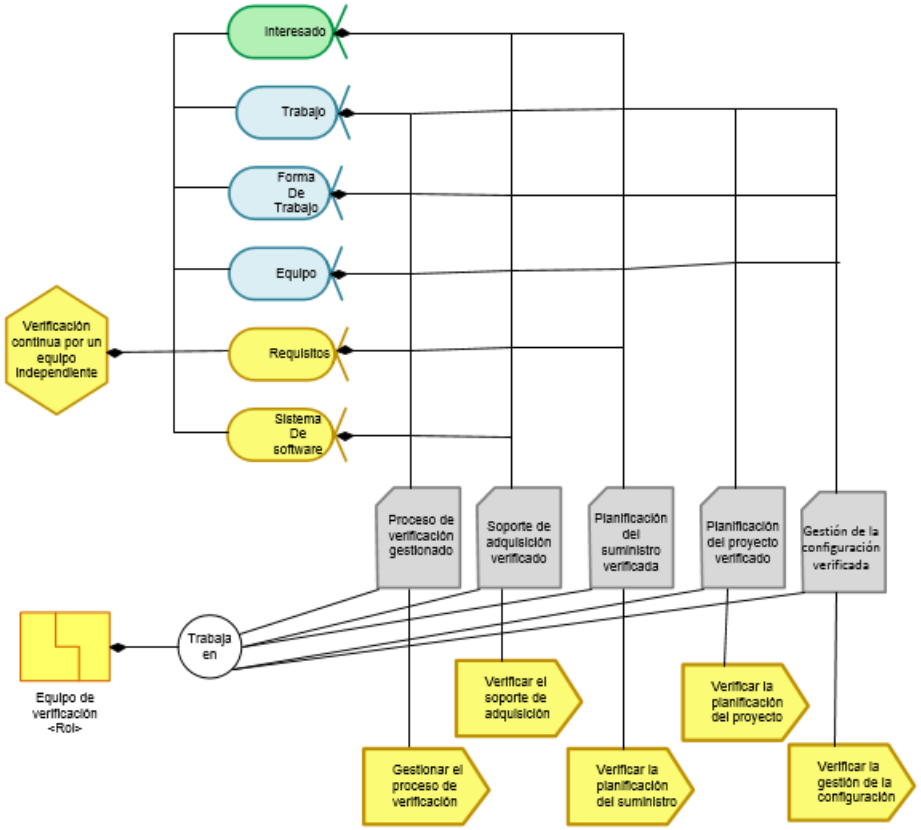
Anexo 1: Representación gráfica en el núcleo de la Esencia de Semat, de la práctica planificación del proceso de verificación basada en la norma IEEE 829, mediante alfas, actividades, roles y productos de trabajo.
Elaboración propia del autor.



Anexo 2: Representación gráfica en el núcleo de la Esencia de Semat, de la práctica planificación del proceso de verificación basada en la norma IEEE 829, mediante espacios de actividad, actividades, competencias y sus asociaciones.
Elaboración propia del autor.



Anexo 3: Representación gráfica en el núcleo de la Esencia de *Sema*t, de la práctica verificación continua por un equipo independiente basada en la norma IEEE 1012, mediante los alfas, actividades, productos de trabajo, roles y sus asociaciones.
Elaboración propia del autor.



Anexo 4: Representación gráfica en el núcleo de la Esencia de *Sema*t, de la práctica verificación continua por un equipo independiente basada en la norma IEEE 1012, mediante espacios de actividad, actividades, competencias y sus asociaciones.
Elaboración propia del autor.